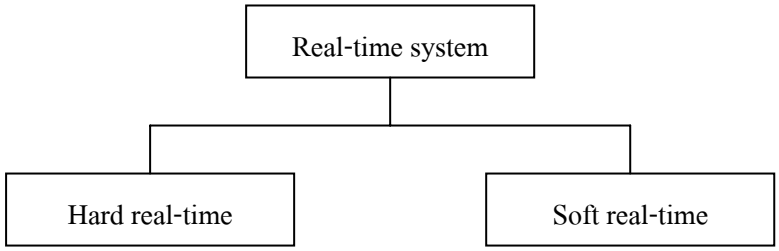
บทที่ 2

การศึกษาค้นคว้าข้อมูล

การออกแบบและพัฒนาระบบปฏิบัติการเชิงเวลาจริง uC/OS-II บนระบบคอมพิวเตอร์ฝังตัวให้ตระหนักถึงกำลังงานที่ใช้ต้องศึกษาค้นคว้าข้อมูลที่เกี่ยวข้องก่อน ด้วยเหตุนี้จึงได้ทำการศึกษาบทความและทฤษฎีที่เกี่ยวข้องเพื่อเป็นข้อมูลประกอบการวิจัยซึ่งจะกล่าวต่อไปในรายละเอียดของบทนี้

2.1 ระบบเชิงเวลาจริง

ระบบเชิงเวลาจริงเป็นระบบที่ต้องการความถูกต้องทั้งผลลัพธ์ที่ได้และเวลาที่ใช้ในการทำงาน โดยต้องทำงานให้เสร็จทันตามเวลาที่กำหนด เป็นระบบที่ฮาร์ดแวร์และซอฟต์แวร์ต้องสนับสนุนการทำงานเชิงเวลาจริงและสามารถสนับสนุนงานที่มีการทำงานพร้อมกันหลาย ๆ งานได้ โดยไม่ส่งผลกระทบต่อเวลาที่ใช้ในการทำงานกับเวลาที่กำหนดไว้ นั่นคือ จะต้องทำงานให้เสร็จภายในเวลาที่แย่ที่สุดในการทำงานของแต่ละงาน (Worst Case Execution Time : WCET) แอพลิเคชันที่ใช้ในระบบเชิงเวลาจริง ได้แก่ ระบบความปลอดภัย ระบบควบคุมและสั่งการหุ่นยนต์ การสื่อสาร และมัลติมีเดีย เป็นต้น ระบบเชิงเวลาจริงแบ่งออกเป็น 2 ประเภท คือ hard real-time system และ soft real-time system ดังภาพประกอบ 2-1



ภาพประกอบ 2-1 ประเภทของระบบเชิงเวลาจริง

Hard real-time system เป็นระบบที่สนับสนุนงานเชิงเวลาจริง ต้องทำงานให้เสร็จทันตามเวลาที่กำหนด หากใช้เวลาในการทำงานเกินเวลาที่กำหนดไว้จะก่อให้เกิดผลกระทบร้ายแรงได้ ดังนั้นระบบนี้เป็นระบบที่ต้องการการทดสอบที่แน่นอน และต้องทนต่อความผิดพลาดได้สูงโดยจะต้องมีวิธีการแก้ปัญหาที่เหมาะสมเมื่อมีข้อผิดพลาดเกิดขึ้น

Soft real-time system เป็นระบบที่สนับสนุนการทำงานเชิงเวลาจริงเช่นเดียวกับ hard real-time system แต่ระบบนี้มีข้อจำกัดน้อยกว่าเนื่องจากมีความยืดหยุ่นในเรื่องของเวลาในการทำงานมากกว่า นั่นคือ เมื่อใช้เวลาในการทำงานเกินเวลาที่กำหนดเพียงเล็กน้อยก็ยังสามารถยอมรับได้ไม่มีผลกระทบร้ายแรงเกิดขึ้น

โดยทั่วไปการทำงานเชิงเวลาจริงรันอยู่บนระบบเชิงเวลาจริงและทำงานร่วมกับระบบปฏิบัติการเชิงเวลาจริงซึ่งระบบดังกล่าวมีคุณสมบัติพิเศษเฉพาะดังนี้

คุณลักษณะของระบบเชิงเวลาจริง

- ทำงานอยู่ภายใต้ข้อจำกัดของเวลา นั่นคือจะต้องทำงานให้เสร็จทันตามเวลาที่กำหนดไว้ในแต่ละงาน
- สามารถคืนสภาพปกติได้เมื่อมีปัญหา
- ทำงานเป็นกลุ่มและมีการกระจายงานกันทำ
- การติดต่อสื่อสารระหว่างงานแต่ละงานไม่ขึ้นต่อกัน
- มีการปล่อยพลังงานในรูปแบบของความร้อนสูง เนื่องจากระบบเชิงเวลาจริงโดยมากแล้วมักจะทำงานต่อเนื่องเป็นระยะเวลานานดังนั้นกำลังงานที่ใช้จะสูญเสียอยู่ในรูปแบบของความร้อน
- ต้องทราบเวลาที่แย่ที่สุด (WCET) และต้องทำงานให้เสร็จภายใน WCET

คุณลักษณะของระบบปฏิบัติการเชิงเวลาจริง

- ระบบปฏิบัติการสามารถจัดการงานหลาย ๆ งานได้
- สามารถคาดคะเนได้ว่าในแต่ละงานจะได้รับการบริการเมื่อไร
- มี interrupt latency ต่ำ นั่นคือเวลาที่ใช้ในการตอบสนองกลับของระบบปฏิบัติการเชิงเวลาจริงเมื่อมี interrupt เกิดขึ้นต้องต่ำกว่าระบบปฏิบัติการทั่วไป
- ต้องทำงานได้น่าเชื่อถือในสภาวะที่มีความผิดพลาดเกิดขึ้น โดยความผิดพลาดนั้น ๆ อาจจะเกิดจากภายในหรือภายนอก ซึ่งภายในมาจากฮาร์ดแวร์หรือซอฟต์แวร์ของระบบ processor restart บอร์ด และลิงค์มีปัญหา เป็นต้น

ส่วนความผิดพลาดที่เกิดจากภายนอกมาจากพฤติกรรมที่ไม่ถูกต้อง และการเชื่อมต่อสายรับส่งข้อมูล เป็นต้น

2.2 ARM-7 embedded controller board

ARM-7 embedded controller board ประกอบด้วย 2 ส่วน คือ ส่วนที่เป็นบอร์ดฐานหรือเรียกว่า ET-ARM7 START KIT V1.0 / EXP และส่วนของหน่วยประมวลผลหรือเรียกว่า ET-ARM7 STAMP LPC2119

2.2.1 สถาปัตยกรรมของ ARM-7 LPC2119

ARM7TDMI-S เป็นไมโครโพรเซสเซอร์ (microprocessor) ขนาด 32 บิตที่ออกแบบให้ใช้กำลังงานต่ำ สถาปัตยกรรมของ ARM เป็นสถาปัตยกรรมแบบ Reduced Instruction Set Computer (RISC) ชุดคำสั่งที่ใช้และกระบวนการถอดรหัสทำได้ง่ายกว่าสถาปัตยกรรมแบบ Complex Instruction Set Computer (CISC) ด้วยเหตุนี้จึงทำให้ได้จำนวนคำสั่งที่ใช้ในการประมวลผลในช่วงเวลาหนึ่งน้อยกว่าสถาปัตยกรรมแบบ CISC และหน่วยประมวลผลสามารถตอบสนองต่อ interrupt เชิงเวลาจริงได้ดี

เทคนิคการทำ pipeline เป็นเทคนิคที่ช่วยให้การประมวลผลสามารถทำได้อย่างต่อเนื่อง นั่นคือ ในขณะที่มีการประมวลผลหรือรัน (execute) คำสั่งถัดมาหรือคำสั่งในส่วนที่สองก็จะทำการถอดรหัสไปพร้อม ๆ กันและในขณะเดียวกันจะทำการอ่าน (fetch) คำสั่งในส่วนที่สามจากหน่วยความจำ

ARM มีสถาปัตยกรรมแบบ Super-Reduced Instruction Set ขนาด 32 บิตที่สามารถประมวลผลได้ 2 ชุดคำสั่งคือ

- ชุดคำสั่งของ ARM แบบมาตรฐาน 32 บิต
- ชุดคำสั่ง 16 บิตแบบ Thumb

สถาปัตยกรรมของหน่วยประมวลผล ARM7TDMI-S สามารถประมวลผลได้ทั้ง 2 แบบ ชุดคำสั่งแบบ Thumb ซึ่งเป็น 16 บิตเมื่อใช้คำสั่ง ARM ที่มีขนาด 32 บิตจะต้องโหลดชุดคำสั่ง 2 ครั้งเพื่อให้การทำงานแบบ 16 บิตสามารถทำงานแบบ 32 บิตได้

ชุดคำสั่งทั้ง 2 แบบมีข้อดีข้อเสียที่แตกต่างกันคือ Thumb code ใช้ขนาดหน่วยความจำได้มีประสิทธิภาพมากกว่า ARM code นั่นคือสามารถลดขนาดของหน่วยความจำที่

ต้องใช้ได้ถึง 65% แต่เมื่อพิจารณาประสิทธิภาพหรือความเร็วในการทำงานกับหน่วยความจำ ARM code มีประสิทธิภาพดีกว่า Thumb code ถึง 160% [33]

2.2.1.1 Flash program memory บนชิพ

LPC2119 มีหน่วยความจำแฟลช (flash memory) ขนาด 128 กิโลไบต์ (kB) โดยหน่วยความจำส่วนนี้สามารถเก็บได้ทั้งโปรแกรมและข้อมูล การเขียนโปรแกรมลงแฟลชสามารถทำได้หลายทาง เช่น ผ่านทาง serial port หรือเขียนลงบนหน่วยความจำแฟลชโดยตรง สามารถลบ หรือโปรแกรมกับแอพลิเคชันโปรแกรมบนแฟลชได้ในขณะที่แอพลิเคชันกำลังทำงานอยู่และสามารถอัพเกรด (upgrade) เฟิร์มแวร์ (firmware) ได้ การเขียนข้อมูลลงบนหน่วยความจำแฟลชสามารถทำงานได้ก็ต่อเมื่อบู๊ตโหลดเดอร์ (boot loader) บนชิพมีการใช้งาน

หน่วยความจำแฟลช LPC2119 สามารถลบหรือเขียนได้อย่างต่ำ 100,000 รอบ และสามารถเก็บข้อมูลได้นานถึง 20 ปี

บู๊ตโหลดเดอร์ (รุ่นที่มีการปรับปรุงใหม่ 1.60) มีความสามารถที่จะจัดการให้หน่วยความจำแฟลชมีคุณสมบัติเป็น Code Read Protection (CRP) ได้ บู๊ตโหลดเดอร์ที่อยู่ในสถานะ CRP on จะทำให้การเข้าถึง JTAG debug port และ ISP commands ที่อยู่บน RAM และ/หรือที่อยู่บนหน่วยความจำแฟลชไม่สามารถทำได้ และสามารถเข้าถึงได้เมื่อ CRP อยู่ในสถานะ CRP off อย่างไรก็ตามคำสั่ง ISP flash erase สามารถทำงานได้ตลอดเวลาโดยไม่ขึ้นกับสถานะของ CRP การแก้ไขหรือเปลี่ยนสถานะของ CRP ทำได้โดยการลบข้อมูลบนแฟลชทั้งหมดให้สมบูรณ์

2.2.1.2 Static RAM บนชิพ

Static RAM (SRAM) บนชิพใช้สำหรับเก็บโปรแกรมโค้ดและ/หรือข้อมูลโดยสามารถใช้งานได้ 16 kB สำหรับการเข้าถึงอาจจะเข้าถึงด้วย 8 บิต 16 บิต หรือ 32 บิต

2.2.1.3 ตัวควบคุมทิศทางของ interrupt (Vectored Interrupt Controller : VIC)

