การพัฒนาซอฟต์พีแอลซีสำหรับภาษา IL ตามมาตรฐาน IEC 61131-3

THE DEVELOPMENT OF IEC 61131-3 SOFTPLC FOR IL LANGUAGE

ภานุ วัชระนฤมล
PANU VACHARANARUMOL

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2550

การพัฒนาซอฟต์พีแอลซีสำหรับภาษา IL ตามมาตรฐาน IEC 61131-3

THE DEVELOPMENT OF IEC 61131-3 SOFTPLC FOR IL LANGUAGE

ภานุ วัชระนฤมล

PANU VACHARANARUMOL

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2550

THE DEVELOPMENT OF IEC 61131-3 SOFTPLC FOR IL LANGUAGE

PANU VACHARANARUMOL

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN COMPUTER ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2007

หัวข้อวิทยานิพนธ์	การพัฒนาซอฟต์พีแอลซี สำหรับภาษา IL ตามมาตรฐาน IEC 61131-3
นักศึกษา	นายภานุ วัชระนฤมล
รหัสนักศึกษา	45061218
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2550
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผศ. อภิเนตร อูนากูล

บทคัดย่อ

การพัฒนาให้พีแอลซีมีขีดความสามารถสูงขึ้น และมีประสิทธิภาพดีขึ้น เพื่อให้สามารถรองรับกับงานที่ใช้พีแอลซีเป็นตัวควบคุม ในปัจจุบันที่มีความซับซ้อนมากขึ้น จำนวนงานมากขึ้น เป็นสิ่งที่น่าสนใจอย่างหนึ่ง วิทยานิพนธ์ฉบับนี้เสนอการออกแบบและพัฒนาสร้างพีแอลซียุคใหม่ในชื่อซอฟต์พีแอลซี โดยสามารถ Implement ส่วนหนึ่งของ IEC 61131-3 สามารถแสดงผลลัพธ์ที่ได้จากการทดลองว่าโปรแกรมที่สร้างมีประสิทธิภาพและมีความถูกต้อง ผลลัพธ์ที่ได้แสดงว่าโปรแกรมซอฟต์พีแอลซี ที่ทำงานแบบเรียวไทม์ และมีความสามารถในการทำมัลติทาสก์ได้ สามารถแก้ปัญหาของพีแอลซีทั่วไป ที่มีการทำงานแบบกระบวนการเดียวได้

Thesis Title	The Development of IEC 61131-3 SoftPLC for IL Language
Student	Mr. Panu Vacharanarumol
Student ID	45061218
Degree	Master of Engineering
Program	Computer Engineering
Year	2007
Thesis Advisor	Asst. Prof. Apinetr Unakul

ABSTRACT

The current challenge of conventional PLC is to allow for higher performance and greater efficiency in controlling more complex and numerous tasks, especially for users in factory automation. This paper proposes the design of a modern PLC program, code name SoftPLC, using partial implementation of IEC 61131-3. It presents the results of an experiment to test the performance and correctness of the program execution. The test results show that SoftPLC program functions in real-time with multi-tasking capacity and program-level portability. It solves the limitations of conventional, single-process PLC and demonstrates the potential of a new generation of PLC.

กิตติกรรมประกาศ

วิทยานิพนธ์เล่มนี้ สำเร็จได้ด้วยความกรุณาจากอาจารย์ที่ปรึกษา ผศ.อภิเนตร อูนากูล ซึ่งได้ให้ คำแนะนำ คำปรึกษา และกำลังใจ ในการทำวิทยานิพนธ์เล่มนี้มาโดยตลอด รวมทั้งอาจารย์ในภาควิชาวิศวกรรมคอมพิวเตอร์ทุกท่าน ที่ได้ให้ความรู้เพื่อใช้ในการทำวิทยานิพนธ์เล่มนี้ ข้าพเจ้ารู้สึกทราบซึ้งในความอนุเคราะห์จากท่านอาจารย์ และขอขอบพระคุณเป็นอย่างสูง

ขอขอบคุณ น้อง ๆ ร่วมรุ่น ทุกท่านที่ให้คำแนะนำและเป็นกำลังใจมาโดยตลอด

สุดท้ายนี้ข้าพเจ้าขอกราบขอบพระคุณ บิดา มารดา และครอบครัวของข้าพเจ้าที่เป็นกำลังใจ และให้การสนับสนุนในทุกเรื่องๆ ทำให้ข้าพเจ้าสามารถทำวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงด้วยดี

คุณค่าและประโยชน์อันพึงมาจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบแด่ผู้มีพระคุณทุกท่าน

ภานุ วัชระนฤมล

สารบัญ

หน้า

บทคัดย่อภาษาไทย I

บทคัดย่อภาษาอังกฤษ II

กิตติกรรมประกาศ III

สารบัญ IV

สารบัญตาราง VII

สารบัญรูป VIII

บทที่ 1 บทนำ 1

1.1 ความเป็นมาและความสำคัญของปัญหา 1

1.2 จุดมุ่งหมายและวัตถุประสงค์ของการศึกษา 1

1.3 แนวคิดที่ใช้ในงานวิจัย 2

1.4 ขอบเขตของการวิจัย 2

1.5 ขั้นตอนการศึกษา 3

1.6 เนื้อหาของวิทยานิพนธ์ 3

บทที่ 2 ทฤษฎีพื้นฐานและงานวิจัยที่เกี่ยวข้อง 4

2.1 พีแอลซี (PLC : Programmable Logic Controller) 4

2.1.1 โครงสร้างพื้นฐานของ พีแอลซี 7

2.1.2 วงรอบการทำงานของ พีแอลซี 8

2.1.3 ซอฟต์แวร์โมเดลของ พีแอลซี 9

2.2 IEC 61131 Standards 9

2.2.1 Common Elements 10

2.2.2 Programming Languages 11

2.2.3 รูปแบบซอฟต์แวร์ (Software Model) 13

2.2.4 Instruction List (IL) 14

2.3 ระบบปฏิบัติการเรียวไทม์ (RTOS : Real Time Operating System) 17

2.3.1 ระบบเรียวไทม์ (Real Time System) 17

2.3.2 ทาสก์ (Task) 17

2.3.3 ฟอร์กราวน์ด / แบคกราวน์ด (Foreground / Background) 18

2.3.4 การมัลติทาสก์ (Multitasking) 19

2.3.5 สถานะของทาสก์ (Task States) 21

สารบัญ (ต่อ)

หน้า

2.3.6 การเปลี่ยนบริบท (Context Switch) 21
2.3.7 การกำหนดเวลา (Scheduling) 22
2.3.8 ระบบปฏิบัติการเรียวไทม์ในกลุ่มเชิงพาณิชย์ 24
2.3.9 ไมโครคอนโทรลเลอร์ โอเพอเรติงซิสเต็ม เวอร์ชันทู (μC/OS II) 28

บทที่ 3 การออกแบบซอฟต์แวร์และฮาร์ดแวร์ 32
3.1 โครงสร้างซอฟต์แวร์ (Software Structure) 32
3.1.1 Soft PLC Software Model 32
3.1.2 โครงสร้างโปรแกรม (Program structure) 33
3.1.3 โครงสร้างข้อมูล (Data Structure) 34
3.1.4 ไฟล์โครงแบบ (Configuration file) 35
3.2 กลไกและขั้นตอนการทำงาน (Mechanism and Algorithms) 35
3.2.1 แผนผังการทำงาน (Activity Diagram) 36
3.2.2 การจัดการทาสก์และวิธีกำหนดตาราง (Task Management & Scheduling Policy) 38
3.2.3 Communication Path 41
3.2.4 Resource Sharing และ Race Condition 42
3.2.5 Hardware 42
3.3 IL Interpreter 43
3.4 Hardware Design 44

บทที่ 4 การทดลองและผลการทดลอง 52
4.1 Testing system methodology 52
4.2 Analysis of results 53
บทที่ 5 สรุปผลการวิจัย และข้อเสนอแนะ 56
5.1 สรุปผลการวิจัย 56
5.2 ข้อเสนอแนะ 56

เอกสารอ้างอิง 57

สารบัญ (ต่อ)

หน้า

ภาคผนวก 58
ผลงานวิจัยที่ได้รับการตีพิมพ์ 59

ประวัติผู้เขียน 66

สารบัญตาราง

ตารางที่ หน้า

2.1 IEC 61131-3 Programming Languages 12

4.1 Task Period Assignment 52

4.2 Task Execution Time 53

สารบัญรูป

รูปที่ หน้า

2.1 อุปกรณ์ต่าง ๆ ที่ประยุกต์ใช้งานกับ พีแอลซี 4

2.2 การใช้ พีแอลซี ควบคุมการผลิตอัตโนมัติในโรงงานอุตสาหกรรม 5

2.3 ตู้ควบคุมวงจรไฟฟ้า 6

2.4 พีแอลซีขนาดต่าง ๆ 6

2.5 โครงสร้างพื้นฐานของพีแอลซี 7

2.6 วงรอบการทำงานของพีแอลซี [12] 8

2.7 ซอฟต์แวร์โมเดล ของ PLC [12] 9

2.8 IEC 61131-3 Standard [12] 10

2.9 IEC 61131-3 Standard data types [5] 11

2.10 IEC 61131-3 Software Model 13

2.10 IL Operations [5] 14

2.11 ตัวอย่างโปรแกรมภาษา IL [5] 14

2.12 การเปลี่ยนจาก Ladder Diagram → Instruction List [5] 15

2.13 Ladder Diagram → Instruction List แบบ Allen Bradley [5] 16

2.14 Foreground/Background Systems [14] 18

2.15 เสมือนทำงานไปพร้อม ๆ กัน [11] 19

2.16 ทาสก์สอดแทรกเปลี่ยนกันทำงาน [11] 19

2.17 Cooperative multitasking [14] 20

2.18 Preemptive multitasking [14] 20

2.19 สถานการณ์ทำงานของทาสก์ [3] 21

2.20 Context Switch [3] 22

2.21 Round-Robin Scheduling [3] 23

2.22 Priority-based Scheduling [3] 23

2.23 RTLinux [15] 25

2.24 แสดงการเปรียบเทียบ Commercial RTOSs [15] 27

2.25 ผลการประเมินลำดับของ RTOSs [11] 28

2.26 Multiple Task [9] [10] 29

2.27 Priority Table [9] 30

2.28 State Transiton Diagram [9] [10] 30

สารบัญรูป (ต่อ)

รูปที่ หน้า

3.1 IEC 61131-3 Software Model [12] 32
3.2 Soft-PLC Software Model 33
3.3 Component diagram 33
3.4 Class diagram 34
3.5 โค้ดของ plcStruc.h 35
3.6 การอินเตอร์รัพซีพียูแบบมีคาบเวลา 36
3.7 Activity diagram 36
3.8 Link list ของ Ready List [10] 38
3.9 Task control flows 38
3.10 Source code ของ (ก) PLC Tsk scheduling (ข) Timer Task 39
3.11 Preemptive Priority Based Scheduling [14] 39
3.12 Priority Table ของ µCOS-II [9] 40
3.13 การจัดระดับ Priority ของซอฟต์พีแอลซี 41
3.14 การกำหนดเลขประจำทาสก์ (ID) และลำดับทาสก์ (Priority) 41
3.15 การกำหนดตำแหน่งทาง Hardware โดยตรง (Direct IO Port Map) 42
3.16 โครงสร้างทาง Hardware 44
3.17 โครงสร้างของ IO Device 45
3.18 แผนภาพการจัดวางหน่วยความจำในส่วนของ I/O ของบอร์ด x86 45
3.19 Block Diagram ของ IO Device 46
3.20 Spartan-II User I/O (XC2Sxx) 46
3.21 Block Diagram ที่ได้จาก ISE WebPACK 8.102i 47
3.22 RTL Schematic ของ ISA8 (IO Device) 47
3.23 RTL Schematic ของ U0 : IOPORT 48
3.24 RTL Schematic ของ U1 : Data_en 48
3.25 RTL Schematic ของ U2 : Latch_buf (Input Port → Data Bus) 49
3.26 RTL Schematic ของ U2 : Latch_buf (Data Bus → Output Port) 50
3.27 (ก) วงจรส่วน Input (ข) วงจรส่วน Output 51
4.1 แสดงการทำงานของซอฟต์พีแอลซี ที่ 4 พีแอลซีทาสก์ 52
4.2 Task Execution Timing Diagram 53

สารบัญรูป (ต่อ)

รูปที่ หน้า

4.3 แสดงการทำงานของซอฟต์พีแอลซี ที่ 12 พีแอลซีทาสก์ 54
4.4 Timing Diagram ของการทำงานที่ High load 55

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

พีแอลซี (PLC : Programmable Logic Controller) หรือ เอสซี (SC : Sequence Controller) เป็นเครื่องที่ใช้ในการควบคุมการทำงานแบบตามลำดับ (Sequential control) ถูกนำมาใช้มากในการ ควบคุมขบวนการผลิตอัตโนมัติในโรงงานอุตสาหกรรม (Factory Automation) เนื่องจากสามารถปรับเปลี่ยนการทำงานของพีแอลซีให้เป็นไปตามลักษณะของงานที่ต้องการควบคุม เพราะพีแอลซี ทำงานตามโปรแกรมที่ป้อนไว้ในหน่วยความจำของมัน การปรับเปลี่ยนการทำงานของพีแอลซี ทำได้โดยการเปลี่ยนโปรแกรมเพื่อควบคุมการทำงานของพีแอลซีใหม่ ตัวอย่างการประยุกต์ใช้งาน เช่น การควบคุมสายพานที่ใช้ในการลำเลียงวัตถุดิบ ผลิตภัณฑ์ หรือ การคัดแยกวัสดุ เป็นต้น ภาษาที่ใช้สำหรับเขียนโปรแกรมเพื่อควบคุมการทำงานของพีแอลซี มีรูปแบบเป็นของตัวเอง แตกต่างไปจากภาษาที่ใช้กับคอมพิวเตอร์ทั่วไป บุคลากรที่เขียนโปรแกรมควบคุมส่วนใหญ่มาจากช่างเทคนิค หรือ ผู้ที่ศึกษาทางเทคนิคหรือวิศวกรรมศาสตร์

ในปัจจุบัน ลักษณะของงานที่นำพีแอลซีไปควบคุม มีความซับซ้อนมากขึ้น มีจำนวนงานมากขึ้น เมื่อพิจารณาการทำงานของพีแอลซีในปัจจุบัน จะพบว่าถูกออกแบบให้มาทำงานในลักษณะกระบวนการเดียว (Single Process) นั่นหมายความว่าไม่ว่างานนั้นจะมีความซับซ้อนเพียงใดก็ตาม จะใช้เพียงโปรแกรมเดียวเพื่อควบคุมการทำงานทั้งหมด ผลที่ตามมาคือ เขียนโปรแกรมได้ยาก โปรแกรมมีความซับซ้อนและมีขนาดใหญ่ งานบางลักษณะที่มีความซับซ้อนมาก มีหลายงานที่ต้องทำพร้อม ๆ กัน เช่น การควบคุมแขนกลหลายแกน การทำงานแบบกระบวนการเดียวไม่สามารถรองรับได้

นอกจากนี้อีกปัญหาที่ผู้ใช้ประสบก็คือ ภาษาที่ใช้ในการเขียนโปรแกรม เพราะแต่ละบริษัทต่างก็มี เทคนิค รูปแบบ หรือ วิธีการของตัวเอง แม้จะเป็นภาษาเดียวกันก็จะมีข้อกำหนดปลีกย่อยที่แตกต่างกัน การที่จะนำโปรแกรมที่พัฒนาจนใช้งานได้ดีกับเครื่องของบริษัทหนึ่ง ไปใช้กับเครื่องของอื่นโดยตรง แทบเป็นไปไม่ได้ ต้องมีการแก้ไขหรือปรับเปลี่ยนให้เป็นไปตามที่บริษัทนั้น ๆ กำหนดก่อน จึงจะสามารถนำไปใช้งานได้ ทำให้เสียเวลาในการพัฒนาบุคลากรเพื่อเขียนโปรแกรมสำหรับเครื่องของแต่ละบริษัท ซึ่งเป็นการสิ้นเปลือง [6]

1.2 จุดมุ่งหมายและวัตถุประสงค์ของการศึกษา

วิทยานิพนธ์ฉบับนี้เป็นการออกแบบและพัฒนาซอฟต์พีแอลซี สำหรับภาษา IL ตามมาตรฐาน IEC 61131-3 โดยมีวัตถุประสงค์ดังต่อไปนี้

- เพื่อศึกษาพื้นฐานการทำงานของพีแอลซี และการนำไปประยุกต์ใช้งาน
- เพื่อศึกษามาตรฐานของพีแอลซี ตามที่ไอซีอี (IEC : International Electrotechnical Commission) ออกข้อกำหนดไว้
- เพื่อศึกษารูปแบบของภาษาไอแอล (IL : Instruction List) ตาม IEC 61131-3
- เพื่อศึกษา ระบบปฏิบัติการเรียวไทม์ (RTOS : Real Time Operating System) และการทำงานแบบมัลติทาส (Multi Tasking)
- เพื่อศึกษาการทำงานของพีแอลซีแบบกระบวนการเดียว (Single process) และแบบหลายกระบวนการ (Multi process)
- เพื่อเปรียบเทียบประสิทธิภาพที่ได้จากการทำงาน แบบกระบวนการเดียว กับ แบบหลายกระบวนการ

1.3 แนวคิดที่ใช้ในงานวิจัย

ปัจจุบันขบวนการผลิตอัตโนมัติในโรงงานอุตสาหกรรมมีความซับซ้อนมากขึ้น ปริมาณงานที่ควบคุมก็มีจำนวนมากขึ้น [6] การเขียนโปรแกรมเพื่อควบคุมการทำงานของพีแอลซีแบบธรรมดาทำได้ยากและมีขนาดใหญ่ ด้วยข้อจำกัดในการทำงานแบบกระบวนการเดียว (Single Process)

รูปแบบของภาษาที่มีความแตกต่างกันก็เป็นปัญหาที่สำคัญ เพราะต้องพัฒนาบุคลากรสำหรับควบคุมการทำงานของแต่ละเครื่อง นอกจากนี้ยังส่งผลต่อการศึกษาพีแอลซีในสถานศึกษาที่ใช้เครื่องพีแอลซีของบริษัทใด ก็ต้องเขียนโปรแกรมตามรูปแบบของบริษัทนั้น ๆ [6]

งานวิจัยนี้นำเสนอซอฟต์แวร์ '*ซอฟต์พีแอลซี*' ที่ทำงานภายใต้ระบบปฏิบัติการ µC/OS-II ซึ่งเป็นระบบปฏิบัติการเรียวไทม์ (RTOS : Real time Operating System) มีความสามารถในการทำงานแบบมัลติทาสก์ (Multi Tasking) เพื่อให้โปรแกรมซอฟต์พีแอลซี สามารถทำงานแบบหลายกระบวนการ (Multi Process) รองรับกับงานควบคุมที่มีความซับซ้อนมากขึ้นได้ ในส่วนภาษา IL ซึ่งเป็นหนึ่งในห้าภาษามาตรฐานตามที่ IEC กำหนดไว้ ได้ออกแบบให้มีโครงสร้างและรูปแบบของภาษาตามที่ IEC กำหนดไว้ในส่วนที่ 3 (IEC 61131-3 Data types and programming)

1.4 ขอบเขตของการวิจัย

วิทยานิพนธ์ฉบับนี้เป็นการออกแบบและพัฒนาซอฟต์แวร์ ซอฟต์พีแอลซี สำหรับภาษา IL ตามมาตรฐาน IEC 61131-3 โดยโปรแกรม ซอฟต์พีแอลซี สามารถทำงานแบบมัลติทาสก์ได้ และมีสมรรถภาพสูงพอ เมื่อนำไปทำงานบนคอมพิวเตอร์พีซี สามารถรองรับได้ถึง 12 พีแอลซีทาสก์ และสามารถนำไปพอร์ทลงบนบอร์ด x86

1.5 ขั้นตอนการศึกษา

- ศึกษาและออกแบบโปรแกรมซอฟต์พีแอลซีที่สามารถทำงานแบบมัลติทาสก์ได้บนระบบปฏิบัติการเรียวไทม์ μC/OS-II
- ศึกษาและออกแบบตัวแปลภาษา IL ให้เป็นไปตาม IEC 61131-3
- ทดสอบโปรแกรมเพื่อหาสมรรถภาพของโปรแกรม จำนวน พีแอลซี ทาสก์ สูงสุดที่สามารถรองรับได้
- ทดสอบโปรแกรมเพื่อหาเสถียรภาพของโปรแกรม
- ทดลองเพื่อหาประสิทธิภาพที่ได้จากการทำงานแบบหลายกระบวนการ เปรียบเทียบกับการทำงานแบบกระบวนการเดียว

1.6 เนื้อหาของวิทยานิพนธ์

วิทยานิพนธ์ฉบับนี้ แบ่งเนื้อหาออกเป็น 5 บท โดยแต่ละบทมีรายละเอียด ดังต่อไปนี้

บทที่ 1 กล่าวถึงความเป็นมาของงานวิจัย ความมุ่งหมายและวัตถุประสงค์ แนวคิดที่ใช้ในการวิจัย ขอบเขตของการวิจัย และขั้นตอนการศึกษา

บทที่ 2 กล่าวถึงทฤษฎีพื้นฐานต่าง ๆ ที่เกี่ยวข้องในการวิจัย คือ พื้นฐานการทำงานของพีแอลซี มาตรฐานและข้อกำหนดของ IEC 61131-3 และ ระบบปฏิบัติการเรียวไทม์ และ การทำมัลติทาสก์ ซึ่งเนื้อหาทั้งหมดนี้นำมาใช้ในการออกแบบโปรแกรมซอฟต์พีแอลซี

บทที่ 3 กล่าวถึงการออกแบบโปรแกรม โครงสร้างของโปรแกรม และ การออกแบบฮาร์ดแวร์

บทที่ 4 กล่าวถึงการหาค่าสมรรถภาพของโปรแกรม เปรียบเทียบประสิทธิภาพการทำงานแบบหลายกระบวนการ กับ การทำงานแบบกระบวนการเดียว

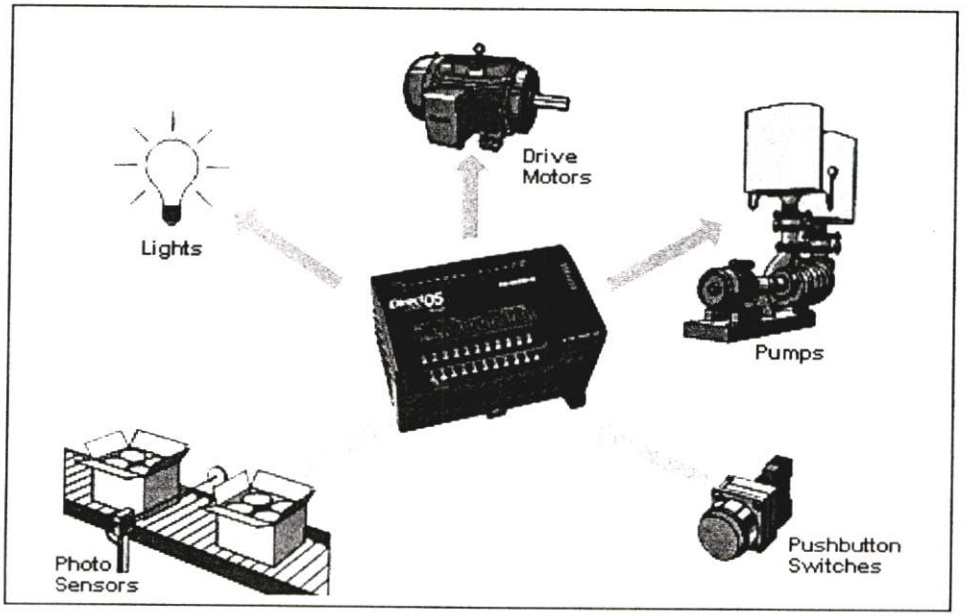
บทที่ 5 บทสรุปผลการวิจัยและข้อเสนอแนะ

บทที่ 2
ทฤษฎีพื้นฐานและงานวิจัยที่เกี่ยวข้อง

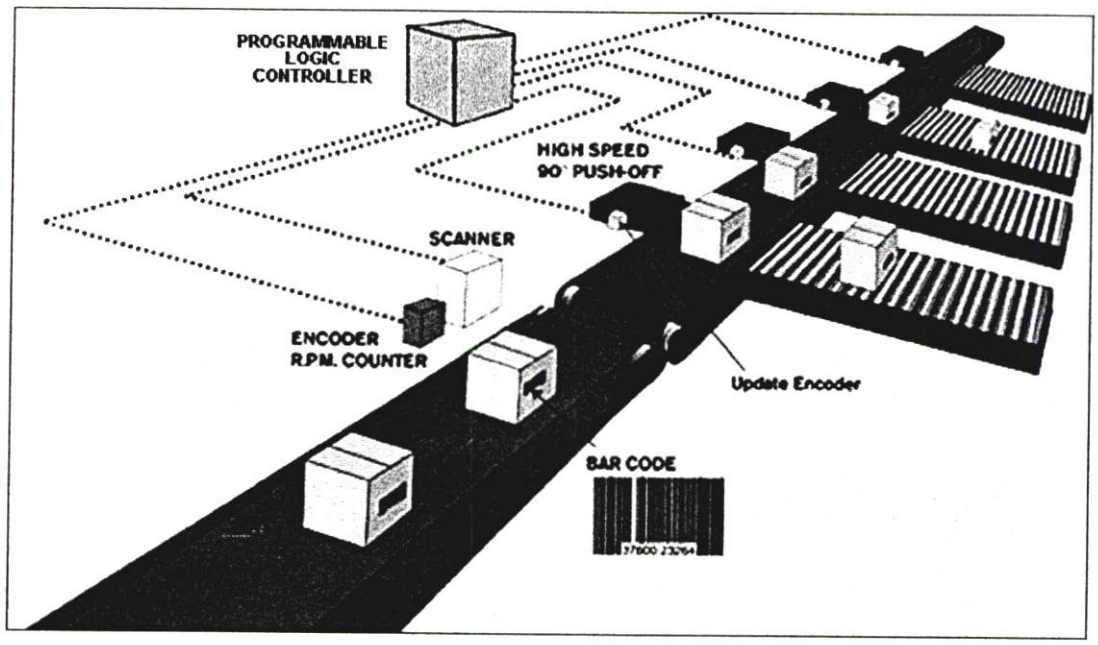
ในบทนี้จะกล่าวถึงทฤษฎีพื้นฐานต่าง ๆ ที่เกี่ยวข้องในการวิจัย คือ พื้นฐานการทำงานของพีแอลซี (PLC : Programmable Logic Controller) รูปแบบข้อมูล และภาษามาตรฐานของ พีแอลซี ตามข้อกำหนดของไออีซี (IEC 61131-3 Data type and programming) และระบบปฏิบัติการเรียวไทม์ (RTOS : Real Time Operating System) ซึ่งเนื้อหาทั้งหมดนี้นำมาใช้ในการออกแบบ และสร้างซอฟต์แวร์พีแอลซี หรือ ซอฟต์พีแอลซี (SoftPLC)

2.1 พีแอลซี (PLC : Programmable Logic Controller)

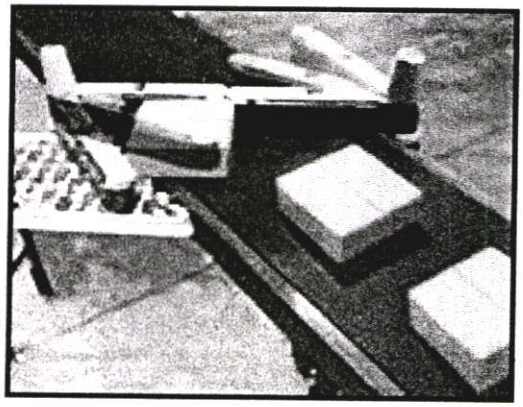
พีแอลซี เป็นเครื่องมือที่ถูกคิดค้นขึ้นมาเพื่อใช้ในการควบคุมอุปกรณ์ต่าง ๆ ได้หลายชนิด ดังแสดงในรูปที่ 2.1 ถูกนำมาใช้มากในการควบคุมขบวนการผลิตอัตโนมัติในโรงงานอุตสาหกรรม ชนิดของอุปกรณ์หรือเครื่องจักรที่สามารถนำพีแอลซีไปควบคุม มีอยู่หลากหลาย ตามแต่ขบวนการผลิตของโรงงานอุตสาหกรรมนั้นๆ เช่น ระบบสายพานลำเลียง การตรวจคัดแยกวัสดุ หรือ สายงานการประกอบผลิตภัณฑ์ เป็นต้น เนื่องจากพีแอลซีทำงานตามโปรแกรมที่ถูกป้อนไว้ก่อนในหน่วยความจำ ทำให้พีแอลซีมีความอ่อนตัว สามารถนำไปใช้ควบคุมงานได้หลายลักษณะ ขึ้นอยู่กับผู้ใช้พัฒนาโปรแกรมเพื่อให้พีแอลซีควบคุมการทำงาน ตามลักษณะงานที่ต้องการควบคุม



