

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การพัฒนาระบบสมองกลฝังตัวแบบทันเวลาบนเอฟพีจีเอ

โดยใช้ไมโครเบลสและไมโครซีไอเอสทูวี

Development of Embedded Real-time System on FPGA

Using MicroBlaze and MicroC/OS-II



ปริญญาานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปริญญาานิพนธ์ปีการศึกษา 2550

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาระบบสมองกลฝังตัวแบบทันเวลาบนเอฟพีจีเอ โดยใช้ไมโครเบลสและไมโครซีไอเอสทูว

Development of Embedded Real-time System on FPGA Using MicroBlaze and MicroC/OS-II

ผู้จัดทำ

1. นายพิณพงศ์ ศิลป์สกุลสุข

รหัสนักศึกษา 47010517



 อาจารย์ที่ปรึกษา
(ผู้ช่วยศาสตราจารย์อภิเนตร อุนากุล)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การพัฒนาระบบสมองกลฝังตัวแบบทันเวลาบนเอฟพีจีเอ

โดยใช้ไมโครเบลสและไมโครซีไอเอสทูว์

นายพิณพงศ์ ศิลป์สกุลสุข 47010517

ผศ.อภิเนตร อุนากุล อาจารย์ที่ปรึกษา

ปีการศึกษา 2550

บทคัดย่อ

ในปัจจุบันนี้ฮาร์ดแวร์ที่มีความยืดหยุ่นและสามารถทำการ โปรแกรมได้นั้นเริ่มเข้ามามีบทบาทสำคัญในการพัฒนาระบบสมองกลฝังตัวค่อนข้างมาก แต่อย่างไรก็ตาม ข้อมูลต่างๆ ที่จำเป็นต่อการเรียนรู้ นั้นยังมีค่อนข้างที่จะจำกัดซึ่งทำให้การศึกษานั้นเป็นไปได้ค่อนข้างที่จะลำบาก ในโครงการนี้เราได้ทำการ นำเสนอแนวทางในการสร้างระบบร่วมระหว่างซอฟต์แวร์โปรเซสเซอร์คอร์ ไมโครเบลส กับเรียลไทม์ไอเอส ไมโครซีไอเอสทูว์ บนเอฟพีจีเอ ซึ่งเราได้ทำการเรียบเรียงเอกสารสำหรับนำไปศึกษา และทำการพัฒนาไลบรารีพื้นฐานต่างๆ ไว้บางส่วน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Development of Embedded Real-time System on FPGA

Using MicroBlaze and MicroC/OS-II

Mr. Pinnapong Silpsakulsuk 47010517

Asst. Prof. Apinetr Unakul Advisor

Academic Year 2007

ABSTRACT

In recent years, agile and reprogrammable hardware has become an important requirements in embedded system development. However, the there are limited resources with steep learning curve. In this project we introduced an integrated implementation of highly agile solution on FPGA with soft processor core using Xilinx MicroBlaze and with RTOS using uC/OS-II. We have developed relevant reference materials, user manual, and basic component library.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติกรรมประกาศ

ปริญญาบัตรฉบับนี้คงไม่อาจสำเร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลายๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึง เพราะเป็นส่วนสำคัญที่ทำให้ปริญญาบัตรเสร็จลงได้ก็คือ ผศ. อภิเนตร อุณาภูล ซึ่งเป็นที่ปรึกษาปริญญาบัตร ที่ให้ความเอาใจใส่ แนะนำความรู้และความช่วยเหลือในทุกๆ ด้านเสมอมา ซึ่งต้องขอกราบขอบพระคุณเป็นอย่างยิ่ง นอกจากนี้ผู้ที่ต้องขอขอบพระคุณเป็นอย่างสูงอีกท่านหนึ่ง คือ คุณธนพร แสงไพฑูรย์ และบริษัทดีไซน์เทคโนโลยี จำกัดที่เอื้อเฟื้อ อุปกรณ์ และเป็นผู้ที่คอยให้คำแนะนำดีๆ ในหลายๆ อย่างอีกท่านหนึ่ง

และต้องขอขอบพระคุณบุคคลสำคัญที่สุด ที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ เอาใจใส่เสมอมาในทุกๆ ด้าน อันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

พินพงษ์ ศิลป์สกุลสุข

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา III
และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

หน้า

บทคัดย่อภาษาไทย.....	I
บทคัดย่อภาษาอังกฤษ.....	II
กิตติกรรมประกาศ.....	III
สารบัญ.....	IV
สารบัญรูป.....	IX
สารบัญตาราง.....	XIV

บทที่ 1 บทนำ.....	1
-------------------	---

1.1 ความเป็นมาของปัญหา.....	1
1.2 วัตถุประสงค์ของโครงการ.....	2
1.3 ประโยชน์ที่คาดว่าจะได้รับ.....	2
1.4 ขอบเขตของโครงการ.....	2
1.5 ส่วนประกอบของรายงาน.....	2

บทที่ 2 พื้นฐานเกี่ยวกับเรียล ไทม์ซิสเต็ม.....	3
--	---

2.1 บทนำ.....	3
2.2 ชนิดของเรียล ไทม์ซิสเต็ม.....	4

2.2.1 ฟอรักราวด์และแบคกราวด์ซิสเต็ม(Foreground / Background System).....	4
2.2.2 นอนพรีเอมทีฟเคอร์เนล(Non-Preemptive Kernel).....	5
2.2.3 พรีเอมทีฟเคอร์เนล(Preemptive Kernel).....	6

2.3 ทาส์ และ มัลติทาสกิ้ง.....	7
--------------------------------	---

2.3.1 ทาส์.....	7
2.3.2 มัลติทาสกิ้ง.....	8
2.3.3 คอนเท็กซ์สวิตซิง (context switching).....	9
2.3.4 เคอร์เนล.....	9

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ(ต่อ)

หน้า

2.3.5	ตัวจัดลำดับงาน	9
2.3.6	ความสำคัญของทาสก์.....	10
2.3.7	พรีออริตีอื่นเวอร์ชัน.....	10
2.4	ซิงโครไนเซชัน(Synchronization)	13
2.4.1	ทรัพยากรและทรัพยากรที่ใช้ร่วมกัน(Resource and Share Resource).....	13
2.4.2	รีเอนทรานซ์(Reentrancy).....	13
2.4.3	มิวทอลเอ็กซ์คลูชัน(Mutual Exclusion).....	15
2.4.4	เดดล็อก(Deadlock).....	20
2.4.5	ซิงโครไนเซชัน(Synchronization).....	21
2.4.6	อีเวนท์แฟลก(Event Flags).....	23
2.4.7	การสื่อสารระหว่างทาสก์.....	24
2.5	อินเตอร์รัพต์(Interrupt).....	26
2.5.1	ความล่าช้าจากอินเตอร์รัพต์(Interrupt Latency).....	28
2.5.2	การตอบสนองต่ออินเตอร์รัพต์(Interrupt Response).....	28
2.5.3	การออกจากอินเตอร์รัพต์(Interrupt Recovery).....	28
2.5.4	คล็อกทิก(Clock Tick).....	30
2.6	คริติคอลเซกชันของโค้ด.....	32
2.7	ความต้องการหน่วยความจำ.....	32
2.8	ข้อดีและข้อเสียของเรียลไทม์เคอร์เนล.....	33
บทที่3	สถาปัตยกรรมของไมโครเบลส.....	34
3.1	ภาพรวม	34
3.1.1	คุณสมบัติ.....	34

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ(ต่อ)

	หน้า
3.2 ชนิดและการจัดเรียงตัวของข้อมูล.....	35
3.3 ชุดคำสั่ง.....	36
3.4 รีจิสเตอร์.....	37
3.5 สถาปัตยกรรมของไปป์ไลน์.....	37
3.5.1 ไปป์ไลน์แบบ 3 สเตจ.....	37
3.5.2 ไปป์ไลน์แบบ 5 สเตจ.....	37
3.5.3 แบรินช์(Branches).....	38
3.6 รีเซ็ต(Reset), อินเทอร์รัพต์(Interrupts), เอ็กเซพชั่น(Exceptions) และเบรก(Break).....	38
3.6.1 รีเซต.....	39
3.6.2 อินเทอร์รัพต์.....	39
บทที่4 การสร้างระบบไมโครเบลสอย่างง่าย.....	41
4.1 ภาพรวม.....	41
4.2 การสร้างระบบไมโครเบลสด้วย เบสซิสเต็มบิลเดอร์วิซาร์ด (Base System Builder Wizard).....	41
4.3 ภาพของระบบที่ถูกสร้างขึ้น.....	47
4.4 การนำระบบไมโครเบลสลงไปทำงานบนบอร์ดเฟิร์มแวร์.....	48
4.5 การพัฒนาซอฟต์แวร์บนระบบไมโครเบลสที่สร้างขึ้น.....	51
บทที่5 การใช้งานจีพีไอ โอบนระบบไมโครเบลส.....	54
5.1 ภาพรวม.....	54
5.2 ทำการเพิ่มไอพีกอร์ลงในระบบไมโครเบลส.....	54
5.3 การเขียนโปรแกรมเพื่อควบคุมจีพีไอโอ.....	60
บทที่6 การใช้งานอินเทอร์พท์และไทม์เมอร์ในระบบไมโครเบลส.....	65
6.1 ภาพรวม.....	65

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ(ต่อ)

หน้า

6.2	การเพิ่มไทม์เมอร์และอินเทอร์รัพท์คอนโทรลเลอร์ลงในระบบ.....	65
6.3	การเขียนโปรแกรมเพื่อใช้งานไทม์เมอร์ และอินเทอร์รัพท์คอนโทรลเลอร์.....	68
6.3.1	การเขียนโปรแกรมเพื่อใช้งานไทม์เมอร์.....	68
6.3.2	การเขียนโปรแกรมเพื่อใช้งานอินเทอร์รัพท์.....	70
บทที่7	การสร้างไอพ็คคอร์เพื่อใช้กับระบบไมโครเบลส.....	74
7.1	ภาพรวม.....	74
7.2	ไอพ็คอินเตอร์เฟส.....	74
7.3	การสร้างยูเซอร์ไอพ็คคอร์.....	76
7.4	การเขียนซอฟต์แวร์เพื่อส่งงานไอพ็คคอร์ที่สร้างขึ้นเอง.....	81
บทที่8	การนำไมโครซีไอเอสทูว์มาใช้งานบนระบบไมโครเบลส.....	84
8.1	ภาพรวม.....	84
8.2	ไมโครซีไอเอสทูว์.....	84
8.2.1	เตรียมความพร้อมของระบบก่อนใช้งานไมโครซีไอเอสทูว์.....	84
8.2.2	การเขียนโปรแกรมให้ไมโครซีไอเอสทูว์เริ่มทำงาน.....	88
บทที่9	ตัวอย่างโปรแกรมประยุกต์.....	92
9.1	ภาพรวม.....	92
9.2	อุปกรณ์ภายนอกที่ใช้ในการทดลองนี้.....	92
9.2.1	เซ็นเซอร์วัดอุณหภูมิ.....	92
9.2.2	จอแอลซีดีแบบกราฟิก.....	95
9.3	ทาสก์ในระบบ.....	97
9.3.1	AppTaskFirst.....	97