Vectored Interrupt Controller (VIC) เป็นตัวรับการร้องขอ interrupt โดยแบ่งออกเป็น 3 ประเภทตามการกำหนดค่าจากการโปรแกรมซึ่งได้แก่ Fast Interrupt reQuest (FIQ),

vectored IRQ และ non-vectored IRQ การกำหนดค่าในการโปรแกรมหมายถึงการกำหนดลำดับความสำคัญให้กับอุปกรณ์ในแต่ละชนิดโดยสามารถกำหนดและปรับเปลี่ยนได้ตามความเหมาะสม

Interrupt ที่มีความสำคัญสูงที่สุดก็คือ FIQ ถ้ามีการร้องขอที่เป็น FIQ มากกว่าหนึ่ง VIC จะทำหน้าที่รวมการร้องขอเหล่านั้นและส่งสัญญาณ FIQ ไปยังหน่วยประมวลผล ARM เมื่อหน่วยประมวลผลได้รับสัญญาณก็จะทำการตรวจสอบและกรณีที่เป็นสัญญาณ FIQ จะตอบสนองกลับเร็วที่สุดเท่าที่สามารถทำได้เพราะ FIQ service routine สามารถจัดการกับอุปกรณ์เหล่านั้นได้โดยตรง และถ้ามีการร้องขอที่เป็น FIQ มากกว่าหนึ่งแหล่งระบบจะอ่านข้อความจาก VIC เพื่อตรวจว่ามี FIQ จากแหล่งใดบ้างที่ต้องการการร้องขอ interrupt

Vectored IRQ มีลำดับความสำคัญปานกลาง สามารถกำหนดได้สูงสุด 16 การร้องขอ สำหรับการร้องขอใด ๆ ก็ตามจะส่งไปยัง IRQ slot ซึ่งมีทั้งหมด 16 vector โดย slot 0 จะมีความสำคัญสูงที่สุดและ slot 15 มีความสำคัญต่ำที่สุด

VIC มีหน้าที่รวมการร้องขอทั้งหมดจาก IRQ ทุก ๆ ตัวทั้งที่เป็น vector และ non-vector เพื่อสร้างสัญญาณ IRQ และส่งไปยังหน่วยประมวลผล ARM จากนั้น IRQ service routine จะเริ่มต้นอ่านรีจิสเตอร์จาก VIC และกระโดดไปทำงานยังจุดนั้น โดย VIC จะจัดให้แอดเดรส (address) ที่มีลำดับความสำคัญสูงทำงานก่อน ส่วน non-vectored IRQ จะถูกจัดให้ทำงานเป็นลำดับสุดท้ายซึ่งเป็นตำแหน่ง default routine ที่ non-vectored IRQ ทุก ๆ ตัวใช้งานร่วมกัน default routine สามารถอ่านรีจิสเตอร์ VIC ตัวอื่น ๆ ได้เพื่อดูว่ามี IRQ ตัวใดบ้างที่กำลังทำงานอยู่

อุปกรณ์แต่ละตัวสามารถติดต่อ VIC ได้เพียงหนึ่ง interrupt line เท่านั้นแต่สามารถมี internal interrupt flag ได้มากกว่าหนึ่ง โดยที่ interrupt flag แต่ละตัวสามารถใช้ interrupt source ได้มากกว่าหนึ่ง แสดงดังตาราง 2-1 และการทำงานของ VIC แสดงดังภาพประกอบ 2-2 [34]

ตาราง 2-1 การติดต่อของ interrupt source ไปยัง Vectored Interrupt Controller

Block	**Flag(s)**	**VIC channel #**
WDT	Watchdog Interrupt (WDINT)	0
-	สงวนไว้สำหรับ software interrupt	1
ARM Core	Embedded ICE, DbgCommRx	2
ARM Core	Embedded ICE, DbgCommTx	3

ตาราง 2-1 (ต่อ)

Block	Flag(s)	VIC channel #
TIMER0	Match 0 – 3 (MR0, MR1, MR2, MR3) Capture 0 – 3 (CR0, CR1, CR2, CR3)	4
TIMER1	Match 0 – 3 (MR0, MR1, MR2, MR3) Capture 0 – 3 (CR0, CR1, CR2, CR3)	5
UART0	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI)	6
UART1	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Status Interrupt (MSI)	7
PWM0	Match 0 – 6 (MR0, MR1, MR2, MR3, MR4, MR5, MR6)	8
I^2C	SI (state change)	9
SPI0	SPI Interrupt Flag (SPIF) Mode Fault (MODF)	10
SPI1	SPI Interrupt Flag (SPIF) Mode Fault (MODF)	11
PLL	PLL Lock (PLOCK)	12
RTC	Counter Increment (RTCCIF) Alarm (RTCALF)	13
System Control	External Interrupt 0 (EINT0)	14
System Control	External Interrupt 1 (EINT1)	15
System Control	External Interrupt 2 (EINT2)	16

ตาราง 2-1 (ต่อ)

Block	Flag(s)	VIC channel #
System Control	External Interrupt 3 (EINT3)	17
A/D	A/D Converter	18
CAN	CAN and Acceptance Filter (1 ORed CAN, LUTerr int)	19
	CAN1 Tx	20
	CAN2 Tx	21
	CAN3 Tx	22
	CAN4 Tx	23
	สงวน	24-25
	CAN1 Rx	26
	CAN2 Rx	27
	CAN3 Rx	28
	CAN4 Rx	29
	สงวน	30-31

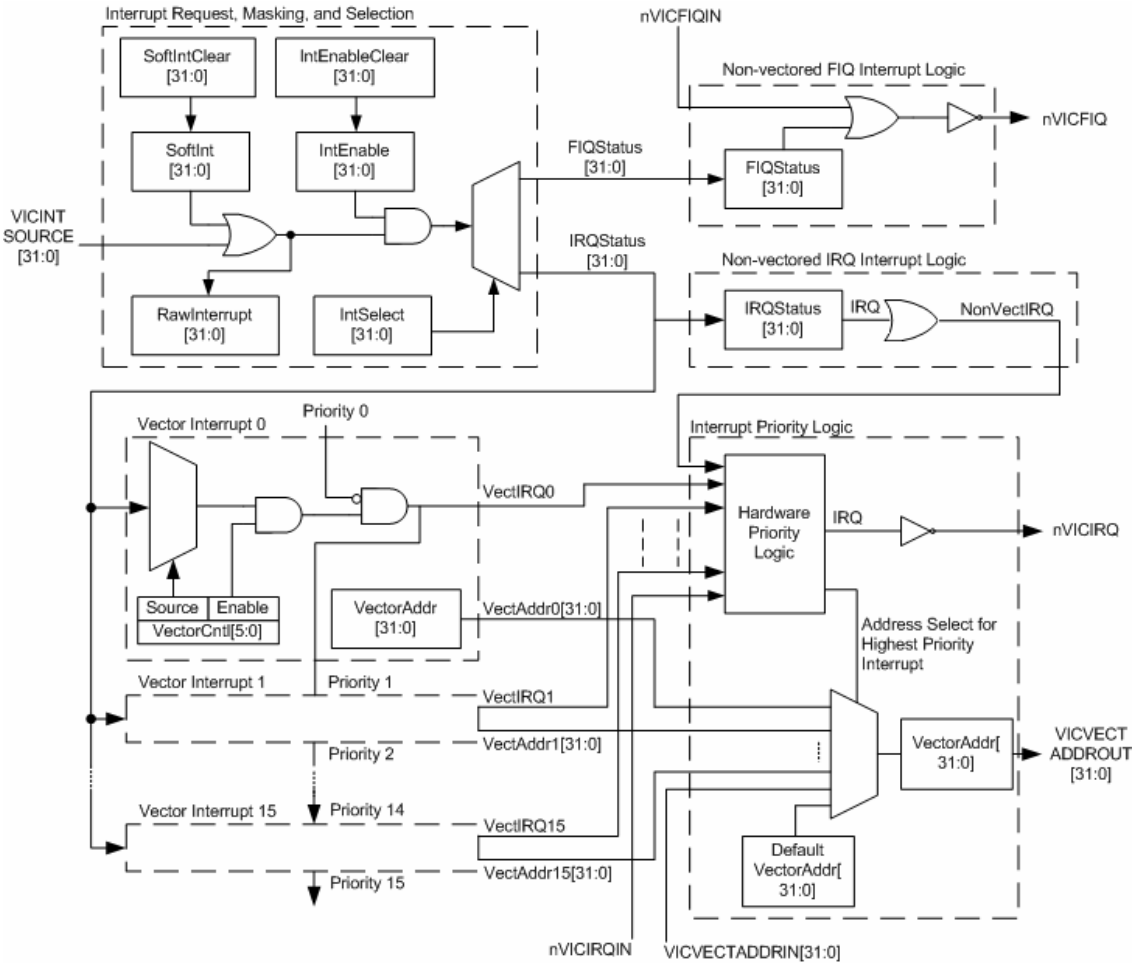
2.2.1.4 Interrupt ที่รับมาจากภายนอก (External interrupt input)

LPC2119 มีขาที่ทำหน้าที่เป็น external interrupt input ทั้งหมด 4 ขาซึ่ง external interrupt input สามารถใช้เป็นตัวปลุกหน่วยประมวลผลให้ออกจากสถานะ power down ได้

รีจิสเตอร์ที่เกี่ยวข้องกับ external interrupt มีทั้งหมด 4 ตัว คือ EXTINT, EXTWAKEUP, EXTMODE, และ EXTPOLAR

- EXTINT เป็นรีจิสเตอร์ที่ใช้สำหรับ interrupt flag
- EXTWAKEUP เป็นรีจิสเตอร์ที่มีบิตที่สามารถ enable external interrupt ด้วยการปลุก LPC2119 ให้ตื่นจากสถานะ power down

- EXTMODE และ EXTPOLAR เป็นตัวกำหนดค่าพารามิเตอร์ว่าต้องการ interrupt ที่ edge หรือ level ซึ่งสรุปดังตาราง 2-2



ภาพประกอบ 2-2 Block diagram ของ Vectored Interrupt Controller

2.2.1.5 การควบคุมกำลังงาน (Power control)

LPC2119 มีสถานะที่สนับสนุนการลดกำลังงาน 2 สถานะ คือ สถานะ idle และสถานะ power down สำหรับสถานะ idle จะมีการพักการรันคำสั่งไว้ชั่วคราวจนกระทั่งมีการรีเซ็ต (reset) และ/หรือ มีการ interrupt เกิดขึ้น ในส่วนอุปกรณ์ต่อพ่วงภายนอกยังสามารถทำงานได้อย่างต่อเนื่องและอาจจะเป็นตัวสร้าง interrupt ให้หน่วยประมวลผลกลับมาทำงานต่ออีก

ครั้ง เมื่อระบบเข้าสู่สถานะนี้หน่วยประมวลผล หน่วยความจำและตัวควบคุมที่เกี่ยวข้อง และบัสจะไม่มีการใช้กำลังงานใด ๆ

ตาราง 2-2 External interrupt register

ตำแหน่ง	ชื่อ	รายละเอียด	การเข้าถึง
0xE01FC140	EXTINT	External interrupt flag register ที่ประกอบด้วย interrupt flag สำหรับ EINT0, EINT1, EINT2, และ EINT3	อ่าน/เขียน
0xE01FC144	EXTWAKE	External interrupt register ใช้ 3 บิตในการควบคุมการ enable ของ external interrupt แต่ละตัวที่มีหน้าที่ในการปลุกหน่วยประมวลผลให้ออกจากสภาวะ power down	อ่าน/เขียน
0xE01FC148	EXTMODE	External interrupt mode register ควบคุมแต่ละขาในการพิจารณาที่ edge หรือ level	อ่าน/เขียน
0xE01FC14C	EXTPOLAR	External interrupt polarity register ควบคุมเหตุในการเกิด interrupt ของขาที่ edge หรือ level	อ่าน/เขียน