รูปที่ 2.1 อุปกรณ์ต่าง ๆ ที่ประยุกต์ใช้งานกับ พีแอลซี



(ก)

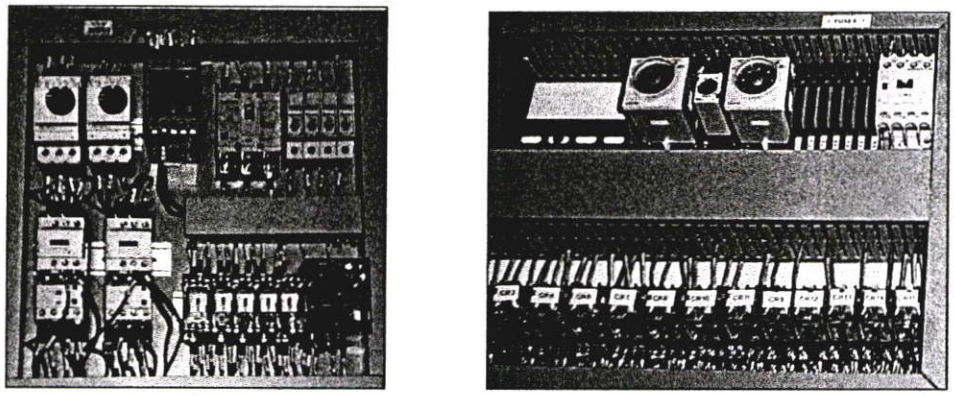


(ข) การตรวจคัดแยกวัสดุ

(ค) ระบบสายพานลำเลียง

รูปที่ 2.2 การใช้ พีแอลซี ควบคุมการผลิตอัตโนมัติในโรงงานอุตสาหกรรม

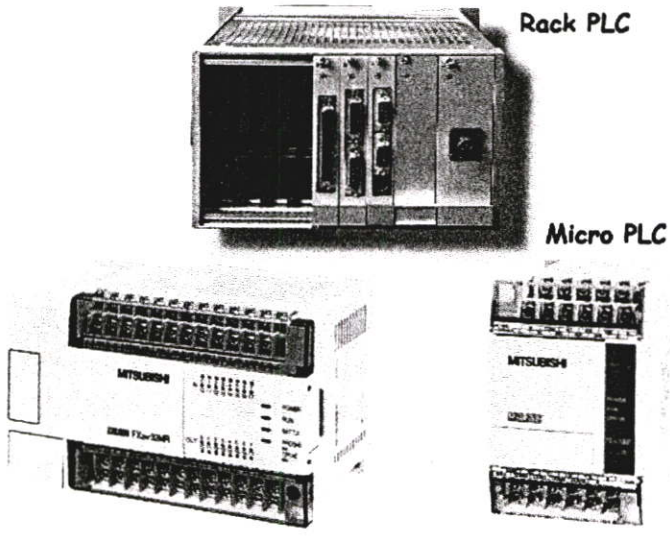
พีแอลซี ถูกออกแบบให้ทำงานแทนการควบคุมด้วยวงจรไฟฟ้าแบบเดิม ที่ประกอบด้วยอุปกรณ์ฟ้าเช่น รีเลย์ (Relay) เครื่องจับเวลา (Timer) หรือตัวนับ (Counter) ประกอบกันเป็นวงจรไฟฟ้าใช้สำหรับควบคุมเฉพาะงานใดงานหนึ่ง ดังแสดงในรูปที่ 2.3



รูปที่ 2.3 ตู้ควบคุมวงจรไฟฟ้า

การทำงานจะเป็นการควบคุมสถานะของอุปกรณ์ให้ “เปิด” หรือ “ปิด” ตามเงื่อนไขที่กำหนด แต่วงจรควบคุมแบบนี้มีข้อเสียคือ การปรับเปลี่ยนการทำงานต้องรื้อวงจรเดิมออกแล้วเดินสายไฟใหม่ (Hard wiring) และข้อเสียอีกอย่างก็คือ ใช้พลังงานฟ้าฟ้าสูง

พีแอลซี เข้ามาทดแทนโดยการสร้างอุปกรณ์หลักเหล่านี้ไว้ในตัวพีแอลซี เป็นอุปกรณ์เสมือน ใช้สถานะทางโลจิก “1” หรือ “0” แทน การ “เปิด” หรือ “ปิด” การต่อสายจึงทำโดยการใช้โปรแกรม (Soft wiring) มีการสร้างภาษาสำหรับควบคุมการทำงานของพีแอลซีโดยเฉพาะขึ้นมา



รูปที่ 2.4 พีแอลซีขนาดต่าง ๆ

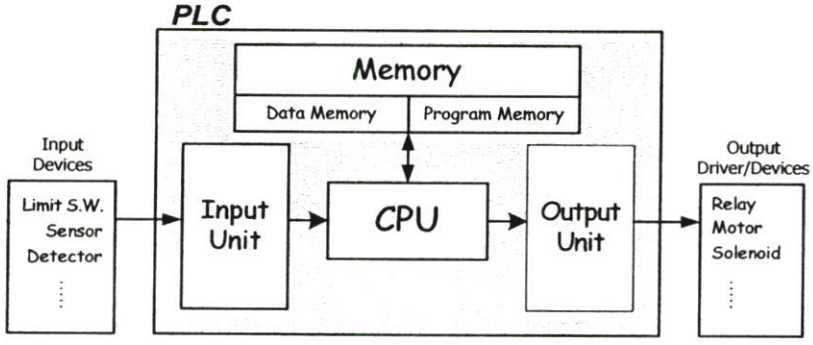
การจัดขนาดของพีแอลซีจะแบ่งตามจำนวนจุดต่ออินพุทและเอาท์พุท (I/O point) และจำนวนคำสั่งหรือ Step ที่สามารถรับได้ในแต่ละวงรอบการทำงานของพีแอลซี เช่น 4K (4,000 คำสั่ง) หรือ 10K (10,000 คำสั่ง) เป็นต้น รูปที่ 2.4 แสดงพีแอลซีขนาดต่าง ๆ ที่ใช้ในปัจจุบัน

จากที่ พีแอลซี ถูกออกแบบให้ทำงานทดแทนการควบคุมแบบเดิมที่เป็นวงจรไฟฟ้า ดังนั้นภาษาที่นำมาใช้ในการโปรแกรมให้พีแอลซีทำงาน จึงเลียนแบบมาจากวงจรฟ้า โดยใช้สัญญาลักษณ์ของ หน้าสัมผัส (Contact) และขดลวด (Coil) มาเขียนเป็น Diagram เป็นแบบภาษารูปภาพ (Graphics Langauge) เรียกภาษานี้ว่า แลดเดอร์ไดอะแกรม (LD : Ladder Diagram) เป็นภาษาที่ผู้ศึกษาทางด้านเทคนิคหรือวิศวกรรม ศึกษาเป็นภาษาแรก แต่ภาษาที่ใช้สำหรับเขียนโปรแกรมเพื่อควบคุมการทำงานของ พีแอลซี มีอีกหลายภาษา แต่ที่นิยมใช้มีอยู่ 5 ภาษา คือ [5]

1. IL (Instruction List)
2. ST (Structured Text)
3. LD (Ladder Diagram)
4. FBD (Function Block Diagram)
5. SFC (Sequential Function Charts)

2.1.1 โครงสร้างพื้นฐานของ พีแอลซี

โครงสร้างพื้นฐานของ พีแอลซี ประกอบด้วย 4 ส่วนหลัก ดังแสดงในรูปที่ 2.5



รูปที่ 2.5 โครงสร้างพื้นฐานของพีแอลซี

หน่วยความจำ (Memory Unit) แบ่งออกเป็น 2 ส่วน คือ

- *หน่วยความจำข้อมูล* (Data Memory) ทำหน้าที่เก็บสถานะของอุปกรณ์อินพุต (Input devices) ที่ได้รับมาจากอุปกรณ์ภายนอก และข้อมูลที่ใช้งานต่าง ๆ (Working data)
- *หน่วยความจำโปรแกรม* (Program Memory) ทำหน้าที่เก็บ โปรแกรมหรือชุดคำสั่งที่ใช้ควบคุมการปฏิบัติงานของ พีแอลซี

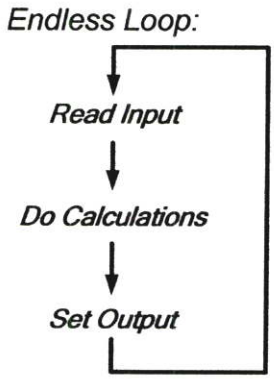
หน่วยรับข้อมูล (Input Unit) ทำหน้าที่รับข้อมูลจากอุปกรณ์อินพุตต่าง ๆ เช่น ลิมิตสวิท (Limit Switch) โฟโตดีเทคตอร์ (Photo detector) พร็อกซิมิตีดีเทคเตอร์ (Proximity detector) หรือ รีเลย์ (Relay)

หน่วยส่งออกข้อมูล (Output Unit) ทำหน้าที่รับข้อมูลจาก ซีพียู แล้วส่งไปยังตัวขับของอุปกรณ์ต่าง ๆ เช่น โซลินอยด์ (Solenoid) มอเตอร์ หรือ รีเลย์

หน่วยประมวลผลกลาง (CPU : Central Processing Unit) ทำหน้าควบคุมจังหวะการทำงานของพีแอลซี อ่านสถานะของอุปกรณ์อินพุต ประมวลผลตามโปรแกรม แล้วส่งผลลัพธ์ออกที่หน่วยส่งออกข้อมูล

2.1.2 วงรอบการทำงานของ พีแอลซี

วงรอบการทำงาน (Execution cycle) ของ PLC ประกอบ 3 ขั้นตอน ต่อเนื่องกันไปตลอดเวลา [12] ดังแสดงในรูปที่ 2.6



รูปที่ 2.6 วงรอบการทำงานของพีแอลซี [12]

วงรอบการทำงานของ พีแอลซี ประกอบด้วยขั้นตอนหลัก ๆ มีอยู่ 3 ขั้นตอน คือ

1. อ่านสถานะของอุปกรณ์อินพุตทั้งหมด (Read Input) นำมาเก็บไว้ในหน่วยความจำ
2. นำข้อมูลที่ได้มาประมวลผลตามโปรแกรม (Do Calculations) ตั้งแต่คำสั่งแรกไปจนจบโปรแกรม
3. ส่งผลลัพธ์ที่ได้จากการประมวลผลออกที่หน่วยส่งออกข้อมูล (Set Output)

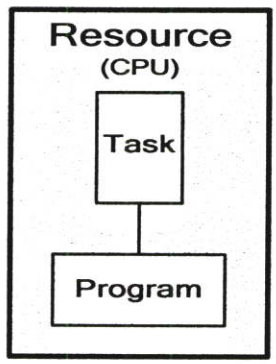
หลังจากทำงานครบทั้ง 3 ขั้นตอนแล้วก็จะวนกลับไปทำขั้นตอนที่ 1 ใหม่ ทำต่อเนื่องไปเรื่อย ๆ เป็นการวนรอบไม่รู้จบ (Endless loop) จากวงรอบการทำงานของ พีแอลซี ที่มีอยู่ 3 ขั้นตอน เวลาที่ใช้ในการทำงานแต่ละรอบเรียกว่า สแกนไทม์ (Scan time)

เวลาที่ใช้ในการทำงานแต่ละรอบ มีความสัมพันธ์โดยตรงกับการเปลี่ยนแปลงของสัญญาณอินพุต นั่นคือ เวลาที่ใช้ในการทำงานแต่ละรอบ ต้องน้อยกว่าเวลาของการเปลี่ยนแปลงของสัญญาณอินพุต เพราะถ้าเวลาที่ใช้ในการทำงานแต่ละรอบ มากกว่าเวลาของการเปลี่ยนแปลงของสัญญาณอินพุต จะทำให้สัญญาณอินพุตบางสัญญาณหายไป

2.1.3 ซอฟต์แวร์โมเดลของ พีแอลซี

ซอฟต์แวร์โมเดลของ พีแอลซี โดยทั่วไป [12] ดังแสดงในรูปที่ 2.7 ประกอบด้วย

- Resource หรือ CPU ทำหน้าที่ควบคุมการทำงานทั้งหมด
- Task ทำหน้าที่อ่านคำสั่งจากโปรแกรม แล้ว execute
- โปรแกรม ซึ่งเขียนโดยภาษาใดภาษาหนึ่งใน 5 ภาษามาตรฐาน ตาม IEC 61131-3



รูปที่ 2.7 ซอฟต์แวร์โมเดล ของ PLC [12]

พีแอลซี โดยทั่วไปที่ประกอบด้วย หนึ่ง Resource หนึ่ง Running task และ ควบคุมหนึ่งโปรแกรม เรียกการทำงานลักษณะนี้ว่า *Single Process* [9]

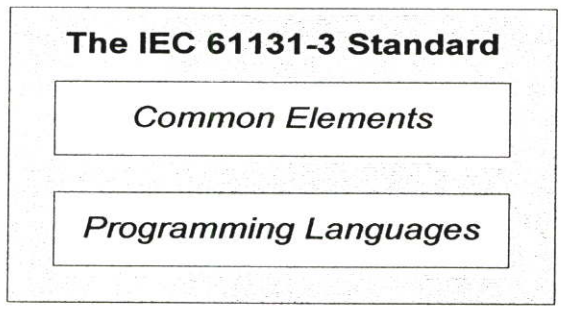
2.2 IEC 61131 Standards

ในอดีตที่ผ่านมาหรือแม้กระทั่งในปัจจุบัน บริษัทผู้ผลิตพีแอลซี เป็นผู้ถือครองลิขสิทธิ์ทั้ง Hardware และ Software เป็นลักษณะของระบบปิด (Closed System) ทำให้บริษัทผู้ผลิตพีแอลซีสามารถควบคุมเทคโนโลยีและมีอำนาจในการต่อรอง โดยผู้ใช้ถูกจำกัดในการเลือกใช้ ขยายหรือดัดแปลง เพื่อให้เหมาะสมกับการใช้งาน ต้องคอยเครื่องรุ่นใหม่ โดยหวังว่าจะมีลักษณะที่ต้องการหรือต้องจ้างบริษัทนั้นๆ มาดำเนินการให้ โดยเฉพาะภาษาที่ใช้ในการเขียนโปรแกรม การที่จะนำโปรแกรมที่พัฒนาจนใช้งานได้สมบูรณ์แล้วไปใช้กับเครื่องพีแอลซี ที่ผลิตจากต่างบริษัทโดยตรงเป็นไปไม่ได้ ต้องปรับเปลี่ยนให้อยู่ในรูปแบบที่บริษัทนั้น ๆ กำหนดก่อน ทำให้เสียเวลา ต้องพัฒนาบุคลากรขึ้นสำหรับเขียนโปรแกรมเพื่อใช้กับของแต่ละบริษัท ในปี ค.ศ. 1992 ได้มีการรวมตัวกันของ กลุ่มกำหนดมาตรฐาน (Standard groups) และกลุ่มบริษัทผู้ผลิตพีแอลซี เพื่อแก้ปัญหานี้ ในชื่อ International Electrotechnical Commission (IEC) และได้ออกข้อกำหนดเป็นมาตรฐานคือ IEC 11131 standards มีการนำมาทบทวนใหม่เป็น IEC 611131 standards [5] เป็นข้อกำหนดของระบบพีแอลซียุคใหม่ ซึ่งข้อกำหนดเหล่านี้จะเกี่ยวข้องกับพีแอลซี ทั้งทางด้านฮาร์ดแวร์ และการโปรแกรม แบ่งออกเป็น 5 ส่วน คือ

1) IEC 61131-1 Overview เป็นคำจำกัดความทั่วไปและลักษณะการทำงานที่ทำให้พีแอลซีแตกต่างไปจากระบบอื่น ๆ
2) IEC 61131-2 Requirements and Test Procedures เกี่ยวข้องกับทางด้านฮาร์ดแวร์ เป็นข้อกำหนด ทางไฟฟ้า ทางกล การทำงานของอุปกรณ์ และ วิธีการตรวจสอบเพื่อให้เป็นตามข้อกำหนด
3) IEC 61131-3 Data types and programming กล่าวถึง ภาษามาตรฐานที่ใช้ในการโปรแกรมพีแอลซี ข้อกำหนดเกี่ยวกับรูปแบบ, Syntax, Semantic ของภาษา การกำหนดชนิดและรูปแบบของข้อมูล
4) IEC 61131-4 User Guidelines เป็นข้อกำหนดเกี่ยวข้องกับหนังสือคู่มือ วิธีการใช้และการบำรุงรักษาอุปกรณ์
5) IEC 61131-5 Communications เกี่ยวข้องกับการติดต่อสื่อสารกับ ระบบพีแอลซีอื่นหรือ การติดต่อกับอุปกรณ์อื่น ๆ

โดยข้อกำหนดมาตรฐานเหล่านี้จะอยู่ในขอบเขตที่ บริษัทผู้ผลิตพีแอลซี สามารถใช้เทคนิคของตัวเอง แต่รูปแบบของข้อมูลหลัก การอินเตอร์เฟซกับผู้ใช้ จะเป็นไปตามข้อกำหนด [4]

ส่วนที่ 3 IEC 61131-3 Data types and programming เป็นส่วนที่มีผลโดยตรงกับผู้ใช้ เป็นส่วนที่เกี่ยวข้องกับภาษาที่ใช้ในการเขียนโปรแกรมควบคุมการทำงานของ พีแอลซี รายละเอียดและข้อกำหนดต่าง ๆ ที่ระบุไว้ใน IEC 61131-3 สามารถพิจารณาได้หลายลักษณะ พิจารณาโดยแบ่ง IEC 61131-3 ออกเป็นสองส่วน [12] ดังแสดงในรูปที่ 2.8



รูปที่ 2.8 IEC 61131-3 Standard [12]

2.2.1 Common Elements

ภายใน Common element เป็นการประกาศตัวแปรที่จะใช้ในการเก็บข้อมูลหรือประมวลผล โดย เป็นการประกาศตัวแปรพร้อมทั้งระบุชนิดของข้อมูล (INT, REAL) เช่น

Temperature_Sensor_1 : Integer

กรณีตัวแปรที่เป็น อินพุทหรือเอาท์พุท จะต้องระบุตำแหน่งที่แท้จริงทางกายภาพ ชนิดของข้อมูลที่ IEC 61131-3 กำหนดไว้เป็นมาตรฐานดังแสดงในรูปที่ 2.9

Name	Type	Bits	Range
BOOL	boolean	1	0 to 1
SINT	short integer	8	-128 to 127
INT	integer	16	-32768 to 32767
DINT	double integer	32	-2.1e-9 to 2.1e9
LINT	long integer	64	-9.2e19 to 9.2e19
USINT	unsigned short integer	8	0 to 255
UINT	unsigned integer	16	0 to 65536
UDINT	unsigned double integer	32	0 to 4.3e9
ULINT	unsigned long integer	64	0 to 1.8e20
REAL	real numbers	32	
LREAL	long reals	64	
TIME	duration	not fixed	not fixed
DATE	date	not fixed	not fixed
TIME_OF_DAY. TOD	time	not fixed	not fixed
DATE_AND_TIME. DT	date and time	not fixed	not fixed
STRING	string	variable	variable
BYTE	8 bits	8	NA
WORD	16 bits	16	NA
DWORD	32 bits	32	NA
LWORD	64 bits	64	NA

รูปที่ 2.9 IEC 61131-3 Standard data types [5]

2.2.2 Programming Languages

ภาษามาตรฐานตามที่ IEC 61131-3 กำหนดไว้มี 5 ภาษา [4][7] คือ

1) IL (Instruction List)

2) ST (Structured Text)

3) LD (Ladder Diagram)

4) FBD (Function Block Diagram)

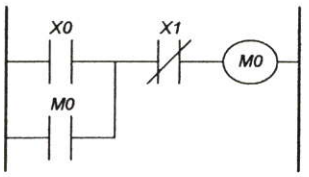
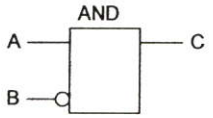
5) SFC (Sequential Function Charts)

ทั้ง 5 ภาษาจะมี รูปแบบและลักษณะโครงสร้างของภาษาแตกต่างกัน แบ่งเป็น

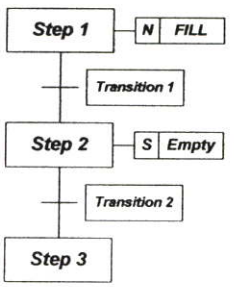
- แบบรูปภาพ (Graphical) 3 ภาษา คือ Ladder Diagram (LD), Function Block Diagram (FBD) และ Sequential Function Chart (SFC)
- แบบตัวอักษร (Textual) 2 ภาษา คือ Instruction List (IL) และ Structured Text (ST)

ตัวอย่างดังแสดงในตารางที่ 2.1 โดย IEC 61131-3 จะกำหนดรูปแบบ องค์ประกอบต่าง ๆ ของแต่ละภาษา เพื่อให้มีลักษณะเป็นมาตรฐานเดียวกัน เช่น ลักษณะการประกาศตัวแปร ฟังก์ชั่น และฟังก์ชันบล็อก เป็นต้น ทำให้ผู้ใช้สามารถที่จะนำโปรแกรมที่พัฒนาจนทำงานได้สมบูรณ์แล้วไปใช้กับ พีแอลซี ที่ผลิตจากบริษัทอื่นที่ใช้มาตรฐาน IEC 61131-3 ได้ โดยไม่ต้องปรับเปลี่ยนแก้ไขโปรแกรมใหม่

ตารางที่ 2.1 IEC 61131-3 Programming languages

LD : *Ladder Diagram* 	เป็นภาษาที่เขียนอยู่ในรูปของกราฟิค มีพื้นฐานมาจากวงจรควบคุมแบบรีเลย์ และวงจรไฟฟ้า แลดเดอร์ ไดอะแกรม จะประกอบด้วย ราง (Rail) ทั้งซ้ายและขวา ของไดอะแกรม เพื่อใช้สำหรับเชื่อมต่ออุปกรณ์ที่เป็นสวิตช์หน้า สัมผัส เพื่อเป็นทางผ่านของกระแส และมี ขดลวดหรือคอยล์ เป็นเอ้าท์พุต
IL : *Instruction List* LD X0 OR M0 ANDN X1 ST M0	IL จะเป็นภาษาที่เขียนอยู่ในรูปของข้อความ และมีลักษณะคล้ายกับภาษาแอสเซมบลี (Assembly) และภาษาเครื่อง (Machine code) ซึ่งภายในหนึ่งคำสั่งควบคุมจะประกอบด้วย ส่วนปฏิบัติการ (Operator) และส่วนที่ถูกดำเนินการ (Operand)
FBD : *Function Block Diagram* 	เป็นภาษาที่แสดงฟังก์ชั่นการทำงานในรูปของกราฟิคเช่นเดียวกัน และเชื่อมต่อกันเป็นโครงข่าย โดยการเขียนโปรแกรมในรูปของฟังก์ชั่นบล็อคไดอะแกรม จะมีพื้นฐานมาจาก ลอจิกไดอะแกรม

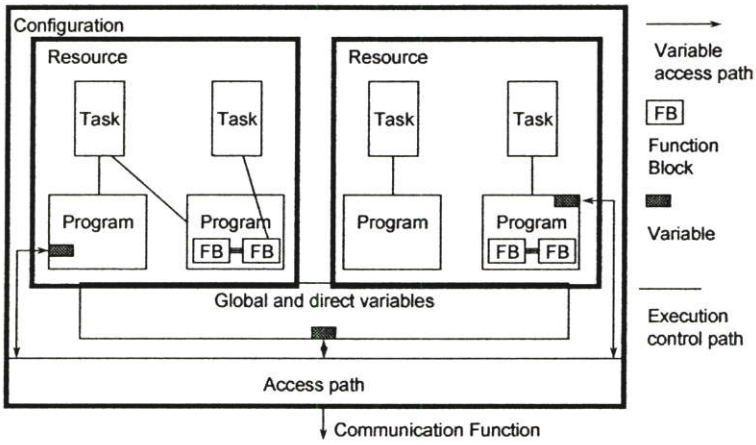
ST : *Structured Text*:
FUNCTION_BLOCK divide
VAR_INPUT
a: INT;
b: INT;
END_VAR
VAR_OUTPUT
c: INT;
END_VAR
IF b <> 0 THEN
c := a / b;
ELSE
c := 0;
END_IF;
END_FUNCTION_BLOCK
	ST จะเป็นภาษาในระดับสูง โดยมีพื้นฐานมาจากภาษา Pascal ซึ่งจะประกอบไปด้วย นิพจน์ และคำสั่ง โดยคำสั่งทั่วไปจะอยู่ในรูปของคำสั่งเกี่ยวกับการเลือกทำงาน เช่น
	IF……..THEN……..ELSE เป็นต้น
	คำสั่งเกี่ยวกับการทำงานซ้ำ เช่น FOR , WHILE เป็นต้น

SFC : *Sequential Function Chart* 	SFC จะเป็นภาษาที่รองรับการเขียนโปรแกรมที่มีโครงสร้างการทำงานเป็นแบบซีเควนซ์ ซึ่งส่วนประกอบของ SFC จะประกอบด้วย Step (คำสั่งในการปฏิบัติการในแต่ละขั้นตอน) และ Transition (เงื่อนไขที่กำหนดให้กระทำคำสั่งในแต่ละ Step) นอกจากนี้ยังสามารถกำหนดลักษณะการทำงาน เช่น Alternative step sequence และ Parallel step sequence เป็นต้น

บริษัทผู้ผลิตส่วนใหญ่ได้ปรับเปลี่ยนให้เป็นไปตาม IEC 61131-3 ให้ภาษาที่ใช้ มีรูปแบบตามมาตรฐานที่กำหนดไว้ ยกเว้น Function Block Diagram ยังคงมีส่วนที่แตกต่างกัน เพราะ IEC 61131-3 กำหนดมาตรฐานไว้เพียง Standard Function เท่านั้น ส่วน Function อื่น ๆ ที่เพิ่มเข้ามาขึ้นมาอยู่กับบริษัทผู้ผลิต [5]

2.2.3 รูปแบบซอฟต์แวร์ (Software Model)

IEC 61131-3 สามารถพิจารณาได้อีกลักษณะโดยพิจารณาจาก Software Model ของ IEC 61131-3 จาก Software Model ระดับบนสุดคือ โครงแบบ (Configuration) ของระบบพีแอลซี และภายใน Configuration อาจมีหนึ่งรีซอร์ส (Resource) หรือมากกว่า ภายในแต่ละ Resource อาจมีหนึ่งทาสก์ หรือมากกว่าแต่ละทาสก์ควบคุมการทำงานของโปรแกรม และแต่ละโปรแกรมอาจเขียนด้วยภาษาใดภาษาหนึ่งในห้าภาษามาตรฐานของ IEC [12] ดังแสดงในรูปที่ 2.10



รูปที่ 2.10 IEC 61131-3 Software Model

จากรูปที่ 2.10 เป็น Software Model ของระบบ พีแอลซี หนึ่งระบบ จะมีการกำหนดค่าต่าง ๆ ในแต่ละระดับชั้น [12] ดังนี้

โครงแบบ (Configuration) ภายในโครงแบบจะ

- กำหนดตัวแปรร่วม (Global Variables) ที่ใช้ในโครงแบบนี้
- รวมรีซอร์สทั้งหมดของระบบเข้าด้วยกัน
- กำหนด Access path ระหว่างโครงแบบ
- กำหนดตำแหน่งทางฮาร์ดแวร์ของพีแอลซี

รีซอร์ส (Resource) ภายรีซอร์สจะ

- กำหนดตัวแปรร่วม (Global Variables) ที่ใช้ในรีซอร์ส
- กำหนด Task และ โปรแกรม ของรีซอร์ส
- กระตุ้นให้ทำงานด้วยค่าที่ได้จากอินพุทหรือเอ้าท์พุท
- กำหนดตำแหน่งทางฮาร์ดแวร์ของพีแอลซี

ทาสก์ (Task)

- กำหนดคุณสมบัติของ Run-time

โปรแกรม

- กำหนดคุณสมบัติของ Run-time ให้กับ โปรแกรม หรือ Function block

การติดต่อสื่อสารระหว่างโครงแบบจะกระทำผ่าน Communication Function

2.2.4 Instruction List (IL)

Instruction list (IL) เป็นภาษามาตรฐาน ตามที่ IEC 61131-3 ระบุไว้ คำสั่งของภาษา ILจะเป็นคำสั่งพื้นฐานไม่ซับซ้อน ข้อดีของภาษา IL คือ ภาษาอื่นสามารถเปลี่ยนมาเป็นภาษา IL ได้ รูปแบบและลักษณะของภาษา IL คล้ายภาษาเครื่อง อาจเรียกว่าเป็นภาษาแอสเซมบลีของพีแอลซี [5]