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา VII
และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ(ต่อ)

หน้า

9.3.2	TempLoggerTask1(2,3).....	97
9.3.3	DispTask.....	98
9.3.4	UARTCmdTask.....	98
9.4	ลำดับการทำงานของโปรแกรม.....	98
9.5	ภาพรวมการทำงานของระบบ.....	98
บทที่10	บทวิจารณ์และบทสรุป.....	103
10.1	สรุปผลการทำงาน.....	103
10.2	ปัญหาและอุปสรรคในการทำงาน.....	103
10.3	แนวทางแก้ไข.....	103
10.4	บทวิจารณ์โครงการ.....	104
10.5	แนวทางการพัฒนา.....	104

สารบัญรูป

หน้า

รูปที่ 2.1ฟอร์กราวด์/แบ็กกราวด์ซิสเต็ม (Foreground/Background System).....	4
รูปที่ 2.2นอนพรีเอมทีฟเคอร์เนล (non preemptive kernel)	5
รูปที่ 2.3พรีเอมทีฟเคอร์เนล (preemptive kernel)	6
รูปที่ 2.4มัลติเพิลทาสก์ (Multiple Task).....	7
รูปที่ 2.5สถานะของทาสก์	8
รูปที่ 2.6ปัญหาไพรออริตีอินเวอร์ชัน (priority inversion problem).....	10
รูปที่ 2.7เนลซึ่งรองรับ ไพรออริตีอินเฮอริแตนส์ (kernel support priority inheritance).....	12
รูปที่ 2.8รีเอนทรานฟังก์ชัน (reentrant function).....	13
รูปที่ 2.9นอนรีเอนทรานฟังก์ชัน (non-reentrant function).....	14
รูปที่ 2.10นอนรีเอนทรานฟังก์ชัน (non reentrant function).....	14
รูปที่ 2.11การดิสเอเบิลและเอนาเบิลอินเตอร์รัพต์ (disable and enable interrupt).....	15
รูปที่ 2.12 test and set	16
รูปที่ 2.13การดิสเอเบิลและเอนาเบิลการจัดลำดับ (disable and enable scheduler).....	16
รูปที่ 2.14การเข้าใช้ทรัพยากร โดยการใช้งานเซมาฟอว์.....	18
รูปที่ 2.15การซ็อนเซมาฟอว์จากทาสก์.....	19
รูปที่ 2.16เคาท์ดิง เซมาฟอว์ (counting semaphore).....	19
รูปที่ 2.17การจัดการบัฟเฟอร์โดยการใช้งานเซมาฟอว์.....	20
รูปที่ 2.18การซิงโครไนซ์ของทาสก์และ ISR	21
รูปที่ 2.19ทาสก์ซิงโครไนซ์กิจกรรมร่วมกัน	22
รูปที่ 2.20 code bilateral rendezvous	22
รูปที่ 2.21ดิซจังก์ทีฟและคอนจังก์ทีฟ ซิงโครไนซ์เซชัน	23
รูปที่ 2.22อีเวนท์เฟลก.....	23
รูปที่ 2.23แมสเสจเมลบี๊อก	25
รูปที่ 2.24แมสเสจคิว	26

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา IX ต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป(ต่อ)

หน้า

รูปที่ 2.25 อินเทอร์รัพต์แบบซ้อน	27
รูปที่ 2.26 Interrupt latency , response และ recovery (foreground/background)	29
รูปที่ 2.27 Interrupt latency , response และ recovery (non-preemptive kernel)	29
รูปที่ 2.28 Interrupt latency , response และ recovery (preemptive kernel)	30
รูปที่ 2.29 ดีเลย์ทาสก์ กรณีที่ 1	31
รูปที่ 2.30 ดีเลย์ทาสก์ กรณีที่ 2	31
รูปที่ 2.31 ดีเลย์ทาสก์ กรณีที่ 3	32
รูปที่ 3.1 ผังการทำงานโดยรวมของ ไมโครเบลส คอร์	34
รูปที่ 3.2 ไปป์ไลน์แบบ 3 สเตจ	37
รูปที่ 3.3 ไปป์ไลน์แบบ 5 สเตจ	38
รูปที่ 4.1 การเลือกเบสซิมบิลเคอร์รี่ชาร์ด	41
รูปที่ 4.2 การเลือกตำแหน่งในการเซฟโปรเจค	42
รูปที่ 4.3 การตั้งค่าเกี่ยวกับออฟฟิซเอด และ โพรเซสเซอร์ที่จะใช้	43
รูปที่ 4.4 การตั้งค่าต่างๆของตัวไมโครเบลส	45
รูปที่ 4.5 การเพิ่มไอโอของระบบ	46
รูปที่ 4.6 ภาพรวมของระบบที่ถูกสร้างขึ้น	47
รูปที่ 4.7 การเปิดยูซีเอฟไฟล์	48
รูปที่ 4.8 ไฟล์ยูซีเอฟที่ถูกแก้ไขแล้ว	49
รูปที่ 4.9 คอนโซลหลังจากทำการสร้างบิตสตรีมแล้ว	49
รูปที่ 4.10 การเลือกให้บูทลูปไปอยู่บนบลอคแรม	50
รูปที่ 4.11 คาวโหลดบิตสตรีมด้วยโปรแกรมอิมแพค	50
รูปที่ 4.12 การตั้งค่าในเฮ็กซ์ทีเอสเอสดีเค	51
รูปที่ 4.13 โปรแกรม "Hello Microblaze!"	52
รูปที่ 4.14 การตั้งค่าการรัน	52

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา **X** และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป(ต่อ)

หน้า

รูปที่ 4.15ปุ่มรัน	53
รูปที่ 4.16ผลการทำงานของ โปรแกรม.....	53
รูปที่ 5.1การเพิ่มจีพีไอ โอ	55
รูปที่ 5.2การเชื่อมต่อจีพีไอ โอเข้ากับ ไอพีบี.....	56
รูปที่ 5.3การแก้ความกว้างบัสของจีพีไอ โอ	57
รูปที่ 5.4การทำขาพินให้เชื่อมต่อกับภายนอก.....	58
รูปที่ 5.5พอร์ตของระบบหลังจากทำการเพิ่มขาภายนอก	58
รูปที่ 5.6การทำแอดเดรสแมพ.....	59
รูปที่ 5.7ภาพรวมของระบบหลังจากเพิ่มสวิตช์แล้ว.....	59
รูปที่ 5.8การแมพขานนยูซีเอฟไฟล์.....	60
รูปที่ 5.9โปรแกรมอ่านค่าจากสวิตช์.....	62
รูปที่ 5.10ค่าที่พิมพ์ออกมาทางเทอร์มินอลหลังจากเปลี่ยนสวิตช์.....	62
รูปที่ 5.11 โปรแกรมในการอ่านค่าสวิตช์และเขียนออกแอลอีดี.....	63
รูปที่ 5.12การทำงานของสวิตช์และแอลอีดี.....	64
รูปที่ 6.1เพิ่มไทม์เมอร์และอินเทอร์รัพท์คอน โทรลเข้าสู่ระบบ	65
รูปที่ 6.2การตั้งค่าเน็ทของไทม์เมอร์.....	66
รูปที่ 6.3การตั้งค่าเน็ท Intr ของอินเทอร์รัพท์คอน โทรล.....	67
รูปที่ 6.4การตั้งค่าเน็ท Irq ของอินเทอร์รัพท์คอน โทรล.....	67
รูปที่ 6.5การตั้งค่าเน็ท INTERRUPT ของตัวไมโครเบลส.....	68
รูปที่ 6.6โปรแกรมไทม์เมอร์นับลง.....	69
รูปที่ 6.7ผลลัพธ์ของโปรแกรมไทม์เมอร์นับลง.....	70
รูปที่ 6.8ฟังก์ชันตอบสนองการทำงานของอินเทอร์รัพท์.....	71
รูปที่ 6.9โปรแกรมรีจิสเตอร์ฟังก์ชันตอบสนองอินเทอร์รัพท์.....	72
รูปที่ 6.10 ผลลัพธ์ของโปรแกรมตอบสนองอินเทอร์รัพท์.....	73

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา XI ต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป(ต่อ)

หน้า

รูปที่ 7.1การตั้งค่าไอพีไอเอฟ.....	75
รูปที่ 7.2การเชื่อมต่อไอพีที่สร้างขึ้นกับไอพีบี.....	76
รูปที่ 7.3การเพิ่มพอร์ตในยูเซอร์ลอคจิก.....	76
รูปที่ 7.4 "signal" ในส่วน "architecture".....	77
รูปที่ 7.5โค้ดการทำงานของเซว่นเซกเมนต์คอนโทรลเลอร์.....	78
รูปที่ 7.6การแปลงพอนท์ของเซว่นเซกเมนต์.....	79
รูปที่ 7.7พอร์ตที่เพิ่มในส่วนทอปเลเวล.....	79
รูปที่ 7.8การทำพอร์ตแมพที่ทอปเลเวล.....	80
รูปที่ 7.9เพิ่มพอร์ตลงในเอ็มพีดีไฟล์.....	80
รูปที่ 7.10การแมพขาอุปกรณ์ออกมาเป็นภายนอก.....	81
รูปที่ 7.11 โปรแกรมสั่งงานเซว่นเซกเมนต์โมดูล.....	82
รูปที่ 7.12 ผลลัพธ์ที่เซว่นเซกเมนต์.....	83
รูปที่ 8.1ระบบที่จะนำไมโครซีไอเอสทูว์เข้ามาทำงาน.....	85
รูปที่ 8.2เลือก "OS" เป็น "uCOS-II".....	86
รูปที่ 8.3การแก้ไข"FILE_LOCATION".....	87
รูปที่ 8.4แก้ไขไฟล์ "bsp.c".....	88
รูปที่ 8.5 โปรแกรมส่วน "main" ในการเริ่มการทำงานของไอเอส.....	89
รูปที่ 8.6การทำงานภายในทาสก์แรกของเรา.....	90
รูปที่ 8.7โปรแกรมพิมพ์ตัวเลขทุกวินาที.....	90
รูปที่ 8.8ผลการทำงานของโปรแกรมนับเลขในทุกวินาที.....	90
รูปที่ 8.9ทาสก์ที่สองในโปรแกรม.....	91
รูปที่ 8.10การสร้างทาสก์ที่สอง.....	91
รูปที่ 8.11ผลลัพธ์ของโปรแกรมที่มีสองทาสก์.....	91
รูปที่ 9.1สัญญาณรีเซตของโปรโตคอลวันไวร์.....	93

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา **XII** และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป(ต่อ)