ในสถานะ power down ระบบจะหยุดการทำงานของ oscillator อุปกรณ์ย่อยภายในทุกตัวจะไม่ได้รับสัญญาณนาฬิกาภายใน ค่าต่าง ๆ และสถานะของ SRAM รีจิสเตอร์ของอุปกรณ์ต่อพ่วงภายนอก ระดับสัญญาณที่ขาต่าง ๆ หน่วยประมวลผล และรีจิสเตอร์ไม่มีการเปลี่ยนแปลงนั่นคือเป็นการค้างการทำงานของชิพทั้งหมด ระบบจะเข้าสู่สถานะปกติเมื่อมีการ reset หรือสัญญาณจาก external interrupt

การเข้าสู่สถานะ idle และ power down ระบบกับโปรแกรมต้องทำงานประสานกันเนื่องจากการเปลี่ยนสถานะเข้าสู่การทำงานปกติต้องไม่มีคำสั่งใด ๆ ถูกข้ามไป ไม่สมบูรณ์ หรือทำงานซ้ำคำสั่งเดิม

2.2.1.6 รายละเอียดของรีจิสเตอร์ (Register description)

รีจิสเตอร์ที่มีหน้าที่ในการควบคุมกำลังงานมี 2 ตัวโดยรายละเอียดแสดงดังตาราง 2-3

ตาราง 2-3 รีจิสเตอร์ทั้งหมดสำหรับควบคุมกำลังงาน

ตำแหน่ง	ชื่อ	รายละเอียด	การเข้าถึง
0xE01FC0C0	PCON	เป็นรีจิสเตอร์ที่ใช้สำหรับควบคุมการใช้กำลังงาน ซึ่งมีบิตที่ใช้สำหรับควบคุมสถานะของทั้ง 2 โหมด แสดงดังตาราง 2-4	อ่าน/เขียน
0xE01FC0C4	PCONP	เป็นรีจิสเตอร์สำหรับควบคุมกำลังงานที่ใช้สำหรับอุปกรณ์ต่อพ่วง โดยมีบิตที่ทำหน้าที่ควบคุมการทำงานของอุปกรณ์ต่อพ่วงให้สามารถทำงานได้และไม่สามารถทำงานได้เพื่อลดกำลังงานที่ใช้สำหรับอุปกรณ์ต่อพ่วง	อ่าน/เขียน

2.2.1.7 รีจิสเตอร์สำหรับควบคุมกำลังงาน (Power Control Register : PCON - 0xE01FC0C0)

รีจิสเตอร์ PCON ประกอบด้วยบิตจำนวน 2 บิต การกำหนดค่าบิตให้เป็น 1 คือการกำหนดให้ระบบเข้าสู่สถานะ idle หรือ power down กรณีที่ทั้ง 2 สถานะถูกกำหนดค่าให้เป็น 1 ระบบจะเข้าสู่สถานะ power down

2.2.1.8 รีจิสเตอร์สำหรับควบคุมกำลังงานในอุปกรณ์ต่อพ่วง (Power Control for Peripherals Register : PCONP – 0xE01FC0C4)

รีจิสเตอร์ PCONP มีหน้าที่ในการปิดหน้าที่การทำงานของอุปกรณ์ต่อพ่วงแต่ละตัวเพื่อประหยัดกำลังงานที่ใช้ แต่มีอุปกรณ์บางชนิดที่ไม่สามารถปิดได้ เช่น Watchdog timer,

GPIO, Pin connect block และ System control block แต่ละบิตของ PCONP จะทำหน้าที่ในการควบคุมอุปกรณ์เพียง 1 ชนิดเท่านั้น รายละเอียดของ PCONP แสดงดังตาราง 2-5

ตาราง 2-4 รีจิสเตอร์ที่ใช้สำหรับควบคุมกำลังงาน (PCON - 0xE01FC0C0)

PCON	หน้าที่	รายละเอียด	ค่า RESET
0	IDL	เมื่อมีค่าเป็น 1 ระบบจะเข้าสู่สถานะ idle ทำให้สัญญาณนาฬิกาของหน่วยประมวลผลหยุดทำงานในขณะที่อุปกรณ์ต่อพ่วงบนชิพยังสามารถทำงานได้เมื่อมี interrupt จากอุปกรณ์ต่อพ่วงหรือ interrupt อื่น ๆ จากภายนอกและภายในซึ่งจะทำให้หน่วยประมวลผลกลับมาทำงานได้อีกครั้ง	0
1	PD	เมื่อมีค่าเป็น 1 ระบบจะเข้าสู่สถานะ power down ทำให้ oscillator และสัญญาณนาฬิกาบนชิพทั้งหมดหยุดทำงาน เมื่อ interrupt จากภายนอกปลุกจะทำให้ oscillator เริ่มทำงาน จากนั้นจึงเคลียร์ค่า PD บิต หน่วยประมวลผลจึงสามารถทำงานต่อได้	0
7:2	สงวน	เป็นบิตที่สงวนไว้ ซึ่งโปรแกรมของผู้ใช้ทั่วไปไม่สามารถที่จะเขียนข้อมูลลงไปในส่วนนี้ได้ โดยค่าในส่วนนี้จะไม่มีความหมายในการทำงาน	NA

ตาราง 2-5 รีจิสเตอร์ที่ควบคุมกำลังงานสำหรับอุปกรณ์ต่อพ่วงต่าง ๆ ใน LPC2119 (PCONP - 0xE01FC0C4)

PCONP	หน้าที่	รายละเอียด	ค่า RESET
0	สงวน	เป็นส่วนที่สงวนเอาไว้ ผู้ใช้ไม่ควรเขียนข้อมูลลงตรงส่วนนี้ ข้อมูลที่อ่านจากบิตนี้จะไม่ได้กำหนดหน้าที่การทำงานไว้	0

ตาราง 2-5 (ต่อ)

PCONP	หน้าที่	รายละเอียด	ค่า RESET
1	PCTIM0	เมื่อค่าเป็น 1 TIMER0 จะทำงานและเมื่อค่าเป็น 0 TIMER0 จะไม่ทำงานเพื่อประหยัดกำลังงาน	1
2	PCTIM1	เมื่อค่าเป็น 1 TIMER1 จะทำงานและเมื่อมีค่าเป็น 0 TIMER1 จะไม่ทำงานเพื่อประหยัดกำลังงาน	1
3	PCURT0	เมื่อค่าเป็น 1 UART0 จะทำงานและเมื่อค่าเป็น 0 UART0 จะไม่ทำงานเพื่อประหยัดกำลังงาน	1
4	PCURT1	เมื่อค่าเป็น 1 UART1 จะทำงานและเมื่อค่าเป็น 0 UART1 จะไม่ทำงานเพื่อประหยัดกำลังงาน	1
5	PCPWM0	เมื่อค่าเป็น 1 PWM0 จะทำงานและเมื่อมีค่าเป็น 0 PWM0 จะไม่ทำงานเพื่อประหยัดกำลังงาน	1
6	สงวน	เป็นส่วนที่สงวนเอาไว้ ผู้ใช้ไม่ควรเขียนข้อมูลลงตรงส่วนนี้ ข้อมูลที่อ่านจากบิตนี้จะไม่ได้กำหนดหน้าที่การทำงานไว้	0
7	PCI2C	เมื่อค่าเป็น 1 I^2C interface จะทำงานและเมื่อค่าเป็น 0 I^2C interface จะไม่ทำงานเพื่อประหยัดกำลังงาน	1
8	PCSPI0	เมื่อค่าเป็น 1 SPI0 interface จะทำงานและเมื่อค่าเป็น 0 SPI0 interface จะไม่ทำงานเพื่อประหยัดกำลังงาน	1
9	PCRTC	เมื่อค่าเป็น 1 RTC จะทำงานและเมื่อค่าเป็น 0 RTC จะไม่ทำงานเพื่อประหยัดกำลังงาน	1
10	PCSPI1	เมื่อค่าเป็น 1 SPI1 interface จะทำงานและเมื่อค่าเป็น 0 SPI1 interface จะไม่ทำงานเพื่อประหยัดกำลังงาน	1
11	สงวน	ในโปรแกรมผู้เขียนควรกำหนดให้เป็น 0 ในตำแหน่งนี้เพื่อลดการใช้กำลังงาน	1
12	PCAD	เมื่อค่าเป็น 1 ตัวแปลงสัญญาณ A/D จะทำงานและเมื่อมีค่าเป็น 0 ตัวแปลงสัญญาณ A/D จะไม่ทำงาน	1

ตาราง 2-5 (ต่อ)

PCONP	หน้าที่	รายละเอียด	ค่า RESET
13	PCCAN1	เมื่อค่าเป็น 1 CAN controller 1 จะทำงานและเมื่อค่าเป็น 0 CAN controller 1 จะไม่ทำงานเพื่อประหยัดกำลังงาน Note : Acceptance Filter ทำงานได้ก็ต่อเมื่อ CAN controller 1 - 2 ตัวใดตัวหนึ่งใช้งานได้	1
14	PCCAN2	เมื่อค่าเป็น 1 CAN controller 2 จะทำงานและเมื่อค่าเป็น 0 CAN controller 2 จะไม่ทำงานเพื่อประหยัดกำลังงาน	1
31:15	สงวน	เป็นส่วนที่สงวนเอาไว้ ผู้ใช้ไม่ควรเขียนข้อมูลลงตรงส่วนนี้ ซึ่งข้อมูลที่อ่านจากบิตนี้จะไม่ได้กำหนดหน้าที่การทำงานไว้	NA

2.2.2 ET-ARM7 STAMP LPC2119

ET-ARM7 STAMP LPC2119 เป็นบอร์ดไมโครคอนโทรเลอร์ในตระกูล ARM7TDMI-S Core ใช้ไมโครคอนโทรเลอร์ 16/32 บิต ขนาด 64 ขาแบบใช้พลังงานต่ำ สำหรับ MCU เบอร์ LPC2119 ของบริษัท Phillips ที่ใช้ในงานวิจัยนี้มีลักษณะดังภาพประกอบ 2-3 การออกแบบโครงสร้างจะเน้นในเรื่องของการจัดวางบอร์ดให้มีขนาดเล็กเพื่อง่ายต่อการประยุกต์ใช้งาน โดยได้นำ MCU มาต่อวงจรร่วมกับอุปกรณ์พื้นฐานที่จำเป็นและจัดขาออกมาให้ใช้งานภายนอก ซึ่งการจัดเรียงขาสัญญาณจะทำการจัดเรียงอย่างเป็นระเบียบเพื่อให้สามารถต่อใช้งานได้สะดวก ตัวบอร์ดใช้ไฟ +3.3V สามารถรองรับ I/O ที่เป็นสัญญาณ 5 V ได้ ตัวบอร์ดมีตัวเชื่อมต่อ UART0 (RS-232) จำนวน 1 พอร์ตสำหรับทำการดาว์นโหลด Hex ไฟล์หรือใช้สื่อสารด้วย RS-232 สำหรับแอพลิเคชันที่พัฒนาขึ้นเอง

ภาพประกอบ 2-3 ET-ARM7 STAMP LPC2119

LPC2119 สนับสนุนงานเชิงเวลาจริงและระบบฝังตัวโดยมีหน่วยความจำแฟลชแบบฝังตัวความเร็วสูงที่มีขนาด 128 หรือ 256 kB มีตัวเชื่อมต่อระหว่างหน่วยความจำและหน่วยประมวลผลขนาด 128 บิต เป็นสถาปัตยกรรมสามารถที่สามารถประมวลผลสูงสุดครั้งละ 32 บิต ตามความเร็วสูงสุดของรอบสัญญาณนาฬิกา สำหรับโค้ดหรือแอพลิเคชันที่มีขนาดใหญ่สามารถใช้อีกทางเลือกหนึ่งได้ คือ ใช้ 16 บิต Thumb mode เพื่อลดขนาดของโค้ดโดยสามารถลดขนาดโค้ดได้มากถึง 30% ในขณะที่ประสิทธิภาพลดลงเพียงเล็กน้อยเท่านั้น