Operator	Modifiers	Data Types	Description
LD	N	many	set current result to value
ST	N	many	store current result to location
S, R		BOOL	set or reset a value (latches or flip-flops)
AND, &	N, (	BOOL	boolean and
OR	N, (	BOOL	boolean or
XOR	N, (	BOOL	boolean exclusive or
ADD	(	many	mathematical add
SUB	(	many	mathematical subtraction
MUL	(	many	mathematical multiplication
DIV	(	many	mathematical division
GT	(	many	comparison greater than >
GE	(	many	comparison greater than or equal >=
EQ	(	many	comparison equals =
NE	(	many	comparison not equal <>
LE	(	many	comparison less than or equals <=
LT	(	many	comparison less than <
JMP	C, N	LABEL	jump to LABEL
CAL	C, N	NAME	call subroutine NAME
RET	C, N		return from subroutine call
)			get value from stack

Note N - negates an input or output

(- nests an operation and puts it on a stack to be pulled off by ')'

C - forces a check for the currently evaluated results at the top of the stack

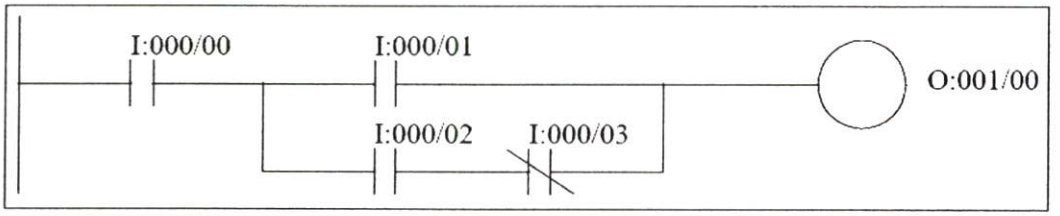
รูปที่ 2.10 IL Operations [5]

รูปที่ 2.10 แสดงรายการคำสั่งของ IL รูปแบบ และลักษณะโปรแกรมแสดงในรูปที่ 2.11

Label	**Opcode**	**Operand**	**Comment**
SequenceOne:	LD	Var1	(* load instruction *)
	ANDN	Var2	(* instruction without bracket *)
	ORN(	Var3	(* instruction with bracket *)
	AND	Var4	
	)		(* end of bracketing *)
	AND	Var5	
	ST	Var6	(* assignment *)
	S	Var7	(* S/R instruction *)

รูปที่ 2.11 ตัวอย่างโปรแกรมภาษา IL [5]

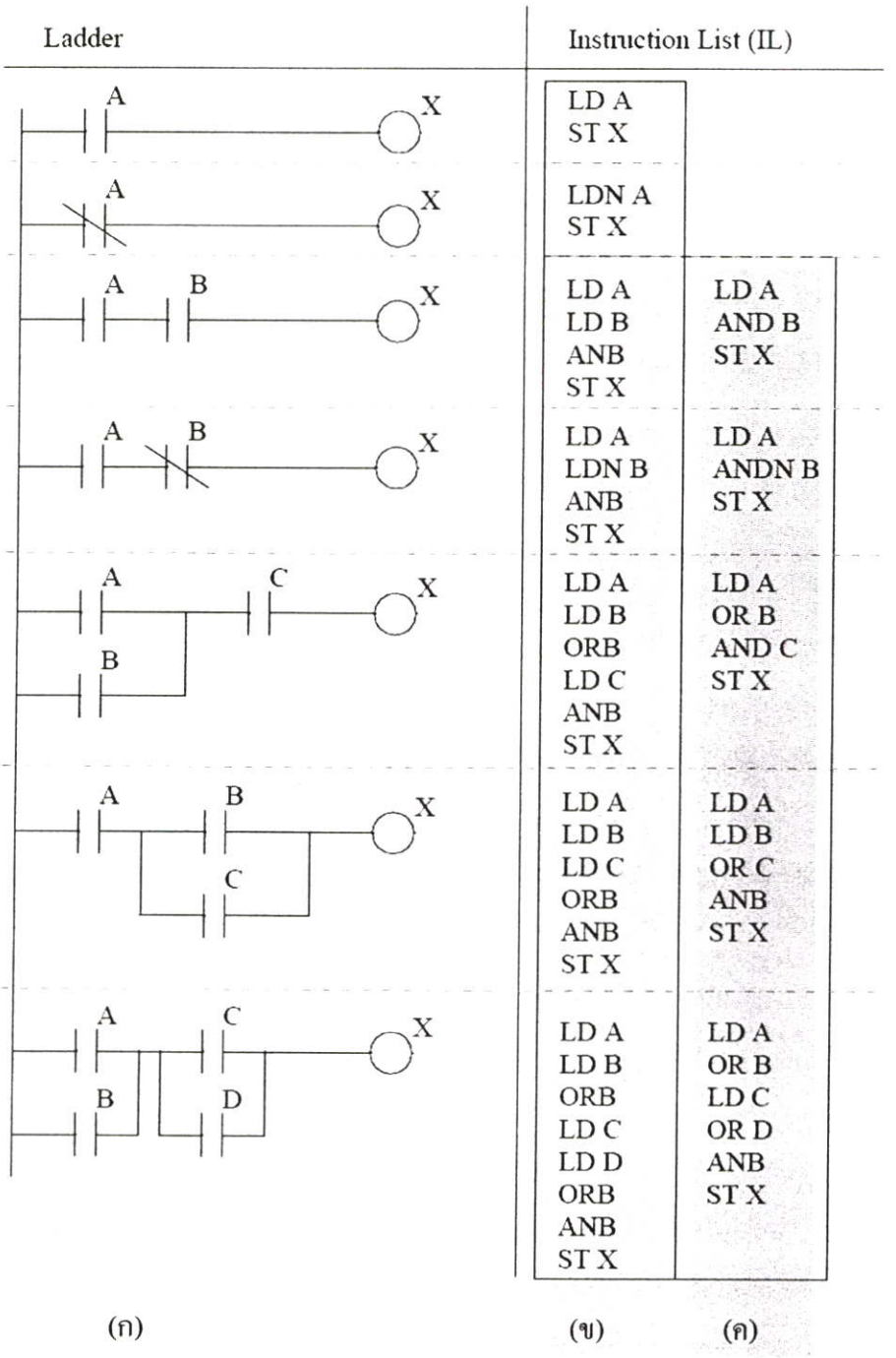
เนื่องจากภาษาอื่น ๆ ตามมาตรฐาน IEC 61131-3 สามารถเปลี่ยนมาเป็นภาษา IL ได้ และโดยปกติผู้ที่ทำหน้าควบคุมหรือเขียนโปรแกรมเพื่อควบคุมการทำงานของพีแอลซี จะเป็นผู้ที่ศึกษามาทางด้านวิศวกรรมฯ หรือช่างเทคนิค และการเรียนการสอนพีแอลซีในสถานศึกษา ส่วนใหญ่จะเริ่มต้นจากภาษา Ladder diagram (LD) ซึ่งเป็นภาษากราฟฟิค เพราะลักษณะจะคล้ายกับวงจรไฟฟ้า (LD : ใช้สัญญาลักษณ์ของหน้าสัมผัสและขดลวด) การเปลี่ยนจาก LD ไปเป็น IL สามารถกระทำได้โดยตรง ดังแสดงในรูปที่ 2.12



Label	Opcode	Operand	Comment
START:	LD	%I:000/00	(* Load input bit 00 *)
	AND(	%I:000/01	(* Start a branch and load input bit 01 *)
	OR(	%I:000/02	(* Load input bit 02 *)
	ANDN	%I:000/03	(* Load input bit 03 and invert *)
	)		
	)		
	ST	%O:001/00	(* SET the output bit 00 *)

รูปที่ 2.12 การเปลี่ยนจาก Ladder Diagram → Instruction List [5]

การเปลี่ยนจาก Ladder Diagram ไปเป็น Instruction List ยังสามารถทำได้อีกลักษณะ คือ แบบ Allen Bradley ดังแสดงในรูปที่ 2.13



รูปที่ 2.13 Ladder Diagram → Instruction List แบบ Allen Bradley [5]

จากรูปที่ 2.13 เป็นการเปลี่ยนจาก Ladder Diagram (LD) ไปเป็น Instruction List (IL) แบบ Allen Bradley ซึ่ง LD อยู่ในรูปภาษาแบบกราฟิค และ IL อยู่ในรูปของ Text แต่เมื่อนำ LD (ก) มาเปลี่ยนเป็น IL จะได้ดังแสดงในรูป (ข) หรือ (ค) โดยแบบ (ค) จะทำงานได้เร็วกว่า [5]

โปรแกรม IL ที่ได้ไม่เป็นไปตามมาตรฐานที่ IEC 61131-3 กำหนดไว้ แต่จะเป็นรหัสนีโมนิค (Mnemonic Code) ของโปรแกรม LD นั้น ๆ สำหรับใช้ในกรณีที่ผู้ใช้ต้องป้อนโปรแกรมเข้า

พีแอลซี ผ่านแผงป้อนโปรแกรม (Keypad) ด้วยตัวเอง โดย พีแอลซี ขนาดเล็กใช้จะวิธีนี้ในการนำโปรแกรมเข้าเครื่อง พีแอลซี

2.3 ระบบปฏิบัติการเรียวไทม์ (RTOS : Real Time Operating System)

2.3.1 ระบบเรียวไทม์ (Real Time System)

เวลาเป็นตัวแปรสำคัญที่นำมาใช้เป็นเงื่อนไขในงานควบคุมอุปกรณ์ต่าง ๆ เช่น การควบคุมวาล์วโซเลนอยด์ (Solenoid valve) ให้เปิดหรือปิดต

ามเวลาที่ต้องการ หรือ การควบคุมไฟจราจร เป็นต้น ในกรณีพีแอลซีทั่วไปที่ควบคุมการทำงานเพียงหนึ่งโปรเซส (Single process) จะไม่มีปัญหาในเรื่องเวลา แต่ในกรณีซอฟต์พีแอลซี ที่มีพีแอลซีทาสก์ หลาย ๆ ทาสก์ ทำงานพร้อม ๆ กัน ระบบปฏิบัติการที่นำมาใช้จะต้องสามารถทำการมัลติทาสก์ได้ และเป็นเรียวไทม์

ความหมายของระบบเรียวไทม์ คือ "ระบบที่ความถูกต้องของระบบทั้งหมดจะขึ้นอยู่กับความถูกต้องของทั้งการทำงานและเวลา ความถูกต้องของเวลามีความสำคัญไม่น้อยกว่าความถูกต้องของการทำงาน"

"*Real-time systems are defined as those systems in which the overall correctness of the system depends on both the functional correctness and the timing correctness. The timing correctness is at least as important as the functional correctness.*" [1]

ระบบปฏิบัติการเรียวไทม์ เป็น ระบบปฏิบัติการที่ใช้เวลาเป็นหลักในการ กำหนดการทำงาน (Schedule) การจัดสรรทรัพยากรของระบบ การจัดเตรียมทรัพยากรตามความต้องการของโปรแกรมประยุกต์ ระบบปฏิบัติการเรียวไทม์ที่ดี ต้องสามารถปรับเปลี่ยน (Scalable) ไปตามความต้องการที่แตกต่างกันของโปรแกรมประยุกต์ได้อย่างพอเพียง

ระบบปฏิบัติการเรียวไทม์ โดยทั่วไปมี 2 แบบ คือ SOFT Real Time และ HARD Real Time

- แบบ Soft real-time ในแบบนี้ ทาสก์ถูกให้ทำงานโดยระบบเร็วที่สุดเท่าที่จะเป็นไปได้ แต่การทำงานนั้นอาจจะไม่เสร็จสิ้นตามเวลาที่กำหนด
- แบบ Hard real-time ในแบบนี้การทำงานของทาสก์ ไม่เพียงแต่จะทำงานถูกต้องอย่างเดียว ยังต้องเป็นไปตามเวลาที่กำหนดด้วย

โดยทั่วไประบบปฏิบัติการเรียวไทม์จะทำงานได้ทั้งสองแบบ ทำให้สามารถนำไปใช้งานได้กว้าง แต่การออกแบบให้ทำงานแบบ Hard real-time จะยุ่งยากกว่าแบบ Soft real-time [9]

2.3.2 ทาสก์ (Task)

หน้าที่ของระบบปฏิบัติการ คือ การจัดสรรทรัพยากรของระบบ ระบบปฏิบัติการจะต้องจัดเตรียมทรัพยากรให้กับผู้ที่ต้องการใช้ทรัพยากรเหล่านั้น ผู้ที่นำทรัพยากรไปใช้ก็ คือ ทาสก์

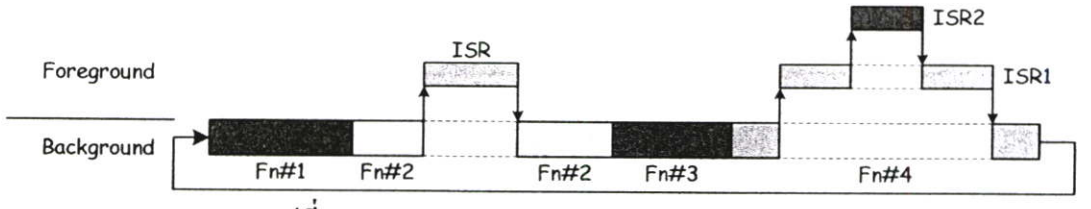
ความหมายของคำว่า "ทาสก์" ยังไม่มีมาตรฐานที่เป็นที่ยอมรับ บางครั้งอาจใช้ความหมายเดียวกันกับ โปรเซส (Process) หรือ เทรด (Thread) ในวิทยานิพนธ์ฉบับใช้ความหมายเดียวกับโปรเซส นั่นคือ ทาสก์หมายถึง "โปรแกรมที่กำลังดำเนินการ (program in execution)"

ทาสก์มีส่วนประกอบต่าง ๆ ดังนี้

1) ชื่อและหมายเลขประจำทาสก์ (Task ID) ต้องไม่ซ้ำกับทาสก์อื่น
2) หมายเลขบอกลำดับความสำคัญ (Priority) ของทาสก์ ต้องไม่ซ้ำกับทาสก์อื่น
3) โค้ดของโปรแกรม (program code) เป็นคำสั่งที่สามารถทำงานได้ทันที
4) ข้อมูลที่โปรแกรมต้องการหรือจัดการ ข้อมูลนี้อาจเป็นของทาสก์ใดทาสก์หนึ่ง หรืออาจใช้ได้ร่วมกับทาสก์อื่น ๆ ก็ได้
5) บล็อกควบคุมทาสก์ (TCB : Task control block) ระบบปฏิบัติการจะกำหนดเนื้อที่บางส่วนในหน่วยความจำเพื่อทำเป็นบล็อกควบคุมทาสก์ เป็นโครงสร้างข้อมูลชนิดหนึ่งไว้เก็บข้อมูลที่สำคัญ ๆ ของทาสก์นั้น ๆ เอาไว้ ข้อมูลเหล่านี้ได้แก่
   - สถานะของทาสก์ (Task State) ที่เป็นอยู่ในปัจจุบัน
   - หมายเลขประจำทาสก์
   - ลำดับความสำคัญของทาสก์
   - พอยน์เตอร์ชี้ไปยังตำแหน่งที่อยู่ของทาสก์ในหน่วยความจำ
   - พอยน์เตอร์ชี้ไปยังทรัพยากรต่าง ๆ ที่ทาสก์ครอบครอง
   - พื้นที่ที่เก็บค่าของรีจีสเตอร์ (register save area)

2.3.3 ฟอร์กราวน์ด / แบคกราวน์ด (Foreground / Background)

ในระบบที่ไม่มีความซับซ้อนมาก จะถูกออกแบบให้ทำงานในลักษณะ ฟอร์กราวน์ด / แบคกราวน์ด หรือ ซุปเปอร์ลูป ประกอบด้วย โปรแกรมหลักทำงานในลักษณะวนรอบไม่รู้จบ (Infinite loop) เรียกว่า โมดูล (Function) อยู่ในส่วนแบคกราวน์ด เป็นระดับทาสก์ (Task level) ส่วนโปรแกรมตอบสนองการอินเตอร์รัพ (ISR : Interrupt Service Routine) เป็น ฟอร์กราวน์ด ในระดับอินเตอร์รัพ (Interrupt level) [9] ดังแสดงในรูปที่ 2.14



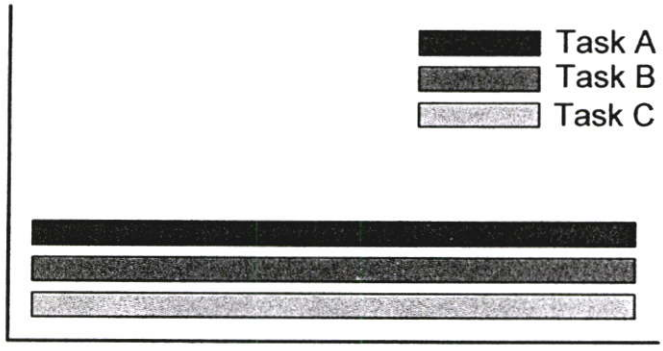
รูปที่ 2.14 Foreground/Background Systems [14]

โปรแกรมหลัก (แบคกราวน์ด) จะวนรอบทำงานตลอดเวลาเมื่อมีการอินเตอร์รัพเข้ามา ISR (ฟอร์กราวน์ด) จะเข้ามาทำงานแทนจนกระทั่งเสร็จสิ้นการทำงาน จึงจะเปลี่ยนให้กลับไป

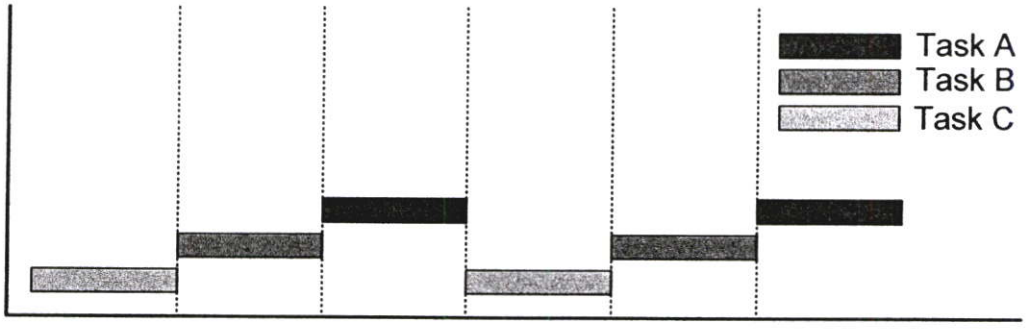
ทำงานที่โปรแกรมหลักอีกครั้ง การทำงานในลักษณะนี้จะพบเห็นโดยทั่วไปในอุปกรณ์ที่ใช้ไมโครคอนโทรลเลอร์ เป็นตัวควบคุม [9]

2.3.4 การมัลติทาสก์ (Multitasking)

การมัลติทาสก์ หมายถึง ความสามารถของระบบที่สามารถทำงานได้มากกว่าหนึ่งทาสก์ในเวลาเดียวกัน การมัลติทาสก์ เพียงแค่ทำให้รู้สึกว่าทาสก์เหล่านั้นทำงานพร้อม ๆ กัน ดังแสดงในรูปที่ 2.15 แต่ในความเป็นจริง แต่ละทาสก์จะสอดแทรกเปลี่ยนกันทำงาน [11] ดังแสดงในรูปที่ 2.16



รูปที่ 2.15 เสมือนทำงานไปพร้อม ๆ กัน [11]



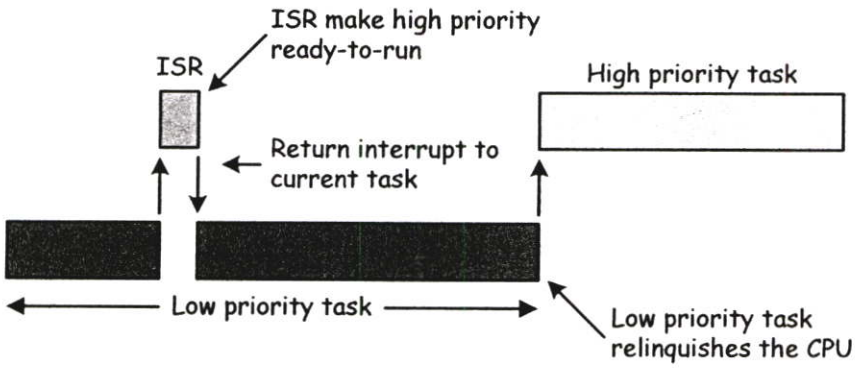
รูปที่ 2.16 ทาสก์สอดแทรกเปลี่ยนกันทำงาน [11]

หน้าที่ของระบบปฏิบัติการแบบมัลติทาสก์กิ้ง คือ สร้างทาสก์ ควบคุมการเริ่มต้นและสิ้นสุดการทำงานของทาสก์ กำหนดการทำงานของแต่ละทาสก์ โดยใช้ระดับ Priority เป็นฐาน ดูแลและจัดการใช้ทรัพยากรร่วมกันระหว่างทาสก์ ควบคุมจังหวะการทำงานของแต่ละทาสก์ให้สอดประสานกัน (Synchronize) และ ป้องกันการ Corrupt ของทาสก์

2.3.4.1 Non preemptive multitasking หรือ Cooperative multitasking

การมัลติทาสในลักษณะนี้ แต่ละทาสก์สามารถครอบครองซีพียูได้นานเท่าใดก็ได้ ตามที่ทาสก์นั้นต้องการ เมื่องานแล้วเสร็จจึงจะคืนการครอบครองซีพียูให้แก่ระบบ เพื่อให้ทาสก์อื่นได้เข้ามาครอบครองซีพียูเพื่อทำงานต่อไป ถ้ามีการอินเตอร์รัพเข้ามาและ ISR จะจัดให้ทาสน์นั้นมี Priority สูง และอยู่ในสถานะพร้อมที่จะทำงาน แต่ ISR จะคืนการครอบครองให้กับทาสก์ปัจจุบัน จนกว่าทาสก์ปัจจุบันจะทำงานแล้วเสร็จ ทาสก์ที่มี Priority สูงสุดถึงจะเข้ามาทำงานได้

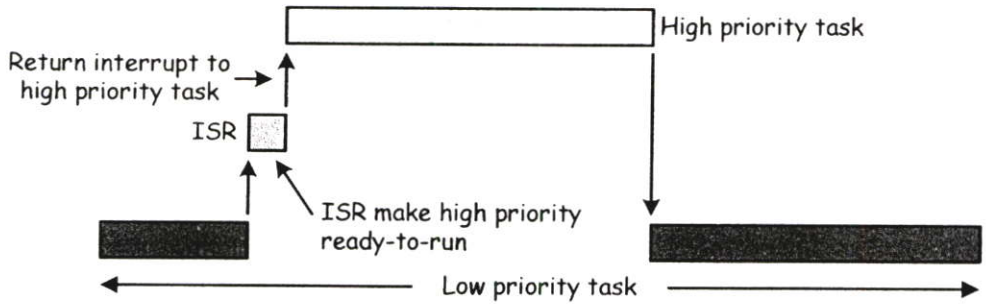
การทำงานลักษณะนี้จะมีปัญหาค่อนข้างมาก เนื่องจากถ้าทาสก์บางทาสก์ไม่คืนการครอบครองซีพียู เป็นผลให้ทาสก์อื่น ๆ ไม่สามารถเข้ามาทำงานได้



รูปที่ 2.17 Cooperative multitasking [14]

2.3.4.2 Preemptive multitasking

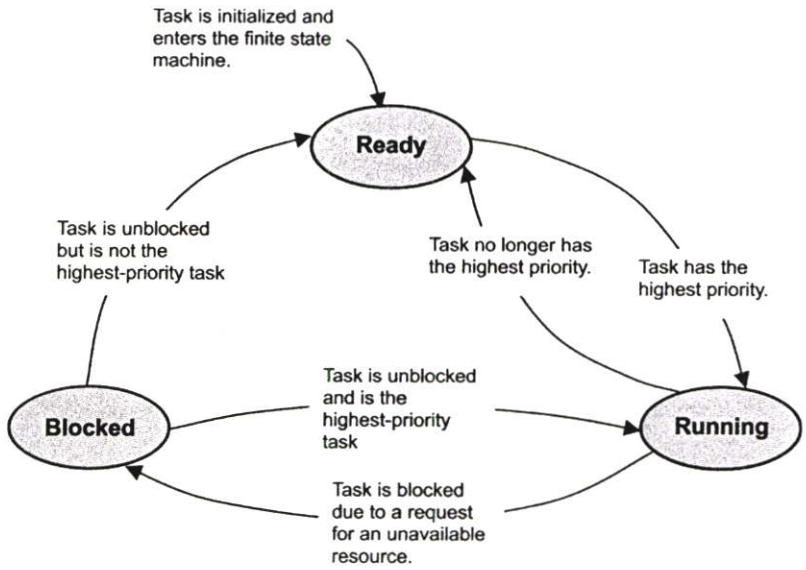
การมัลติทาสก์ในลักษณะนี้ ทาสก์ที่มีระดับ Priority สูงสุดจะได้รับสิทธิในการครอบครองซีพียูก่อนเสมอ เมื่อ ISR ทำให้ทาสก์ที่มีระดับ Priority สูงกว่าอยู่ในสถานะพร้อมจะทำงานแล้ว ก็จะแขวน (Suspend) ทาสก์ปัจจุบัน แล้วมอบการครอบครองซีพียูให้กับทาสก์ที่มีระดับ Priority สูงกว่าเข้าครอบครองซีพียูแทน



รูปที่ 2.18 Preemptive multitasking [14]

2.3.5 สถานะของทาสก์ (Task States)

ระบบปฏิบัติการที่มีการมัลติทาสก์แบบ Preemptive สามารถควบคุมและติดตามการทำงานของทาสก์ต่าง ๆ ได้ โดยใช้การเก็บสถานะของทาสก์ ทั้ง System Task และ Application Task ไว้ ณ. เวลาใดก็ตาม แต่ละทาสก์จะต้องอยู่ในสถานะใดสถานะหนึ่ง นั่นคือ สถานะ Ready, Running หรือ Blocked การเคลื่อนที่จากสถานะหนึ่งไปสู่อีกสถานะหนึ่งเป็นไปตามเงื่อนไขที่กำหนดไว้ใน Finite State Machine (FSM) [3] ดังแสดงในรูป 2.19



รูปที่ 2.19 สถานการณ์ทำงานของทาสก์ [3]

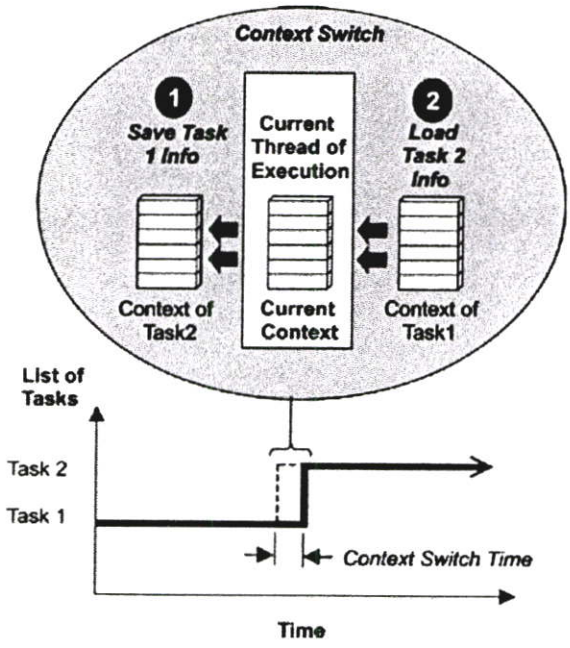
สถานะของทาสก์ที่ระบบปฏิบัติการเรียวไทม์ใช้กำหนดโดยทั่วไปมีอยู่ 3 สถานะ [3] แต่บางระบบปฏิบัติการ อาจมีจำนวนสถานะของทาสก์ แตกต่างไปจากนี้

- Ready state คือ สถานะที่ทาสก์พร้อมทำงาน แต่ยังไม่สามารถทำงานได้ เพราะมีทาสก์ที่ Priority สูงกว่ากำลังทำงานอยู่
- Blocked state คือ สถานะที่ ทาสก์ ถูกกันออก (Blocked) เพราะเหตุการณ์ (Even) ที่ต้องการ ยังไม่ปรากฏ หรือ หยุดตัวเอง (Suspend หรือ Delay)
- Running state คือ สถานะที่ ทาสก์ มี Priority สูงสุด และกำลังทำงาน

2.3.6 การเปลี่ยนบริบท (Context Switch)

ทาสก์ ทุก ทาสก์ ต่างก็มีบริบท (Context) ของตัวเอง ซึ่งก็คือสถานะของ CPU registers ที่ต้องการใช้ทุกครั้งที่ทาสก์นั้นถูกเปลี่ยน (Switch) ให้เข้าทำงาน การเปลี่ยนบริบทจะเกิดเมื่อระบบต้องการสลับเปลี่ยนการทำงานจากทาสก์หนึ่งไปสู่อีกทาสก์หนึ่ง