หน้า

รูปที่ 9.2 การอ่านและเขียนข้อมูลบนโปรโตคอลวันไวร์.....	94
รูปที่ 9.3 การสื่อสารบนโปรโตคอลเอสพีไอ.....	95
รูปที่ 9.4 การส่งภาพไปแสดงผลที่จอ.....	97
รูปที่ 9.5 ภาพรวมของระบบ.....	99
รูปที่ 9.6 เซ็นเซอร์วัดอุณหภูมิ.....	99
รูปที่ 9.7 ตัวเลขแสดงภาวะของซีพียู.....	100
รูปที่ 9.8 กราฟแสดงอุณหภูมิของเซ็นเซอร์ทั้งสามตัว.....	100
รูปที่ 9.9 สวิตช์เลือกให้แสดงผลเซ็นเซอร์ตัวที่สามสามตัวเดียว.....	101
รูปที่ 9.10 เพิ่มอุณหภูมิที่เซ็นเซอร์ตัวที่สาม.....	101
รูปที่ 9.11 กราฟอุณหภูมิของเซ็นเซอร์ตัวที่สาม.....	102
รูปที่ 9.12 ผลการอ่านค่าเซ็นเซอร์ตัวที่สามทางพอร์ตอนุกรม.....	102

สารบัญตาราง

หน้า

ตาราง 3.1 คุณสมบัติที่ปรับแต่งได้ของไมโครเบลส เรียงตามรุ่นของไมโครเบลส.....	35
ตาราง 3.2 ข้อมูลชนิดเวิร์ด.....	36
ตาราง 3.3 ข้อมูลชนิดฮาล์ฟเวิร์ด.....	36
ตาราง 3.4 ข้อมูลชนิดไบต์.....	36
ตาราง 3.5 เวกเตอร์ และ ตำแหน่งของรีเทอร์นแอดเดรสไฟล์.....	39



บทที่ 1

บทนำ

1.1 ความเป็นมาของปัญหา

เนื่องด้วยในปัจจุบันนี้ ระบบสมองกลฝังตัวนั้น ได้เข้ามามีบทบาทในชีวิตประจำวันของเรามากขึ้นทุกวัน และด้วยเทคโนโลยีในปัจจุบันนี้ หน่วยประมวลผลที่ใช้ในระบบสมองกลฝังตัวนั้นเริ่มมีประสิทธิภาพสูงขึ้นเรื่อยๆ จึงทำให้สามารถนำหน่วยประมวลผลเหล่านี้ไปทำงานที่มีความซับซ้อนได้มากขึ้น หรือแม้กระทั่งทำงานหลายๆอย่างไปพร้อมๆกัน ในงานบางชนิดนั้น ระบบมีความจำเป็นอย่างยิ่งที่จะต้องตอบสนองเหตุการณ์ต่างๆให้ทันภายในระยะเวลาที่กำหนด ระบบประเภทนี้เรียกว่าระบบทันเวลา หรือ เรียลไทม์ซิสเต็ม(real-time system) ซึ่งการออกแบบ และสร้างระบบประเภทเรียลไทม์ซิสเต็มนี้ค่อนข้างมีความซับซ้อนและมีสิ่งที่จะต้องระวังอยู่มาก ทางออกที่ดีทางหนึ่งนั้นคือการใช้งานระบบปฏิบัติการแบบทันเวลา หรือ เรียลไทม์โอเปอเรติงซิสเต็มนั่นเอง

ในระบบสมองกลฝังตัวนั้น หน่วยประมวลผลมีความจำเป็นที่จะต้องติดต่อกับอุปกรณ์ต่างๆ หลายชนิด วงจรรวมประเภทไมโครคอนโทรลเลอร์ในปัจจุบันจึงพยายามผนวกเอาอุปกรณ์ต่างๆเข้าไว้ในตัวมากขึ้นเรื่อยๆ และเรียกตัวเองว่าเป็นวงจรรวมชนิด ซิสเต็มออนชิป(system on chip) หรือ เอส โอซี(SoC) แต่อย่างไรก็ตามอุปกรณ์เหล่านี้ก็ยังไม่เพียงพอกับความต้องการ เนื่องจากแต่ละระบบนั้นมีความต้องการอุปกรณ์ที่แตกต่างกันออกไปการเลือกใช้เอฟพีจีเอที่มีโปรเซสเซอร์คอร์ฝังอยู่จึงเป็นอีกทางออกหนึ่งที่น่าสนใจ เพราะเราสามารถออกแบบวงจบบนเอฟพีจีเอให้เป็นอุปกรณ์ที่เหมาะสมกับระบบของเราได้เอง แต่อย่างไรก็ตามเอฟพีจีเอประเภทที่มีโปรเซสเซอร์คอร์ฝังอยู่นี้ ในปัจจุบันยังมีราคาค่อนข้างสูง ไม่คุ้มค่ากับการนำมาใช้งานจริง อีกทางออกหนึ่งที่ค่อนข้างจะลงตัวคือการเลือกใช้ ซอฟท์โปรเซสเซอร์คอร์ ซึ่งใช้พื้นที่ของเอฟพีจีเอในการสังเคราะห์ตัว โปรเซสเซอร์นี้ขึ้นมา ถึงแม้ว่าประสิทธิภาพจะสู้โปรเซสเซอร์คอร์จริงๆไม่ได้ และเปลืองพื้นที่ของตัวเอฟพีจีเอไปบ้าง แต่หากนำเรื่องราคาเข้ามาเปรียบเทียบด้วยแล้ว การใช้ซอฟต์แวร์โปรเซสเซอร์คอร์เช่นนี้คุ้มแก่ต่อการนำมาใช้ในระบบจริงมากกว่าการใช้เอฟพีจีเอที่มีโปรเซสเซอร์คอร์จริงๆ ทางผู้จัดทำโครงการจึงมีแนวคิดที่จะออกแบบระบบสมองกลฝังตัวแบบทันเวลาบนเอฟพีจีเอขึ้น โดยใช้ซอฟต์แวร์โปรเซสเซอร์คอร์และเรียลไทม์โอเปอเรติงซิสเต็มที่ได้กล่าวไว้ในข้างต้น พร้อมทั้งทำเอกสาร

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

อธิบายขั้นตอนการออกแบบโดยละเอียดเพื่อเป็นประโยชน์แก่ผู้ที่ต้องการนำระบบในลักษณะนี้ไปใช้งานจริง

1.2 วัตถุประสงค์ของโครงการ

- เพื่อศึกษาและรวบรวมข้อมูลที่เกี่ยวข้องกับวิธีการสร้างระบบแบบเรียลไทม์บนแพลตฟอร์มที่มีความยืดหยุ่นสูง
- เพื่อสร้างระบบเรียลไทม์ตัวอย่างบนแพลตฟอร์มที่มีความยืดหยุ่นสูง

1.3 ประโยชน์ที่คาดว่าจะได้รับ

- ระบบสมองกลฝังตัวแบบทันเวลาบนเอฟทีอีเอ
- เอกสารอธิบายขั้นตอนการออกแบบของระบบอย่างละเอียด

1.4 ขอบเขตของโครงการ

- รวบรวมข้อมูลเกี่ยวกับการใช้งานไมโครซีไอเอสทูว์ และไมโครเบลส
- ทดลองสร้างระบบตัวอย่างโดยใช้ไมโครซีไอเอสทูว์ บนไมโครเบลส
- เรียบเรียงวิธีการในการสร้างระบบเรียลไทม์โดยใช้ไมโครซีไอเอสทูว์ และไมโครเบลส

1.5 ส่วนประกอบของรายงาน

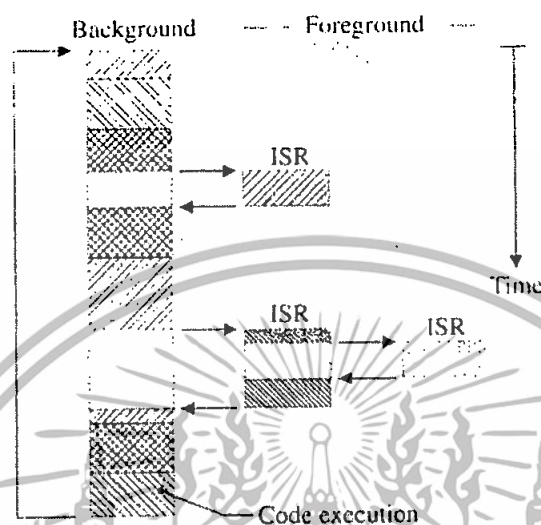
- บทที่ 1 กล่าวถึงความเป็นมาของปัญหา วัตถุประสงค์ของโครงการ ประโยชน์ที่คาดว่าจะได้รับ ขอบเขตของโครงการ และส่วนประกอบของรายงานฉบับนี้
- บทที่ 2 กล่าวถึงทฤษฎีพื้นฐานเกี่ยวกับระบบทันเวลา หรือเรียลไทม์ซิสเต็ม
- บทที่ 3 กล่าวถึงสถาปัตยกรรมของซอฟต์แวร์โพสเซสเซอร์คอร์ด ไมโครเบลส ที่ใช้ในโครงการ
- บทที่ 4 เป็นเอกสารประกอบการทดลองการสร้างไมโครเบลสซิสเต็มอย่างง่าย
- บทที่ 5 เป็นเอกสารประกอบการทดลองการใช้งานจีพีไอโอบนไมโครเบลสซิสเต็ม
- บทที่ 6 เป็นเอกสารประกอบการทดลองการใช้งานไทม์เมอร์และอินเทอร์รัพท์บนไมโครเบลส
- บทที่ 7 เป็นเอกสารประกอบการทดลองการสร้างไอพีคอร์เพื่อใช้งานบนไมโครเบลส
- บทที่ 8 เป็นเอกสารประกอบการทดลองการนำไมโครซีไอเอสทูว์ไปใช้งานบนระบบไมโครเบลส
- บทที่ 9 เป็นเอกสารประกอบโปรแกรมประยุกต์ตัวอย่างที่สร้างบนไมโครซีไอเอสทูว์บนระบบไมโครเบลส
- บทที่ 10 เป็นสรุปผลการดำเนินงานของโครงการ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.2 ชนิดของเรียลไทม์ซิสเต็ม

2.2.1 ฟอ์กราวด์และแบ็กกราวด์ซิสเต็ม(Foreground / Background System)

ระบบขนาดเล็กที่มีความซับซ้อนน้อยมักจะออกแบบดังนี้



รูปที่ 2.1ฟอ์กราวด์/แบ็กกราวด์ซิสเต็ม (Foreground/Background System)

ระบบที่เรียกว่าฟอ์กราวด์/แบ็กกราวด์(foreground/background) หรือซูเปอร์ลูป (super-loop) ตัวแอปพลิเคชันจะประกอบด้วย infinite loop ซึ่งจะเรียกว่าโมดูล (module) หรือฟังก์ชัน (function) เพื่อใช้ในการปฏิบัติการเป็นแบ็กกราวด์หรือเรียกว่าทาสก์เลเวล(task level) ส่วนอินเตอร์รัพต์เซอร์วิสรูทีน (interrupt service routines) หรือ ISRจะจัดการกับเหตุการณ์หรืออีเวนต์ (event) ที่เกิดขึ้น โดยจะทำงานเป็นฟอ์กราวด์หรือจะเรียกอีกอย่างว่าอินเตอร์รัพต์เลเวล (interrupt level) การทำงานที่เกิดขึ้นนั้นจะถูกจัดการโดยจะใช้ ISR เพื่อรับประกันว่าผลที่ได้จะทันเวลาที่ต้องการ