LPC2119 มีขา 64 ขาสำหรับเชื่อมต่อ ใช้กำลังงานต่ำ ใช้ timer 32 บิต มี ADC ขนาด 10 บิตจำนวน 4 ช่อง (channel) มี CAN channel ระดับสูง 2 ช่อง มี PWM 1 ช่อง และมี GPIO 46 line นอกจากนี้ยังมีขา interrupt ขาจากภายนอกมากถึง 9 ขาซึ่งไมโครคอนโทรเลอร์ในลักษณะนี้เหมาะสำหรับงานควบคุมทางอุตสาหกรรม และงานควบคุมแบบอัตโนมัติ เช่น ระบบทางการแพทย์และ fault-tolerant maintenance buses เกตเวย์สำหรับสื่อสารเพราะมีตัวเชื่อมต่อขนาดกว้าง ตัวแปลงโพรโตคอล และแอพลิเคชันอื่น ๆ (รายละเอียดของบอร์ดแสดงในภาคผนวก ก) คุณลักษณะที่สำคัญของบอร์ดที่เกี่ยวข้องกับงานวิจัยโดยตรงสรุปได้ดังนี้ [35]

- ใช้ MCU ตระกูล ARM7TDMI-S เบอร์ LPC2119 ของ Phillips ซึ่งเป็น MCU ขนาด 16/32 บิตที่เป็นแพ็คเก็ตใน tiny LQFP64
- มี static RAM ที่อยู่บนชิพขนาด 16 kB
- มี Flash program memory บนชิพขนาด 128 kB และมี wide interface/accelerator ขนาด 128 บิตในการทำงาน สามารถทำงานได้เร็วสูงสุดถึง 60 MHz
- การโปรแกรมแบบ In-System Programming (ISP) และ In-Application Programming (IAP) ทำผ่านซอฟต์แวร์บนบูตโหลดเดอร์ที่อยู่บนชิพ การ

โปรแกรมบนแฟลช ใช้เวลาเพียง 1 มิลลิวินาที (ms) ต่อ 512 ไบต์การลบข้อมูลในชิพ 1 sector หรือทั้งชิพใช้เวลา 400 ms

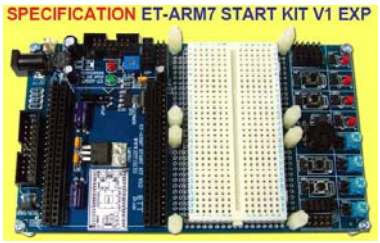
- สำหรับตัวแปลง A/D ขนาด 10 บิต 4 ช่องใช้เวลาในการแปลง 2.44 us
- Serial interface มีหลาย interface ประกอบด้วย UART (16C550) 2 ตัว fast 1^2C (400 kbits/s) และ SPI จำนวน 2 ชุด
- ความเร็วสูงสุดที่สัญญาณนาฬิกาของหน่วยประมวลผลสามารถทำงานได้จากการโปรแกรมบนชิพ คือ 60 MHz โดยกำหนดเวลาที่ใช้ในการทำ Phase-locked loop ประมาณ 100 ms
- Vectored interrupt controller สามารถกำหนดลำดับความสำคัญและ vector address ได้
- มี timer ขนาด 32 บิต จำนวน 2 ตัว (4 capture และ 4 compare channel) มี PWM unit (6 output) นาฬิกาเชิงเวลาจริง และ watchdog
- มีขา general purpose I/O รับแรงดันได้ขนาด 5 V จำนวน 46 ขาโดยมีขาสำหรับ interrupt จากภายนอก 9 ขา
- มีสภาวะ สำหรับการใช้กำลังงานต่ำ 2 สภาวะคือ idle และ power down
- หน่วยประมวลผลจะออกจากสภาวะ power down ด้วย interrupt จากภายนอก
- มีการ enable/disable ในส่วนที่ทำหน้าที่เกี่ยวกับอุปกรณ์ต่อพ่วงเฉพาะตัวเพื่อให้มีการใช้กำลังงานได้อย่างเหมาะสม

2.2.3 ET-ARM7 START KIT V1.0 EXP

ET-ARM7 START KIT V1.0 EXP เป็นบอร์ดที่มีวงจรพื้นฐานจำเป็นสำหรับการศึกษาเรียนรู้และทดลองใช้ทรัพยากร MCU ในตระกูล ARM ลักษณะของบอร์ดแสดงดังภาพประกอบ 2-4 (รายละเอียดของบอร์ดแสดงในภาคผนวก ข) วงจรสำคัญภายในบอร์ดที่เกี่ยวข้องกับงานวิจัยนี้โดยตรง ได้แก่

- วงจร LED แสดงผลใช้ไฟเลี้ยง 3.3 V โดยใช้ LED สีแดงขนาด 3 mm จำนวน 4 ชุดสำหรับใช้ในการทดสอบการทำงานของเอาท์พุท
- วงจรปรับแรงดัน 0 - 3.3 V โดยใช้ตัวต้านทานปรับค่าได้แบบเกือกม้าแบบมีแกนปรับจำนวน 4 ชุดสำหรับใช้ในการทดสอบการทำงานของ A/D

- วงจร push button switch จำนวน 4 ชุด สำหรับใช้ทดสอบการทำงานของอินพุทต่าง ๆ
- ขั้วต่อเฮดเดอร์สำหรับรองรับโมดูล “ET-ARM STAMP LPC2119” หรือโมดูลอื่น ๆ ที่มีขนาดเท่ากันพร้อมตัวเชื่อมต่อสำหรับต่อไปยังวงจรทดลองต่าง ๆ ทั้งแบบตัวผู้และตัวเมีย



ภาพประกอบ 2-4 ลักษณะของบอร์ด ET-ARM7 START KIT V1.0 EXP

2.3 ระบบปฏิบัติการเชิงเวลาจริง uC/OS-II

uC/OS-II เป็นระบบปฏิบัติการขนาดเล็กที่สนับสนุนการทำงานเชิงเวลาจริง และจากการทดสอบจากผู้พัฒนาระบบปฏิบัติการ uC/OS-II มีความปลอดภัยและทนทานเพียงพอที่จะนำมาใช้ในงานที่ต้องการความน่าเชื่อถือสูง

uC/OS-II เป็นเคอร์เนล (kernel) ที่พัฒนาด้วยภาษา ANCI C ผสมกับ assembly เล็กน้อย และสามารถปรับมาใช้ให้เข้ากับหน่วยประมวลผลที่มีสถาปัตยกรรมที่แตกต่างกันได้

uC/OS-II เป็น real-time kernel ที่สามารถนำมาใช้กับแอพลิเคชันที่หลากหลายได้ เช่น กล้อง เครื่องมือทางอิเล็กทรอนิกส์เกี่ยวกับการบิน high-end audio equipment เครื่องมือแพทย์ เครื่องดนตรี ส่วนควบคุมเครื่องจักร เครื่องมือที่ใช้ในเครือข่าย highway telephone call boxes เครื่อง ATM หุ่นยนต์ และอื่น ๆ

คุณลักษณะของ uC/OS-II [36]

- Source code มีขนาดเล็ก ไม่ซับซ้อน สามารถแก้ไขและปรับเปลี่ยนได้
- Portable สามารถนำเข้าไปใช้กับระบบอื่น ๆ ได้
- ROMable สามารถนำไปใช้กับแอพลิเคชันแบบฝังตัวได้

- Scalable ใช้งานเท่าที่จำเป็นจะช่วยประหยัดหน่วยความจำ (RAM และ ROM)
- Preemptive ยอมให้มีการ interrupt หรือบริการงานอื่นที่เข้ามาขอใช้บริการ โดยที่ยังให้บริการงานปัจจุบันอยู่
- Multitasking สามารถจัดการงานได้มากถึง 64 งาน แต่จำนวนงานที่เหมาะสมไม่ควรเกิน 8 งาน โดยแต่ละงานจะถูกกำหนดลำดับความสำคัญไว้เฉพาะในแต่ละงาน ซึ่งหมายความว่า uC/OS-II ไม่สามารถจัดตารางการทำงานแบบ round-robin ได้ และมี ลำดับความสำคัญทั้งหมด 64 ระดับ
- Deterministic เวลาที่ใช้ในการประมวลผลของ uC/OS-II สามารถคาดคะเนได้ นั่นคือสามารถทราบเวลาในการประมวลผลการให้บริการงานและฟังก์ชันต่าง ๆ ยกเว้นการทำงานของฟังก์ชัน OSTimeTick() และ event flag บางตัว ซึ่งทั้ง 2 ส่วนนี้ไม่ขึ้นกับจำนวนงานที่กำลังทำงานอยู่
- Task stack มีการตรวจสอบสแต็ก (stack) ที่จะใช้ล่วงหน้า ทำให้ทราบว่าจะต้องใช้ stack จำนวนเท่าไรในการทำงาน
- Services มีบริการของระบบหลายบริการ เช่น semaphores, mutual exclusion semaphores, event flags, message mailboxes, message queues, fixed-sized memory partitions, task management, time management functions และอื่น ๆ
- Interrupt management สามารถจัดการ interrupt ที่ซ้อนกันได้ลึกถึง 255 ระดับ
- Robust and reliable การทำงานน่าเชื่อถือและทนต่อความผิดพลาดได้ดี

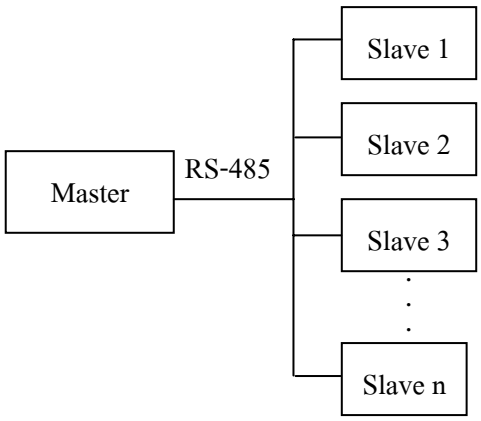
2.4 โพรโตคอล Modbus

โพรโตคอล Modbus เป็นโพรโตคอลที่ใช้มากในการติดต่อสื่อสารของอุปกรณ์บนระบบเชิงเวลาจริงเพื่อให้อุปกรณ์แต่ละตัวทั้งที่เป็นชนิดเดียวกันและต่างชนิดกันสามารถติดต่อสื่อสารกันได้ ซึ่งสามารถติดต่อผ่านเครือข่ายเดียวกันหรือต่างเครือข่ายก็ได้ โพรโตคอล Modbus มีการใช้งานอย่างแพร่หลายจนเป็นมาตรฐานที่ยอมรับกันโดยไม่ต้องรับรองมาตรฐานจากองค์กรหรือหน่วยงานที่ทำหน้าที่รับรองมาตรฐาน (defacto) กรณีที่มีการส่งข้อมูลแบบ point to point ต้องติดต่อสื่อสารผ่าน serial interface ที่เป็น RS-232C และส่งผ่าน RS-485 ในกรณีที่มี

การเชื่อมต่อเป็นเครือข่ายหรือมีการส่งข้อมูลไปยังอุปกรณ์หลายตัว นอกจากนี้ต้องกำหนดรูปแบบการเชื่อมต่อ เช่น connector, baud rate, stop bit และการตรวจสอบพาริตี (parity check) ทั้งฝ่ายรับและฝ่ายส่งให้ถูกต้องและตรงกันก่อนที่จะเริ่มส่งข้อมูล [37]

2.4.1 สถาปัตยกรรมของ Modbus

การรับส่งข้อมูลโดยผ่านโพรโตคอล Modbus ประกอบด้วย 2 ส่วนใหญ่ ๆ คือ ส่วนที่ทำหน้าที่เป็น master และส่วนที่ทำหน้าที่เป็น slave โดยที่ master 1 ตัวสามารถต่อกับ slave ได้หลายตัวแต่ไม่เกิน 247 ตัว ซึ่งการเชื่อมต่อกันระหว่าง master และ slave แสดงได้ดังภาพประกอบ 2-5