ทุกครั้งที่มีการสร้างทาสก์ขึ้นใหม่ ระบบจะสร้างและเก็บข้อมูลความสัมพันธ์เหล่านี้ไว้ในบล็อกควบคุมทาสก์ (TCB : Task control block) ของทาสก์นั้น ๆ โดยจะบันทึกเฉพาะข้อมูลที่จำเป็นใช้ในการทำงานของแต่ละทาสก์ เมื่อทาสก์นั้นทำงาน (Running) บริบทของมันจะมีการเปลี่ยนแปลงซึ่งการเปลี่ยนแปลงของบริบทจะถูกปรับปรุงและเก็บรักษาในบล็อกควบคุมทาสก์เพื่อนำกลับใช้อีก เมื่อทาสก์นั้นถูกเปลี่ยนให้กลับเข้ามาทำงานอีกครั้ง [3]



รูปที่ 2.20 Context Switch [3]

การทำการเปลี่ยนบริบท ดังแสดงในรูปที่ 2.20 เมื่อ Scheduler ตัดสินว่าต้องการหยุดการทำงานของ ทาสก์ 1 เพื่อสลับให้ ทาสก์ 2 เข้ามาทำงานแทน จะมีลำดับขั้นดังนี้ [3]

1) บันทึกข้อมูลบริบทของทาสก์ 1 ไว้ในบล็อกควบคุมทาสก์ของทาสก์ 1
2) โหลดข้อมูลบริบทของของทาสก์ 2 จากบล็อกควบคุมทาสก์ของทาสก์ 2 ซึ่งเป็นรันนิ่งทาสก์
3) บริบทของ ทาสก์ 1 ถูกเก็บไว้ ในขณะที่ทาสก์ 2 ทำงาน แต่ถ้า Scheduler ต้องการให้ทาสก์ 1 ทำงานอีกครั้ง ทาสก์ 1 ก็กลับมาทำงานต่อจากจุดเดิม ก่อนที่จะโดนเปลี่ยนบริบทออกไป

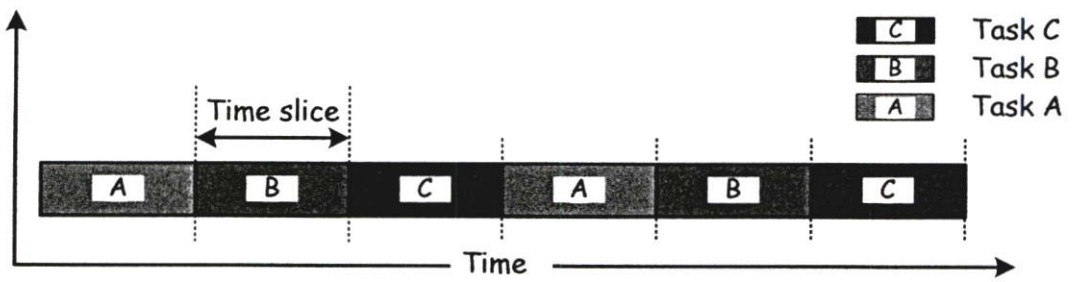
เวลาที่ใช้ในการทำการเปลี่ยนบริบทจะน้อยมากเมื่อเปรียบเทียบกับการทำงานอื่น ๆ

2.3.7 การกำหนดเวลา (Scheduling)

การทำงานแบบมัลติทาสก์ จะมีการเปลี่ยนให้ทาสก์สอดแทรกกันทำงานตลอดเวลา กระบวนการในการตัดสินใจว่า ควรจะเป็นทาสก์ใด และเมื่อไร ขึ้นอยู่กับการกำหนดเวลา

(Scheduling) ของระบบ ขั้นตอนวิธีในการตัดสินใจเรียกว่า *Scheduling Algorithms* หรือ *Scheduling Policy* และโปรแกรมที่ทำหน้าที่นี้เรียกว่า *Scheduler*

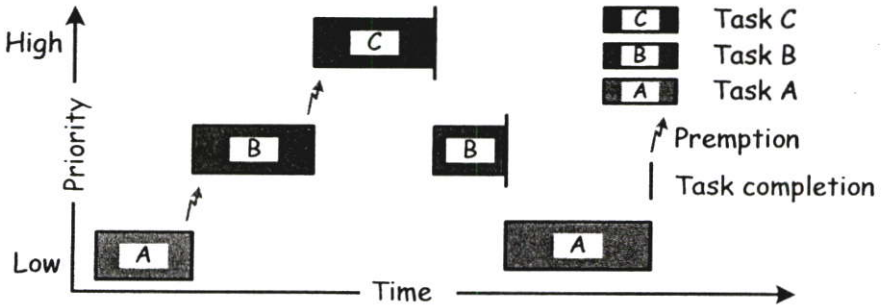
2.3.7.1 Round-Robin Scheduling การกำหนดเวลาแบบนี้ ใช้วิธีการแบ่งเวลา (Time slice) ให้แต่ละทาสก์ ได้ช่วงเวลาในการครอบครองซีพียูนานเท่า ๆ กัน โดยหมุนเวียนสลับกันเข้ามาครอบครองซีพียู ดังแสดงในรูปที่ 2.21



รูปที่ 2.21 Round-Robin Scheduling [3]

ข้อเสียของ Round-Robin Scheduling คือ ทาสก์ที่มี Piority สูง ต้องการใช้ซีพียู แต่ไม่สามารถสลับเข้ามาได้ต้องรอจนกว่าจะถึงคิว แต่ในระบบเรียวไทม์ ทาสก์ที่มี Piority สูงต้องได้สิทธิในการครอบครองซีพียูก่อนเสมอ ดังนั้นการกำหนดเวลาแบบนี้จึงไม่สามารถใช้ได้กับระบบเรียวไทม์ [3]

2.3.7.2 Priority-based Scheduling การกำหนดเวลาแบบนี้ ใช้วิธีการจัดระดับ (Piority) ให้แต่ละทาสก์ โดยใช้ตัวเลขเป็นตัวกำหนดระดับ ถ้ามี 64 ระดับ ระดับสูงสุดคือ 0 และ 63 คือระดับต่ำสุด ทาสก์ที่มีระดับสูงสุด และอยู่ในกลุ่มของทาสก์ที่มีสถานะ Ready State จะได้สิทธิในการครอบครองซีพียูก่อน [3] ดังแสดงในรูปที่ 2.22



รูปที่ 2.22 Priority-based Scheduling [3]

ระบบปฏิบัติการเรียวไทม์ ที่นิยมใช้ในปัจจุบัน แบ่งได้เป็น 2 กลุ่ม คือ

- กลุ่มเชิงพาณิชย์ (Commercial Real-time Operating Systems)
- กลุ่มโอเพนซอร์ส (Open Source Real-time Operating Systems)

2.3.8 ระบบปฏิบัติการเรียวไทม์ในกลุ่มเชิงพาณิชย์

ระบบปฏิบัติการเรียวไทม์ในกลุ่มเชิงพาณิชย์ นี้ แตกต่างจากกลุ่มทั่วไปที่มีความสามารถในการคาดหมาย (Predictability) ได้สูงกว่า ถูกนำไปใช้ใน ระบบสมองกลฝังตัว หรือ ระบบควบคุมในโรงงานอุตสาหกรรม เช่น LynxOS, VxWorks, pSoS, QNX เป็นต้น สามารถนำไปใช้กับระบบ Uniprocessor, Multiprocessor หรือ Distributed Real-Time OS [15]

Lynx OS

ออกแบบให้มี Kernel ขนาดเล็ก (Microkernel) มีขนาดเพียง 28 KB Kernel จะให้บริการเฉพาะที่จำเป็นคือ การจัดเวลาการทำงาน (Scheduling) การส่งผ่านการอินเตอร์รัพ (Interrupt Dispatching) และการควบคุมจังหวะการทำงาน (Synchronization) ส่วนการให้บริการอื่น ๆ ให้บริการโดย Kernel Plug-Ins (KPIs) แทน โดยสามารถเพิ่ม KPI ใหม่ให้ทำงานร่วมกับ Microkernel ได้ [15]

Lynx OS ใช้การปกป้องหน่วยความจำทาง Hardware โดย MMUs โปรแกรมประยุกต์สามารถเรียกใช้ IO ของระบบผ่านทาง System call และ Kernel จะเรียกใช้ IO โดยตรงที่ Device driver แต่ละ Device driver มี Interrupt Handler และ Kernel Thread ของตัวเอง และ Interrupt Handler จะเป็นขั้นแรกที่ตอบสนองต่อการ Interrupt ถ้าไม่สามารถทำงานให้แล้วเสร็จตามกระบวนการได้ ก็จะสับไปยัง Kernel หลังจากนั้น Kernel จะจัดเวลาให้ชั่วขณะจนกว่าการตอบสนองการ Interrupt นั้นจะเสร็จสมบูรณ์ [15]

QNX/ Neutrino

ออกแบบตามมาตรฐาน POSIX (Unix operating system) มี Kernel ขนาดเล็ก (Microkernel) ให้บริการเฉพาะงานที่จำเป็น และงานบริการด้าน Real-Time ที่ Microkernel ยอมให้ผู้ใช้ปิดฟังชันที่ไม่ได้ใช้งานได้ โดยไม่มีผลต่อระบบ โปรแกรมมีขนาดเล็ก สามารถบรรจุลงใน Floppy Disc 1 แผ่น Footprint ของ Microkernel มีขนาดเพียงแค่ 12 KB โดยที่ Driver ทุกตัว โปรแกรมประยุกต์ Protocol Stack และ File system ทำงานนอก Kernel ในส่วนเซฟทีของ Memory-protected user space ผลก็คือ Virtually ของ Component ใดๆ สามารถที่จะ ล้มเหลว และ เริ่มใหม่ โดยอัตโนมัติ โดยไม่มีผลกระทบกับ Component อื่น หรือ Kernel [15]

VxWorks

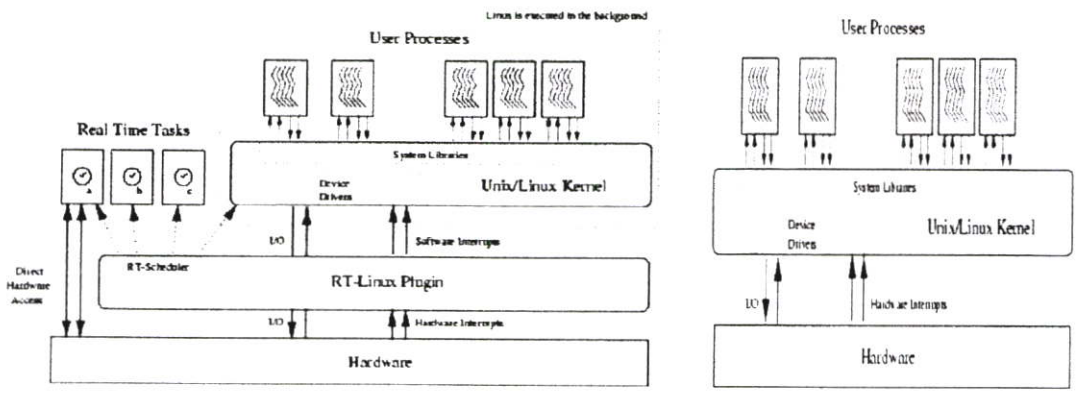
เป็นไปตามมาตรฐาน POSIX 1003.1, .1b, .1c การจัดตารางเวลาเป็นแบบ Preemptive Priority with Round Rbin Scheduling รองรับได้ทั้ง Real-Time process และ Non-real time process ใช้ MMU เป็นฐานในการปกป้องหน่วยความจำ ลดเวลาในการทำ Context switch โดยเก็บเฉพาะ Register ที่ใช้จริงเท่านั้น และเมื่อมีการสลับทาสก์กลับมาทำงานใหม่จะเรียกคืน (Restore) เฉพาะ Register ที่เกี่ยวข้อง เป็น RTOS ที่มีสมรรถนะภาพสูง เชื่อถือได้ คาดหมายได้ Latency ต่ำ และ ปรับเปลี่ยนขนาดได้ (Scalability) [15]

eCOS (Embedded Configurable OS)

Kernel footprint ขนาดเล็กทำงานหลายหน้าที่ กำหนดตารางการทำงาน จัดสรรหน่วยความจำ และควบคุมการอินเตอร์รัพ นำไปพอร์ทใช้กับไมโครโปรเซสเซอร์ได้หลายตระกูล ทำงานได้ทั้งบนวินโดว์และดอส เป็นโอเพนซอร์สและไม่มีค่าลิขสิทธิ์ [15]

RTLinux

สามารถแพท (Patch) ลงบน Linux Kernel เป็น Hybrid OS โดยจะรัน Idle thread ที่ linux kernel การกำหนดตารางการทำงานได้หลายแบบ FIFO (ใช้การส่งผ่านข่าวสารระหว่าง โปรเซสเรียวไทม์ และ โปรเซสลินุกต์ปกติ) หรือแบบ Rate-monotonic scheduler การเข้าครอบครอง Kernal จะไม่ใช้เวลามากเพราะโปรแกรมมีขนาดเล็กและทำงานเร็ว ในขณะที่การอินเตอร์รัพจากลินุกถูกดิสเอเบิลไว้ [15]



(ก) RTLinux patch to the Linux kernel (ข) Regular Linux kernel

รูปที่ 2.23 RTLinux [15]

RTAI (Real Time Application Interface)

เป็นส่วนขยาย Hard real-time ให้กับ Linux kernel การแพทลงบน Linux Kernel โดย HAL สามารถทำงานร่วมกับโปรแกรม Linux ปกติได้โดยไม่ต้องเปลี่ยนแปลงใด ๆ การทำงาน RTAI จะมอง Linux ว่าเป็นเสมือน Background task เมื่อไม่มีการทำงานแบบเรียวไทม์ปรากฏ มีโมดูลในการกำหนดตารางการทำงานให้ใช้หลายโมดูล เช่น Task functions, Timing functions, Semaphore functions, Mailbox functions หรือ Intertask communication functions เป็นต้น ใน RTAI ซอฟท์เรียวไทม์ในยูสเซอร์สเปซจะขึ้นกับฮาร์ดเรียวไทม์ในเคอแนลสเปส [15]

MicroC/OSII (µC/OS II)

µC/OS II ถูกออกแบบให้มี Footprint ขนาดเล็ก จนสามารถนำไปใช้กับซีพียู ขนาด 8 bit ได้ เป็น Preemptive real-time และ Multitasking kernel แต่ละทาสก์จะมี Piority และ Stack ของตัวเอง ไม่รองรับการกำหนดตารางการทำงานแบบ Round robin การทำมัลติทาสก์จะออกแบบให้ทุกทาสก์ทำงานแบบวนรอบไม่รู้จบเป็น Infinite loop โดยทุกทาสก์จะอยู่ในสถานะใดสถานะหนึ่งในห้าสถานะ คือ Dormant, Ready, Running, Waiting หรือ ISR สามารถพอร์ทลงบนซีพียูได้หลายตระกูล [15]

รูปที่ 2.24 แสดงผลการเปรียบเทียบระบบปฏิบัติการเรียวไทม์ในกลุ่มเชิงพาณิชย์ [15]
ส่วนรูปที่ 2.25 แสดงผลที่ได้จากการประเมินผล เรียงตามลำดับการเลือกใช้งาน ของระบบปฏิบัติการเรียวไทม์ ในกลุ่มเชิงพาณิชย์ [11]

RTOS, Vendor	*Thread priority levels*	*Synchronization mechanisms*	*Priority inversion prevention provided*	*Development hosts, kernel characteristics*
AMX, *KADAK Products Ltd.*	N/A	Mailboxes: wait-wake requests	Yes	Windows. predictable memory block availability
C Executive, *JMI Software Systems, Inc.*	32	Messages. dynamic data queues	Yes	Windows. Solaris
CORTEX, *Australian Real-time Embedded Systems.*	62	Recursive locks. mutexes	Yes. uses priority ceiling	Windows/Unix. CPU-independent software interrupt manager: statically and dynamically segmented memory models
Delta OS, *CoreTek Systems, Inc.*	256	Semaphores. timers. message queues	Yes	Windows. Linux.
Ecos *RedHat, Inc.*	1-32	Semaphores, timers and counters	Yes, uses priority ceiling	Windows, Linux For soft real-time embedded applications in small devices
Emboss *SEGGER Microcontroller Systems.*	255	Mailbox. binary and counting semaphore	No	Windows. Linux. profiling to collect timing information for every task: task activation time independent of number of tasks.
ERTOS *JK Microsystems, Inc.*	256	Inter-thread messaging. queues. semaphores	No	Windows. DOS. OS/2. High-speed interrupt driven serial port routines
INTEGRITY *GreenHills Software, Inc.*	255	Semaphores. breakpoints can be placed any where in the system including ISRs.	Yes. mutex. semaphore	Used in critical embedded applications: object-oriented: supports distributed processing: task execution profiling.
IRIX 1.1.1.1.1 SGI	255	Message queues	Yes	SGI. Double-precision matrix support: Multi-pipe scalability
Nuclear Plus *Accelerated Technology, Inc.*	N/A	Mailboxes. pipes and queues	Yes	Windows.
OS-9 *Microware Systems Corporation.*	65535	Semaphore and queues	Yes	Windows.
OSE *OSE Systems.*	32	Message passing	Yes	Windows. Solaris. Linux. User-defined system clock resolution: fault-tolerant: suited for wireless applications.
RT-Linux *Finite State Machine Labs.*	1024	Shared memory or via files	Yes. lock free data structures and priority ceiling	Linux: supports hard real-time applications
ThreadX *Express Logic, Inc.*	32	Mutexes. counting semaphores and messaging	Yes. by disabling preemption over ranges of priorities and by priority inheritance	Windows.
QNX Neutrino *QNX Software Systems Ltd.*	64	Message passing	Yes. using priority inheritance	Windows. Solaris. Linux. Symmetrical multiprocessor systems. Every OS component runs in its own MMU-protected address space

Table 2. Features of commercial *RTOSs*

รูปที่ 2.24 แสดงการเปรียบเทียบ Commercial RTOSs [15]

QNX/Neutrino™	1st
OS-9™	
Precise/MQX™	2nd
OSE™	
Delta OS™	3rd
RTEMS™	
Lynx OS™	
Integrity™	
VxWorks™	
Nucleus Plus™	4th
VRTX™	
TTPos™	
C Executive™	
Emb OS™	
CMX™	
ECOS™	5th
uC/OS-II™	
SuperTask™	
AMX™	
Cortex™	6th

Evaluation results

รูปที่ 2.25 ผลการประเมินลำดับของ RTOSs [11]

2.3.9 ไมโครคอนโทรลเลอร์ โอเพอเรติงซิสเต็ม เวอร์ชันทู (µC/OS II)

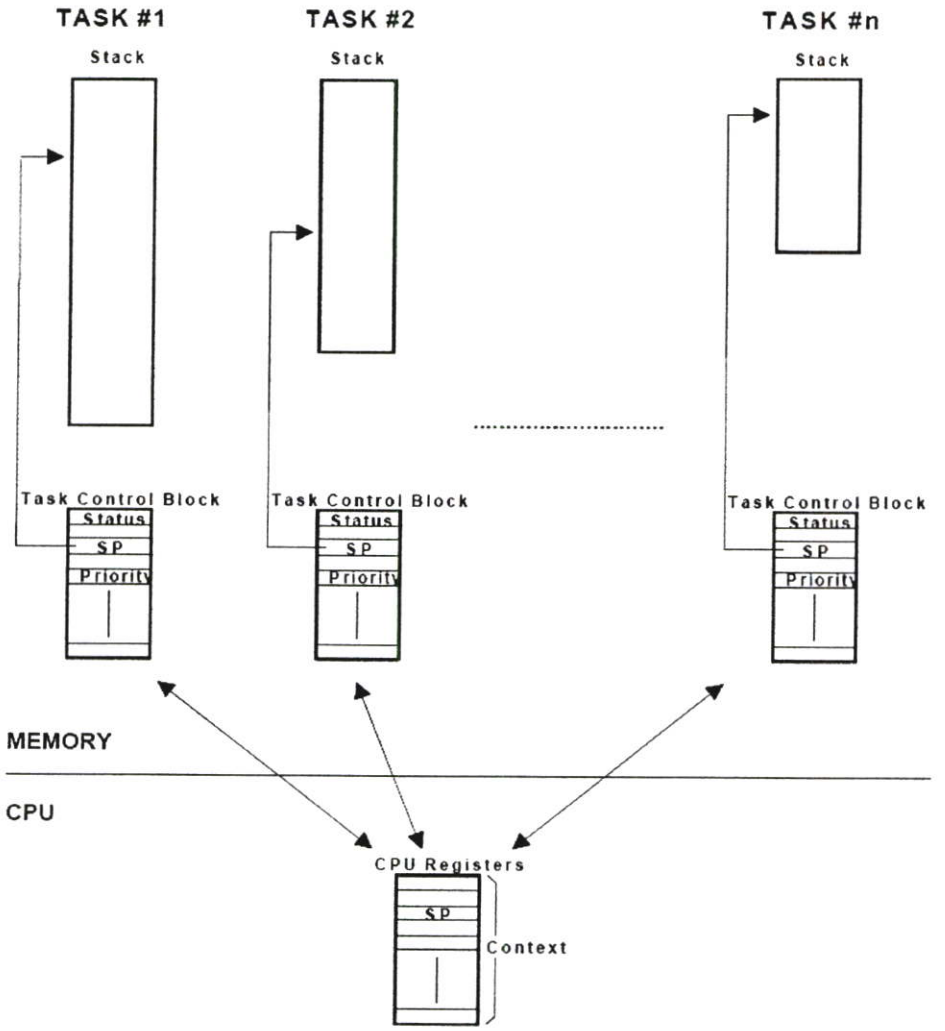
ระบบปฏิบัติการเรียวไทม์ ที่งานวิจัยนี้เลือกใช้ คือ µC/OS II ซึ่งเป็นระบบปฏิบัติการในกลุ่มเชิงพาณิชย์ [10] ดังแสดงในรูปที่ 2.25 ข้อดีของ µC/OS II คือ ถูกออกแบบให้มี Footprint ขนาดเล็ก สามารถนำไปใช้กับไมโครโปรเซสเซอร์ ขนาด 8 bit , 16 bit, 32 bit, 64 bit หรือไมโครคอนโทรลเลอร์ ได้ สามารถปรับเปลี่ยนขนาดได้ตามความต้องการ โดยผู้ใช้สามารถเลือกใช้บริการของโอเอสเฉพาะที่ต้องการใช้งานเท่านั้น นอกจากนี้การนำมาใช้ในงานการศึกษาไม่ต้องเสียค่าลิขสิทธใด ๆ [8][9]

µC/OS II เป็น Preemptive real-time kernel โดยโอเอสจะเปลี่ยนให้ทาสก์ที่มี Piority สูงสุด และอยู่ในสถานะ Ready ทำงาน (Running) ก่อนเสมอ ผู้ใช้เป็นผู้กำหนด Piority ให้แต่ละทาสก์ ตามลำดับความสำคัญ ทาสก์ทุกทาสก์จะมีหน่วยความจำของตัวเอง ซึ่งโอเอสจะกำหนดไว้ในตอนสร้างทาสก์ขึ้นมาใหม่ ขนาดของ Stack ของแต่ละทาสก์สามารถกำหนดให้มีขนาดแตกต่างกันได้ตามความต้องการของผู้ใช้ [9]

รูปที่ 2.26 แสดงความสัมพันธ์ระหว่าง ทาสก์สแตค บล็อกควบคุมทาสก์ (TCB) และ ซีพียู รีจิสเตอร์ การเปลี่ยนจากทาสก์หนึ่งไปสู่อีกทาสก์หนึ่งขึ้นอยู่กับเงื่อนไขที่กำหนด ในระบบเรียวไทม์เงื่อนไขก็คือเวลา โดยมี เคอเนล (Kernel) ควบคุมการเปลี่ยนแปลงนี้

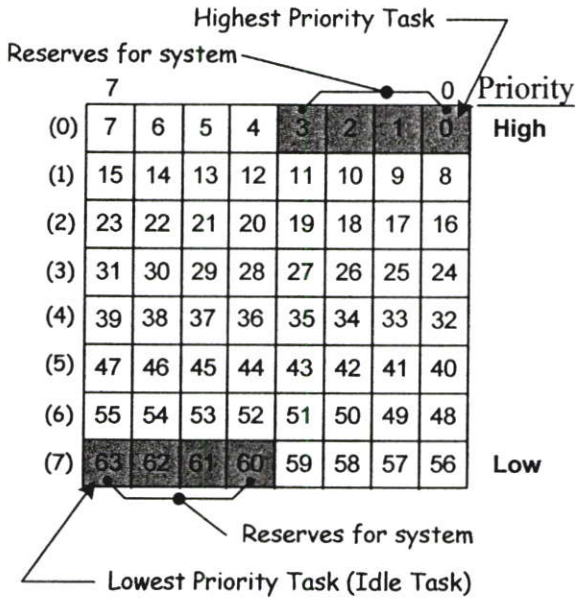
คำว่า เคอเนล ในระบบเรียวไทม์ หมายถึงซอฟต์แวร์ที่ทำหน้าที่ดูแลจัดการเวลาของ ซีพียู

การเปลี่ยนบริบท (Context Switch) คือ การเปลี่ยนจากทาสก์หนึ่งไปสู่อีกทาสก์หนึ่ง เริ่มจากเก็บบริบท (Context) ของทาสก์ปัจจุบัน เก็บตัวชี้สแตค (Stack Pointer) ของทาสก์ปัจจุบันไว้ในบล็อกควบคุมทาสก์ (TCB) ของทาสก์นั้น โหลดตัวชี้สแตค (SP) ของทาสก์ใหม่ที่จะเข้ามาแทนที่ และตามด้วยโหลดบริบทของทาสก์ใหม่เป็นขั้นตอนสุดท้าย



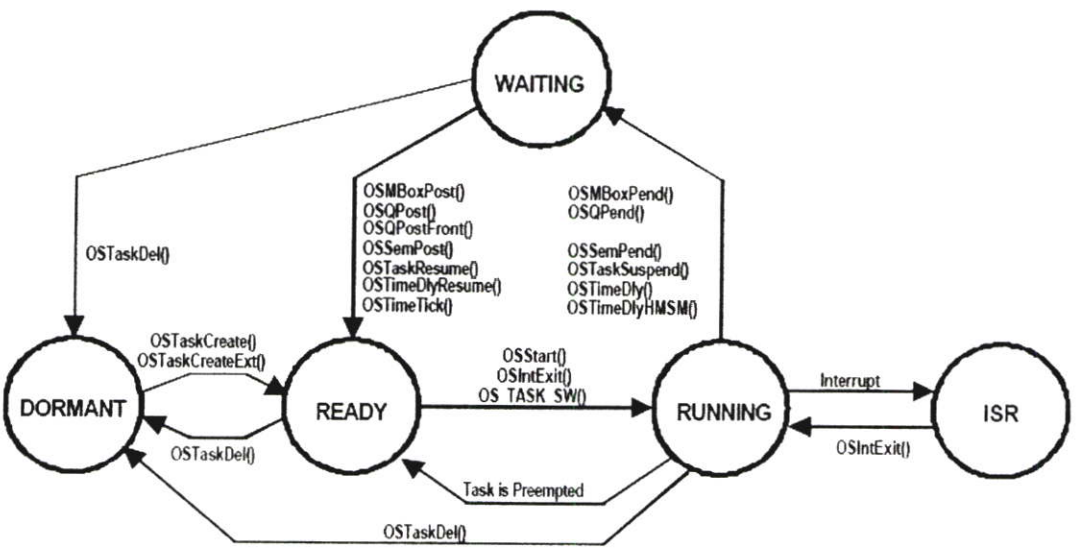
รูปที่ 2.26 Multiple Task [9] [10]

μC/OS II สามารถบริหารจัดการทาสก์ได้ 64 ทาสก์ เรียงลำดับตาม Piority 0 – 63 โดยที่ 0 คือทาสก์ที่มีระดับ Piority สูงสุด และ 63 คือทาสก์ที่มีระดับ Piority ต่ำสุด รูปที่ 2.27 แสดงการจัดระดับ Priority ของ μC/OS II โดยจัดแบ่งออกเป็น 8 กลุ่ม (0) → (7) แต่ละกลุ่มมี 8 ระดับ Priority โดยจะสงวนไว้สำหรับระบบ 8 Piority คือ 4 Piority สูงสุด (0, 1, 2, 3) และ 4 Piority ต่ำสุด (60, 61, 62, 63) คงเหลือระดับ Piority ให้ผู้ใช้กำหนดได้ 56 ระดับ หรือ 56 ทาสก์ และการกำหนด Priority ให้แต่ละทาสก์จะต้องไม่ซ้ำกัน (Unique) [9]



รูปที่ 2.27 Priority Table [9]

เพื่อให้สามารถติดตามการทำงานของทาสก์แต่ละทาสก์ได้ μC/OS II จะกำหนดสถานะการทำงานของทาสก์ไว้ 5 สถานะ คือ DORMANT, READY, RUNNING, WAITING, และ ISR ณ เวลาใดก็ตาม ทาสก์จะอยู่ในสถานะใดสถานะหนึ่ง ใน 5 สถานะ [9] [10] ดังแสดงในรูปที่ 2.28