สำหรับข้อมูลของแบ็กกราวด์โมดูลที่ถูกจัดเตรียมโดย ISR นั้นจะไม่ถูกนำไปใช้งานจนกว่าแบ็กกราวด์รูทีนจะเริ่มทำงาน ซึ่งจะเรียกว่าการตอบสนองของทาสก์เลเวล (task level response) กรณีที่แย่ที่สุดของเวลาในการตอบสนองของทาสก์เลเวลขึ้นอยู่กับระยะเวลาที่แบ็กกราวด์ลูปทำงาน เพราะว่าเวลาในการทำงานของโค้ด(code)จะไม่คงที่ โดยขึ้นกับเวลาของลูปซึ่งไม่สามารถคาดการณ์ได้ ฉะนั้น ถ้ามีการเปลี่ยนแปลงโค้ด การทำงานของลูปจะมีผลต่อเวลา

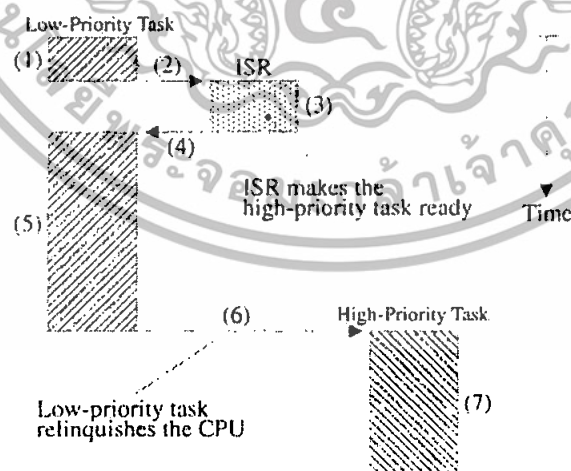
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.2.2 นอนพรีเอมทีฟเคอร์เนล(Non-Preemptive Kernel)

นอนพรีเอมทีฟเคอร์เนล (Non-preemptive kernel) จะให้แต่ละทาสก์ที่ทำงานอยู่ภายในซีพียู ทำงานให้เสร็จก่อน ซึ่งการจัดลำดับแบบนอนพรีเอมทีฟ (Non-preemptive scheduling) จะเรียกว่า โคโอเปอร์เรทีฟมัลติทาสกิง (cooperative multitasking) โดยทาสก์จะร่วมมือกับทาสก์อื่น ๆ ในการใช้งานซีพียูร่วมกัน ส่วนอีเวนต์ที่เกิดขึ้นจะถูกจัดการ โดย ISR ซึ่ง ISR จะทำหน้าที่ให้ทาสก์ที่มีความสำคัญสูงกว่าและพร้อมที่จะทำงานได้ทำงานก่อน แต่หลังจากเสร็จงานที่ทำโดย ISR แล้ว จะย้อนกลับไปทำทาสก์ที่ทำก่อนเกิดอินเตอร์รัพต์ และเมื่อมีทาสก์ใหม่ที่มีความสำคัญสูงกว่าต้องการทำงาน จะต้องรอให้ทาสก์ที่กำลังทำงานอยู่ใน ซีพียูทำงานให้เสร็จก่อน จึงจะสามารถเข้าใช้งานซีพียูได้

ประโยชน์ของนอนพรีเอมทีฟเคอร์เนล คือ ที่ระดับทาสก์เวลานั้น นอนพรีเอมทีฟเคอร์เนลสามารถจะใช้นอนรีเอนทรานฟังก์ชัน(non-reentrant function) ได้ ซึ่งทาสก์สามารถใช้นอนรีเอนทรานฟังก์ชันได้โดยไม่ต้องกลัวว่าจะเกิดการเปลี่ยนแปลงค่าโดยทาสก์อื่น เพราะว่าแต่ละทาสก์จะต้องทำงานให้เสร็จก่อนที่เลิกใช้งานซีพียู

ประโยชน์อีกอย่างหนึ่งของนอนพรีเอมทีฟเคอร์เนล คือ ไม่ต้องกังวลเรื่องการร่วมกันใช้ข้อมูล เพราะแต่ละทาสก์จะได้ทำงานให้เสร็จโดยไม่มีทาสก์อื่นเข้ามาใช้งานซีพียู และจะมีการใช้เซมาฟออร์ (semaphore) เพื่อช่วยในการจัดการเรื่องการเข้าใช้งานอุปกรณ์ I/O ต่าง ๆ ร่วมกัน ตัวอย่างเช่น การเข้าใช้งานปริ้นเตอร์



รูปที่ 2.2นอนพรีเอมทีฟเคอร์เนล (non preemptive kernel)

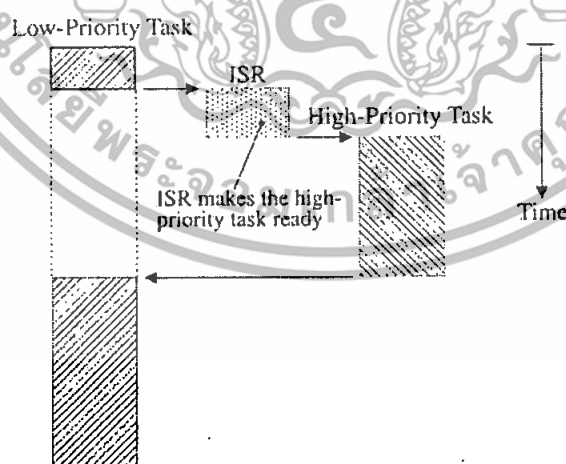
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ที่หมายเลข 1 ขณะที่ทาสก์กำลังทำงานอยู่ มีการเกิดอินเตอร์รัพต์ขึ้น ถ้าอินเตอร์รัพต์ได้มีเปิดการใช้งานซีพียูจะทำการกระโดดเข้าไปทำงานใน ISR (หมายเลข 2) ซึ่ง ISR จะจัดการกับอีเวนต์ที่เกิดขึ้น (หมายเลข 3) และจะให้ทาสก์ที่มีความสำคัญสูงกว่าพร้อมที่จะทำงาน ซึ่งหลังจากทำงานในส่วนของ ISR เสร็จแล้วจะย้อนกลับมาทำงานในทาสก์เดิม (หมายเลข 4) และทำงานต่อไปเรื่อย ๆ (หมายเลข 5) หลังจากนั้นเมื่อทาสก์ทำงานเสร็จ มันจะทำการเรียกใช้งานในส่วนเคอร์เนลเพื่อเลิกใช้งานซีพียู (หมายเลข 6) และทาสก์ตัวใหม่จะทำงาน ซึ่งเป็นทาสก์ที่ทำงานเมื่อเกิดเหตุการณ์ขึ้นจาก ISR (หมายเลข 7)

อุปสรรคสำคัญของนอนพรีเอมทีฟเคอร์เนล คือ การเวลาในการตอบสนอง โดยทาสก์ที่มีความสำคัญสูงและพร้อมที่จะทำงาน อาจจะต้องคอยนาน เพราะจะต้องคอยให้ทาสก์ที่กำลังทำงานอยู่ทำงานเสร็จก่อน โดยในระบบฟอร์กราวด์/แบ็กกราวด์ เวลาการตอบสนองของทาสก์เลเวลในนอนพรีเอมทีฟเคอร์เนลจะไม่สามารถคาดการณ์ได้ ทำให้ไม่รู้ว่าเมื่อไหร่ทาสก์ที่มีความสำคัญสูงจะได้ใช้งานซีพียูซึ่งก็ขึ้นอยู่กับแอปพลิเคชันที่สร้างขึ้น

2.2.3 พรีเอมทีฟเคอร์เนล (Preemptive Kernel)

พรีเอมทีฟเคอร์เนล (preemptive kernel) จะใช้งานเมื่อต้องการการตอบสนองเป็นหลัก โดยทาสก์ที่มีความสำคัญสูงและพร้อมที่จะทำงานจะได้ใช้งานซีพียูทันที โดยเมื่อทาสก์ตัวหนึ่งซึ่งมีความสำคัญสูงพร้อมที่จะทำงาน ทาสก์ตัวเดิมที่กำลังทำงานอยู่จะหยุดการทำงานชั่วคราวและให้ทาสก์ที่มีความสำคัญสูงจะเข้าใช้งานซีพียูทันที



รูปที่ 2.3 พรีเอมทีฟเคอร์เนล (preemptive kernel)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แอปพลิเคชันที่ใช้งานพริอเมทีฟเคอร์เนลจะไม่ใช้ฮอนรีเอนทรานฟังก์ชัน เพราะไม่สามารถป้องกันการเข้าใช้งานฟังก์ชันได้ โดยทาสก์ที่มีความสำคัญสูงและต่ำจะสามารถเข้าใช้งานฟังก์ชันทั่วไปได้ ทำให้ข้อมูลอาจเกิดการซ้อนทับกันได้ กรณีเมื่อทาสก์ที่มีความสำคัญสูง ต้องการใช้งานฟังก์ชันขณะที่ทาสก์ที่มีความสำคัญต่ำกำลังใช้งานอยู่

2.3 ทาสก์ และ มัลติทาสก์

2.3.1 ทาสก์

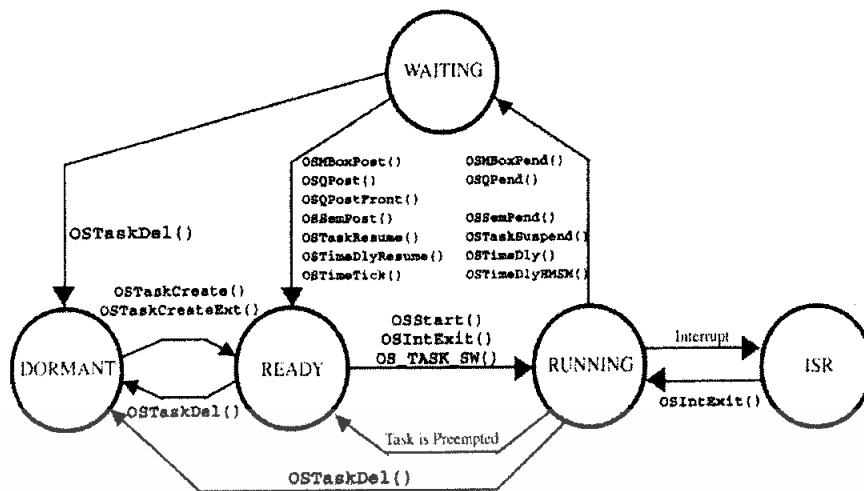
ทาสก์ (task) คือ ตัวโปรแกรมที่คิดว่ามีซีพียูเป็นของตัวเองทั้งหมด การออกแบบการทำงานของเรียลไทม์แอปพลิเคชันจะเริ่มจากการแยกหน้าที่ให้แก่ทาสก์รับผิดชอบสำหรับการใช้แก้ปัญหาต่าง ๆ โดยในแต่ละทาสก์จะกำหนดค่าความสำคัญ (priority), กำหนดคอนเท็กซ์(context)ภายในรีจิสเตอร์ (register)ของซีพียู และกำหนดพื้นที่สแต็ก (stack) ของแต่ละทาสก์