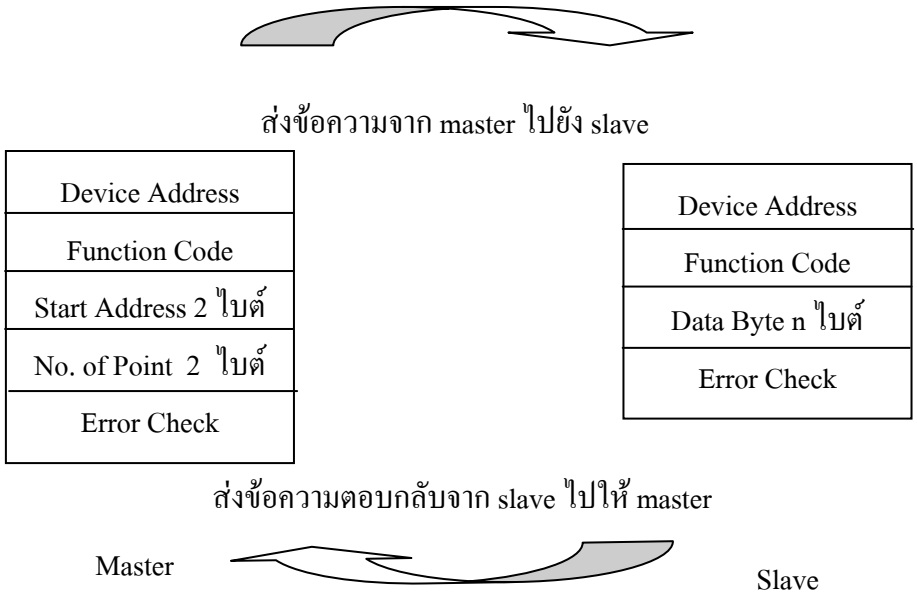
ภาพประกอบ 2-5 การเชื่อมต่อของอุปกรณ์หลายตัวที่มีการติดต่อสื่อสารผ่านโพรโตคอล Modbus

การทำงานของโพรโตคอล Modbus เกี่ยวข้องกับการรับ/ส่งข้อมูล ดังนั้นในการสื่อสารระหว่างกันทั้งฝ่ายรับและฝ่ายส่งต้องเข้าใจรูปแบบของการรับและการส่งข้อมูลที่ตรงกันซึ่งจะกล่าวถึงในหัวข้อการรับส่งข้อมูลระหว่าง master และ slave

2.4.2 การรับส่งข้อมูลระหว่าง master และ slave

รูปแบบทั่วไปในการรับส่งข้อมูลระหว่าง master และ slave มีวงจรการรับส่งข้อมูลแสดงดังภาพประกอบ 2-6

การส่งคำถามจาก master ไปยัง slave นั้น master จะระบุตัว slave ด้วย slave address ส่วน function code เป็นการบอกถึงหน้าที่ที่ slave ต้องทำ เช่น กรณีที่ function code มีค่าเป็น 03 หมายถึง master สั่งให้ slave อ่านค่าใน holding register ส่งกลับไปให้ slave ซึ่งสิ่งที่ส่งกลับไปเป็นตำแหน่งเริ่มต้นของรีจิสเตอร์และจำนวนที่ต้องการอ่าน นอกจากนี้ต้องตรวจสอบความถูกต้องของข้อมูลที่ส่งด้วย error check



ภาพประกอบ 2-6 วงจรการถามตอบของ master และ slave

การตอบกลับไปยัง master ถ้าเป็นการตอบกลับแบบธรรมดา slave จะส่ง device address และ function code เหมือนกันกับที่ได้รับมาจาก master ในส่วนของข้อมูล slave จะทำการรวบรวมตามหน้าที่ที่ได้รับ เช่น ถ้า master ต้องการทราบค่าหรือสถานะรีจิสเตอร์ของ slave ทางด้าน slave จะทำการรวบรวมค่าและสถานะของรีจิสเตอร์ใส่ไว้ใน data byte และตรวจสอบความถูกต้องเช่นเดียวกับการส่งคำถามจาก master ไปยัง slave ในการรับ/ส่งข้อมูลระหว่าง master และ slave ข้อมูลที่แต่ละฝ่ายได้รับมีรูปแบบที่แตกต่างกันโดยรูปแบบข้อมูลที่ทั้งสองฝ่ายส่งจะกล่าวถึงในหัวข้อถัดไป

2.4.3 รูปแบบข้อมูล

รูปแบบข้อมูลหรือรูปแบบเฟรมที่ master และ slave รับ/ส่งข้อมูลระหว่างกันจะไม่เหมือนกัน นั่นคือ รูปแบบเฟรมที่ master ส่งไปยัง slave จะมีขนาดที่แน่นอนแต่รูปแบบเฟรมที่ slave ส่งไปยัง master จะมีขนาดที่ไม่แน่นอนขึ้นอยู่กับฟังก์ชันที่ master ถาม แสดงดังภาพประกอบ 2-7 และจะกล่าวถึงรายละเอียดของรูปแบบเฟรมในหัวข้อรูปแบบเฟรมที่ส่งด้วยโพรโตคอล Modbus

DEVICE ADDRESS	FUNC	START ADDR. (HO)	START ADDR. (LO)	NO. OF POINT (HO)	NO. OF POINT (LO)	CRC/LRC

ภาพประกอบ 2-7a เฟรมข้อมูลที่ส่งออกจาก master เพื่อถาม slave

DEVICE ADDRESS	FUNC	BYTE COUNT	---------- DATA ----------	CRC/LRC

ภาพประกอบ 2-7b เฟรมข้อมูลที่ส่งออกจาก slave เพื่อตอบกลับไปยัง master

ภาพประกอบ 2-7 รูปแบบเฟรมทั่วไปในการรับ/ส่งข้อมูล

จากภาพประกอบ 2-7 ข้อมูลในแต่ละส่วนมีความหมายที่แตกต่างกันและไม่สามารถสลับลำดับของข้อมูลได้ ความหมายในแต่ละส่วนแสดงดังตาราง 2-6

ฟังก์ชันที่ใช้ในโพรโตคอล Modbus มีหลายฟังก์ชันด้วยกันในที่นี้จะแสดงให้เห็นถึงตัวอย่างที่มีการใช้งานบ่อยดังตาราง 2-7

ตาราง 2-6 ความหมายและจำนวนไบต์ที่ต้องการในแต่ละฟิลด์ของข้อมูล

ฟิลด์ข้อมูล	ความหมาย	จำนวนไบต์
DEVICE ADDRESS	หมายเลขอุปกรณ์ slave ที่ไม่ซ้ำกัน	1
FUNC	หมายเลขฟังก์ชันของ Modbus ที่ master ใช้สำหรับส่งให้ slave ตามที่ต้องการ ตัวอย่างฟังก์ชันแสดงดังตาราง 2-7	1

ตาราง 2-6 (ต่อ)

ฟิลด์ข้อมูล	ความหมาย	จำนวนไบต์
START ADDR.	ตำแหน่งเริ่มต้นของข้อมูลหรืออุปกรณ์ที่ต้องการอ่านทั้ง HO และ LO (HO คือ High Order และ LO คือ Low Order)	2
NO. OF POINT	จำนวนข้อมูลที่ต้องการอ่านทั้ง HO และ LO	2
BYTE COUNT	จำนวนข้อมูลที่ slave ต้องการส่ง โดยส่วนนี้อาจจะเป็นคำตอบที่บางฟังก์ชันต้องการ	1
CRC/LRC	ตรวจสอบความผิดพลาดของข้อมูลที่ส่ง	2/1

ตาราง 2-7 ตัวอย่างฟังก์ชันที่ใช้ในโพรโตคอล Modbus

Function Code	Description
01	Read coil status
02	Read input status
03	Read holding registers
04	Read input registers
05	Force single coil
06	Preset single registers
0F	Force multiple coils
10	Preset multiple registers

การส่งข้อมูลนอกจากรูปแบบเฟรมที่ถูกต้องแล้วอีกสิ่งหนึ่งที่สำคัญคือการกำหนดโหมดในการส่งข้อมูลเนื่องจากการส่งข้อมูลในโหมดที่แตกต่างกันรูปแบบของข้อมูลที่ส่งก็จะแตกต่างกันด้วย

2.4.4 โหมดการส่งข้อมูล

มาตรฐานการส่งข้อมูลผ่านโพรโตคอล Modbus แบ่งออกเป็น 2 โหมด คือ โหมด ASCII และ RTU ซึ่งความแตกต่างระหว่างโหมด ASCII และ RTU แสดงดังตาราง 2-8

การส่งข้อมูลด้วยโพรโตคอล Modbus จะส่งข้อมูลเป็นลำดับเรียงติดต่อกัน การส่งข้อมูลทั้งสองโหมดมีลำดับและรูปแบบการจัดส่งที่แตกต่างกันซึ่งรายละเอียดจะกล่าวถึงต่อไปในหัวข้อการส่งอักขระข้อมูลติดต่อกันเป็นลำดับ

ตาราง 2-8 ความแตกต่างระหว่างโหมด ASCII และ RTU

รายละเอียด	**ASCII**	**RTU**
ระบบ Coding	Hexadecimal, ASCII character (0-9, A-F) ข้อมูล 1 ชุดมี 1 Hexadecimal character	8 bit binary, hexadecimal (0-9, A-F) ใน 8 บิตประกอบด้วยข้อมูล 2 Hexadecimal character
Bits per byte Start bit Data bit Even/odd parity Stop bit	 1 7 ส่ง least significant bit ก่อน 1 กรณีไม่มีพาริตีจะไม่เพิ่มบิตนี้ 1 กรณีที่มีการใช้พาริตี 2 กรณีที่ไม่มีการใช้พาริตี	 1 8 ส่ง least significant bit ก่อน 1 กรณีไม่มีพาริตีจะไม่เพิ่มบิตนี้ 1 กรณีที่มีการใช้พาริตี 2 กรณีที่ไม่มีการใช้พาริตี
Error check field	Longitudinal Redundancy Check (LRC)	Cyclical Redundancy Check (CRC)
ข้อดี	ยอมให้มี delay ในการส่งข้อมูลในแต่ละ character 1 วินาที และตรวจสอบได้ง่าย	ความหนาแน่นของข้อมูลที่ส่งดีกว่าโหมด ASCII (กรณีที่ baud rate เท่ากัน)

2.4.5 รูปแบบเฟรมที่ส่งด้วยโพรโตคอล Modbus (Modbus message framing)

รูปแบบข้อมูลที่ส่งด้วยโพรโตคอล Modbus มีลักษณะเป็นเฟรมข้อมูลที่มีจุดเริ่มต้นและจุดสิ้นสุดเฟรม ระบุถึงตำแหน่งหรืออุปกรณ์ที่ต้องการติดต่อซึ่งอาจจะติดต่อกับอุปกรณ์บางตัวหรือทุกตัวก็ได้ และที่สำคัญเฟรมข้อมูลที่ส่งทุกเฟรมต้องมีการตรวจสอบความผิดพลาดที่เกิดจากการส่งข้อมูลในทุกๆ ครั้งซึ่งการส่งข้อมูลด้วยโพรโตคอล Modbus สามารถส่งได้ 2 โหมดดังที่ได้กล่าวมาในหัวข้อที่แล้ว และนอกจากรูปแบบการส่งที่แตกต่างกันแล้วรูปแบบเฟรมที่ส่งก็แตกต่างกันด้วย

2.4.5.1 การแบ่งเฟรมแบบแอสกี (ASCII framing)

การส่งข้อมูลด้วยเฟรม ASCII จะต้องส่งในโหมด ASCII เท่านั้นโดยรูปแบบของเฟรมจะต้องขึ้นต้นด้วยโคลอน colon (:) หรือ 3A ฐานสิบหกและจบด้วย carriage return-line feed (CRLF) หรือ 0D และ 0A ฐานสิบหก

รูปแบบของอักขระที่ส่งเป็นเลขฐานสิบหก (0-9, A-F) เมื่อไรก็ตามที่อุปกรณ์เครือข่ายได้รับโคลอนก็จะทำการถอดรหัสหาตำแหน่งอุปกรณ์ในฟิลด์ถัดมาทันที รูปแบบเฟรม ASCII แสดงดังภาพประกอบ 2-8 และข้อมูลที่ส่งแต่ละอักขระสามารถหน่วงเวลาได้ 1 วินาที