รูปที่ 2.28 State Transiton Diagram [9] [10]

- สถานะ DORMANT คือ สถานะที่ทาสก์ยังฝังตัวอยู่ใน Program memory ไม่อยู่ในสถานะพร้อมทำงาน

- สถานะ READY คือ สถานะที่ทาสก์พร้อมจะทำงานแต่มีระดับ Priority ต่ำกว่าทาสก์ที่ทำงานอยู่ในปัจจุบัน
- สถานะ RUNNING คือ สถานะของทาสก์ที่กำลังทำงาน (Execute)
- สถานะ WAITING คือ สถานะของทาสก์ที่รอเหตุการณ์ (Event) บางอย่าง เช่น รอการทำงานของ I/O เสร็จสิ้น หรือ รอให้หมดเวลา (Time to expire) เป็นต้น
- สถานะ ISR คือ สถานะของทาสก์ที่ถูกหยุดไว้ เพราะซีพียูถูกอินเตอร์รัพ

การเปลี่ยนสถานะของทาสก์ เป็นไปตาม State Transition Diagram ดังแสดงในรูป 2.28 เมื่อทาสก์อยู่ในสถานะ **DORMANT** คือ สถานะที่ทาสก์ยังอยู่ในโปรแกรม ยังไม่สามารถทำงานได้ ทาสก์จะสามารถทำงานได้โดยการเรียกใช้ *OSTaskCreate()* หรือ *OSTaskCreateExt()* เมื่อทาสก์ถูกสร้างขึ้นจะถูกกำหนดให้มีสถานะ **READY** คือพร้อมที่จะทำงาน ทาสก์อาจถูกสร้างก่อนการมัลติทาสก์จะเริ่มต้น หรือถูกสร้างโดยทาสก์ที่กำลังทำงานอยู่ ถ้าถูกสร้างด้วยทาสก์ และถูกสร้างให้มี Piority สูงกว่าทาสก์ที่สร้างมัน ทาสก์นั้นก็จะได้ควบคุมซีพียูทันที ทาสก์สามารถที่จะย้อนกลับไปสู่สถานะ **DORMANT** ได้โดยเรียกใช้ *OSTaskDel()*

การมัลติทาสก์เริ่มต้นโดยการเรียกใช้ *OSStart()* โดยที่ *OSStart()* จะให้ทาสก์ที่มี Piority สูงสุดและมีสถานะ **READY** ทำงานโดยจะเปลี่ยนสถานะของทาสก์นั้นเป็นสถานะ **RUNNING** ณ เวลาใด ๆ ก็ตามจะมีเพียงทาสก์เดียวเท่านั้นที่ทำงาน ทาสก์ที่มีสถานะ READY จะไม่ทำงานจนกว่าทาสก์ที่มี Piority สูงกว่าจะมีสถานะเป็น **WAITING** หรือถูกลบออกโดย *OSTaskDel()*

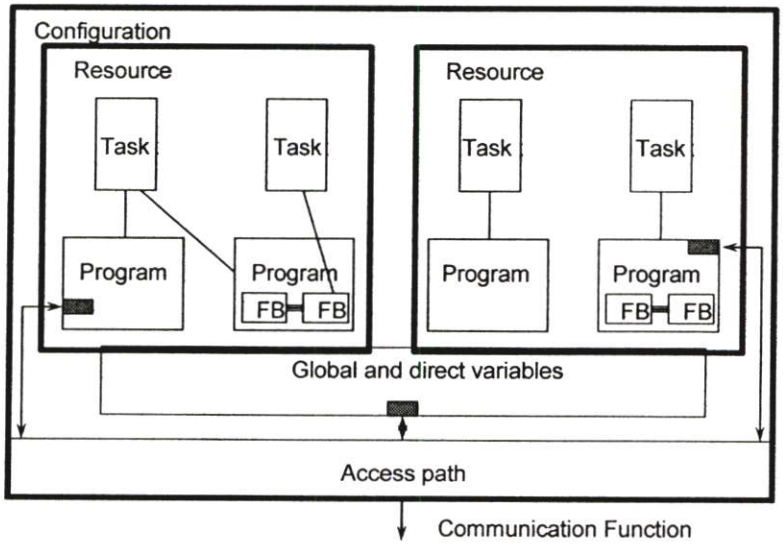
ทาสก์ที่กำลังทำงานอาจจะหยุดตัวเองตามช่วงเวลาที่ต้องการโดยการเรียกใช้ *OSTimeDly()* หรือ *OSTimeDlyHMSM()* ทาสน์นั้นจะเข้าสู่สถานะ **WAITING** จะอยู่ในสถานะนี้จนถึงเวลาที่กำหนด ทาสก์นั้นจะเข้าสู่สถานะ **READY** และถ้าเป็นทาสก์ที่มี Piority สูงสุด ทาสก์นั้นจะได้รับสิทธิในการควบคุมซีพียูทันที

ทาสก์ที่กำลังทำงานอาจถูกอินเตอร์รัพได้ตลอดเวลา เว้นแต่ว่า ทาสก์ หรือ µC/OS II ได้ดิสเอเบิลอินเตอร์รัพไว้ เมื่อมีการอินเตอร์รัพ ทาสก์จะถูกหยุดการทำงานและ เข้าสู่สถานะ **ISR** เพื่อที่โปรแกรมตอบสนองอินเตอร์รัพ (ISR : Interrupt service routine) จะเข้ามาควบคุมซีพียูแทน และเมื่อการตอบสนองการอินเตอร์รัพนั้นเสร็จสิ้น µC/OS II จะตัดสินใจว่าจะรีเทินอินเตอร์รัพไปที่ใด ถ้า Priority ของทาสก์ที่ถูกหยุดไว้ยังคงมี Priority สูงสุด ก็จะรีเทินอินเตอร์รัพไปที่ทาสก์นั้น แต่ถ้ามีทาสก์อื่นที่รออยู่ใน Ready Queue มี Priority สูงกว่า ก็จะรีเทินอินเตอร์รัพไปที่ทาสก์ใหม่ที่มี Priority สูงกว่านั้นแทน ส่วนทาสน์ที่มีสถานะ ISR ก็จะเปลี่ยนสถานะเป็น READY เข้าไปรออยู่ใน Ready Queue เพื่อรอกลับเข้ามาทำงานใหม่เมื่อทาสก์นั้นมี Piority สูงสุดอีกครั้ง

ถ้าไม่มีทาสก์ใดที่มีสถานะ READY รออยู่ใน Ready Queue แล้ว µC/OS II ก็จะไปทำงานที่ Idle ทาสก์ (*OSTaskIdle()*) จนกว่า Clock Tick เข้ามาอินเตอร์รัพ จึงจะออกจาก Idle ทาสก์ เข้าสู่ขบวนการตัดสินใจว่าจะให้ทาสก์ใดใน Ready Queue ทำงานต่อไป [9]

บทที่ 3
การออกแบบซอฟต์แวร์และฮาร์ดแวร์

ใน IEC 61131-3 นอกจากข้อกำหนดรูปแบบของภาษามาตรฐานทั้ง 5 ภาษาแล้ว ยังได้นำเสนอรูปแบบของซอฟต์แวร์ (Software Model) เพื่อเป็นแนวทางในการออกแบบ เพื่อให้โปรแกรม ซอฟต์พีแอลซี เป็นไปตามมาตรฐานที่ IEC 61131 กำหนด จึงได้นำรูปแบบของซอฟต์แวร์ตามที่ IEC 61131-3 แนะนำไว้มาเป็นแนวทางในการออกแบบโปรแกรม รูปแบบของซอฟต์แวร์ของ IEC 61131-3 [12] ดังแสดงในรูปที่ 3.1



รูปที่ 3.1 IEC 61131-3 Software Model [12]

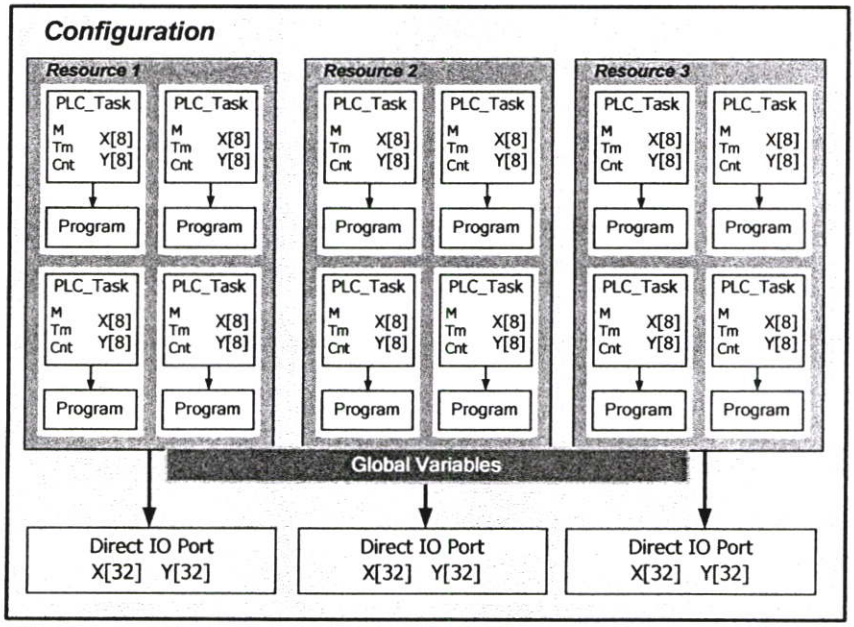
3.1 โครงสร้างซอฟต์แวร์ (Software Structure)

รูปแบบของซอฟต์แวร์ตามที่ IEC 61131-3 เสนอแนะจัดแบ่งออกเป็นระดับชั้น เริ่มจากระดับบนสุด คือ โครงแบบ (Configuration) ภายในโครงแบบประกอบด้วย รีสอร์ซ (Resource) และ แต่ละรีซอร์สประกอบด้วย ทาสก์ (Task) และโปรแกรมของทาสก์ นั้น ๆ [12]

3.1.1 Soft PLC Software Model

โปรแกรมซอฟต์พีแอลซี จัดวางรูปแบบซอฟต์แวร์ ตาม IEC 61131-3 แนะนำไว้ โดยจัดเป็นระดับชั้น ระดับบนสุดคือโครงแบบระบบ (System configuration) ภายในโครงแบบประกอบด้วย 3 รีซอร์ส แต่ละรีซอร์สมี 4 ทาสน์ และแต่ละทาสน์ประกอบด้วย ตัวแปรเฉพาะ (Local variables : Input, Output, Timer, Counter และ Auxiliary relay) และโปรแกรมของทาสน์นั้น ๆ ดังแสดงในรูปที่ 3.2

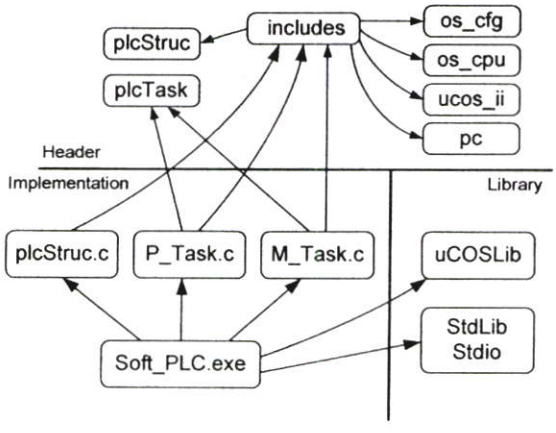
โปรแกรมซอฟต์พีแอลซี รองรับเฉพาะภาษา IL ซึ่งไม่มี Function Block (FB) การส่งผ่านข้อมูลระหว่างทาสก์ทำโดยการส่งข้อมูลผ่าน Global Variable แต่ละทาสก์ทำงานเป็นอิสระต่อกัน จึงไม่มี Access path ส่วนการติดต่อกับอุปกรณ์ภายนอกจะกระทำโดยตรงที่ IO Port



รูปที่ 3.2 Soft-PLC Software Model

3.1.2 โครงสร้างโปรแกรม (Program structure)

เพื่อให้การทำงานเป็นแบบ Real-time โดยใช้ uCOS-II ซึ่งเป็น Real Time OS จากรูปที่ 3.3 แสดงโครงสร้างของโปรแกรม ซอฟต์พีแอลซี แบ่งออกเป็น 3 ส่วนหลัก คือ Header file, Library และ Implementation



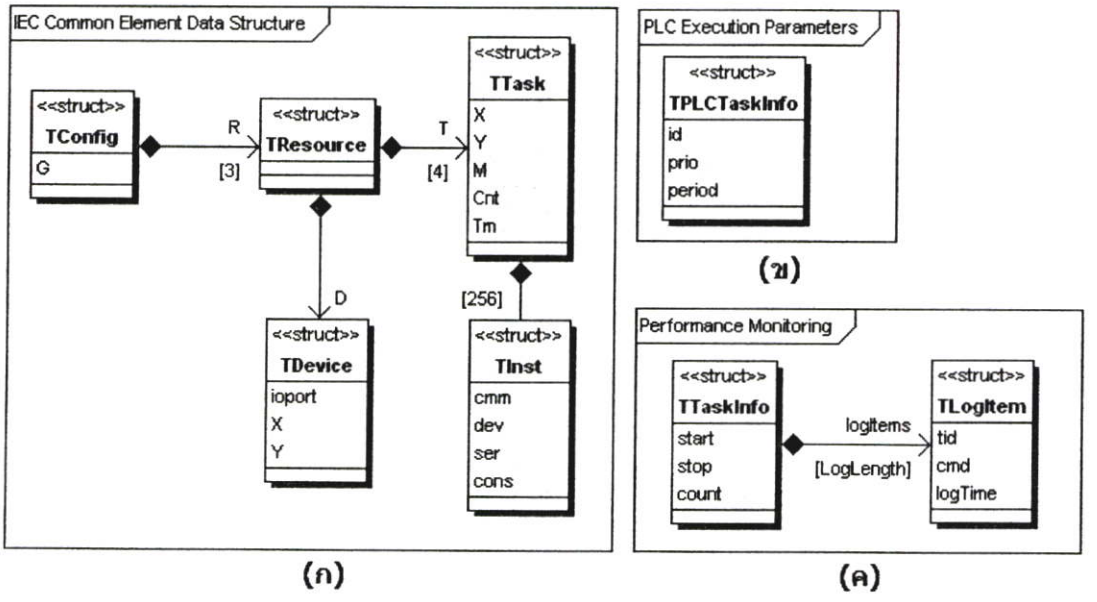
รูปที่ 3.3 Component diagram

Soft_PLC.exe เป็น Executing file ได้มาจาก P_Task.c M_Task.c และ plcStruc.c โดย

- M_Task.c เป็นโปรแกรมทำหน้าที่กำหนดองค์ประกอบทั้งหมดของทาสก์
- P_Task.c เป็นโปรแกรมแปลภาษา Instruction List (IL interpreter)
- plcStruc.c เป็น Direct IO Port map

3.1.3 โครงสร้างข้อมูล (Data Structure)

รูปที่ 3.4 (ก) แสดง Class diagram โครงสร้างข้อมูลภาษา C ของโปรแกรมซอฟต์พีแอลซี เป็นไปตาม IEC กำหนด โดย System configuration (TConfig) ประกอบ 3 รีซอร์ส (TResource) ในแต่ละรีซอร์สประกอบด้วยหนึ่ง Device (TDevice) และ 4 พีแอลซีทาสก์ (TTask) แต่ละทาสก์จะมีโปรแกรม ซึ่งมีได้ไม่เกิน 256 คำสั่ง



รูปที่ 3.4 Class diagram

รูปที่ 3.4 (ข) แสดงค่าพารามิเตอร์ที่ส่งผ่านไปยังพีแอลซีทาสก์ ประกอบด้วยหมายเลขประจำทาสก์ (id) ระดับความสำคัญ (prio) และ คาบเวลา (period)

รูปที่ 3.4 (ค) ตรวจจับการทำงาน โดย Task Information (TtaskInfo : Start, Stop และ Count) ได้มาจากบันทึกการทำงานแต่ละ Task (TlogItem : tid, cmd และ logTime)

3.1.4 ไฟล์โครงแบบ (Configuration file)

plcStruc.h เป็น Header file สำหรับกำหนดโครงแบบของระบบ (System configuration) เป็นส่วนที่ ผู้ใช้สามารถปรับเปลี่ยนโครงแบบของระบบได้ตามความเหมาะสมของการใช้งาน ภายใน plcStruc.h ประกอบด้วยการกำหนด

- จำนวนรีซอร์ส (Resource)
- ตัวแปรร่วม (Global Variables)
- จำนวนทาสก์ (Task) ในแต่ละรีซอร์ส
- ตัวแปรเฉพาะ (Local Variables) ของแต่ละทาสก์ และ
- จำนวน อินพุท และ เอาท์พุท ของระบบ

โค้ดของ plcStruc.h ดังแสดงในรูปที่ 3.5

```
typedef struct
{
  int       ioport;
  BYTE      X[32];          // Number of system Input port.
  BYTE      Y[32];          // Number of system Output port.
} TDevice;

typedef struct
{
  BYTE      X[8];           // Number of input port for each Task.
  BYTE      Y[8];           // Number of output port for each Task.

  BYTE      M[M_Max];       // Number of auxiliary relay for each Task.
  BYTE      Cnt[Cnt_Max];   // Number of counter for each Task.
  BYTE      Tm[Tm_Max];     // Number of timer for each Task.
} TTask;  /*tid*/

typedef struct
{
  TTask     T[4];           // Number of Task for each Resource.
  TDevice   D;
} TResource; /*rid*/

typedef struct
{
  INT8U       G[MaxG];
  TResource   R[3];
} TConfig;
```

รูปที่ 3.5 โค้ดของ plcStruc.h

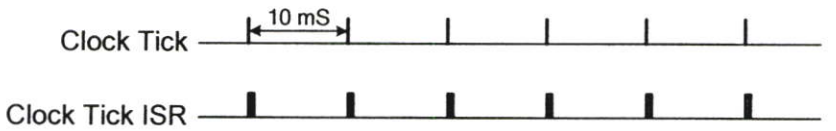
3.2 กลไกและขั้นตอนการทำงาน (Mechanism and Algorithms)

ระบบปฏิบัติการ uCOS-II ใช้กลไกในการรักษาการทำงานให้เป็นไปตามเวลาที่กำหนด โดยการสร้างการอินเตอร์รัพซีพียู แบบมีคาบเวลา เรียกว่า Interrupt Tick หรือ Clock Tick

โดยทั่วไป จะมีคาบเวลาอยู่ที่ 20 – 100 ครั้ง/วินาที และ uCOS-II เปิดให้ผู้ใช้กำหนดคาบเวลาการอินเตอร์รัพนี้ได้ใน Header file (OS_CFG.H) โดยกำหนดเป็นจำนวนครั้งต่อวินาที [4]

```
#define OS_TICKS_PER_SEC        100     /* Set the number of ticks in one second*/
```

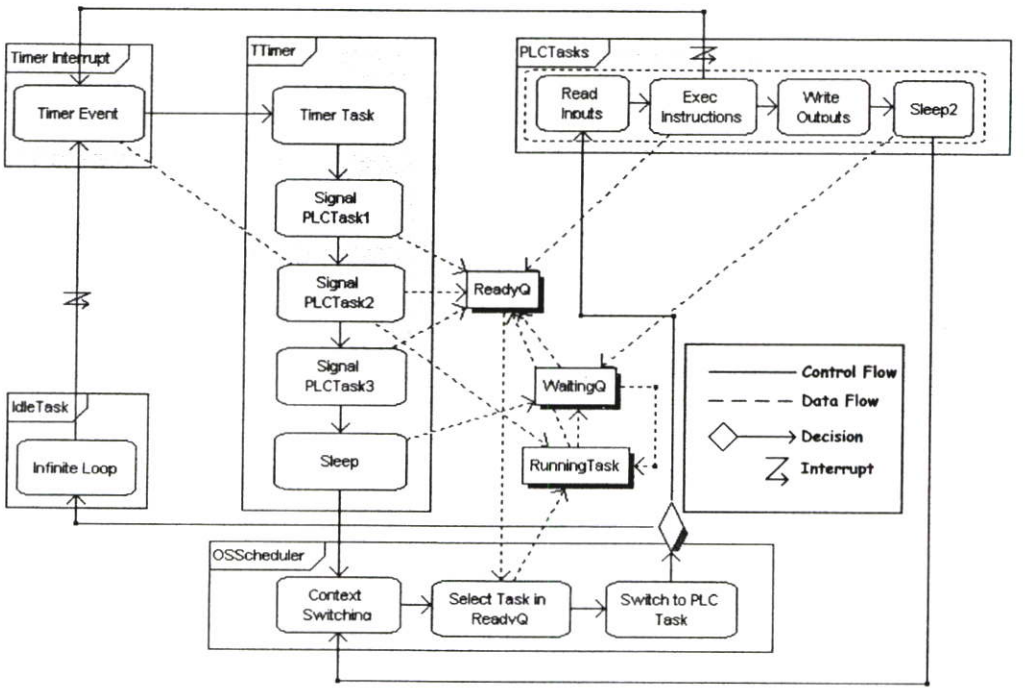
สำหรับโปรแกรมซอฟต์พีแอลซี กำหนดไว้ที่ 100 ครั้ง/วินาที หรือ 10mS (โค๊ดดังข้างบน) ทุกครั้งที่ Clock Tick ปรากฏ โปรแกรมตอบสนองการอินเตอร์ (Clock Tick ISR) จะทำให้ทาสก์ที่มีสถานะ READY และมี Priority สูงสุด อยู่ในสภาวะพร้อมทำงาน แล้วทำการเปลี่ยนบริบทไปที่ทาสก์นั้น



รูปที่ 3.6 การอินเตอร์รัพซีพียูแบบมีคาบเวลา

3.2.1 แผนผังการทำงาน (Activity Diagram)

การทำงานของโปรแกรมซอฟต์พีแอลซี ดังแสดงในรูปที่ 3.7 โดยมีทาสก์ที่ทำงานเกี่ยวข้องกัน คือ Timer Interrupt, TTimer, OSSchedule, PLCTasks และ Idle Task



รูปที่ 3.7 Activity diagram

รูปที่ 3.7 แสดงแผนผังการทำงานของโปรแกรมซอฟต์พีแอลซี โดยมีการทำงานตามผังการทำงาน ดังต่อไปนี้

ณ เวลาใด ๆ เมื่อมีการอินเตอร์รัพท์เข้ามา ISR จะทำการเปลี่ยนบริบทไปที่ Timer ทาสก์ เพราะมี Priority สูงกว่าทาสก์อื่น

Timer ทาสก์จะปลุกพีแอลซีทาสก์ ทีละทาสก์ให้เข้าสู่สถานะ *READY* แล้วเข้าไปรออยู่ใน Ready Queue ส่วน Timer ทาสก์จะส่งการควบคุมไปยัง OSScheduler โดยการนำตัวเองเข้าสู่ภาวะ Sleep ทำให้ Timer ทาสก์เข้าสู่สถานะ *WAITING* เข้าไปรออยู่ใน Waiting Queue จนกว่าจะครบเวลาที่กำหนด (time to expire) ก็จะเปลี่ยนสถานะเป็น *READY* เข้าไปรออยู่ใน Ready Queue

```
OSTimeDly(1);          /* Sleep mode delay 1 tick */
```

OSScheduler ทำหน้าที่เลือกทาสก์ที่อยู่ใน Ready Queue โดยเลือกทาสก์ที่มีระดับ Piority สูงสุดให้เข้าสู่สถานะ *RUNNING* แล้วทำการเปลี่ยนบริบทไปที่ทาสก์นั้น แต่ถ้าไม่มีทาสก์ใดรออยู่ใน Ready Queue ก็จะเลือก Idle ทาสก์ *OSTaksIdle()* โดยส่งการควบคุมซีพียูให้ Idle ทาสก์

Idle ทาสก์ เป็นทาสก์ที่มีระดับ Pririty ต่ำสุด ทำงานแบบวนรอบไม่รู้จบ (Infinite Loop) จะทำงานไปเรื่อยจนกว่าจะมีการอินเตอร์รัพเข้ามาจึงจะออกจาก Idle ทาสก์ นี้ได้

```
// Idle Task
void OSTaskIdle (void *pdata)
{
        pdata = pdata;
        for (;;) {
                OS_ENTER_CRITICAL();
                OSIdleCtr++;
                OS_EXIT_CRITICAL();
        }
}
```

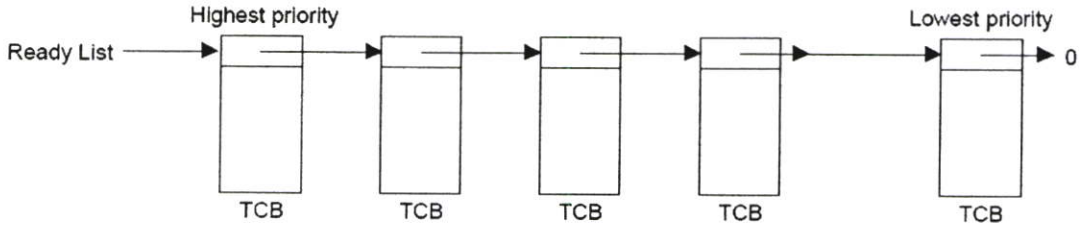
ในกรณีที่ Ready Queue มีพีแอลซีทาสก์ที่อยู่ในสถานะ *READY* รออยู่ OSScheduler จะเลือกพีแอลซีทาสก์ที่มีระดับ Piority สูงสุด แล้วทำการเปลี่ยนบริบทไปที่พีแอลซีทาสก์นั้น

พีแอลซีทาสก์ที่ OSScheduler ส่งการควบคุมมาให้ เริ่มทำการอ่านข้อมูลจาก IO Port ของมันมาเก็บไว้ นำข้อมูลที่ได้มาประมวลผลตามโปรแกรมของมัน โดยอ่านคำสั่งจากโปรแกรมทีละคำสั่ง กระทำการตามคำสั่ง (Execute) นั้น ตั้งแต่คำสั่งแรกไปจนจบโปรแกรม เสร็จแล้วจัดส่งผลที่ได้จากการประมวลผลไปที่ IO Port แล้วพักตัวเองโดยการนำตัวเองเข้าสู่ภาวะ Sleep คืนการควบคุมซีพียูให้กับ OSScheduler และสถานะของพีแอลซีทาสก์นั้น ก็เป็นสถานะ *WAITING* เข้าไปอยู่ใน Waiting Queue จนกว่าจะครบเวลาที่กำหนด (time to expire) ก็จะเปลี่ยนสถานะเป็น *READY* เข้าไปรออยู่ใน Ready Queue

ถ้าในระหว่างที่พีแอลซีทาสก์กำลังประมวลผล มีการอินเตอร์รัพ หรือ Clock Tick ใหม่ปรากฏ ก่อนที่จะทำการเปลี่ยนบริบทไปที่ Timer ทาสก์ ISR จะเปลี่ยนสถานะของพีแอลซีทาสก์นั้นให้มีสถานะเป็น *READY* แล้วเข้าไปรอใน Ready Queue

เมื่อ OSScheduler ได้รับการควบคุมคืนจากพีแอลซีทาสก์ OSScheduler ก็จะทำหน้าที่ซ้ำเดิมคือ เลือกทาสก์ที่อยู่ใน Ready Queue โดยเลือกทาสก์ที่มีระดับ Piority สูงสุดให้เข้าสู่สถานะ *RUNNING* แล้วทำการเปลี่ยนบริบทไปที่ทาสก์นั้น

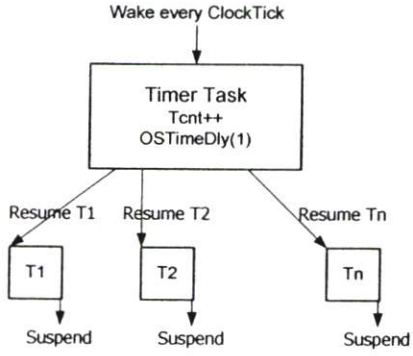
Ready Queue หรือ Ready List เป็น Link list ของทาสก์ที่มีสถานะ *READY* พร้อมจะทำงาน ทุกครั้งที่มีการนำทาสก์เข้าหรือออกจาก Ready Queue จะมีการปรับปรุง Link list นี้เสมอ เพื่อให้มั่นใจได้ว่า ทาสก์ที่มีระดับ Priority สูงสุดจะเป็นคิวแรกเสมอ แล้วเรียงลำดับตามระดับ Priority ต่อกันไปเรื่อย ๆ [5] ดังแสดงในรูปที่ 3.8



รูปที่ 3.8 Link list ของ Ready List [10]

3.2.2 การจัดการทาสก์และวิธีกำหนดตาราง (Task Management & Scheduling Policy)