รูปที่ 2.4 มัลติเพิลทาสก์ (Multiple Task)

ในแต่ละทาสก์มักจะอยู่ในสถานะซึ่งจะเป็น 1 ใน 5 สถานะ ได้แก่ DORMANT , READY , RUNNING , WAITING (สำหรับสถานการณ์ต่าง ๆ) และ ISR

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.5สถานะของทาสก์

- สถานะ DORMANT จะทำหน้าที่สร้างทาสก์ขึ้นมา(แต่ยังไม่พร้อมที่จะทำงาน) และทำลายทาสก์
- ทาสก์จะเป็นสถานะ READY เมื่อทาสก์ที่สร้างมาสามารถพร้อมจะทำงาน แต่ความสำคัญมีค่าต่ำกว่าทาสก์ที่กำลังทำงานอยู่ในสถานะ RUNNING
- สถานะ RUNNING เป็นสถานะที่ทาสก์กำลังทำงานโดยเข้าใช้งานโปรเซสเซอร์
- ทาสก์จะเป็นสถานะ WAITING หรือ BLOCKING เมื่อทาสก์ต้องการให้อีเวนต์เกิดขึ้น เช่น การรอที่จะให้ I/O ทำงานเสร็จ , รอจะใช้งานแชร์รีซอสหรือการรอให้เกิดพัลส์ (pulse) เป็นต้น
- ทาสก์จะเป็นสถานะ ISR เมื่อมีอินเตอร์รัพท์เกิดขึ้น และซีพียูก็จะไปทำงานในส่วนของอินเตอร์รัพท์

2.3.2 มัลติทาสกิง

มัลติทาสกิง (Multitasking) คือกระบวนการจัดลำดับการทำงาน (scheduling) และสวิตชิง (switching) ของ CPU (Central Processing Unit) ระหว่างหลายๆ ทาสก์โดยซีพียูตัวเดียวจะพิจารณาหลายๆ ลำดับของทาสก์ ซึ่งมัลติทาสกิงจะคล้ายกับฟอร์คราวด์/แบ็กกราวด์ด้วย โดยจะเป็นในรูปแบบหลายๆ แบ็กกราวด์ การทำมัลติทาสกิงเป็นการใช้ประโยชน์ของซีพียูให้คุ้มค่าที่สุดและยังเป็นการจัดเตรียมองค์ประกอบให้กับแอปพลิเคชันอีกด้วย ลักษณะสำคัญอีกประการหนึ่งของมัลติทาสกิงคือสามารถให้โปรแกรมเมอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ผู้พัฒนาแอปพลิเคชันสามารถจัดการความซับซ้อนที่มีในตัวเรียลไทม์แอปพลิเคชันได้โดยไม่ต้องคำนึงถึงแอปพลิเคชันตัวอื่น ทำให้แอปพลิเคชันโปรแกรมง่ายต่อการออกแบบและดูแล

2.3.3 คอนเท็กซ์สวิตชิง (context switching)

เมื่อมัลติทาสกิ้งเคอร์เนลทำการเปลี่ยนทาสก์ที่จะทำงาน จะทำการเก็บค่าคอนเท็กซ์ของทาสก์ (task context) หรือซีพียูรีจิสเตอร์ (CPU registers) เอาไว้ในพื้นที่สำหรับเก็บซึ่งเป็นสแตคของทาสก์นั้น เมื่อได้ทำการเก็บค่าเสร็จแล้วทาสก์ตัวใหม่จะทำการดึงค่าคอนเท็กซ์ (context) จากพื้นที่สำหรับเก็บค่าเพื่อนำกลับมาการทำงานต่อ ซึ่งการทำแบบนี้เรียกว่าคอนเท็กซ์สวิตชิง (context switch) หรือทาสก์สวิตชิง (task switch) โดยคอนเท็กซ์สวิตชิงจะมีการเพิ่มโอเวอร์เฮด (overhead) ให้กับแอปพลิเคชัน โดยซีพียูรีจิสเตอร์มากเท่าไรก็จะมีโอเวอร์เฮดมากขึ้น ส่วนเวลาในการทำคอนเท็กซ์สวิตชิงจะขึ้นกับจำนวนรีจิสเตอร์ที่ทำการเก็บและดึงกลับมาโดยซีพียู

2.3.4 เคอร์เนล

เคอร์เนล (kernel) เป็นส่วนหนึ่งของระบบมัลติทาสกิ้ง ซึ่งจะรับผิดชอบ โดยทำหน้าที่ในการจัดการทาสก์ เช่น การจัดการเวลาของซีพียูและใช้ในการติดต่อระหว่างทาสก์ การให้บริการทั่วไปของเคอร์เนล คือ การทำคอนเท็กซ์สวิตชิง ซึ่งการใช้งานเรียลไทม์เคอร์เนลทั่วไปจะใช้ในการจัดการทาสก์ โดยการออกแบบตัวระบบให้แอปพลิเคชันสามารถแบ่งออกได้หลาย ๆ ทาสก์ โดยเคอร์เนลจะมีการเพิ่มโอเวอร์เฮดเข้าไปในระบบเพราะต้องการใช้รอม (ROM) หรือพื้นที่โค้ด (code space) และแรม (RAM) ในส่วนของโครงสร้างข้อมูลของเคอร์เนลเพิ่ม ซึ่งต้องระวังเพราะธรรมดาแล้ว แต่ละทาสก์จะต้องการพื้นที่สแตคเป็นของตัวเอง ซึ่งมีแนวโน้มว่าจะกินพื้นที่ในแรมสูงพอสมควร

2.3.5 ตัวจัดลำดับงาน

ตัวจัดลำดับงาน (scheduler) หรือเรียกว่าดิस्पัทเชอร์ (dispatcher) เป็นส่วนหนึ่งในเคอร์เนลที่จะช่วยตัดสินใจว่าจะให้ทาสก์ตัวไหนทำงานต่อไป ซึ่งเรียลไทม์เคอร์เนลจะคิดจากค่าความสำคัญ (priority) เป็นหลัก โดยจะให้ทาสก์ที่มีความสำคัญสูงสุดและพร้อมที่จะทำงานได้ทำงานก่อน แต่อย่างไรก็ตามการตัดสินใจว่าจะให้ทาสก์ตัวไหนทำงาน จะขึ้นกับชนิดของเคอร์เนลด้วย โดยเคอร์เนลที่อาศัยค่าความสำคัญเป็นหลัก ประกอบไปด้วย 2 ชนิด นอนพรีเอมทีฟเคอร์เนล (non-preemptive kernel) และพรีเอมทีฟเคอร์เนล (preemptive kernel)

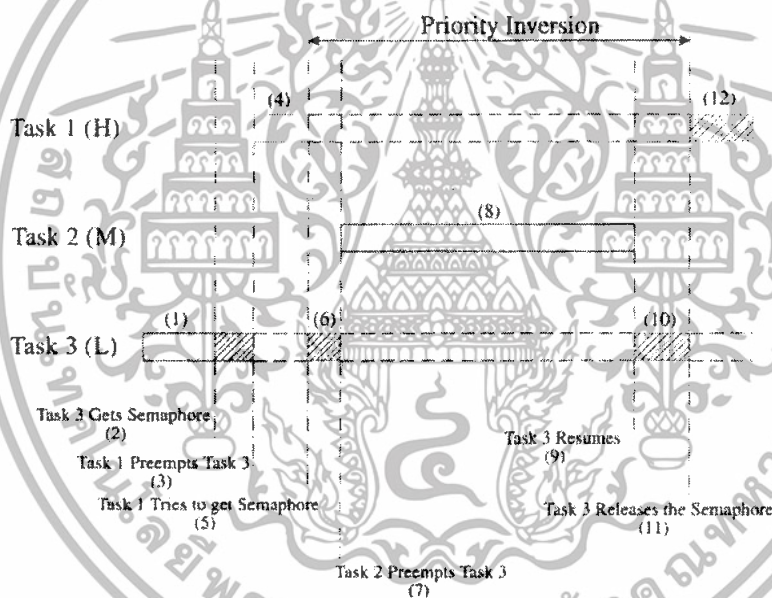
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.3.6 ความสำคัญของทาสก์

ความสำคัญ (priority) จะกำหนดให้ในแต่ละทาสก์ โดยทาสก์ที่มีความสำคัญจะถูกกำหนดให้มีความสำคัญสูงซึ่งแบ่งได้เป็นความสำคัญแบบสแตติก (static priority) และไดนามิก (dynamic priority) ซึ่งทาสก์จะเป็นแบบสแตติกเมื่อแต่ละทาสก์ไม่มีการเปลี่ยนความสำคัญระหว่างที่แอฟพลิเคชันกำลังทำงาน โดยแต่ละทาสก์จะถูกกำหนดความสำคัญมาตั้งแต่แรกแล้ว ส่วนความสำคัญแบบไดนามิก ทาสก์สามารถเปลี่ยนความสำคัญได้ระหว่างที่แอฟพลิเคชันกำลังทำงาน แต่อาจจะเกิดปัญหาคือไพรอริตีอินเวอร์ชัน (priority inversion) ได้ ซึ่งในเรียลไทม์เคอร์เนลจะมีการป้องกันการเกิดไพรอริตีอินเวอร์ชัน

2.3.7 ไพรอริตีอินเวอร์ชัน

ไพรอริตีอินเวอร์ชัน (priority inversion) เป็นปัญหาในเรียลไทม์ซิสเต็มและมักจะเกิดขึ้นกับเรียลไทม์เคอร์เนล



รูปที่ 2.6 ปัญหาไพรอริตีอินเวอร์ชัน (priority inversion problem)

กำหนดให้ทาสก์ 1 มีค่าความสำคัญสูงที่สุด ทาสก์ 2 รองลงมาและทาสก์ 3 มีความสำคัญต่ำที่สุด โดยทาสก์ 1 และทาสก์ 2 จะรอให้เกิดอีเวนต์ขึ้นก่อน

- ที่หมายเลข 1 ในขณะที่ทาสก์ 3 กำลังทำงานอยู่ ทาสก์ 3 ต้องการใช้งานทรัพยากรจึงเข้าใช้งานเซมาฟอว์ (semaphore) ที่หมายเลข 2