Start	Address	Function	Data	LRC	End
1 Char	2 Chars	2 Chars	n Chars	2 Chars	2 Chars CRLF

ภาพประกอบ 2-8 รูปแบบเฟรม ASCII

2.4.5.2 การแบ่งเฟรมแบบอาร์ทียู (RTU framing)

การส่งข้อมูลด้วยเฟรม RTU จะต้องส่งในโหมด RTU เท่านั้น โดยจุดเริ่มต้นของเฟรมหาได้จากการเว้นช่วงห่างระหว่างการส่งข้อมูลอย่างน้อย 3.5 เท่าของระยะเวลาในการส่งข้อมูลแต่ละอักขระ และการหาจุดสิ้นสุดของเฟรมก็หาได้เช่นเดียวกันกับจุดเริ่มต้นของเฟรม ข้อมูลที่ส่งจะอยู่ในรูปแบบของเลขฐานสิบหก (0-9, A-F) ซึ่งการส่งข้อมูลในโหมดนี้จะส่งข้อมูลอย่างต่อเนื่อง และเมื่อไรก็ตามที่ไม่มีการส่งข้อมูลในช่วงเวลา 1.5 เท่าของระยะเวลาในการส่งข้อมูลแต่ละอักขระก็จะเริ่มต้นรับข้อมูลจากเฟรมใหม่และเคลียร์ข้อมูลที่ไม่สมบูรณ์ในบัฟเฟอร์

ในกรณีที่ข้อมูลที่ส่งมาส่งอย่างต่อเนื่องโดยมีช่วงห่างระหว่างแต่ละไบต์ไม่ต่ำกว่า 1.5 เท่าของระยะเวลาในการส่งข้อมูลแต่ละอักขระก็จะนับว่าเป็นข้อมูลในเฟรมเดียวกันแต่ถ้าช่วงห่างระหว่างไบต์สูงกว่า 3.5 เท่าของระยะเวลาในการส่งข้อมูลจะเป็นการเริ่มต้นเฟรมใหม่ สำหรับฟิลด์สุดท้ายของเฟรมใช้ตรวจสอบความถูกต้องของข้อมูลที่ส่ง นั่นคือ CRC รูปแบบเฟรม RTU แสดงดังภาพประกอบ 2-9

Address	Function	Data	CRC Check
8 Bits	8 Bits	n x 8 Bits	16 Bits

ภาพประกอบ 2-9 รูปแบบเฟรม RTU

รูปแบบการแบ่งฟิลด์ที่เหมือนกันสำหรับการแบ่งเฟรมทั้ง 2 แบบ คือ มีฟิลด์ที่บอกตำแหน่งของอุปกรณ์ หน้าที่ที่ส่งให้ slave ทำงาน ข้อมูลที่ส่ง และการตรวจสอบความถูกต้องของข้อมูลซึ่งรายละเอียดในการจัดการฟิลด์แต่ละฟิลด์จะกล่าวถึงในลำดับถัดไป

2.4.6 การจัดการฟิลด์ ADDRESS

ฟิลด์ address ในโหมด ASCII จะใช้อักขระจำนวน 2 อักขระ และ 8 บิตในโหมด RTU เพื่อแทน address ของ slave โดยค่าของ address ต้องอยู่ในช่วง 0-247 ซึ่งค่าที่กำหนดในอุปกรณ์จะต้องไม่ซ้ำกันและจะต้องอยู่ในช่วง 1-247 เมื่อไรก็ตามที่ master ต้องการติดต่อกับ slave จะระบุตัว slave ด้วย slave address และ slave ก็จะตอบกลับไปยัง master โดยส่ง address ของตนเองไว้ที่ฟิลด์ address เพื่อให้ master ทราบว่าเฟรมที่ได้รับมาจาก slave ตัวไหน

address 0 เป็น address ที่ใช้ในการ broadcast ไปยัง slave ทุกตัวแต่เมื่อไรก็ตามที่นำโพรโตคอล Modbus ไปใช้บนเครือข่ายที่มีระดับสูงกว่าก็จะไม่สามารถ broadcast ได้อีก หรืออาจจะใช้วิธีอื่นแทนที่การ broadcast เช่น การใช้ฐานข้อมูลร่วมกันใน Modbus plus เพื่อให้สามารถปรับปรุงข้อมูลทั้งหมดได้ในครั้งเดียว

2.4.7 การจัดการฟิลด์ FUNCTION

ฟิลด์ function ในโหมด ASCII จะใช้อักขระจำนวน 2 อักขระ และ 8 บิต ในโหมด RTU ซึ่งค่าที่ใช้ในฟิลด์นี้อยู่ในช่วง 1-255

ข้อมูลในฟิลด์นี้แสดงถึงสิ่งที่ master ต้องการให้ slave ทำ เช่น การอ่านค่าสถานะ ON/OFF ของกลุ่ม coil หรืออินพุท อ่านข้อมูลของกลุ่มรีจิสเตอร์ อ่านสถานะของ slave เขียนข้อมูลลงใน coil หรือรีจิสเตอร์ที่ต้องการ หรือทำการโหลด บันทึก ตรวจสอบโปรแกรมที่อยู่บน slave

การตอบกลับของ slave มี 2 แบบ คือ การตอบกลับแบบธรรมดาเป็นการตอบกลับด้วยค่าเดิมกับที่ master ส่งมา (normal, error-free) และการตอบกลับแบบพิเศษเป็นการเปลี่ยนค่า most-significant bit ของฟิลด์ function ให้เป็น 1 เพื่อบอกให้ทราบว่ามีความผิดพลาดบางอย่างเกิดขึ้นโดยส่งข้อความผิดพลาดที่ฟิลด์ data ที่ส่งกลับไปยัง master เพื่อบอกว่ามีความผิดพลาดอะไรเกิดขึ้นหรือกรณีเฉพาะบางอย่างซึ่งโปรแกรมที่ทำงานบน master ต้องมีความสามารถในการจัดการกับกรณีต่าง ๆ ที่เกิดขึ้นได้ สำหรับตัวอย่างการส่งข้อมูล สมมติให้ master ส่งฟังก์ชันสำหรับอ่านค่า holding register ไปยัง slave ที่ต้องการ ดังรหัส

0000 0011 (03 ฐานสิบหก)

การตอบกลับแบบธรรมดาค่าที่ส่งไปในฟิลด์นี้จะส่งกลับด้วยค่าเดิมที่ได้รับมา ส่วนการตอบกลับแบบพิเศษจะตอบกลับด้วยค่า

1000 0011 (83 ฐานสิบหก)

2.4.8 ข้อมูลในฟิลด์ DATA

ข้อมูลในฟิลด์นี้จะใช้เลขฐานสิบหก 2 หลักในการแทนค่าโดยมีค่าอยู่ในช่วง 00 ถึง FF ข้อมูลในฟิลด์นี้เมื่อส่งจาก master ไปยัง slave จะบอกถึงข้อมูลเพิ่มเติมจาก function code ว่าต้องการให้ slave ทำอะไรโดยอาจจะรวมถึงค่า address ของรีจิสเตอร์ จำนวนอุปกรณ์ที่ต้องการติดต่อ และจำนวนข้อมูลจริง ๆ ในฟิลด์ข้อมูล เช่น ถ้า master ร้องขอไปยัง slave เพื่ออ่านค่ากลุ่มของ holding register โดยใช้ function code 03 ดังนั้นข้อมูลในฟิลด์ข้อมูลจะประกอบด้วยค่าเริ่มต้นของ holding register และจำนวนรีจิสเตอร์ที่ต้องการอ่าน ในกรณีที่ master ต้องการเขียนค่าลงในกลุ่ม holding register ของ slave ด้วย function code 10 ข้อมูลในส่วนของฟิลด์ข้อมูลจะระบุถึงตำแหน่งเริ่มต้นของรีจิสเตอร์ จำนวนรีจิสเตอร์ที่ต้องการเขียน จำนวนข้อมูลที่อยู่ในฟิลด์ข้อมูล และข้อมูลที่จะเขียนลงบนรีจิสเตอร์

กรณีที่ไม่มีความผิดพลาดเกิดขึ้น master ก็จะได้รับข้อความตอบกลับจาก slave แต่ถ้ามีความผิดพลาดเกิดขึ้น slave จะส่ง exception code ในฟิลด์ข้อมูลเพื่อให้แอพลิเคชันที่ทำงานอยู่บน master จัดการกับความผิดพลาดที่เกิดขึ้น

ข้อมูลในฟิลด์ข้อมูลสามารถมีความยาวเป็นศูนย์ได้ขึ้นอยู่กับสิ่งที่ master ร้องขอ เช่น การร้องขอล็อค (log) เหตุการณ์ในการติดต่อสื่อสารของ slave โดยใช้ function code 0B ซึ่งฟังก์ชันนี้ไม่จำเป็นต้องมีข้อมูลเพิ่มเติมเพื่อบอก slave ก็สามารถทำงานได้ตาม function code ที่ master ส่งมา

2.4.9 ข้อมูลในฟิลด์ ERROR CHECKING

ข้อมูลในฟิลด์ error checking แบ่งเป็น 2 แบบ ขึ้นอยู่กับรูปแบบเฟรมและโหมดในการส่งข้อมูล นั่นคือข้อมูลในฟิลด์นี้แบ่งตามโหมดในการส่งข้อมูล

ASCII ฟิลด์ error checking ประกอบด้วยอักขระ ASCII 2 อักขระซึ่งข้อมูลในฟิลด์นี้เป็นผลที่ได้จากการคำนวณหาค่า LRC ซึ่งเป็นอักขระสุดท้ายของเฟรมก่อนที่จะปิดเฟรมด้วยอักขระ CRLF

RTU ฟิลด์ error checking ประกอบด้วยข้อมูลจำนวน 16 บิต หรือ 2 ไบต์ซึ่งค่าที่เก็บในฟิลด์นี้เป็นผลที่ได้มาจากการคำนวณหาค่า CRC ซึ่งเป็นฟิลด์สุดท้ายของเฟรม โดยจะส่งไบต์ที่เป็น low-order ก่อนและตามด้วย high-order ดังนั้นไบต์สุดท้ายของเฟรมก็คือ high-order ของ CRC

วิธีการตรวจสอบความถูกต้องโดยการคำนวณหาค่า LRC และ CRC จะกล่าวต่อไปในหัวข้อ วิธีการตรวจสอบความถูกต้อง

2.4.10 การส่งอักขระข้อมูลติดต่อกันเป็นลำดับ

การส่งข้อมูลด้วยโพรโตคอล Modbus ข้อมูลที่ส่งเป็นอักขระหรือไบต์มีลำดับในการส่งจากซ้ายไปขวา ดังนี้

Least Significant Bit (LSB) … Most Significant Bit (MSB)

ลักษณะลำดับบิตในการส่งอักขระด้วยเฟรมแอสกีมีลำดับในการส่งแสดงดังภาพประกอบ 2-10 และลำดับบิตในการส่งอักขระด้วยเฟรมอาร์ทียูดังภาพประกอบ 2-11

จากภาพประกอบ 2-10 และ 2-11 เฟรมข้อมูลชนิดเดียวกันมีลำดับการส่งที่แตกต่างกันได้ขึ้นอยู่กับการกำหนดค่าเริ่มต้นก่อนการส่งว่าให้มีการใช้พาริตีบิตจำนวนเท่าไร ซึ่งสรุปได้ว่ารูปแบบการส่งอักขระข้อมูลนอกจากจะขึ้นอยู่กับรูปแบบเฟรมของข้อมูลที่ส่งแล้วยังขึ้นอยู่กับการกำหนดค่าเริ่มต้นในการส่งด้วย