โปรแกรมซอฟต์พีแอลซี ให้ Timer ทาสก์ เป็นทาสก์ที่ควบคุมการทำงานของพีแอลซีทาสก์ โดย Timer ทาสก์ เป็นทาสก์ที่มีคาบ (Periodic task) มีคาบการตื่นทุก 10 mS หรือทุก 1 Clock tick ดังแสดงในรูปที่ 3.9 Timer ทาสก์ทำหน้าที่ตัดสินว่า จะให้พีแอลซีทาสก์ใดทำงาน ตามช่วงเวลาที่แต่ละพีแอลซีทาสก์กำหนดไว้ใน Schedule ของพีแอลซีทาสก์นั้น ๆ (โค้ดดังแสดงในรูปที่ 3.10 (ก)) โดย Timer ทาสก์จะส่ง Resume signal ไปยัง OSScheduler หลังจากส่ง signal แล้ว Timer ทาสก์จะหยุดตัวเอง (Delay 1 tick) ส่วนพีแอลซีทาสก์ที่ถูกปลุกจะมีสถานะ Ready เข้าไปรอใน Ready Queue



รูปที่ 3.9 Task control flows

```
// Initialize PLC Task scheduling period counted in clock tick
PLCTaskInfos[0].id      = 4;    /* PLCTask ID */
PLCTaskInfos[0].prio    = 15;   /* PLCTask priority */
PLCTaskInfos[0].period  = 5;    /* PLCTask piriod : wake every 5 clock tick */
```

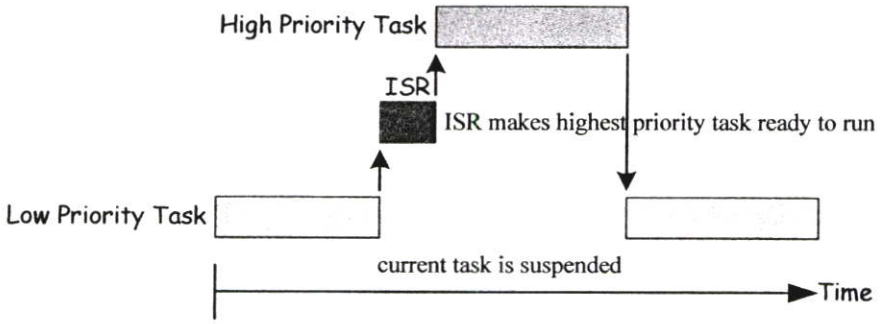
(ก)

```
// Timer Task
for (; ;timerCounter++ ) {
        for (i=0; i < PLC_TASKS; i++ ) {
                if (timerCounter % PLCTaskInfos[i].period == 0) {
                        OSTaskResume(PLCTaskInfos[i].prio);
                }
        }
        OSTimeDly(1);           /* Delay one clock tick */
}
```

(ข)

รูปที่ 3.10 Source code ของ (ก) PLC Tsk scheduling (ข) Timer Task

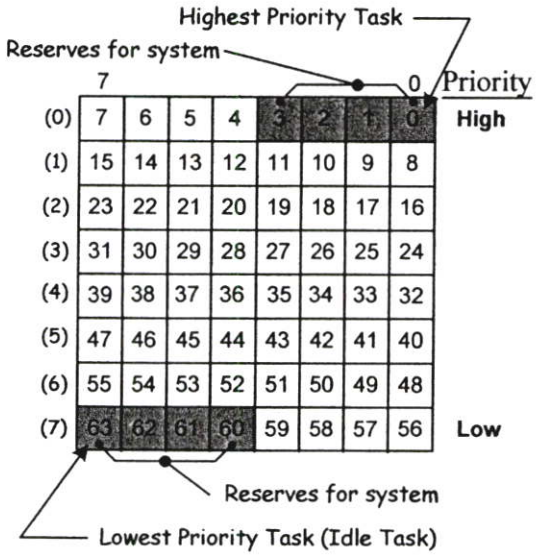
3.2.2.1 Scheduling policy สำหรับโปรแกรม ซอฟต์พีแอลซี เป็นแบบ Preemptive Priority Based Scheduling นั่นคือ OSScheduler จะเลือกทาสก์ที่มี Priority สูงสุด ที่อยู่ใน Ready Queue ให้ทำงานก่อนเสมอ โดยทาสก์ที่ถูกเลือกจะเปลี่ยนสถานะจาก READY เป็นสถานะ RUNNING



รูปที่ 3.11 Preemptive Priority Based Scheduling [14]

3.2.2.2 Priority Table

เพราะ OSScheduler จะเลือกทาสก์ที่มีระดับ Priority สูงสุดให้ทำงานก่อนเสมอ ดังนั้นการกำหนดระดับ Priority ให้แต่ละทาสก์จึงมีความสำคัญ รูปที่ 3.12 แสดง Priority Table ของ μCOS-II เตรียมไว้ให้ผู้ใช้ ใช้ในการจัดระดับ Priority ให้แต่ละทาสก์



รูปที่ 3.12 Priority Table ของ μCOS-II [9]

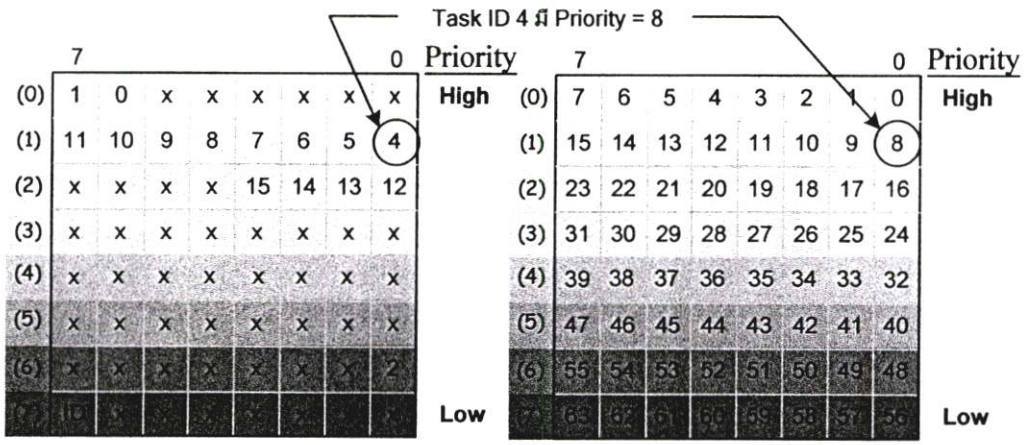
μCOS-II แบ่งตารางการจัดระดับ Priority ออกเป็น 8 กลุ่ม กลุ่มละ 8 ระดับ โดยเรียงระดับจาก Priority น้อยไปมาก เริ่มจาก 0 คือระดับ Priority สูงสุด และ 63 คือระดับ Priority ต่ำสุด โดยจะสงวนไว้สำหรับระบบ 8 Priority คือ 4 Priority สูงสุด (0, 1, 2, 3) และ 4 Priority ต่ำสุด (60, 61, 62, 63:Idle Task) [9]

ทาสก์ที่ใช้เวลาในการทำงานมากกว่า 1 Clock tick ควรกำหนดให้ทาสก์เหล่านี้มี Priority ต่ำ เพื่อให้ทาสก์ที่มีการทำงานเร็วกว่าเข้ามาทำงานก่อน ในทำนองเดียวกันทาสก์ที่มีกำหนดเวลาที่ต้องทำให้เสร็จ (Deadline) ควรให้มี Priority สูง การกำหนดระดับ Priority ให้แต่ละทาสก์ขึ้นอยู่กับความสำคัญของงานและเงื่อนไขเรื่องเวลา โดยผู้ใช้สามารถกำหนดได้ด้วยตนเอง

สำหรับโปรแกรมซอฟต์พีแอลซี กำหนดระดับ Priority ของทาสก์ต่าง ๆ ไว้ดังนี้

- Task ID 0 คือ Task_Start เป็นทาสก์ที่กำหนดองค์ประกอบทั้งหมดของระบบ กำหนดให้มี Priority สูงสุด (Priority = 6)
- Task ID 1 คือ Task_Timer เป็นทาสก์ควบคุมการทำงานของทาสก์ทั้งหมดทำงานต่อจาก Task_Start กำหนดให้มี Priority สูงรองลงมา (Priority = 7)
- Task ID 4 – 15 (12 ทาสก์) คือ Task_PLC 1 - 12 ตามลำดับ พีแอลซีทาสก์ทั้ง 12 ทาสก์ต่างก็ทำงานเป็นอิสระต่อกัน จึงจัดระดับความสำคัญให้เรียงต่อกัน (Priority = 8→19) โดยที่ Task_PLC ID 4 – 11 อยู่ที่ระดับ (1) (8→15) และ Task_PLC ID 12 – 15 อยู่ที่ระดับ (2) (16→19)
- Task ID 2 คือ Task_Disp เป็นทาสก์ที่อ่านสถานะ IO ของ PLC แต่ละทาสก์มาแสดงผลที่มอนิเตอร์ ให้ Priority ต่ำ (Priority = 48)

ส่วน Idle Task เป็น Task ที่มี Priory ต่ำสุด (Priority = 63) ดังแสดงในรูปที่ 3.13 (ข)



รูปที่ 3.13 การจัดระดับ Priority ของซอฟต์พีแอลซี

หมายเหตุ ตัวเลขที่ปรากฏในรูปที่ 3.13 (ก) เป็นตัวเลขเพื่อใช้อ้างอิงตำแหน่งของ Priority ส่วนตัวเลขในรูปที่ 3.13 (ข) เป็นเลขประจำทาสก์ (ID) ของแต่ละทาสก์

ตัวอย่างโค๊ดบางส่วนของการกำหนดค่าลำดับความสำคัญ ดังแสดงในรูปที่ 3.14

```
/********** Task ID ************/
#define     TASK_START_ID       0
#define     TASK_TIMER_ID       1
#define     TASK_PLC_ID         4       // PLC Task 1 (ID)
#define     Task_Disp_ID        2

/********** Task Priority **********/
#define     TASK_START_PRIO     6
#define     TASK_TIMER_PRIO     7
#define     TASK_PLC_PRIO       8       // PLC Task 1 (Priority)
#define     Task_Disp_PRIO      48
```

รูปที่ 3.14 การกำหนดเลขประจำทาสก์ (ID) และลำดับทาสก์ (Priority)

3.2.3 Communication Path

การติดต่อสื่อสารระหว่าง PLC Task กับ Input Output Port ของแต่ละทาสก์ จะติดต่อโดยตรงกับตำแหน่งจริงทาง Hardware โดยทำการกำหนด (Port Map) ไว้ในการกำหนดโครงแบบของระบบ (System configuration) PLC Task แต่ละทาสก์ เป็นอิสระต่อกัน ไม่เกี่ยวข้องหรือสัมพันธ์กัน การรับหรือส่งข้อมูลจาก Port จะกระทำโดยตรงผ่านคำสั่ง Inport () หรือ Outport () ตัวอย่างโค๊ดบางส่วนของการทำ IO Port Map แสดงในรูปที่ 3.15

```
/********** IO Port Map *************/

// Input Port (Resource 0, Task 0)
C -> R[0].T[0].X[0] = 0;  // Device.X[0]
C -> R[0].T[0].X[1] = 1;  // Device.X[1]
C -> R[0].T[0].X[2] = 2;  // Device.X[2]
C -> R[0].T[0].X[3] = 3;  // Device.X[3]
C -> R[0].T[0].X[4] = 4;  // Device.X[4]
C -> R[0].T[0].X[5] = 5;  // Device.X[5]
C -> R[0].T[0].X[6] = 6;  // Device.X[6]
C -> R[0].T[0].X[7] = 7;  // Device.X[7]

// Output Port (Resource 0, Task 0)
C -> R[0].T[0].Y[0] = 0;  // Device.Y[0]
C -> R[0].T[0].Y[1] = 1;  // Device.Y[1]
C -> R[0].T[0].Y[2] = 2;  // Device.Y[2]
C -> R[0].T[0].Y[3] = 3;  // Device.Y[3]
C -> R[0].T[0].Y[4] = 4;  // Device.Y[4]
C -> R[0].T[0].Y[5] = 5;  // Device.Y[5]
C -> R[0].T[0].Y[6] = 6;  // Device.Y[6]
C -> R[0].T[0].Y[7] = 7;  // Device.Y[7]

// IO Port Address (Resource 0, Task 0 - 3)
C -> R[0].D.ioport = 0x0320;
```

รูปที่ 3.15 การกำหนดตำแหน่งทาง Hardware โดยตรง (Direct IO Port Map)

3.2.4 Resource Sharing และ Race Condition

ภาษา IL เป็นภาษาที่ใช้ในการโปรแกรมพีแอลซีมาตั้งแต่ช่วงเริ่มต้น ซึ่งเป็นการทำงานแบบกระบวนการเดียว (Single Process) โครงสร้างของภาษา IL ไม่ได้ถูกจัดเตรียมไว้สำหรับการทำงานที่ซับซ้อนและมีการใช้ทรัพยากรร่วมกัน สำหรับโปรแกรมซอฟต์พีแอลซี การสื่อสารระหว่างพีแอลซีทาสก์กระทำผ่านตัวแปรร่วม (Global Variable) ในลักษณะของการส่ง Flag หรือสัญญาณ เพื่อกำหนดจังหวะในการทำงาน ไม่มีการส่งผ่านข้อมูลเป็นชุดหรือกลุ่มข้อมูล แต่ละพีแอลซีทาสก์ต่างก็ทำงานเป็นอิสระต่อกัน ทรัพยากรที่ใช้ถูกจัดสรรไว้เฉพาะ (Local) เป็นส่วน ๆ เพื่อหลีกเลี่ยงปัญหา Resource Sharing และสภาวะการช่วงชิง (Race Condition) ในการใช้ตัวแปรร่วม

3.2.5 Hardware

ในงานควบคุมเป็นการทำงานร่วมกันระหว่างพีแอลซีกับอุปกรณ์ภายนอกอื่น ๆ ทั้งอุปกรณ์อินพุทและอุปกรณ์เอาท์พุท ซึ่งปัญหาที่เกิดจากอุปกรณ์ต่อร่วมเหล่านี้จะส่งผลกระทบโดยตรงกับพีแอลซี ปัญหาที่พบเกิดจาก 2 สาเหตุ คือ

- เกิดจากระบบจ่ายไฟขัดข้อง เป็นเหตุให้พีแอลซีหยุดการทำงาน เมื่อระบบจ่ายไฟกลับมาทำงานเป็นปกติ ปัญหาก็คือ การกลับมาทำงานใหม่ของพีแอลซีจะสามารถทำงานต่อจากเดิมได้หรือไม่ หรือต้องเริ่มทำงานใหม่จากจุดเริ่มต้น พีแอลซีโดยทั่วไปแก้ปัญหานี้โดยการแบ่งหน่วยความจำที่เก็บสถานะของอุปกรณ์เสมือนต่าง ๆ ไว้ ส่วนหนึ่ง จัดให้มีแบตเตอรี่ Backup แล้วระบุในรายละเอียดให้ผู้ใช้รู้ว่าอุปกรณ์ใด เบอร์ใด

ที่มีแบตเตอรี่ Backup ไว้ เพื่อที่ผู้ใช้สามารถเลือกใช้ในการพัฒนาโปรแกรมเพื่อควบคุมการทำงานของพีแอลซี ในส่วนนี้ซอฟต์พีแอลซี ไม่ได้จัดเตรียมไว้

- เกิดจากอุปกรณ์ภายนอกที่นำมาทำงานร่วมกับพีแอลซีชำรุด ซึ่งจะส่งต่อซอฟต์พีแอลซีที่ทำงานแบบมัลติทาสก์ในสองกรณี คือ ถ้าโปรแกรมให้พีแอลซีทาสก์แต่ละทาสก์ทำงานร่วมกัน ในกรณีนี้ต้องหยุดการทำงานทั้งระบบ เพื่อเปลี่ยนอุปกรณ์ที่ชำรุดนั้น ส่วนอีกกรณี คือ ถ้าโปรแกรมให้พีแอลซีทาสก์แต่ละทาสก์ทำงานเป็นอิสระต่อกัน กรณีนี้สามารถหยุดเฉพาะพีแอลซีทาสก์ที่มีปัญหาได้ โดยที่พีแอลซีทาสก์อื่นยังคงสามารถทำงานตามปกติ (แต่ละพีแอลซีทาสก์ต้องมี Hardware สำหรับควบคุมการเปิด/ปิด ของทาสก์เอง)

3.3 IL Interpreter

โปรแกรมที่ทำหน้าแปลภาษา IL จะเป็นส่วนหนึ่งใน P_Task.c หลังจากที่ Timer ทาสก์ปลุกพีแอลซีทาสก์ทุกทาสก์ ให้เข้ามารออยู่ใน Ready queue แล้ว Timer ทาสก์ จะเข้าสู่สถานะ *WAITING* (*OSTimeDly(1)*) ส่งการควบคุมให้ OSScheduler เลือกทาสก์ที่มีระดับ Prioriy สูงสุด ซึ่ง พีแอลซีทาสก์ จะมีระดับ Priority สูงรองจาก Timer ทาสก์ จะถูก OSScheduler เลือกให้มีสถานะ *RUNNING* ทีละทาสก์เรียงตามระดับ Priority ของแต่ละพีแอลซีทาสก์

พีแอลซีทาสก์ใดที่ได้สถานะ *RUNNING* จะเริ่มทำงานโดยการเคลียร์สถานะของอุปกรณ์เสมือนต่าง ๆ (Timer, Counter, Relay, Output และ Input) ให้อยู่ในสถานะเริ่มต้น แล้วอ่านโปรแกรมที่ใช้ควบคุมพีแอลซีทาสก์ทาสก์นั้น เข้ามาเก็บในหน่วยความจำในส่วนของทาสก์ที่ µCOS-II จัดสรรให้ตอนสร้างทาสน์ (*OSTaskCreateExt()*) แล้วหยุดตัวเองเข้าไปรออยู่ใน Waiting queue (*OSTaskSuspend(OS_PRIO_SELF)*) รอการทำงานในรอบถัดไป

เมื่อพีแอลซีทาสก์ ได้สถานะ *RUNNING* จะคืนกลับมาทำงานอีกครั้ง จะทำงานต่อจากเดิม คือเข้าสู่ Infinite loop (*for (; ;) {code...}*) เริ่มจาก

- อ่านข้อมูลจาก Input port นำข้อมูลที่ได้เก็บไว้ในหน่วยความจำเพื่อใช้ในระหว่างประมวลผล
- ปรับปรุงข้อมูลและสถานะของไทม์เมอร์ที่ถูกใช้งาน
- เริ่มอ่านคำสั่งทีละคำสั่ง ซึ่งคำสั่งแต่ละบรรทัดจะถูกแยกเก็บเป็น 3 ฟิลด์ คือ Operator, Device และ Serial No. ตรวจว่าสั่งให้ทำอะไรแล้วทำตามทีละคำสั่ง ทำต่อเนื่องไปจนกระทั่งถึงคำสั่งสุดท้าย
- ส่งผลลัพธ์ที่ได้จากการประมวลผลออกที่ Output port แล้ว
- หยุดตัวเอง (Suspend) เข้าไปรอใน Waiting queue รอการถูก Resume กลับเข้ามาทำงานอีกครั้ง

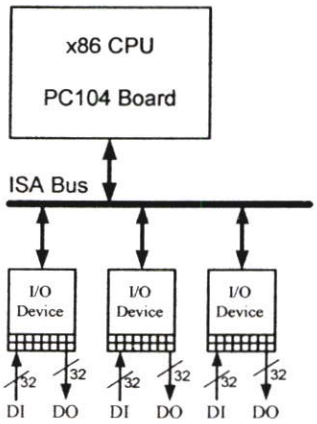
```
if (!(strcmp(ptr[i].cmm,"LD"))) {
   switch (*ptr[i].dev) {
       case 'G' :  buf = G((ptr[i].ser)); break;
       case 'X' :  buf = X((ptr[i].ser)); break;
       case 'M' : buf = M(ptr[i].ser);   break;
       case 'T' :  tc[(ptr[i].ser)] = !Tm((ptr[i].ser)) && ts[(ptr[i].ser)];
                   buf = tc[(ptr[i].ser)]; break;
       case 'C' :  buf = cc[(ptr[i].ser)]; break;
    }
}

if (!(strcmp(ptr[i].cmm,"RST"))) {
  if (buf) {
    switch (*ptr[i].dev) {
        case 'T' : Tm((ptr[i].ser)) = 0; tc[(ptr[i].ser)] = 0; ts[(ptr[i].ser)] = 0;
                   break;
        case 'C' : Cnt((ptr[i].ser)) = 0; cc[(ptr[i].ser)] = 0; cs[(ptr[i].ser)] = 0;
                   cr[(ptr[i].ser)] = 0; break;
    }
  }
}
```

ด้านบนคือตัวอย่างบางส่วนของโค๊ด

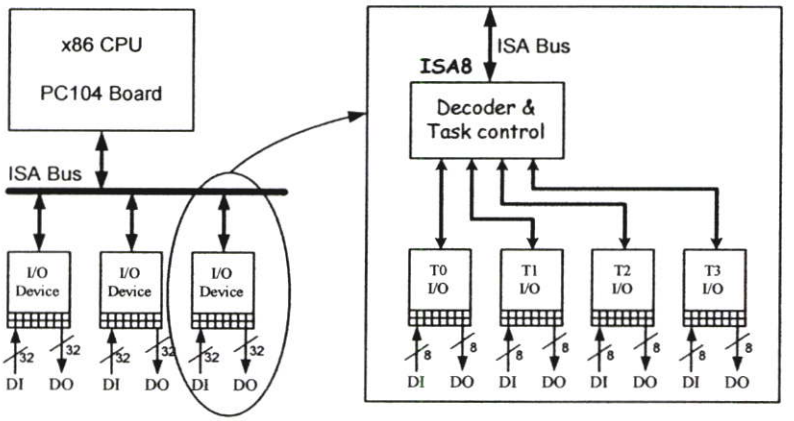
3.4 Hardware Design

วัตถุประสงค์ของรายงานฉบับนี้ โปรแกรม ซอฟต์พีแอลซี ต้องสามารถนำไปใช้กับ Micro Controller (x86) รูปที่ 3.16 แสดงโครงสร้างทาง Hardware ที่จะนำโปรแกรม ซอฟต์พีแอลซี ไปพอร์ทลงบน PC104 Board ที่ใช้ x86 CPU 24 MHz และเชื่อมต่อกับ I/O Device 3 boards ผ่าน ISA Bus แต่ละ I/O Device มี 64 IO (Input 32 bits / Output 32 bits)



รูปที่ 3.16 โครงสร้างทาง Hardware

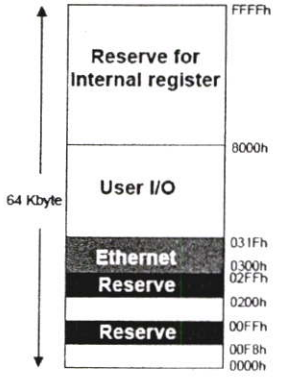
จาก Software Model ของโปรแกรม ซอฟต์พีแอลซี ประกอบด้วย 3 Resource แต่ละ Resource มี 4 PLC Task เพื่อให้ Hardware สอดคล้องกับ Software Model จึงออกแบบให้ทุก Resource มี I/O Device หนึ่งชุดทำหน้าที่เป็นหน่วยรับและส่งออกข้อมูล โครงสร้างภายในของ I/O Device ประกอบด้วย 4 I/O Unit ทำหน้าที่รับเข้าและส่งออกข้อมูลของแต่ละ PLC Task ใน Resource และแต่ละ IO Unit มี 16 IO (Input 8 bits / Output 8 bits) ดังแสดงในรูปที่ 3.17 โดยที่ I/O Unit ทั้งหมด เป็นอิสระต่อกัน สามารถทำการรับหรือส่งออกข้อมูลกับ Unit ใด ๆ ได้โดยตรง



รูปที่ 3.17 โครงสร้างของ IO Device

จากแผนภาพการจัดวางหน่วยความจำในส่วนของ I/O ของบอร์ด x86 ที่ใช้ในการทดลอง กำหนดตำแหน่งของ User I/O ไว้ตั้งแต่ Address 0320H จึงจัดวางตำแหน่งของ IO Device ของแต่ละ Resource ไว้ดังนี้

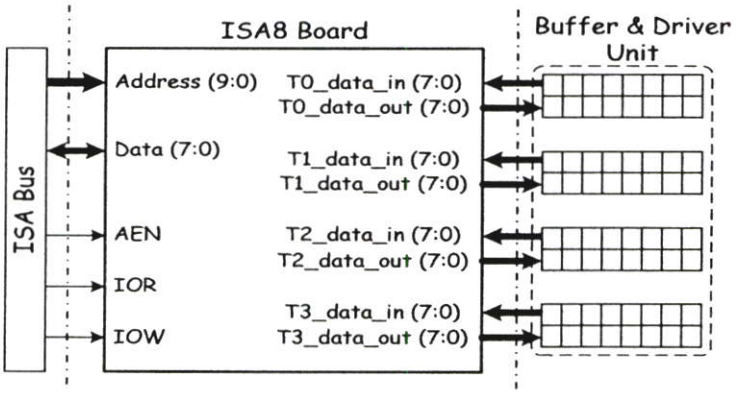
- Resource 0 จัดวาง IO device ที่ตำแหน่ง 0320H – 0323H
- Resource 1 จัดวาง IO device ที่ตำแหน่ง 0324H – 0327H
- Resource 2 จัดวาง IO device ที่ตำแหน่ง 0328H – 032BH



รูปที่ 3.18 แผนภาพการจัดวางหน่วยความจำในส่วนของ I/O ของบอร์ด x86

เขียนเป็น Block Diagram ได้ดังรูปที่ 3.19 แบ่งการทำงานของ ISA8 Board (IO Device) ออกเป็นสองส่วน คือ ส่วนที่

- ติดต่อกับ ISA Bus เพื่อนำสัญญาณ Address Bus (9:0) Data Bus (7:0) และสัญญาณควบคุม AEN, IOW, IOR มากำหนด Address Port และ ทิศทางของข้อมูล
- ติดต่อกับ Buffer & Driver Unit ติดต่อกับอุปกรณ์ภายนอกที่จะนำมาใช้ทำงานร่วมกับ PLC เช่น Limit Switch, Photo Sensor หรือ Coil ของ Relay



รูปที่ 3.19 Block Diagram ของ IO Device

เพราะมีจุดต่อที่ต้องติดต่อกับอุปกรณ์ภายนอกจำนวนมาก (Buffer & Driver Unit (16×4) + ISA Bus (10+8+3) = 85 จุดต่อ) เลือกใช้ Chip FPGA ของบริษัท Xilink เบอร์ xc2s50-6tq144 ซึ่งมีจำนวน User IO 92 จุด ดังแสดงในรูปที่ 3.20 เป็นบอร์ดรุ่น FPGA Adventure XC2S50A ของบริษัท เอเพก อินสตรูเมนต์ จำกัด บอร์ดที่ถูกออกแบบให้ใช้ไฟเลี้ยง D.C. 5V และ I/O ของ FPGA สามารถเชื่อมต่อได้ทั้ง 5V และ 3.3V

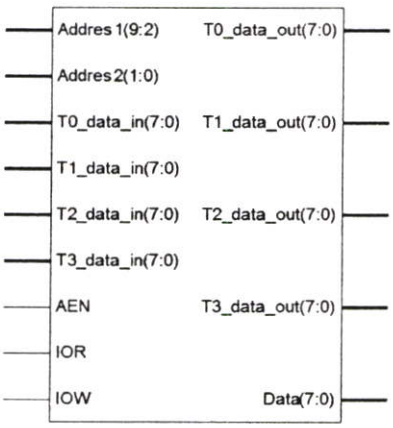
Spartan-II User I/O Chart[1]

Device	Maximum User I/O	Available User I/O According to Package Type					
		VQ100	TQ144	CS144	PQ208	FG256	FG456
XC2S15	86	60	86	86	-	-	-
XC2S30	132	60	92	92	132	-	-
XC2S50	176	-	92	-	140	176	-
XC2S100	196	-	92	-	140	176	196
XC2S150	260	-	-	-	140	176	260
XC2S200	284	-	-	-	140	176	284

Notes:
1. All user I/O counts do not include the four global clock/user input pins.