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- ในขณะที่ทาสก์ 3 กำลังทำงานอยู่กับทรัพยากร(หมายเลข 4)นั้น ทาสก์ 1 ก็ได้เริ่มทำงานขึ้น(หมายเลข 2) ซึ่งทาสก์ 3 ก็จะหยุดทำงานชั่วคราว

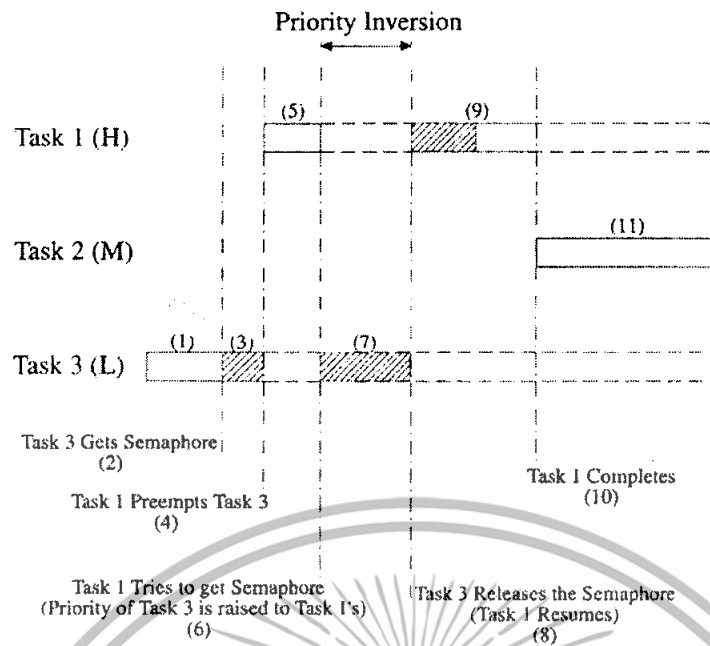
- เมื่อทาสก์ 1 ทำงานไปขณะหนึ่ง ต้องการที่จะใช้งานทรัพยากร(หมายเลข 5) แต่ทาสก์ 3 กำลังทำงานอยู่ในทรัพยากร ดังนั้นทาสก์ 1 จึงพยายามเข้าใช้งานเซมาฟอร์ทำให้ทาสก์ 1 ถูกหยุดการทำงานไปชั่วขณะและทาสก์ 3 ก็จะกลับมาทำงานต่ออีกครั้ง(หมายเลข 6)

- เมื่อทาสก์ 3 ทำงานไปได้ซักพักจะมีอีเวนต์เกิดขึ้นมาและทาสก์ 2 เริ่มทำงาน (หมายเลข 7 และ 8) ทาสก์ 3 จึงหยุดทำงานเพื่อให้ทาสก์ 2 ทำงาน เมื่อทาสก์ 2 ทำงานเสร็จแล้ว ทาสก์ 3 จึงกลับมาทำงานในทรัพยากรต่อ(หมายเลข 9)

- เมื่อทาสก์ 3 ทำงานกับทรัพยากรเสร็จ (หมายเลข 10) จึงหยุดการทำงานกับเซมาฟอร์ (หมายเลข 11) ซึ่งในขณะนี้เคอร์เนลได้รู้ว่ามิตาสก์ที่มีความสำคัญสูงรอใช้งานเซมาฟอร์ดังนั้นจึงทำคอนเท็กซ์สวิตช์ให้ทาสก์ 1 กลับมาทำงาน ทาสก์ 1 จึงได้กลับมาทำงานและเข้าใช้งานทรัพยากร (หมายเลข 12) จนกระทั่งทาสก์ 1 ทำงานเสร็จ

จากเหตุการณ์ข้างต้นเป็นปัญหาไพร์ออริตีอินเวอร์ชัน โดยการแก้ปัญหาไพร์ออริตีอินเวอร์ชันที่เกิดขึ้นสามารถแก้ไขได้โดยการเปลี่ยนความสำคัญของทาสก์ 3 เมื่อได้เข้าไปใช้งานทรัพยากร และเปลี่ยนความสำคัญกลับเป็นค่าเดิมเมื่อทาสก์ทำงานเสร็จ ซึ่งมีลิตทาสกิ้งเคอร์เนลสามารถที่จะเปลี่ยนความสำคัญได้เพื่อช่วยป้องกันการเกิดไพร์ออริตีอินเวอร์ชัน แต่อย่างไรก็ตามการเปลี่ยนความสำคัญก็ต้องการใช้เวลาในการเปลี่ยน และถ้าทาสก์ 3 ทำงานเสร็จสิ้นก่อนที่จะเกิดทาสก์ 1 หรือทาสก์ 2 จะทำให้เกิดการเปลี่ยนแปลงค่าความสำคัญของทาสก์ที่สูงเกินไป โดยเคอร์เนลที่สามารถเปลี่ยนค่าความสำคัญได้อัตโนมัติ จะเรียกว่าไพร์ออริตีอินฮีริแตนซ์ (priority inheritance)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.7 เกล็ดซึ่งรองรับไพโรอริตีอินเฮริแตนซ์ (kernel support priority inheritance)

จากภาพเคอร์เนลจะรองรับการทำไพโรอริตีอินเฮริแตนซ์

- เมื่อทาสก์ 3 ทำงานที่หมายเลข 1 และได้รับเซมาฟออร์เพื่อเข้าใช้งานทรัพยากร (หมายเลข 2) ทาสก์ 3 ก็จะเข้าไปใช้งานทรัพยากร (หมายเลข 3)

- ระหว่างนั้นทาสก์ 1 ก็ได้เริ่มทำงานขึ้น (หมายเลข 4 และ 5) ทาสก์ 1 พยายามที่จะใช้งานเซมาฟออร์ (หมายเลข 6) ซึ่ง เคอร์เนลจะเห็นว่าทาสก์ 3 กำลังใช้งานอยู่ แต่ทาสก์ 3 มีความสำคัญที่ต่ำกว่าทาสก์ 1 ทำให้เคอร์เนลเปลี่ยนความสำคัญของทาสก์ 3 ให้สูงเท่ากับสำคัญของทาสก์ 1 ดังนั้นเคอร์เนลจะให้ทาสก์ 3 ทำงานกับทรัพยากรต่อไปที่หมายเลข 7

- เมื่อทาสก์ 3 ทำงานกับทรัพยากรเสร็จแล้วก็จะเลิกใช้งานเซมาฟออร์ (หมายเลข 8) ตรงจุดนี้ เคอร์เนลจะทำการเปลี่ยนความสำคัญของทาสก์ 3 ให้กลับมาเป็นเหมือนเดิม และให้ทาสก์ 1 ใช้งานเซมาฟออร์เพื่อทำงานต่อไป (หมายเลข 9)

- เมื่อทาสก์ 1 ทำงานเสร็จแล้ว (หมายเลข 10) ทาสก์ 2 ก็จะใช้งาน ซีพียู โดยที่ทาสก์ 2 พร้อมทั้งจะทำงานในช่วงระหว่างหมายเลข 3 ถึง 10 แต่ไม่ได้ทำงาน ซึ่งตรงจุดนี้ยังเป็นไพโรอริตีอินเวอร์ชัน (ระหว่างทาสก์ 2 กับทาสก์ 3) ที่หลีกเลี่ยงไม่ได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การกำหนดความสำคัญให้แก่ทาสก์ก็เป็นสิ่งจำเป็น โดยที่เรียลไทม์ซิสเต็มจะต้องการทั้งซอฟต์แวร์เรียลไทม์และฮาร์ดแวร์เรียลไทม์ซิสเต็ม โดยที่ซอฟต์แวร์เรียลไทม์ซิสเต็มทาสก์จะทำงานให้เร็วที่สุดแต่อาจจะไม่เสร็จในเวลาที่กำหนด แต่ในฮาร์ดแวร์เรียลไทม์จะต้องเสร็จในเวลาที่กำหนด ด้วยเทคนิคที่เรียกว่า Rate Monotonic Scheduling หรือ RMS จะใช้ในการกำหนดความสำคัญของทาสก์โดยขึ้นกับความถี่ในการทำงาน โดยทาสก์ที่มักจะทำงานบ่อยจะกำหนดความสำคัญให้สูงสุด

2.4 ซิงโครไนเซชัน(Synchronization)

2.4.1 ทรัพยากรและทรัพยากรที่ใช้ร่วมกัน(Resource and Share Resource)

ทรัพยากรหรือรีซอส (Resource) คือ สิ่งต่างๆที่ใช้งานโดยทาสก์ซึ่งทรัพยากรอาจจะเป็นพวกอุปกรณ์ I/O (I/O device) อย่างเช่น ปริ้นเตอร์, คีย์บอร์ด หรือพวก โครงสร้าง (structure) เป็นต้น โดยถ้าทรัพยากรสามารถใช้งานร่วมกันได้มากกว่า 1 ทาสก์จะเรียกว่า แชร์รีซอส (Share resource) ซึ่งแต่ละทาสก์จะต้องกำหนดการเข้าใช้งานไปยังแชร์รีซอสเพื่อป้องกันการที่ข้อมูลของแต่ละทาสก์ปนกันซึ่งจะทำให้ผลลัพธ์ผิดไปได้ ซึ่งเรียกว่า mutual exclusion

2.4.2 รีเอนทรานซ์(Reentrancy)

รีเอนทรานซ์ฟังก์ชัน (reentrant function) คือ ฟังก์ชันที่สามารถใช้งานได้มากกว่า 1 ทาสก์โดยไม่ต้องกังวลว่าข้อมูลจะเกิดการเสียหาย โดยรีเอนทรานซ์ฟังก์ชันสามารถทำงานได้ตลอดเวลาและย้อนกลับมาที่หลังได้ โดยข้อมูลไม่สูญหาย รีเอนทรานซ์ฟังก์ชันจะใช้ทั้งตัวแปรโลคัล (local variable) อย่างเช่น ซีพียูรีจิสเตอร์, ตัวแปรในสแตค หรืออาจจะใช้ในการป้องกันข้อมูลเมื่อมีการใช้งานตัวแปรโกลบัล (global variable)

```
void strcpy(char *dest, char *src)
{
    while (*dest++ = *src++) {
        ;
    }
    *dest = NUL;
}
```

รูปที่ 2.8 รีเอนทรานซ์ฟังก์ชัน (reentrant function)

เนื่องจากการก๊อปปี้ค่าอาร์กิวเมนต์ไปยัง strcpy() จะใช้วิธีใส่ไว้ในทาสก์สแตคทำให้ strcpy() สามารถใช้งาน

ได้หลาย ๆ ทาสก์โดยไม่ต้องกลัวว่าทาสก์จะเกิดการปนกันของพอยน์เตอร์ (pointer) ของทาสก์อื่น ๆ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ลบลอด ทำให้เมื่อกลับมาทำงานในทาสก์ที่ความสำคัญต่ำ ค่า Temp จะมีการเปลี่ยนแปลง (หมายเลข 5) ซึ่งปัญหาที่เกิดขึ้นนี้สามารถแก้ไขได้โดยการ