เฟรมที่มีการตรวจสอบพาริตี

Start	1	2	3	4	5	6	7	Par	Stop

เฟรมที่ไม่มีการตรวจสอบพาริตี

Start	1	2	3	4	5	6	7	Stop	Stop

ภาพประกอบ 2-10 ลำดับบิตการส่งข้อมูลด้วยเฟรมแอสกี

เฟรมที่มีการตรวจสอบพาริตี

Start	1	2	3	4	5	6	7	8	Par	Stop

เฟรมที่ไม่มีการตรวจสอบพาริตี

Start	1	2	3	4	5	6	7	8	Stop	Stop

ภาพประกอบ 2-11 ลำดับบิตการส่งข้อมูลด้วยเฟรมอาร์ทียู

2.4.11 วิธีการตรวจสอบความผิดพลาด (Error checking method)

การตรวจสอบความผิดพลาดในการส่งข้อมูลของโพรโตคอล Modbus แบ่งเป็น 2 ประเภทใหญ่ ๆ คือ การตรวจสอบโดยใช้พาริตี และการตรวจสอบโดยใช้การคำนวณข้อมูลในเฟรม การตรวจสอบโดยใช้พาริตี (even หรือ odd) สามารถกำหนดให้มีการตรวจสอบหรือไม่ก็ได้ขึ้นอยู่กับการกำหนดค่าเริ่มต้นก่อนการส่งข้อมูล ส่วนการตรวจสอบโดยการคำนวณข้อมูลในเฟรม (LRC, CRC) ข้อมูลนั้น ๆ จะเป็นส่วนหนึ่งของเฟรมซึ่งมีการตรวจสอบข้อมูลทั้งเฟรมข้อมูลสำหรับตรวจสอบความถูกต้องทั้ง 2 แบบสร้างโดย master และส่งไปยัง slave จากนั้น slave ก็จะทำการตรวจสอบว่าข้อมูลต้นทางและปลายทางตรงกันหรือไม่ สำหรับการตรวจสอบ

ความถูกต้องในการส่งข้อมูลจาก slave กลับไปยัง master มีลักษณะเช่นเดียวกันกับ master ส่งมาให้ slave

การกำหนดค่าเวลาสำหรับคอยการตอบรับ (time out) จะถูกกำหนดที่ master เพื่อกำหนดเวลาที่ master คอยการตอบรับจาก slave ซึ่งค่านี้จะต้องนานกว่าเวลาที่ใช้ในการตอบรับปกติ ถ้า slave พบว่ามีความผิดพลาดเกิดขึ้น slave จะไม่ส่งข้อมูลตอบกลับไปยัง master นอกจากนี้ความผิดพลาดอาจเกิดขึ้นในระหว่างการรับส่งข้อมูลก็เป็นได้ เมื่อ master รอการตอบกลับจาก slave จนถึงเวลาที่ได้กำหนดไว้สำหรับรอคอยการตอบกลับ master ก็จะจัดการกับความผิดพลาดที่เกิดขึ้น

2.4.11.1 การตรวจสอบความถูกต้องในการส่งข้อมูลด้วยพาริตี

การใช้พาริตีในการตรวจสอบความผิดพลาดที่เกิดขึ้นผู้ใช้สามารถกำหนดเองได้ โดยค่าที่กำหนดมี 3 ค่า คือ even, odd และ no parity

กรณีที่กำหนดค่าพาริตีเป็น even หรือ odd จำนวนบิตที่ใช้สำหรับพาริตีมีจำนวน 1 บิต ซึ่งมีหน้าที่ในการนับจำนวนบิตที่เป็น 1 ในฟิลด์ข้อมูล (โหมดแอสกี 7 บิต และโหมดอาร์ทียู 8 บิต) ถ้าผลที่ได้จากการนับตรงกับค่าที่กำหนดไว้ว่าเป็น even หรือ odd พาริตีบิตก็จะมีค่าเป็น 0 ส่วนถ้าได้ผลไม่ตรงกับค่าที่กำหนดไว้ก็จะได้พาริตีบิตเป็น 1 เช่น สมมติให้มีการส่งข้อมูลด้วยโหมดอาร์ทียูโดยมีข้อมูลในฟิลด์ข้อมูลดังนี้

1100 0101

จะได้ว่าจำนวนบิตที่เป็น 1 มีจำนวน 4 บิต ซึ่งถ้าใช้ even parity พาริตีบิตจะมีค่าเป็น 0 แต่ถ้ากำหนดให้ใช้ odd parity พาริตีบิตจะมีค่าเป็น 1

กรณีที่กำหนดให้มีการตรวจสอบพาริตี master จะส่งข้อมูลไปพร้อมกับพาริตีบิต เมื่อถึงปลายทางที่ slave ก็จะทำการหาค่าพาริตีอีกครั้งหนึ่งและเปรียบเทียบกับพาริตีที่ได้รับว่าตรงกันหรือไม่ ซึ่งถ้าไม่ตรงหมายความว่าข้อมูลที่ได้รับมาไม่ถูกต้อง การตรวจสอบข้อมูลที่ส่งจาก slave มีลักษณะเช่นเดียวกันกับการตรวจสอบข้อมูลด้วยพาริตีที่ส่งมาจาก master นอกจากนี้อุปกรณ์ทุกตัวบนระบบทั้ง master และ slave ต้องกำหนดพาริตีให้เหมือนกันทั้งระบบ

แม้ว่าการตรวจสอบความผิดพลาดด้วยพาริตีบิตสามารถทำได้ง่ายและเร็ว แต่ไม่สามารถที่จะตรวจสอบได้อย่างถูกต้องในทุกกรณี เช่น กรณีที่บิต 2 บิตที่เป็น 1 มีความผิดพลาด

ระหว่างการส่งข้อมูล เมื่อตรวจสอบโดยใช้พาริตีจะไม่สามารถตรวจสอบได้ถูกต้องเนื่องจากจำนวนบิต 1 ที่นับได้จะมีค่าเป็น odd หรือ even เหมือนกับข้อมูลที่ถูกต้อง

กรณีที่กำหนดให้ไม่มีการตรวจสอบพาริตีจะไม่มีการการส่งพาริตีบิต แต่จะแทนตำแหน่งนั้น ๆ ด้วยสต็อป (stop) บิต

2.4.11.2 การตรวจสอบความถูกต้องในการส่งข้อมูลด้วย LRC

การตรวจสอบความผิดพลาดด้วยวิธี LRC ใช้ในโหมดแอสกี การตรวจสอบความถูกต้องของข้อมูลด้วย LRC เป็นการตรวจสอบความถูกต้องของข้อมูลทั้งเฟรมยกเว้นจุดเริ่มต้น (colon) และจุดสิ้นสุดของเฟรม (CRLF)

ข้อมูลในฟิลด์ LRC มีทั้งหมด 1 ไบต์ หรือ 8 บิตโดยฝ่ายส่งจะทำการคำนวณหาค่า LRC จากข้อมูลที่ต้องการส่งก่อนจากนั้นเมื่อข้อมูลไปถึงปลายทางฝ่ายรับก็จะคำนวณ LRC จากข้อมูลที่ได้รับถ้าค่า LRC ที่ได้รับกับค่าที่คำนวณใหม่ไม่ตรงกันแสดงว่ามีความผิดพลาดเกิดขึ้น การทำงานที่ฝ่ายส่งและฝ่ายรับมีลักษณะเดียวกัน

การคำนวณ LRC ทำได้โดยการบวกข้อมูลทั้ง 8 บิต โดยไม่คิดตัวทด และนำค่าที่ได้มาทำ 2’ complement แสดงดัง pseudo code ในภาพประกอบ 2-12

1. หาผลรวมของข้อมูลทุกไบต์ที่จะส่ง
2. นำผลลัพธ์ที่ได้จาก 1 หา 2’ complement ซึ่งก็จะได้ LRC ส่งไปพร้อมกับข้อมูล

ภาพประกอบ 2-12 ขั้นตอนวิธีในการคำนวณหาค่า LRC

2.4.11.3 การตรวจสอบความถูกต้องในการส่งข้อมูลด้วย CRC

การตรวจสอบความถูกต้องของข้อมูลด้วย CRC ใช้ในโหมดอาร์ทียูและสามารถตรวจสอบความถูกต้องของข้อมูลทั้งหมดที่ส่งได้

ข้อมูลในฟิลด์ CRC มีทั้งหมด 2 ไบต์ หรือ 16 บิต ซึ่งการตรวจสอบมีลักษณะเช่นเดียวกันกับ LRC นั่นคือ ฝ่ายส่งจะคำนวณและใส่ค่าที่ได้ไว้ในฟิลด์ error checking โดย

คำนวณจากข้อมูลที่ส่ง และฝ่ายรับก็จะคำนวณ CRC จากข้อมูลที่ได้รับจากนั้นก็ทำการเปรียบเทียบค่า CRC ที่ได้รับกับที่คำนวณได้ว่าตรงกันหรือไม่ ถ้าไม่ตรงกันแสดงว่ามีความผิดพลาดเกิดขึ้น

การหาค่า CRC จะไม่นำบิตเริ่มต้น บิตหยุด และพาริตีบิตมาใช้ในการคำนวณ การหาค่า CRC เริ่มต้นจากการโหลดข้อมูลจำนวน 16 บิตที่มีค่าเป็น 1 ทั้งหมด (FFFFH) ลงในรีจิสเตอร์ จากนั้นนำข้อมูลที่จะส่งจำนวน 8 บิตเก็บไว้ในรีจิสเตอร์ และนำค่าทั้ง 2 ค่ามาทำการ exclusive OR และนำค่าที่ได้ชิพไปทาง Least Significant Bit (LSB) 1 บิตและเติมค่า 0 ในตำแหน่ง Most Significant Bit (MSB) ตรวจสอบค่า 1 บิตที่ถูกชิพ ถ้ามีค่าเป็น 1 ให้ทำการ exclusive OR กับค่าหนึ่งค่าที่ได้กำหนดไว้ก่อน เช่น A001H แต่ถ้าค่าที่ได้เป็น 0 ก็ไม่ต้องทำการ exclusive OR และทำเช่นนี้ไปเรื่อย ๆ จนกระทั่งมีการชิพครบ 8 ครั้งก็จะได้ค่า CRC 1 ไบต์ ค่าที่ได้จะถูกเก็บไว้ในรีจิสเตอร์ และทำเช่นนี้ไปเรื่อย ๆ จนกระทั่งจบเฟรมข้อมูล (ไม่รวมบิตเริ่มต้นและบิตหยุด) นำค่าที่เก็บไว้ในรีจิสเตอร์มาใช้เป็นตัวตั้งในการคำนวณรอบต่อไปแทน FFFFH การคำนวณหา CRC แสดงเป็น pseudo code ได้ดังภาพประกอบ 2-13 โดยลำดับข้อมูลของ CRC จะขึ้นต้นด้วย low-order ก่อนและตามด้วย high-order

1. โหลดค่าคงที่ FFFFH (16 บิต) มาเก็บไว้ที่รีจิสเตอร์ (ACC)
2. นำข้อมูล 8 บิตที่จะส่งมา exclusive OR กับค่าในข้อ 1
3. ชิพขวา 1 บิต
4. ถ้าผลที่ได้จากการชิพเป็น 1 ให้นำ A001H มา exclusive กับค่าในรีจิสเตอร์ (ACC) แต่ถ้าเป็น 0 ก็ไม่ต้องทำอะไร
5. ถ้ายังไม่ครบ 8 บิต กลับไปที่ข้อ 3
6. นำค่าผลลัพธ์ที่ได้ในรีจิสเตอร์ (ACC) จากการคำนวณมาใช้เป็นตัวตั้งในการหา CRC ในไบต์ถัดไป (8 บิต)
7. กลับไปที่ข้อ 2 จนกระทั่งจบเฟรมข้อมูล (ไม่นำบิตเริ่มต้น บิตหยุด และพาริตีบิตมาใช้ในการคำนวณ)