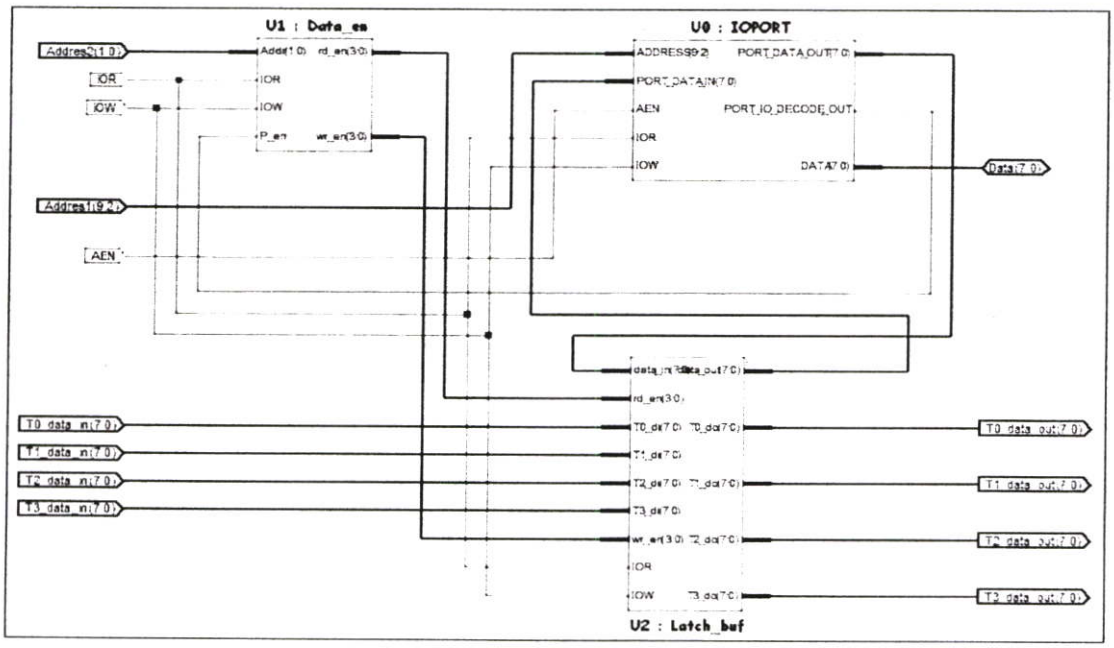
รูปที่ 3.20 Spartan-II User I/O (XC2Sxx)

โดยใช้ซอฟต์แวร์ทูลของบริษัท Xilinx คือ ISE WebPACK 8.102i เป็นเครื่องมือช่วยในการออกแบบ และออกแบบวงจรด้วยภาษา VHDL ผลที่ได้จากซอฟต์แวร์ คัง Block Diagram ในรูปที่ 3.21



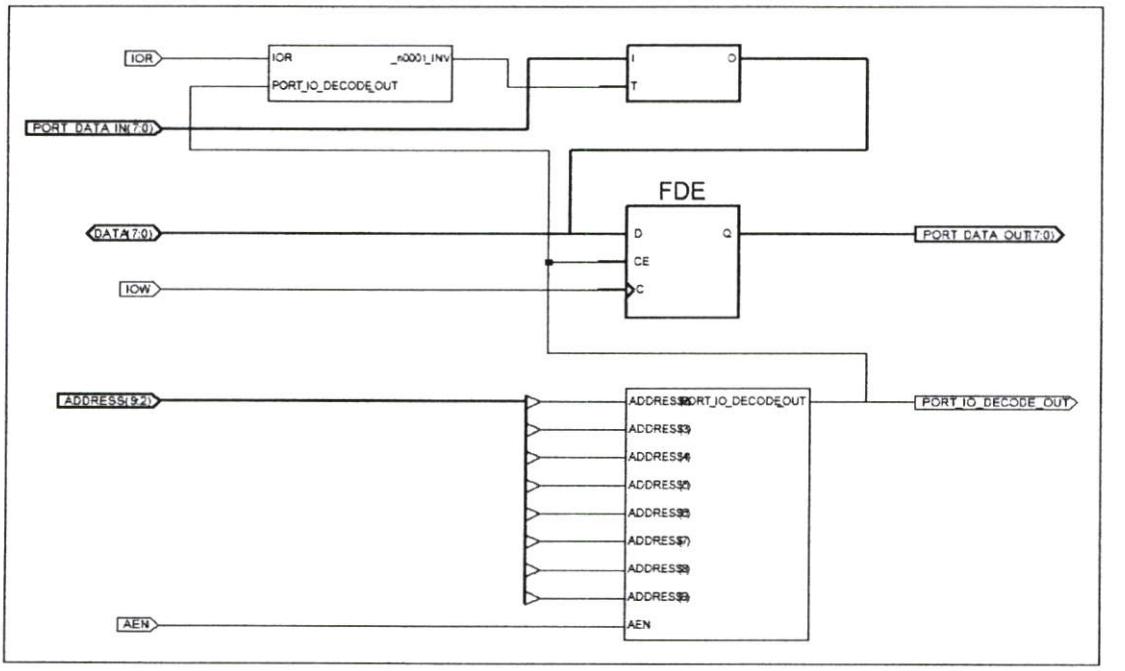
รูปที่ 3.21 Block Diagram ที่ได้จาก ISE WebPACK 8.102i

ใช้ภาษา VHDL เขียนโปรแกรมโดยแบ่งออกเป็น 3 โมดูลหลัก คือ U0 : IOPORT, U1 : Data_en และ U2 : Latch_buf แล้วทำการ port map โมดูลทั้ง 3 เข้าด้วยกัน ได้ RTL Schematic ดังแสดงในรูปที่ 3.22



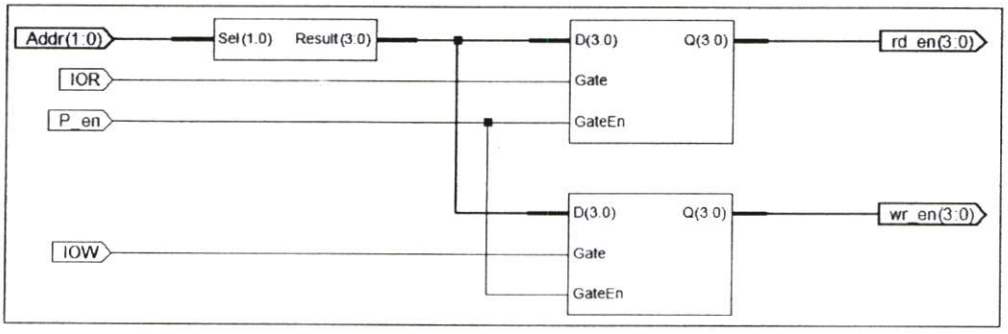
รูปที่ 3.22 RTL Schematic ของ ISA8 (IO Device)

U0 : IOPORT หน้าที่หลักคือควบคุมพอร์ท โดยใช้ ADDRESS(9:2) มาดีโค๊ดหาตำแหน่ง เพื่อสร้างสัญญาณ (PORT_IO_ADDRESS_OUT) และ ควบคุมทิศทางการเคลื่อนที่ของข้อมูลร่วมกับ IOR และ IOW รูปที่ 3.23 แสดง RTL Schematic ของ U0 : IOPORT

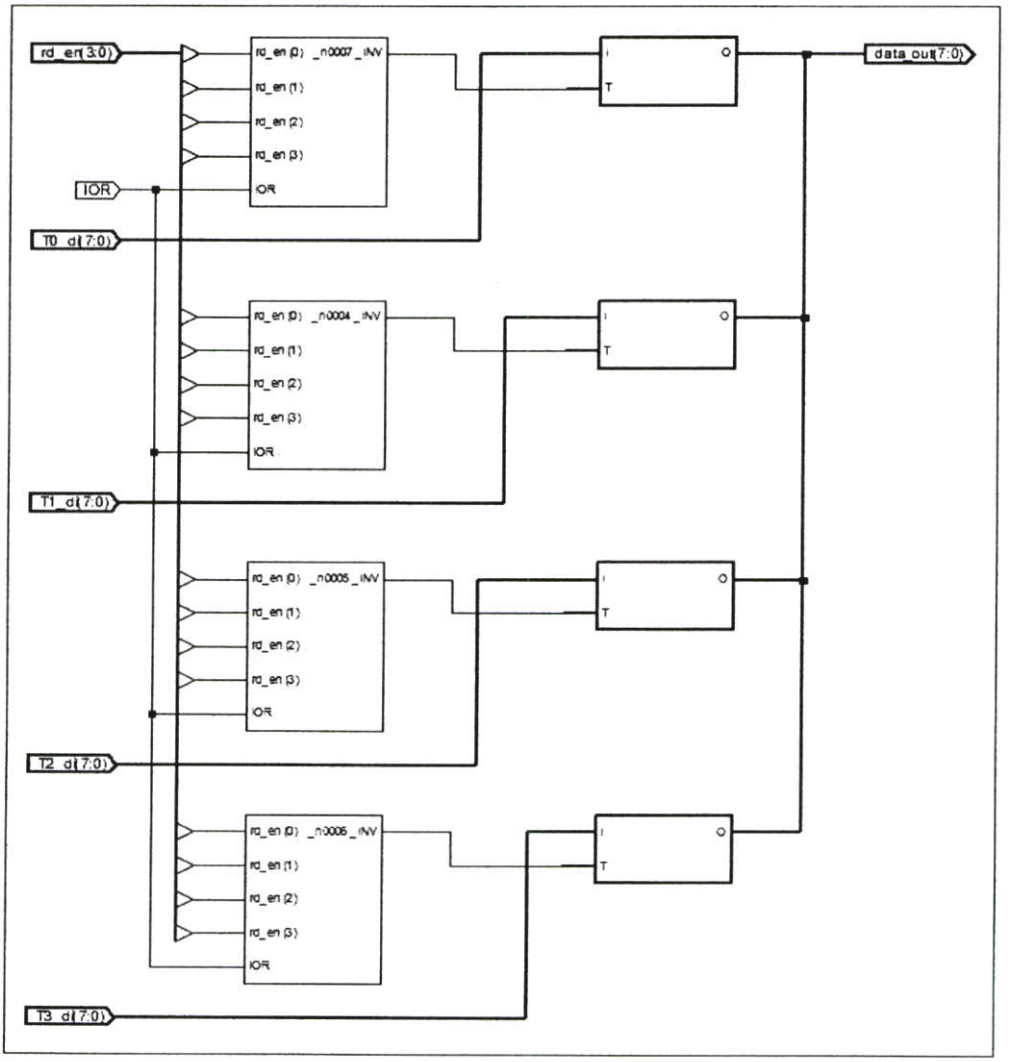


รูปที่ 3.23 RTL Schematic ของ U0 : IOPORT

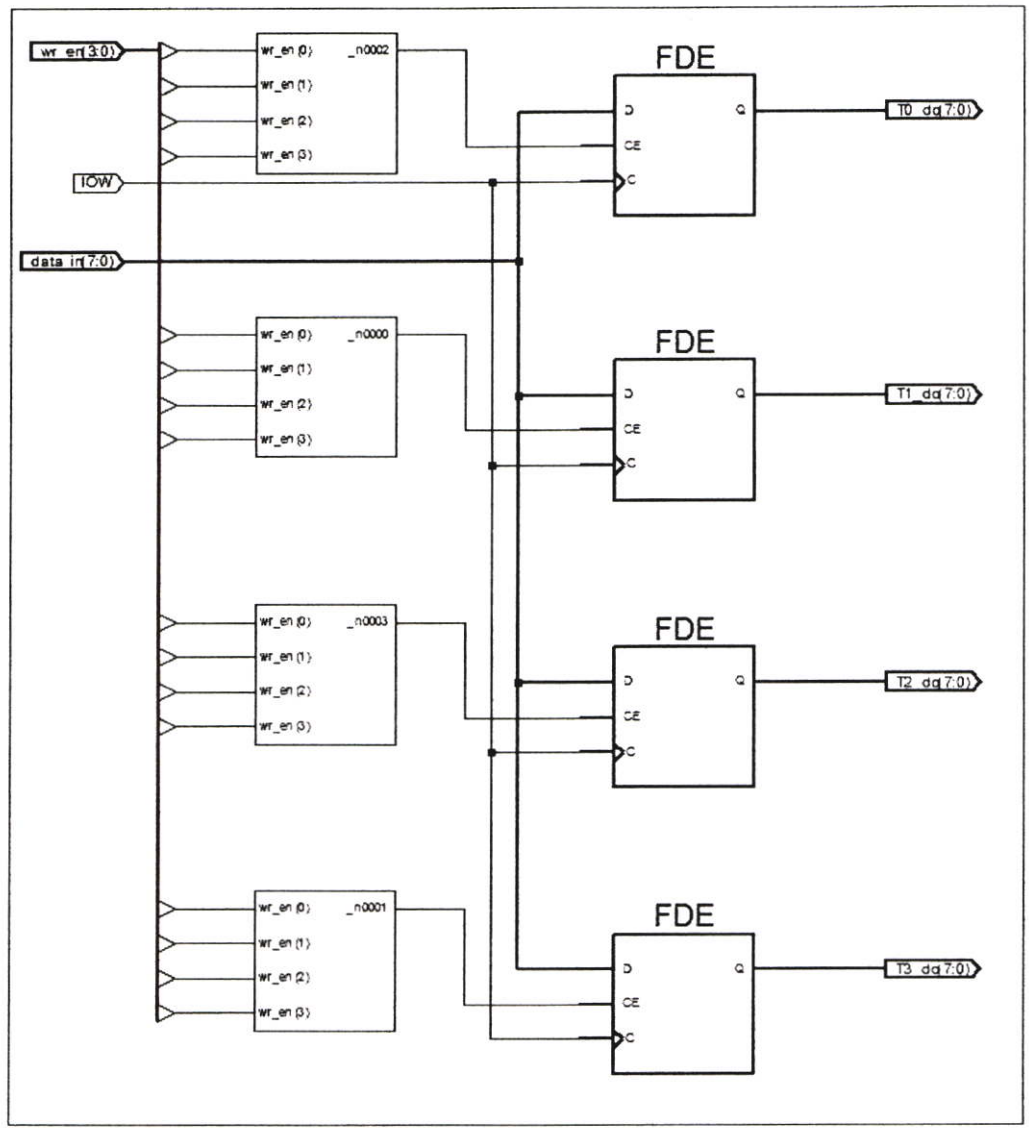
U1 : Data_en หน้าที่หลักคือควบคุมพอร์ททั้งสี่ และทิศทางการเคลื่อนที่ของข้อมูลโดยใช้ ADDRESS(1:0) ดีโค๊ดหาตำแหน่งของพอร์ท ร่วมกับ IOR และ IOW สร้างสัญญาณควบคุมแต่ละพอร์ท [rd_en(3:0) และ wr_en(3:0)] ให้เป็น Input หรือ Output Port รูปที่ 3.24 แสดง RTL Schematic ของ U1 : Data_en



รูปที่ 3.24 RTL Schematic ของ U1 : Data_en



รูปที่ 3.25 RTL Schematic ของ U2 : Latch_buf (Input Port → Data Bus)



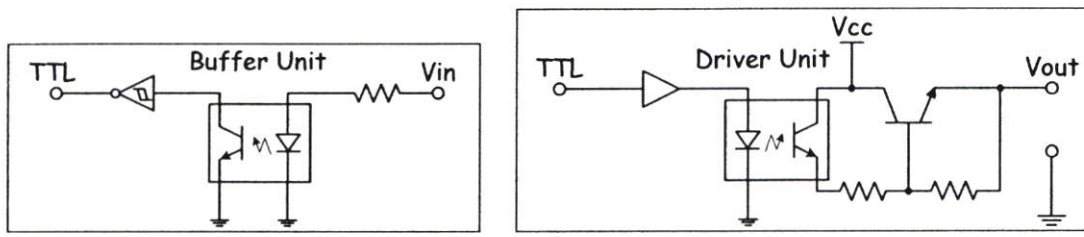
รูปที่ 3.26 RTL Schematic ของ U2 : Latch_buf (Data Bus → Output Port)

U2 : Latch_buf หน้าที่หลักคือ ควบคุมช่องทางผ่านข้อมูลที่จะเข้าหรือออกสู่ Data Bus Data (7:0)

รูปที่ 3.25 ใช้ rd_en(3:0) ร่วมกับ IOR ควบคุมพอร์ทขาเข้า T0_di(7:0), T1_di(7:0), T2_di(7:0) หรือ T3_di(7:0) ยอมให้ข้อมูลจากพอร์ทที่ต้องการเข้าสู่ Data Bus โดยที่ T0_di(7:0), T1_di(7:0), T2_di(7:0) และ T3_di(7:0) ยังทำหน้าที่เป็น Data Buffer ด้วย

รูปที่ 3.26 ใช้ wr_en(3:0) ร่วมกับ IOW ควบคุมพอร์ทขาออก T0_do(7:0), T1_do(7:0), T2_do(7:0) หรือ T3_do(7:0) ยอมให้ข้อมูลจาก Data Bus ออกไปที่พอร์ทที่ต้องการ โดยที่ T0_do(7:0), T1_do(7:0), T2_do(7:0) และ T3_do(7:0) ยังทำหน้าที่เป็น Data Latch ด้วย

เพราะ Chip FPGA ที่ใช้ xc2s50 สามารถรับภาระกระแสได้น้อยไม่เกิน 500 mA ต้องมีอุปกรณ์อื่นมารับภาระในการรับหรือจ่ายกระแสไฟฟ้าแทน และเลือกใช้ Opto Isolate เพื่อป้องกันการรบกวนจากอุปกรณ์ภายนอก วงจรดังแสดงในรูปที่ 3.27



(ก) วงจรส่วน Input (ข) วงจรส่วน Output

รูปที่ 3.27 (ก) วงจรส่วน Input (ข) วงจรส่วน Output

บทที่ 4

การทดลองและผลการทดลอง

จุดประสงค์ของการทดลองเพื่อแสดงว่าโปรแกรมซอฟต์พีแอลซี ที่ออกแบบสามารถทำงานได้ตามวัตถุประสงค์ทั้ง 2 ในบทที่ 1

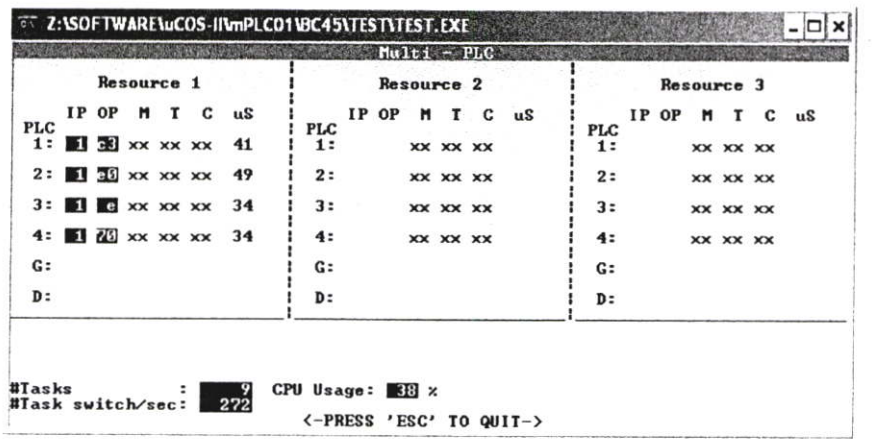
4.1 Testing system methodology

ทำการทดสอบโปรแกรม โดย Execute ทาสก์ ดังต่อไปนี้ :

ตารางที่ 4.1 Task Period Assignment

TaskID	Task Name	Period (ticks) [1 tick=10ms]
0	TaskStart	0
1	TimerTask	1
2	TaskStackDisplay	10
-		
4	PLCTask1	5
5	PLCTask2	5
6	PLCTask3	10
7	PLCTask4	10

เริ่มต้นด้วยการทดสอบเพียง 4 พีแอลซีทาสก์ เพื่อสะดวกในการแสดงให้เห็นว่าระบบสามารถคาดหมายการทำงานได้ (Predictability) และเป็นไปตามที่กำหนดไว้ (Determinism) โดยตั้งคาบการทำงานของแต่ละพีแอลซีทาสก์ไว้ดังแสดงในตาราง 4.1 ทำการบันทึกเวลาที่จุดเริ่มต้นและสิ้นสุดของแต่ละทาสก์ ทำซ้ำ 10 ครั้ง แต่ละครั้งให้ โปรแกรมทำงานเป็นเวลา 10 วินาที เปรียบเทียบกับผลลัพธ์ที่ได้ในแต่ละครั้ง เพื่อดูว่าผลที่ได้จากการทำงานมีเสถียรภาพหรือไม่



รูปที่ 4.1 แสดงการทำงานของซอฟต์พีแอลซี ที่ 4 พีแอลซีทาสก์

4.2 Analysis of results

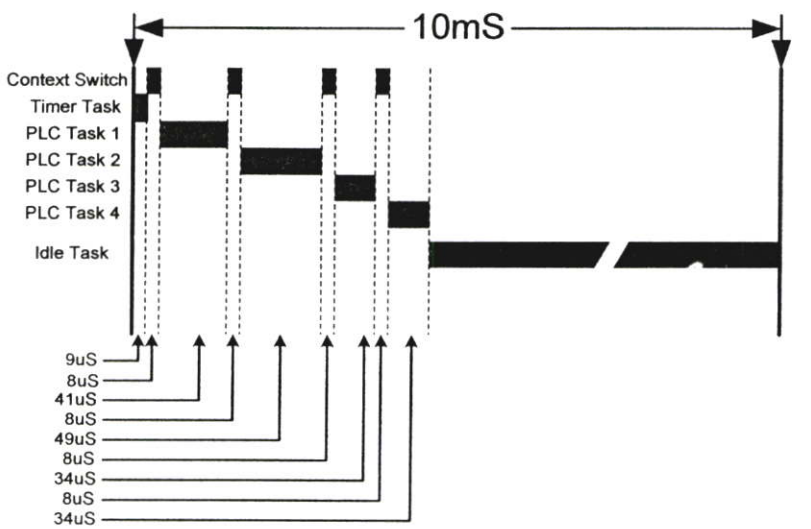
จากตารางที่ 4.2 แสดงให้เห็นว่าการทำงานของโปรแกรมซอฟต์พีแอลซีเป็นไปตามที่คาดหมาย (Predict) ที่ตั้งไว้ในการออกแบบโปรแกรม

หลังจากให้ทำงาน ผลลัพธ์ที่ได้จากการ Execute โปรแกรม พบว่า Execution time ของการทำงานแต่ละครั้งเท่ากัน ในการทำทดลอง 10 ครั้ง

ตารางที่ 4.2 Task Execution Time

Task ID	Status	Time(μS)	Event Description
1	1	8	OS context switch
1	2	9	Timer Task
4	1	8	OS context switch
4	2	41	PLC Task1
5	1	8	OS context switch
5	2	49	PLC Task2
6	1	8	OS context switch
6	2	34	PLC Task3
7	1	8	OS context switch
7	2	34	PLC Task4
Idle		9793	Idle Task
2	1	8	OS context switch
2	2	38	Display Task

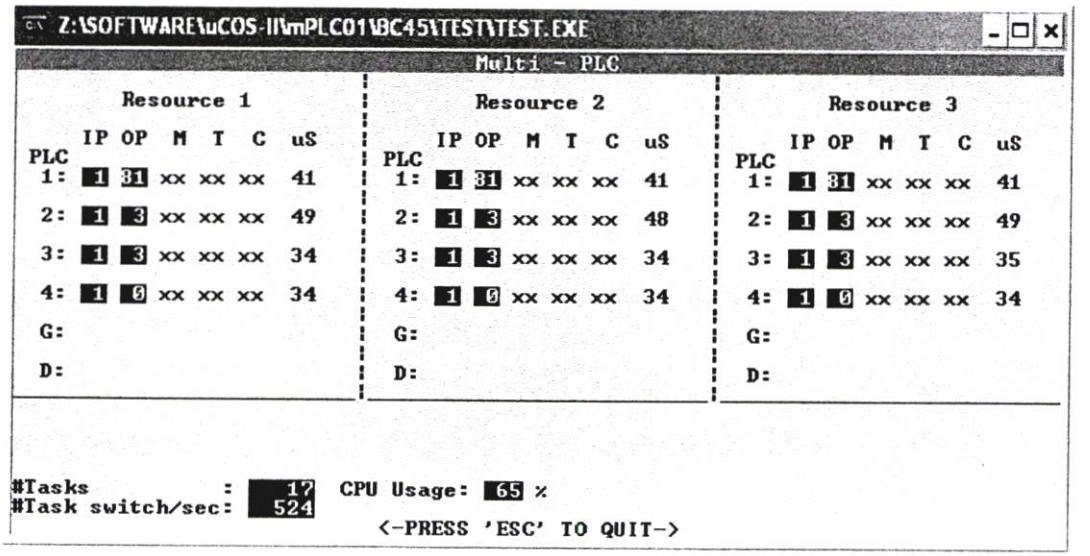
Status: 1=start, 2=stop



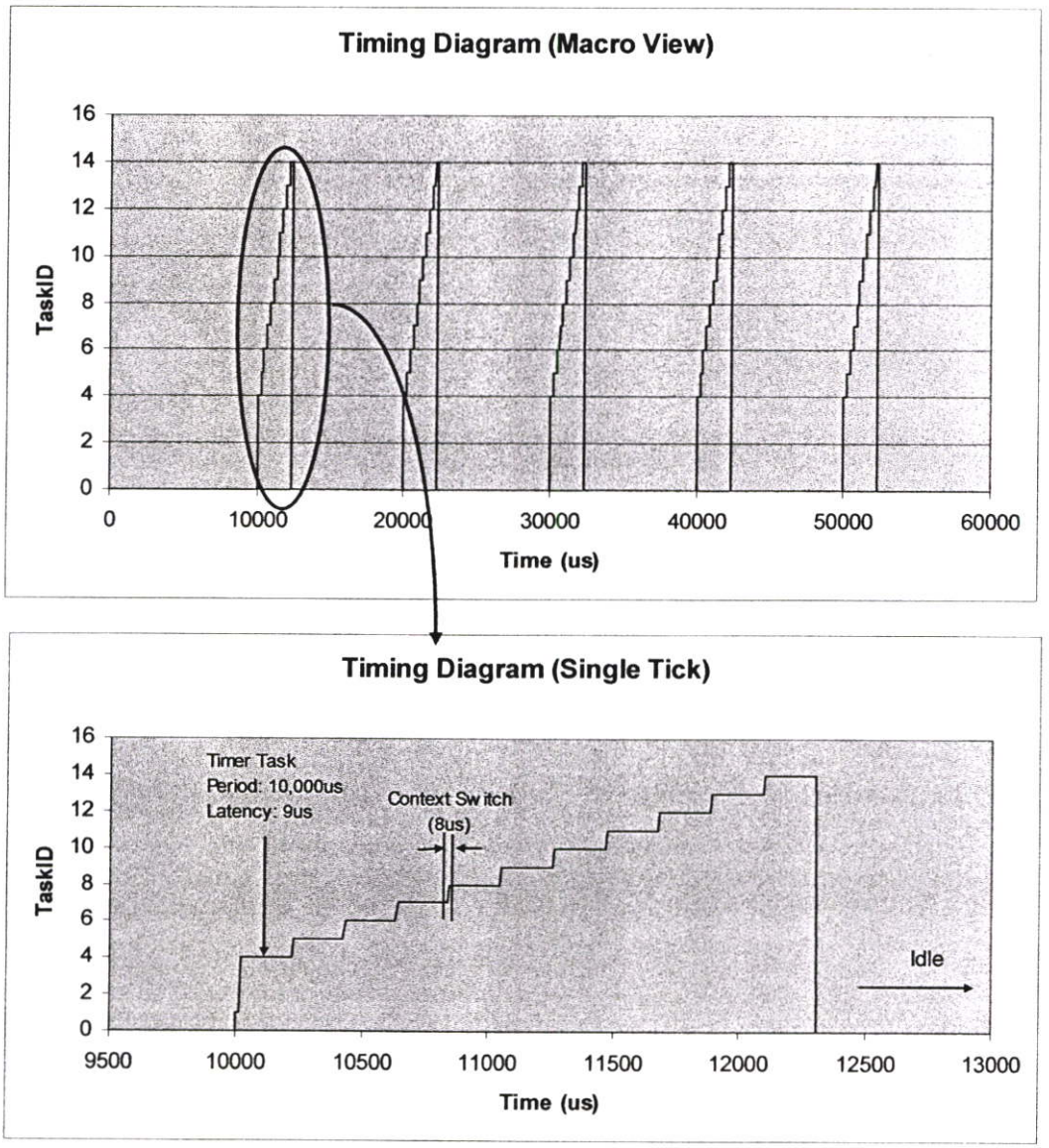
รูปที่ 4.2 Task Execution Timing Diagram

ข้อมูลที่ได้จากตาราง 4.2 นำมาแสดงในรูป Timing Diagram ดังรูปที่ 4.2 แสดงให้เห็นว่าการทำงานของโปรแกรมซอฟต์พีแอลซีเป็นไปตามที่คาดหมายไว้ตอนออกแบบ การทดสอบแสดงว่า OS context switch ใช้เวลาในการ execute คงที่เท่ากับ 8μS ส่วน Timer ทาสก์ ใช้เวลา 9μS และเวลาในการ Execute ของแต่ละพีแอลซีทาสก์แต่ละทาสก์ จะขึ้นอยู่กับจำนวนบรรทัดของคำสั่งในโปรแกรมของทาสก์นั้น ในการทดสอบ พีแอลซีทาสก์1 ถึง พีแอลซีทาสก์ 4 มีจำนวนคำสั่ง 44, 54, 34 และ 34 บรรทัดตามลำดับ เวลาในการ Execute ของพีแอลซีทาสก์แต่ละทาสก์จะคงที่ โดยแปรผันตามความซับซ้อนของโปรแกรม หลังจาก Execute พีแอลซีทาสก์ทุกทาสก์เสร็จแล้ว ซอฟต์พีแอลซีจะเปลี่ยนไปทำงานที่ Idle ทาสก์ เป็นเวลา 9793μS การทำงานของแต่ละ Task ที่ปรากฏเป็นไปตามความคาดหมาย แสดงให้เห็นถึงความถูกต้องของ Scheduling policy และการ Implement