- ประกาศ Temp ให้อยู่ภายใน swap()
- ปิดการใช้งาน อินเตอร์รัพต์ก่อนที่จะทำงาน และเปิดใช้งานหลังจากทำงานเสร็จ
- ใช้เซมาฟอว์ (semaphore)

2.4.3 มิวทอลเอ็กซ์คลูชัน(Mutual Exclusion)

วิธีในการให้ทาสก์แต่ละตัวติดต่อกับทาสก์ตัวอื่น ๆ คือ ผ่านทางการใช้ โครงสร้างข้อมูล (data structure) อย่างเช่น ตัวแปร โกลบอล (global variable) , พอยน์เตอร์ (pointer) , บัฟเฟอร์ (buffer) และอื่น ๆ ใดก็ตามในการใช้ ข้อมูลร่วมกันเพื่อแลกเปลี่ยนข้อมูลจะต้องมั่นใจว่าแต่ละทาสก์ใช้งานข้อมูลแล้วไม่ทำให้ข้อมูลเสียหาย โดยวิธีการในการป้องกันในการใช้งานทรัพยากรร่วมกันจะมีดังนี้

2.4.3.1 การเปิดและปิดอินเตอร์รัพต์

วิธีที่ง่ายและรวดเร็วที่จะป้องกันการเข้าใช้งานทรัพยากรของทาสก์อื่นคือการดิสเอเบิล (disable) การใช้งาน อินเตอร์รัพต์ หลังจากนั้นค่อยทำการ เอนาเบิล (enable) เมื่อทำงานนั้นเสร็จแล้ว แต่ในการใช้การ ดิสเอเบิลการใช้อินเตอร์รัพต์นั้นจะต้องระวังเพราะถ้าเรียกใช้งานนานเกินไป จะมีผลต่อระบบในการตอบสนองอินเตอร์รัพต์ ซึ่งก็คือ interrupt latency โดยจะต้องพิจารณาตรงจุดนี้ เมื่อต้องการจะเปลี่ยนหรือ คัดลอกตัวแปร (variable) บางตัว แต่อย่างไรก็ตามวิธีนี้เป็นทางหนึ่งในการที่ทาสก์จะสามารถใช้งานตัวแปร หรือ โครงสร้างข้อมูลร่วมกับ ISR ซึ่งกรณีนี้ จะต้องดิสเอเบิลการใช้งาน อินเตอร์รัพต์ให้มีช่วงเวลาสั้นที่สุด

```
Disable interrupts;
Access the resource (read/write from/to variables);
Reenable interrupts;
```

รูปที่ 2.11 การดิสเอเบิลและเอนาเบิลอินเตอร์รัพต์ (disable and enable interrupt)

ถ้าใช้งานเคอร์เนลจะสามารถดิสเอเบิลการใช้งานอินเตอร์รัพต์ได้นานเท่ากับที่ไม่ทำให้เกิดปัญหากับ interrupt latency ซึ่งจำเป็นจะต้องรู้ว่าควรดิสเอเบิลการใช้งานอินเตอร์รัพต์ได้นานเท่าไร

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.4.3.2 เทสต์แอนด์เซต(Test-and-Set)

ถ้าไม่ได้ใช้งานเคอร์เนลก็ควรจะใช้การกำหนดค่าตัวแปร โกลบอล ไว้เพื่อใช้ตรวจสอบให้รู้ว่าทรัพยากรมีการใช้งานอยู่หรือไม่ เช่น ถ้ามีการใช้งานทรัพยากรในขณะนั้น ให้กำหนดค่าของตัวแปรโกลบอลเป็น 1 และถ้าไม่มีการใช้งาน ให้กำหนดค่าเป็น 0 ซึ่งวิธีการนี้เรียกว่า Test-And-Set หรือ TAS

```
Disable interrupts;
if ('Access Variable' is 0) {
    Set variable to 1;
    Reenable interrupts;
    Access the resource;
    Disable interrupts;
    Set the 'Access Variable' back to 0;
    Reenable interrupts;
} else {
    Reenable interrupts;
    /* You don't have access to the resource, try back later. */
}
```

รูปที่ 2.12 test and set

2.4.3.3 ปิดและเปิดตัวจัดลำดับงาน

ถ้าทาสก์ไม่ได้ใช้งานตัวแปรหรือโครงสร้างข้อมูลร่วมกับ ISR จะสามารถดิสเอเบิลและเอนาเบิลการจัดลำดับได้ ทำให้ทาสก์ 2 ทาสก์หรือมากกว่านี้สามารถใช้อุปกรณ์ร่วมกันได้โดยไม่ต้องแย่งกัน ซึ่งในขณะที่การจัดลำดับถูกยกเลิก และอินเทอร์รัพต์เปิดใช้งาน เมื่อมีอินเทอร์รัพต์เกิดขึ้นมา ISR จะทำงานโดยทันที หลังจาก ISR ทำงานเสร็จเคอร์เนลจะกลับไปยัง อินเทอร์รัพต์ทาสก์ (interrupt task) หรือทาสก์ที่กำหนดโดย ISR ซึ่งเตรียมพร้อมจะทำงาน และการจัดลำดับจะกลับมาใช้งานอีกครั้งเพื่อทำให้ทาสก์ได้ทำงาน ซึ่งการทำอย่างนี้ใช้งานได้ดี แต่จะเป็นการทำลายจุดประสงค์ของการใช้งานเคอร์เนล

```
void Function (void)
{
    OSSchedLock();
    .
    /* You can access shared data in here (interrupts are recognized) */
    OSSchedUnlock();
}
```

รูปที่ 2.13 การดิสเอเบิลและเอนาเบิลการจัดลำดับ (disable and enable scheduler)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.4.3.4 เซมาฟอว์(Semaphore)

เซมาฟอว์ (Semaphore) ถูกคิดขึ้นมาโดย Edgser Dijkstra ซึ่งจะใช้ในมัลติทาสกิ้งเคอร์เนล โดยจะใช้เพื่อ

- ควบคุมการเข้าใช้งานทรัพยากรที่มีการใช้งานร่วมกัน (mutual exclusion)
- ส่งสัญญาณเมื่อมีการเกิดอีเวนตึ้น
- อนุญาตให้ทาสก 2 ตัวสามารถซิงโครไนซ์ (synchronize) การทำงานได้

เซมาฟอว์เป็นเสมือนคีย์ (key) ที่จะให้การทำงานสามารถทำต่อไปได้ ถ้าเซมาฟอว์ถูกใช้งานอยู่ ทาสกที่ร้องขอการใช้งานจะถูกหยุดการทำงานชั่วคราวและรอไปจนกว่าเซมาฟอว์จะถูกเลิกใช้งานโดย ทาสกที่ใช้งานอยู่ ซึ่งเซมาฟอว์แบ่งออกเป็น 2 ประเภทคือ

- ไบนารีเซมาฟอว์ (binary semaphore) ซึ่งค่าจะมีได้แค่ 0 กับ 1 และ
- เคาน์ติงเซมาฟอว์ (counting semaphore) ซึ่งค่าที่เป็นไปได้คือ 0 กับ 255,65535 หรือ 4292967295 ซึ่งขึ้นกับว่ากระบวนการเซมาฟอว์ที่ใช้งาน ใช้ 8 , 16 หรือ 32 บิต ตามลำดับตามขนาดของ เคอร์เนล โดยระหว่างที่ทาสกกำลังรอการใช้งาน เซมาฟอว์นั้นเคอร์เนลจะเก็บการทำงานของทาสกที่รอ เอาไว้ด้วย

โดยทั่วไปแล้วจะมีการทำงานอยู่ 3 อย่างที่ดำเนินการบนเซมาฟอว์คือ INITIALIZE หรือเรียกว่า CREATE , WAIT หรือ PEND และ SIGNAL หรือ POST ค่าเริ่มต้นของเซมาฟอว์จะถูกกำหนดเมื่อเซมา ฟอว์เริ่มทำงาน และงานใน Waiting List จะว่าง

เมื่องานต้องการใช้งานเซมาฟอว์จะเข้าสู่ WAIT ก่อน ถ้าเซมาฟอว์ว่างค่าของเซมาฟอว์จะมีค่า มากกว่า 0 และเมื่อ เซมาฟอว์ถูกเข้าใช้งานโดยทาสก ค่าของเซมาฟอว์จะถูกลดลง ถ้าค่าของเซมาฟอว์เป็น 0 ทาสกที่ต้องการเข้ามาใช้งานจะเข้าไปอยู่ใน Waiting List และเคอร์เนลสามารถกำหนดไทม์เอาท์ (timeout) ได้ ถ้าเซมาฟอว์ยังไม่ว่างในเวลาที่กำหนดไว้โดยทาสกที่ร้องขอไป ทาสกนั้น ๆ จะเตรียมพร้อมทำงานและ โค้ดแสดงความผิดพลาด (error code) ซึ่งแสดงให้เห็นว่าเกิดไทม์เอาท์จะถูกส่งกลับไปยังผู้เรียกใช้งาน

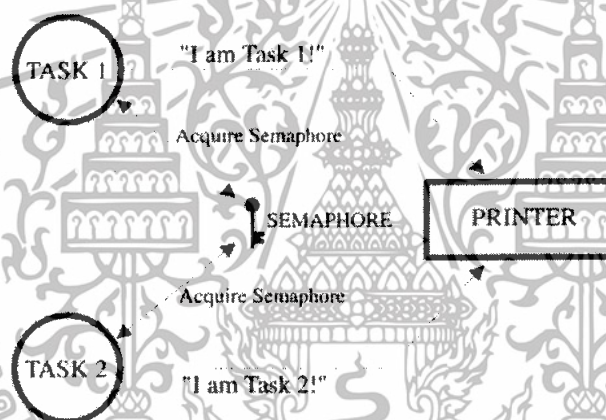
ทาสกจะเลิกการใช้งานเซมาฟอว์โดยการทำงาน SIGNAL ถ้าไม่มีทาสกรอใช้งาน ค่าของเซมาฟอว์ จะถูกเพิ่มค่าขึ้น แต่ถ้ามีทาสกรอการใช้งานเซมาฟอว์ โดยที่ได้มีทาสกตัวหนึ่งพร้อมที่จะทำงานแล้วแต่ค่า

ของเซมาฟอรัยยังไม่ได้เพิ่มค่า คีย์ (key) จะถูกกำหนดให้ทาสก์ที่รอการใช้งาน ซึ่งขึ้นอยู่กับคอร์เนลว่าจะให้ทาสก์ตัวไหนได้รับเซมาฟอร์โดยขึ้นกับ

- ทาสก์ที่มีความสำคัญสูงสุด ที่รอการใช้งาน เซมาฟอร์และ
- ทาสก์ที่มาร้องขอเซมาฟอร์ลำดับแรก ๆ หรือ First In First Out (FIFO)