ภาพประกอบ 2-13 ขั้นตอนวิธีในการคำนวณหาค่า CRC

2.5 Software tools

แอพลิเคชันที่ใช้ในการทดสอบการทำงานเชิงเวลาจริงสำหรับงานวิจัยนี้คือ อุปกรณ์ควบคุมที่รองรับโพรโตคอล Modbus และเพื่อเป็นการยืนยันความถูกต้องในการทำงาน ของระบบที่ทดสอบจึงนำโปรแกรมที่ผู้อื่นได้พัฒนาไว้แล้วมาใช้เป็นเครื่องมือสำหรับทดสอบการ ทำงาน สำหรับโปรแกรมหลักที่เลือกใช้ได้แก่ comDebug และ Modscan ซึ่งเป็นเครื่องมือ สำหรับตรวจสอบการรับส่งข้อมูลด้วยโพรโตคอล Modbus โดยมีรายละเอียดของโปรแกรมดังนี้

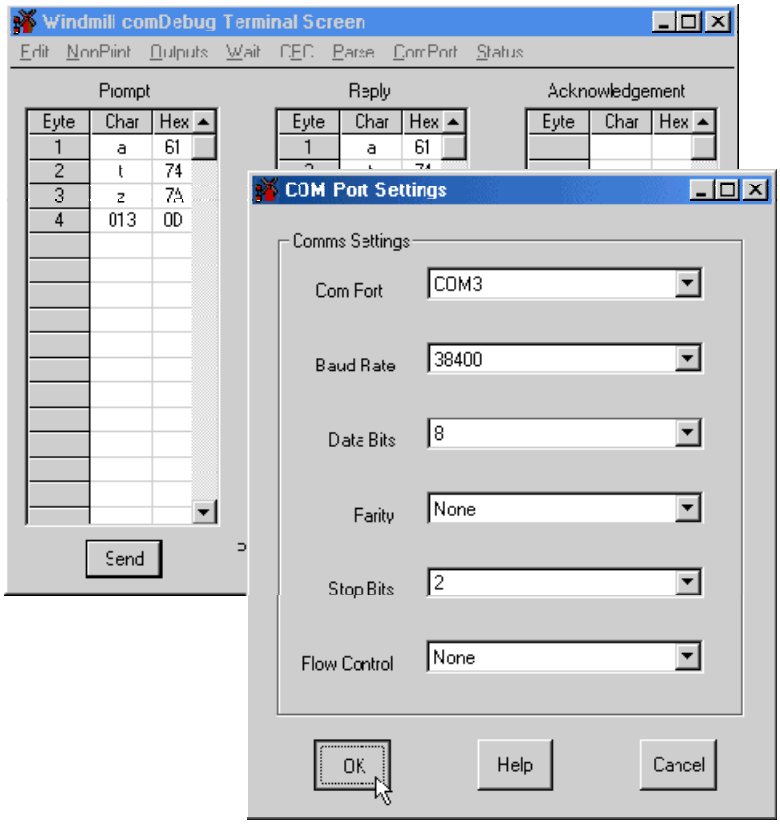
2.5.1 comDebug

comDebug เป็นโปรแกรมที่ดาว์นโหลดมาใช้งานได้ฟรี สามารถอ่านข้อมูล อุปกรณ์ที่มีการติดต่อกับเครื่องไมโครคอมพิวเตอร์ผ่าน COM พอร์ต โปรแกรม comDebug สามารถใช้ในการติดต่อสื่อสารผ่าน RS232 RS422 RS485 หรืออุปกรณ์ที่ติดต่อด้วย Modbus ช่วยในการตรวจสอบปัญหาที่เกิดจากการติดต่อสื่อสาร หรือใช้สำหรับตรวจสอบการสื่อสารของ อุปกรณ์ก่อนที่จะทำการร้องขอข้อมูลได้อย่างรวดเร็ว [38]

คุณสมบัติของโปรแกรม comDebug

- ใช้งานง่ายไม่ต้องเขียนโปรแกรมเพิ่มเติมแสดงดังภาพประกอบ 2-14
- เข้าใจข้อมูลที่เป็น ASCII และไบนารี
- สามารถส่งข้อมูลติดต่อกับอุปกรณ์ได้อย่างต่อเนื่องหรือตามความต้องการ
- สามารถระบุความผิดพลาดที่เกิดขึ้นและกลับคืนสู่การติดต่อสื่อสารที่ถูกต้อง ได้อย่างรวดเร็ว
- แทรก CRC ในข้อมูลได้
- ส่งค่า ASCII เลขจำนวนเต็ม 2’ complement ขนาด 16 บิต เลขจำนวนเต็ม บวก ข้อมูลขนาด 1 บิต และอื่น ๆ ไปยังอุปกรณ์
- ควบคุมสถานะบนช่องทางเอาท์พุท serial port บนเครื่องไมโครคอมพิวเตอร์
- มองเห็นสถานะบนช่องทางอินพุท serial port
- ส่งข้อความแจ้ง (acknowledgement)
- บันทึกค่าที่ได้กำหนดลงบนไฟล์
- มีคู่มือโปรแกรม

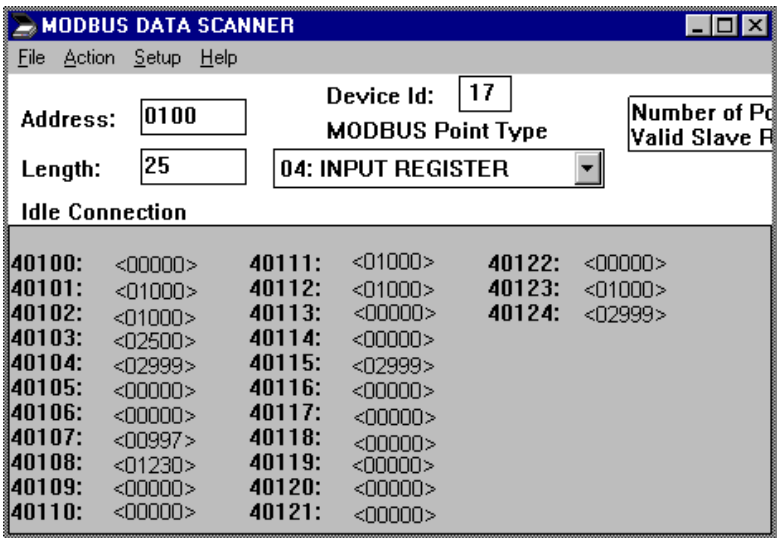
- กำหนดความเร็วการรับส่งได้สูงถึง 38,400 bps
- ใช้ได้บนระบบปฏิบัติการ Windows 95 หรือสูงกว่า



ภาพประกอบ 2-14 การใช้โปรแกรม comDebug

2.5.2 ModScan

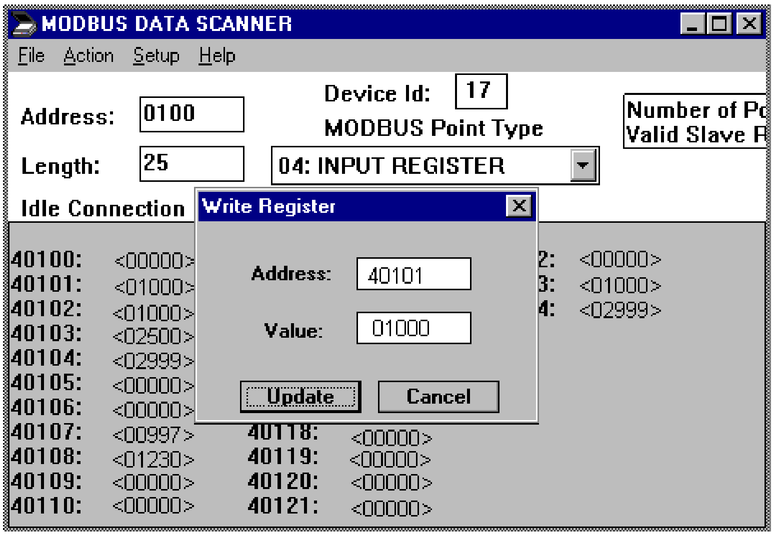
Modscan เป็นแอพลิเคชันโปรแกรมที่สามารถดาว์นโหลดและใช้งานได้ฟรี ทำงานอยู่บนระบบปฏิบัติการ Windows มีหน้าที่จำลองตัวเองให้ทำงานเหมือนกับอุปกรณ์ Modbus master และสามารถเข้าถึงข้อมูลอุปกรณ์ที่เป็น slave ที่เชื่อมต่อกัน โปรแกรมนี้ออกแบบให้เป็นเหมือนอุปกรณ์ที่ใช้สำหรับทดสอบและตรวจสอบความถูกต้องในการทำงานของโพรโตคอลบนระบบใหม่หรือระบบเดิมที่มีอยู่ ข้อมูล coil และรีจิสเตอร์ของ slave สามารถอ่านและ/หรือเขียนได้ด้วยโปรแกรม Modscan โดยใช้คำสั่งของ Modbus ซึ่งมีค่าตั้งแต่ 01 ถึง 06 การทำงานของ Modscan แสดงได้ดังภาพประกอบ 2-15 [39]



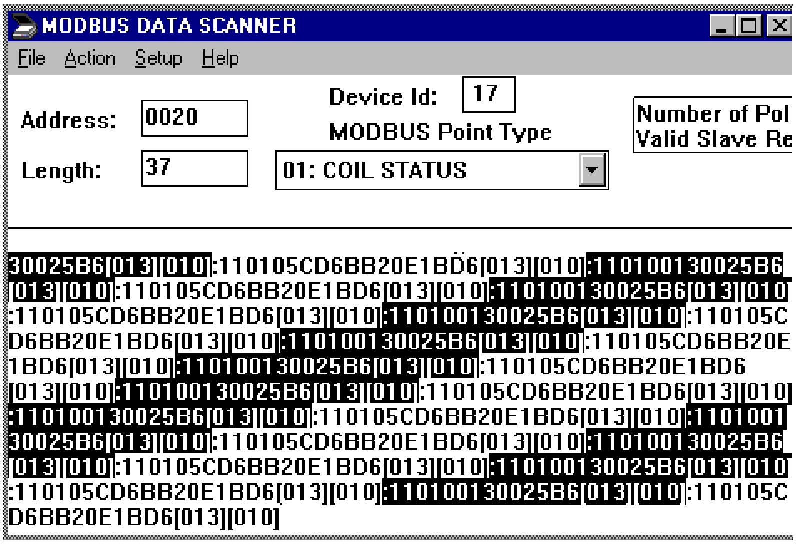
ภาพประกอบ 2-15 การทำงานของ Modscan

ผู้ใช้สามารถร้องขอหรือติดต่ออุปกรณ์ที่เป็น slave ที่ต้องการอย่างต่อเนื่องได้ด้วยโปรแกรม Modscan อุปกรณ์ slave ทุกตัวจะกำหนดตำแหน่งที่เฉพาะเจาะจงสำหรับการเข้าถึง ตำแหน่งของ slave และค่าของข้อมูลที่สัมพันธ์กันจะถูกสแกนและแสดงค่าเหล่านั้น สำหรับความผิดพลาดที่เกิดจากการติดต่อสื่อสารหรือข้อผิดพลาดที่ส่งกลับมาจาก slave จะแสดงบน Modbus status ในโปรแกรม การเขียนหรือแก้ไขค่าทำได้โดยการ double-click ที่ตำแหน่งหรือค่าที่แสดงบนโปรแกรมได้ผลดังภาพประกอบ 2-16

การแสดงผลสามารถแสดงในรูปแบบของเลขฐานสิบหกและฐานสิบ นอกจากนี้ยังสามารถเลือกโหมดการส่งข้อมูลให้เป็น ASCII หรือ RTU ได้ และคุณสมบัติหนึ่งของ Modscan คือสามารถแสดงข้อมูลจริงที่มีการรับส่งระหว่าง master และ slave ดังภาพประกอบ 2-17



ภาพประกอบ 2-16 การเขียนค่าลงบนตำแหน่งของ slave ที่ต้องการ



ภาพประกอบ 2-17 ข้อมูลจริงที่เกิดจากการรับส่งระหว่าง master และ slave