ขั้นตอนต่อมาทำการทดลองเต็มจำนวนทาสก์ คือ 12 พีแอลซีทาสก์ โดยให้ทำงานที่โหลดสูง จำนวนคำสั่ง 400 บรรทัด/ทาสก์ แต่ละพีแอลซีทาสก์มีคาบการทำงาน 1 Clock tick ซึ่งพีแอลซีทาสก์ โดยปกติ อาจจะมีคาบการทำงานทุก ๆ 5 -10 หรือ100 Clock ticks ขึ้นอยู่กับการขนาดของโปรแกรม



รูปที่ 4.3 แสดงการทำงานของซอฟต์พีแอลซี ที่ 12 พีแอลซีทาสก์



รูปที่ 4.3 Timing Diagram ของการทำงานที่ High load

จากรูปที่ 4.3 แสดงสภาพขณะที่โหลดสูง ระบบยังสามารถทำงานได้อย่างเชื่อถือได้ และเป็นไปตามที่คาดหมาย เครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบมีรายละเอียดดังนี้

- Intel Pentium M processor 740, 1.73GHz, 533 MHz FSB, 2 MB L2 Cache, 512 MB DDR2 (support dual-channel) และ
- โดยวัดที่ 400 คำสั่งต่อทาสก์ ตรวจจับเวลาการ execute รวมการเปลี่ยนบริบท (Context switch) จะใช้เวลา = 12305-10000 = 2305 μS

ในกรณีโหลดสูงให้ทั้ง 12 พีแอลซีทาสก์ ทำงานทั้งหมดบนโปรแกรมซอฟต์พีแอลซี และทำการ execute ทุกๆ Clock tick จะได้ CPU utilization = 2305/10000 = .2305 = 23.05%

ผลที่ได้จากการคำนวณใช้เพียง 23.05% ของ CPU utilization

บทที่ 5
สรุปผลการวิจัย และข้อเสนอแนะ

5.1 สรุปผลการวิจัย

ในงานวิจัยนี้ได้นำเสอโปรแกรม "ซอฟต์พีแอลซี" โดยทำงานภายใต้ระบบปฏิบัติการเรียวไทม์ ซึ่งระบบปฏิบัติการเรียวไทม์เป็นระบบปฏิบัติการที่มีความสามารถในการทำงานแบบมัลติทาสก์ได้ดี ในงานวิจัยนี้เลือกระบบปฏิบัติการ μCOS-II ซึ่งเป็นระบบปฏิบัติเรียวไทม์ที่มี Footprint ขนาดเล็ก สมรรถนะภาพสูง คาดหมายการทำงานได้ และสามารถนำพอร์ทลงบนไมโครโปรเซสเซอร์หรือไมโครคอนโทรลเลอร์ได้หลายตระกูล เพื่อให้โปรแกรมซอฟต์พีแอลซีสามารถทำงานได้ดี ทั้งบนคอมพิวเตอร์พีซี หรือคอนโทรลเลอร์บอร์ด ได้อย่างมีเสถียรภาพ และเป็นไปตามความคาดหมาย (Predictability) ในการออกแบบได้ทำตามข้อแนะนำของ IEC ซึ่งเป็นองค์กรที่ทำหน้าที่กำหนดมาตรฐานของพีแอลซี โดยออกแบบให้มีซอฟต์แวร์โมเดลและรูปแบบภาษาตามมาตรฐานของ IEC 61131-3 เมื่อนำโปรแกรม ซอฟต์พีแอลซี ไปทดสอบการทำงาน โดยเริ่มจาก ทดลองที่ 4 พีแอลซีทาสก์ ทำการทดลอง 10 ครั้ง ครั้งละ 10 วินาที บันทึกเวลาที่จุดเริ่มต้นและสิ้นสุดของแต่ละทาสก์ ทุกครั้ง ผลลัพธ์ที่ได้แสดงถึงความมีเสถียรภาพของระบบ และเป็นไปตามที่คาดหมาย ต่อมาทดลองทั้ง 12 พีแอลซีทาสก์ โดยให้ทำงานที่โหลดสูง (400 line instruction / Task) ทุกทาสก์ เมื่อนำผลที่มาคำนวณหา CPU utilization = 23.05% ซึ่งแสดงถึงสมรรถภาพของโปรแกรมซอฟต์พีแอลซี สามารถรองรับการการทำงานแบบมัลติทาสก์ได้

นอกจากนี้รูปแบบของซอฟต์แวร์โมเดลก็เป็นไปตามที่ IEC 61131-3 ให้ Guideline ไว้แสดงการ Implement บางส่วนของ IEC 61131-3 และภาษา IL หนึ่งในห้าภาษามาตรฐานตามที่ IEC กำหนดไว้ ทำโปรแกรมที่พัฒนาโดยภาษา IL มาตรฐานเดียวกัน สามารถนำมาใช้งานทดแทนกันได้

5.2 ข้อเสนอแนะ

ความสามารถของ โปรแกรมซอฟต์พีแอลซี ในการทำงานร่วมกัน การใช้ทรัพยากรร่วมกัน ระหว่าง พีแอลซีทาสก์ ยังมีข้อจำกัด ทำงานได้ในระดับพื้นฐาน ควรมีการพัฒนาให้มีความสามารถมากขึ้น เพื่อให้รองสามารถรับงานที่มีความซับซ้อนมากขึ้นได้

โปรแกรมซอฟต์พีแอลซี สามารถรองรับภาษามาตรฐานได้เพียงหนึ่งภาษา คือ ภาษา IL (Instruction List) ซึ่งเป็นภาษายุคเริ่มต้นโครงสร้างของภาษาไม่ซับซ้อน ไม่ได้ถูกออกแบบให้สามารถรองรับงานที่มีความซับซ้อนมาก ๆ ได้ ควรมีการพัฒนาให้รองรับภาษามาตรฐานทั้ง 5 ภาษา ตามมาตรฐาน IEC 61131-3 กำหนดไว้ เพื่ออำนวยความสะดวกแก่ผู้ใช้

เอกสารอ้างอิง

[1] Dedicate-System, "Definitions",
http://www.realtime-info.be/encyc/publications/faq/ rtfaq.htm#realtime_definition.

[2] Dewi Jones, "Slides on RTOS", MTOS_slides.PDF, http://www.uCOS-II.com.

[3] Dinker Charak, "Real-Time (Realtime) Systems." [Online].
Available: http://sarovar.org/projects/bakar/. 2004.

[4] Engineer Manual IEC 61131-3 programming,
International Electrotechnical Committee. 1999.

[5] Hugh Jack, **Automating Manufacturing Systems with PLCs**,
Berlin Heidelberg : Springer-Verlag Company, Inc. 2001.

[6] Jianghai Huang, Yu-Cheng Li, Wi Luo, Xiao-Xia Liu, Kui-Feng Nan.
"THE DESIGN OF NEW-TYPE PLC BASED ON IEC61131-3",
The Second International Conference on Machine Learning and Cybernetics,
Xi'an, November 2003. pp. 809-813

[7] Karl-Heinz John, Michael Tiegelkamp, **IEC61131-3 Programming Industrial Automation Systems**, 2000.

[8] Labrosse Jean J. **Embedded Systems Building Blocks, Complete and Ready-to-Use Modules in C**, 2nd ed., Lawrence, Kansas : R&D Technical Books, 1999.

[9] Labrosse Jean J. **µC/OS-II, The Real Time Kernel**, 2nd ed., Lawrence, Kansas :
R&D Technical Books, 2002.

[10] Labrosse Jean J. **The Real Time Kernel**, Lawrence, Kansas : R&D Technical Books, 2000.

[11] Philip Melanson, Siamak Tafazoli, "A Selection Methodology for the RTOS Market",
© Canadian Space Agency 2003. Data Systems in Aerospace (DASIA 2003) conference, Prague, Czech Republic, June 2003.

[12] PLCopen. "IEC 61131-3 Programming Languages - providing the basis." [Online].
Available: http://plcopen.org/. 1998.

[13] PLCopen. "IEC 61131-3 Software" [Online].
Available: http://plcopen.org/. 1998.

[14] Qing Li , Caroline Yao. **Real-Time Concepts for Embedded System**,
San Francisco : CMP Books. 2003.

[15] S. Baskiyar, Ph.D., N. Meghanathan. "A Survey of Contemporary Real-time Operating Systems", **Informatica 29**, 2005. pp 233–240

ภาคผนวก

ผลงานวิจัยที่ได้รับการตีพิมพ์

1. Panu Vacharanarumol & Apinetr Unakul., "Design and Implementation Modern PLC.", The 4th International Joint Conference on Computer Science and Software Engineering (JSSE2007)., pp 137 - 141, Khon Kaen, Thailand, May 2-4, 2007.


The 4th International Joint Conference
on Computer Science
and Software Engineering (JCSSE2007)
JCSSE
May 2nd - 4th, 2007
Department of Computer Engineering, Khon Kaen University
Khon Kaen, THAILAND

International Joint Conference on Computer Science & Software Engineering (JCSSE2007)
May 2-4, 2007

Design and Implementation Modern PLC

Panu Vacharanarumol & Apinetr Unakul
Computer Engineering Department, Faculty of Engineering,
King Mongkut's Institute of Technology Ladkrabang
Chalongkrung Rd., Ladkrabang Bangkok 10520 Thailand
Tel: +665-551-698 Email: panu_vach58@yahoo.co.th, apinetr@yahoo.com

Abstract

The current challenge of conventional PLC is to allow for higher performance and greater efficiency in controlling more complex and numerous tasks, especially for users in factory automation. This paper proposes the design of a modern PLC program, code name Multi-PLC, using partial implementation of IEC 61131-3. It presents the results of an experiment to test the performance and correctness of the program execution. The test results show that Multi-PLC program functions in real-time with multi-tasking capacity and program-level portability. It solves the limitations of conventional, single-process PLC and demonstrates the potential of a new generation of PLC.

Key-words: PLC, programmable logic controller, Multi-PLC, real-time, multi-task, IEC 61131, RTOS

I. INTRODUCTION

PLC (Programmable Logic Controller) is widely used in controlling automation in factories. Because of its versatility and flexibility, it can be applied to many types control of applications. [1]

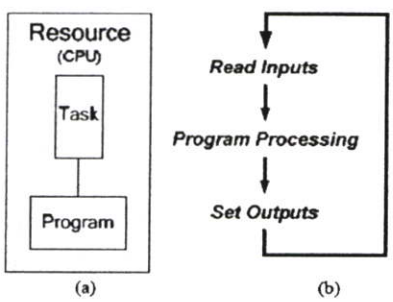


Figure 1a & b. Conventional PLC program structure

Figure 1a shows the structure of PLC in general, which includes (i) resource (CPU), (ii) task which controls the execution of the program, and (iii) program, which is written in one of the languages in the IEC standard. Figure 1b shows the execution cycle, which starts from reading the input, program processing from the first instruction to the final instruction and outputs the result. It repeats as an endless loop. [8]

A. Justification

First, the requirements of the control functions in modern factories have become increasingly more complex. After considering the structure and characteristic of the functioning of PLC, it is clear that when using only single process program to control the work, the program will be difficult to write. The program will be complicated and big. Some kind of work cannot be done, for instance the control of multiple robot arms because of the limitations of conventional PLCs.

In order to control this complex work, Multi-PLC can achieve a much better programmability and functionality than that conventional single PLC.

Second, users need a programming language and implementation for the PLC which is independent from the producer of the PLC. Because each company which produces PLC develops its own variation of the programming language. Even though the same language is used, there are some differences, such as configuration, addressing method, structures of grammar. [3] This makes it impossible to directly use the same program to control PLC which is produced by another company. It is necessary to re-format the program according to the specifications of the manufacturer. This wastes time in developing programmer for PLCs manufactured by each company [8] and also has effect on the teaching of PLC in educational institutions.

In 1999, the International Electrotechnical Commission (IEC) fabricated IEC 61131, which includes IEC 61131-3. This establishes language standards for manufacturers of PLC and directly affects users because it allows users to write programs which can be used by PLCs produced by any manufacturer, without the need to re-format. [2][7]

B. Objective

From the conventional PLC which uses single process, in this paper, we will propose to design a more effective Multi-PLC device using real-time programming techniques which can function in multi-tasking. The Multi-PLC device should have the following characteristics:

(i) Implementation of partial IEC 61131-3 with program-level portability;
(ii) Limited language support, namely Ladder Diagram (LD) and Instruction List (IL).

The program which we will design will accomplish the following:

(i) Deliver high performance Multi-PLC on PC with support up to 12 PLC tasks;
(ii) Demonstrate correctness of the multi-tasking functions.

C. Natures of Multi-PLC device

Real-time
Multi-PLC should function in real-time because in factory applications, the automation must be controlled in real-time. Multi-PLC must be designed so that it can switch from one task to another task in the time required. For conventional PLC, only one task is being controlled. However, for Multi-PLC, it is more challenging to make it real-time due to the increased number of tasks being controlled simultaneously.

Real time in operating systems (RTOS) means the ability of the operating system to provide a required level of service in a bounded response time.[7] Micro-Controller Operating System Version 2 (μC/OS –II) is the real time kernel library. It runs on many microprocessors, including Atmel AVR series and ix86 chips, and supports third-party TCP/IP. [4][5]

Multi-tasking
Because Multi-PLC needs to control multiple tasks, there are many ways to manage multi-tasking. It is important to select the most appropriate way in order to allow all the tasks to be function properly.

Resource sharing
It is challenging to determine between global resources and resources which are distributed for each task. Resource is limited, so they must be shared. In multi-tasking environment, we must use synchronization techniques to ensure mutual exclusion access to shared data.

Complex configuration
To determine structure of each task, we need to configure the system so that each PLC task can use resources according to the different needs of the application. We need to configure the system to determine the use of resources, the use of global variable and map input-output port.

D. Adopting IEC 61131-3

Overview of IEC 61131
IEC 61131 has five parts: (i) Overview, (ii) Requirements and Test Procedures, (iii) Data Types and Programming, (iv) User Guidelines, (v) Communications. For this paper, we are interested and will focus on Part 3, which has the greatest impact on the user [1] [6].

IEC 61131-3
IEC 61131-3 is divided into two sections: (i) common elements and (ii) programming languages. Common elements includes software model, which includes configuration, resources and tasks. For programming languages, this includes 5 languages: (i) Instruction List (IL), (ii) Structured Text (ST), (iii) Ladder Diagram (LD), (iv) Function Block Diagram (FBD), and (v) Sequential Function Chart (SFC). For each language, IEC 61131-3 standardizes and enforces single language definition. [1][6] This allows for programming -level portability. In our paper, we will demonstrate the implementation of IL as a starting point.

IEC 6113-3 is the wave of the future. It will solve many of the industry control and instrumentation problems. Full implementation of IEC 6113-3 will allow for 3 levels of standard compatibility: (i) program level compatibility, (ii) application level compatibility, and (iii) configuration level compatibility. In our paper, we will try to create only a subset of the program level compatibility.

II. MULTI-PLC DESIGN PROPOSAL

A. System architecture

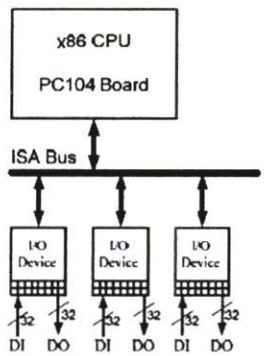


Figure 2. System Architecture Diagram

The Multi-PLC is implemented on PC104 Board with Pentium M 1.73 GHz and is connected to three I/O boards through ISA Bus. Each I/O board has 32 digital inputs and 32 digital outputs.

B. Software model

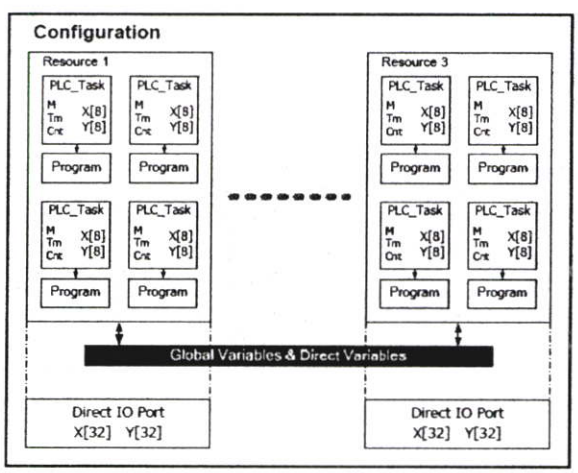


Figure 3. Multi-PLC Software Model

The software model follows IEC 61131-3, with a specific configuration, as follows. Our implementation design fixes that in one configuration system contains global variables and three resources. Each resource contains four tasks, local variables (timer, counter and auxiliary relay), and program and is linked to direct IO port.

C. Program structure

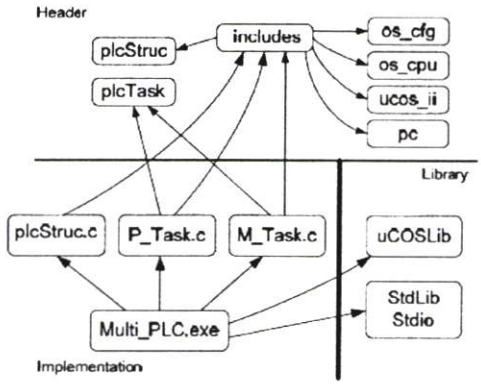


Figure 4. Component diagram

To implement real-time functions, we use uCOS-II RTOS. The Multi_PLC.exe is the executing file, which depends on P_Task.c and M_Task.c. M_Task.c defines all the task decoration. P_Task.c defines the IL interpreter. The plcStruc is intended to be user definable program file which defines the system configuration which includes the number of resources within the configuration and the number of tasks within each resource.

D. Data structure

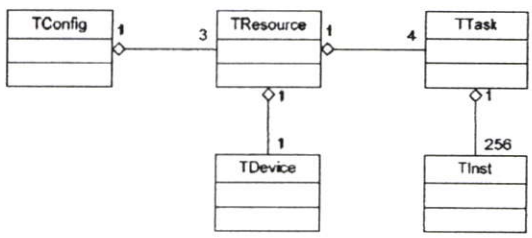


Figure 5. Class diagram of the software model

This is the class diagram representing C data structure of Multi-PLC IEC configuration.

E. Task management and scheduling policy

The TimerTask is a periodic task, waking up every 10 milliseconds and it determines which PLC task is allowed to be executed during the given time period (known as ticks). Depending on which PLC task each scheduled for that tick, it will send resume signal to the scheduled tasks. After sending the signal, TimerTask will suspend itself. Then the PLC tasks will execute in order, one by one. The scheduling policy for the PLC Tasks is prioritized first come first serve. After each task executes, it will suspend itself. When all scheduled PLC tasks have been executed for that tick, the system will invoke the idle task. When the next tick starts, TimerTask will be triggered again. The cycle is then repeated.

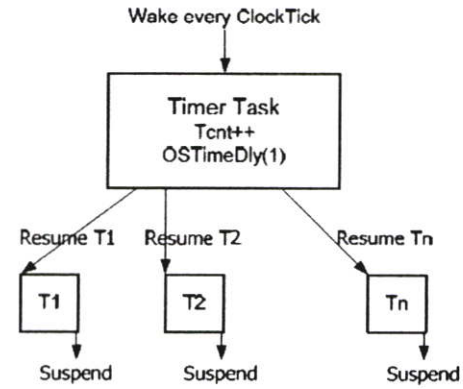


Figure 6. Task control flows

We use TimerTask as the main task controlling the execution of all the PLC tasks. The execution schedule for each PLC task can be different. For instance, certain PLC tasks will occur once every three ticks, or once every four ticks.

For Multi-PLC, it is possible to assign different priority levels to different tasks. From the Task Control Block below, this shows the current default configuration wherein Task 4-15 corresponds to PLC 1-12. As indicated below, Task 4-11 has the same priority level, and is higher than the priority of Task 12-15. Task 1 is the timer task and has the highest priority. The last task is the idle task, which always has the lowest priority. We can modify to have each task at a different priority, by adjusting its position. In this case, the top priority level and bottom two priority levels are reserved. This leaves the middle five priority levels open for task assignment by user.

Task Control Block (TCB)

								Priority
0	1	x	x	x	x	x	x	High
4	5	6	7	8	9	10	11	
12	13	14	15	x	x	x	x	
x	x	x	x	x	x	x	x	
x	x	x	x	x	x	x	x	
x	x	x	x	x	x	x	x	
2	3	x	x	x	x	x	x	
x	x	x	x	x	x	x	ID	Low

Figure 7. Task Control Block (TCB)

In some cases, certain tasks take longer than 1 tick to execute. It is recommended to assign lower priority for these tasks. This will allow the fast-executing tasks to have higher priority and be executed first. At the same time, tasks which have a tight deadline should also have higher priority. In the above example, Tasks 2 and 3 are status display tasks, and have been assigned low priority.

IL Interpreter

IL Interpreter will read the program from the program file and will keep in the memory. When the IL Interpreter task wakes up, it will read in all the inputs, refresh the timers and counters, and read each instruction one by one, check what the instruction says to do, and will then execute each one. It will continue this until the final instruction, update the outputs, and finally, it will suspend itself.

III. EXPERIMENT

A. Purpose

The purpose of the experiment is to demonstrate that Multi-PLC program which we designed can function in accordance with the two objectives in Section 1.2.

B. Testing system methodology

We tested the program by executing the following tasks:

TABLE 1. TASK PERIOD ASSIGNMENT

TaskID	Task Name	Period (ticks) [1 tick=10ms]
0	TaskStart	0
1	TimerTask	1
2	TaskStackDisplay	10
-		
4	PLCTask1	5
5	PLCTask2	5
6	PLCTask3	10
7	PLCTask4	10

We tested with only four PLC Tasks for simplicity of showing system predictability and determinism. We recorded the start and end time for each task. We ran the program ten times. Each time we launched the program, we recorded the results for 10 seconds. We compared the results of each run to see if the results are stable each time.

We also tested implementing all 12 PLC tasks each running at a high load capacity of 400 IL instructions/task. A 400 IL instructions program is considered medium to large PLC program size. Each task is scheduled to execute at every clock tick. A typical PLC task may only execute once every 5-10 or even 100 clock ticks depending on the application.

C. Analysis of results

The results showed that the program execution is highly predictable.

After analyzing three runs, the following table shows the execution time for each task. We found that the execution time for each task is stable during the ten runs.

TABLE 2. TASK EXECUTION TIME

Task ID	Status	Time(μS)	Event Description
1	1	8	OS context switch
1	2	9	Timer Task
4	1	8	OS context switch
4	2	41	PLC Task1
5	1	8	OS context switch
5	2	49	PLC Task2
6	1	8	OS context switch
6	2	34	PLC Task3
7	1	8	OS context switch
7	2	34	PLC Task4
Idle		9793	Idle Task
2	1	8	OS context switch
2	2	38	Display Task

Status: 1=start, 2=stop

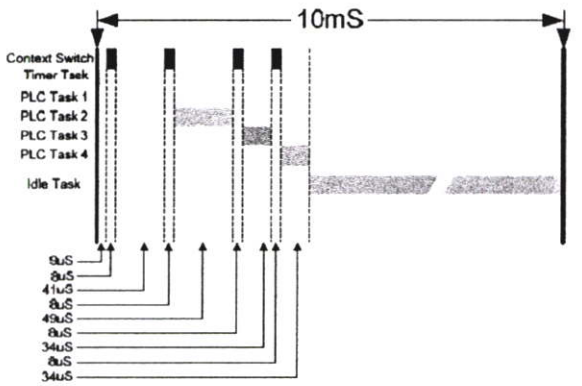


Figure 8. Task Execution Timing Diagram

This data shows that the function of Multi-PLC program is deterministic and in accordance to the program design. The tests show that OS context switch takes 8uS to execute consistently. The timer task takes 9uS. The execution time of each PLC task depends on the number of lines in each PLC program, where PLC Task1 to Task4 has 44, 54, 34 and 34 lines of code, respectively. The execution times for the PLC tasks are consistent and proportional to the complexity of the task. After the execution of all the tasks, the PLC goes into the Idle task for 9793uS. All the events and execution of all tasks occurred in an expected manner. This shows the correctness of the scheduling policy and implementation.

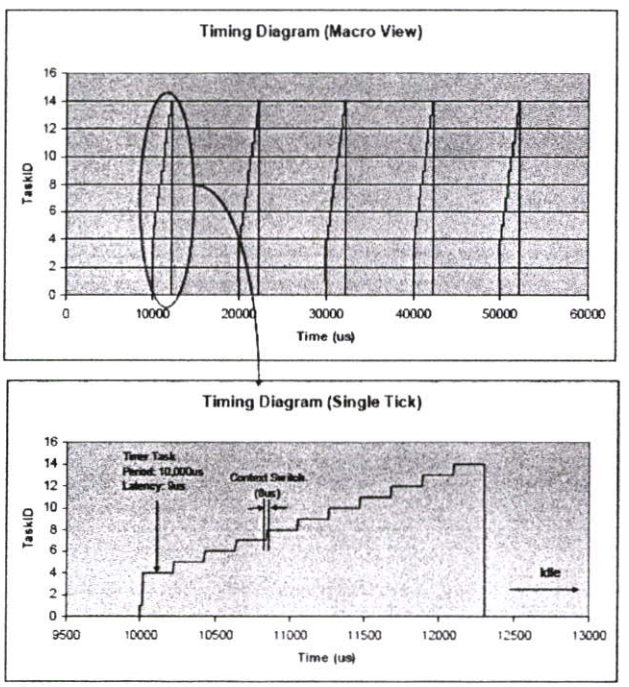


Figure 9. Timing Diagram running at high load

Figure 9 shows that even at a high load, our system still performs reliably as expected. Our system specification is: Intel Pentium M processor 740, 1.73GHz, 533 MHz FSB, 2

MB L2 Cache, 512 MB DDR2 (support dual-channel). On our system, we measured that a 400 instruction task including context switching will take 220uS. In an extreme case where all 12 tasks running concurrently on our Multi-PLC and executing during every tick, the computation takes up only 24.4% of the CPU utilization.

IV. CONCLUSION

This paper describes the design and testing of Multi-PLC device showing partial implementation of IEC 61131-3. It uses real-time programming techniques which can function in multi-process. We have shown that Multi-PLC has the capacity to effectively control multiple tasks effectively and accurately. This addresses the two main problems concerning PLCs used to control automation in the factory setting: meeting complex user needs and adopting a standard language. Finally, because it is more powerful than conventional PLC, it is an excellent learning platform in Thai universities and technical colleges in the area of control.

REFERENCES

[1] Hugh Jack, *Automating Manufacturing Systems with PLCs*, 2005.

[2] International Electrotechnical Committee, *Engineer Manual IEC 61131-3 programming*, 1999.

[3] Jianghai Huang, Yu-Cheng Li, Wi Luo, Xiao-Xia Liu, Kui-Feng Nan, "THE DESIGN OF NEW-TYPE PLC BASED ON IEC61131-3," *Proceeding of the Second International Conference on Machine Learning and Cybernetics, Xi'an*, pp. 809-813, November 2003.

[4] Jean J. Labrosse, *MicroC/OS-II : The Real Time Kernel*, 2nd ed., Cmp Books 2002.

[5] Jean J. Labrosse, *Embedded Systems Building Blocks Complete and Ready-to-Use Modules in C*, 2nd Ed., CMP Books 1999.

[6] Karl-Heinz John, Michael Tiegelkamp, IEC61131-3 Programming Industrial Automation Systems, 2000.

[7] Dedicate-System, "*Definitions*" http://www.realtime-info.be/encyc/publications/faq/rtfaq.htm#realtime_definition

[8] PLCopen, "*IEC 61131-3 Programming Languages - providing the basis*", http://plcopen.org/

ประวัติผู้เขียน

ชื่อ - นามสกุล	นายภานุ วัชระนฤมล
วัน เดือน ปีเกิด	14 มกราคม พ.ศ.2501 จ.นครราชสีมา
ที่อยู่	เลขที่ 2/37 หมู่ที่ 1 ต.น้ำรึม อ.เมืองตาก จ.ตาก
ประวัติการศึกษา	2520 ประกาศนียบัตรวิชาชีพชั้นสูง สาขาอิเล็กทรอนิกส์ วิทยาลัยเทคโนโลยีและอาชีวศึกษา วิทยาเขตตาก จ.ตาก 2522 คุรุศาสตร์อุตสาหกรรมบัณฑิต ไฟฟ้า (ไฟฟ้า-สื่อสาร) วิทยาลัยเทคโนโลยีและอาชีวศึกษา 2544 วิศวกรรมศาสตรบัณฑิต วิศวกรรมไฟฟ้า (อิเล็กทรอนิกส์) คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีราชมงคล
ประวัติการทำงาน	พ.ศ.2523 – ปัจจุบัน อาจารย์ประจำคณะวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ตาก จ.ตาก