โดยคอร์เนลบางตัวสามารถกำหนดได้เมื่อเซมาฟอร์เริ่มต้นทำงาน

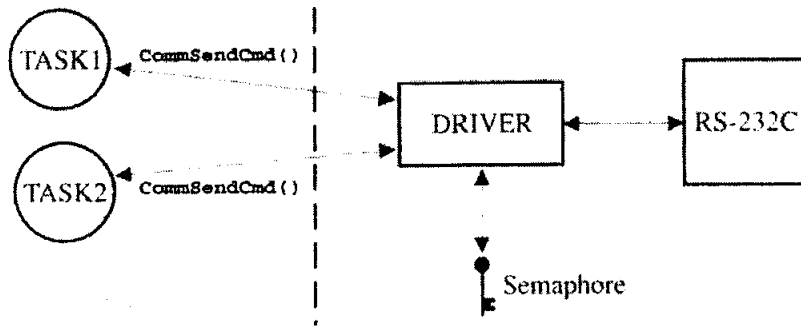
เซมาฟอร์มีประโยชน์เมื่อทาสก์ใช้งาน อุปกรณ์ I/O ร่วมกัน อย่างเช่น ถ้าไม่มีการใช้งานเซมาฟอร์ กรณีที่มีทาสก์ 2 ตัวเข้าใช้งานปริ้นเตอร์พร้อมกัน ทำให้เมื่อปริ้นออกมาแล้ว ข้อความที่ได้ออกมาอาจจะปนกันก็ได้ ซึ่งในกรณีนี้ เมื่อมีการใช้งานเซมาฟอร์ โดยกำหนดค่าเริ่มต้นเป็น 1 (ไบนารีเซมาฟอร์) กฎในการเข้าใช้ปริ้นเตอร์ คือ ต้องได้รับเซมาฟอร์ของทรัพยากรก่อนที่จะเข้าใช้งาน



รูปที่ 2.14 การเข้าใช้ทรัพยากรโดยการใช้งานเซมาฟอร์

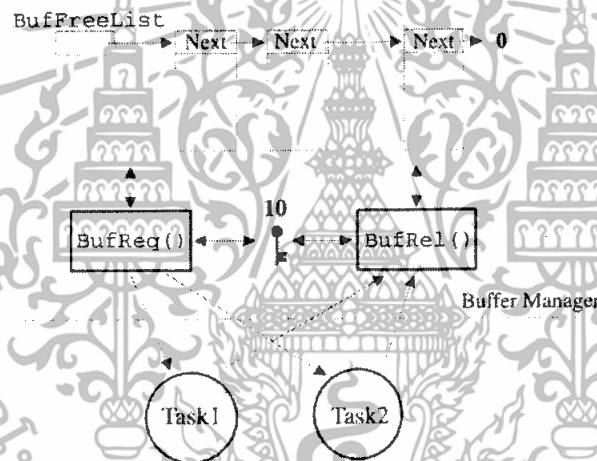
จากภาพจะแสดงเซมาฟอร์ในรูปของกุญแจ ซึ่งจะใช้ในการเข้าใช้งานทรัพยากรทำให้รู้ลำดับการเข้าใช้งานทรัพยากร โดยดูจาก เซมาฟอร์ซึ่งจะดีกว่าถ้าเซมาฟอร์ได้ถูกเอนแคปซูลเลต (encapsulate) และทำให้แต่ละทาสก์ไม่รู้ว่าได้รับเซมาฟอร์เมื่อเข้าใช้งานทรัพยากร ตัวอย่างเช่น อุปกรณ์หรือดีไวซ์ (device) เมื่อทาสก์ต้องการส่งคำสั่งไปยังอุปกรณ์ ก็จะทำการเรียกฟังก์ชัน โดยฟังก์ชันจะเรียกใช้งานเซมาฟอร์ต่ออีกที

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.15 การขออนเซมาฟอรจากทาสก์

เริ่มต้นทาสก์ 1 ติดต่อเข้ามาโดยจะเรียกผ่านฟังก์ชันเพื่อจะได้เซมาฟอรมา จากนั้นอุปกรณ์จะทำงานตามการทำงานของทาสก์ ถ้ามีทาสก์อื่นได้ส่งคำสั่งเพื่อจะใช้งาน แต่พบว่าพอร์ต (port) กำลังถูกใช้งานอยู่ ทาสก์ที่เรียกมาทีหลังจะถูกหยุดการทำงานชั่วคราวจนกว่าเซมาฟอรจะว่าง



รูปที่ 2.16 เคาท์ติง เซมาฟอร (counting semaphore)

เคาท์ติงเซมาฟอร (counting semaphore) จะใช้เมื่อทรัพยากรสามารถใช้งานได้มากกว่า 1 ทาสก์ขึ้นไปในเวลาเดียวกัน อย่างเช่นการจัดการของบัฟเฟอร์พูล (buffer pool) สมมติให้เริ่มต้นบัฟเฟอร์พูลมีจำนวน 10 บัฟเฟอร์ โดยทาสก์สามารถใช้งานบัฟเฟอร์ได้โดยการเรียกใช้ BufReq() และถ้าบัฟเฟอร์ไม่ได้ใช้งานต่ออีกจะคืนด้วยการใช้ BufRel()

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

BUF *BufReq(void)
{
    BUF *ptr;
    Acquire a semaphore;
    Disable interrupts;
    ptr = BufFreeList;
    BufFreeList = ptr->BufNext;
    Enable interrupts;
    return (ptr);
}

```

```

void BufRel (BUF *ptr)
{
    Disable interrupts;
    ptr->BufNext = BufFreeList;
    BufFreeList = ptr;
    Enable interrupts;
    Release semaphore;
}

```

รูปที่ 2.17 การจัดการบัฟเฟอร์โดยการใช้งานเซมาฟอร์

ในการใช้งานบัฟเฟอร์ (ในที่นี้มี 10 บัฟเฟอร์) จะสามารถรองรับการร้องขอได้ 10 อย่างเพราะว่าเสมือนกับมี 10 คีย์ เมื่อเซมาฟอร์ถูกใช้งานทั้งหมด (บัฟเฟอร์เต็ม) และมีทาสก์ต้องการใช้งาน ทาสก์ตัวนั้นจะถูกหยุดการทำงานชั่วคราวจนกว่าเซมาฟอร์จะว่าง เมื่อทาสก์ทำงานเสร็จสิ้น จะทำการคืนบัฟเฟอร์โดยการเรียก BufRel() ซึ่งบัฟเฟอร์จะทำการแทรกเข้าไปในลิงค์ลิสต์ (link list) ของบัฟเฟอร์ก่อนจะทำการคืนเซมาฟอร์

โดยทั่วไปแล้วมักจะใช้งานเซมาฟอร์ แต่ในงานบางอย่าง เช่น การเข้าใช้งานตัวแปรที่ใช้งานร่วมกัน (share variable) ทั่ว ๆ ไปมักจะใช้การการดิสเอเบิลและเอนาเบิลอินเตอร์รัพต์เพราะการใช้เซมาฟอร์จะเสียเวลากว่า

2.4.4 เดดล็อก(Deadlock)

เดดล็อก (Deadlock) หรือ deadly embrace คือกรณีที่ทาสก์ 2 ตัวต่างรอการใช้งานทรัพยากรที่ใช้งานโดยอีกตัว เช่น ทาสก์ T1 ใช้งานทรัพยากร R1 และทาสก์ T2 ใช้งานทาสก์ T2 ถ้า T1 ต้องการใช้งาน R2 และ T2 ต้องการใช้ R1 จะเกิดเดดล็อกขึ้น ซึ่งการแก้ปัญหาทำได้โดย

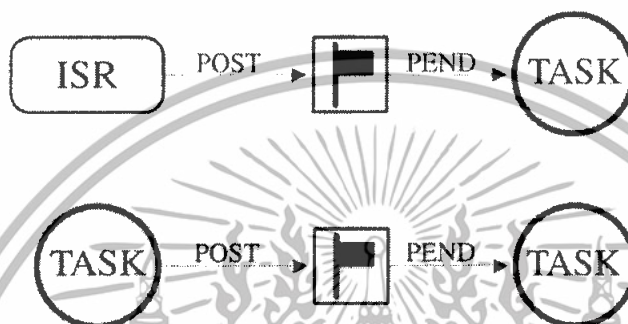
- ก่อนทำงานต้องได้รับทรัพยากรก่อน
- ให้การใช้งานทรัพยากรตามลำดับ
- ในการเลิกใช้งานทรัพยากรต้องเลิกใช้งานแบบย้อนลำดับ

เคอร์เนลทั่วไปจะอนุญาตให้กำหนดไทม์เอาท์ (timeout) ได้เมื่อใช้งานเซมาฟอร์ซึ่งจะช่วยในการป้องกันเกิด เดดล็อก ถ้าทาสก์ที่ต้องการใช้งานเซมาฟอร์ไม่ได้รับในเวลาที่ต้องการ ทาสก์นั้นจะกลับไปเอกสารเป็นเอกสารที่ส่งวนเวียนสำหรับการใช้งานเพื่อการศึกษาเท่านั้น เมื่อนุญาตให้ไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ทำงานต่อ และโค้ดแสดงความผิดพลาด (error code) จะถูกส่งไปเพื่อไปแจ้งว่าเกิดไหมเอ๋ที่ขึ้น ซึ่งโค้ดแสดงความผิดพลาดตรงนี้จะช่วยป้องกันที่การที่ทาสก์คิดว่าได้รับทรัพยากรนั้น โดยเดดล็อกมักจะเกิดกับระบบมัลติทาสก์ที่มีขนาดใหญ่ ไม่ค่อยจะปรากฏในระบบสมองกลฝังตัว (embedded system)

2.4.5 ชิงโครไนซ์(Synchronization)

ทาสก์สามารถชิงโครไนซ์กับ ISR หรือกับทาสก์ที่ไม่มีการเปลี่ยนแปลงข้อมูลได้ โดยการใช้งานเซมาฟอร์



รูปที่ 2.18 การชิงโครไนซ์ของทาสก์และ ISR

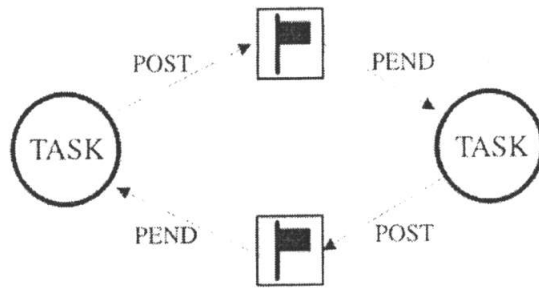
ในกรณีนี้เซมาฟอร์จะถูกแสดงในรูปธง (flag) เพื่อแสดงว่ามันได้รับสัญญาณจากอีเวนท์ โดยเมื่อใช้กระบวนการชิงโครไนซ์เซชัน (synchronization) เซมาฟอร์จะถูกกำหนดค่าเริ่มต้นให้เป็น 0 ซึ่งการใช้เซมาฟอร์แบบนี้จะเรียกว่า unilateral rendezvous โดยทาสก์จะกำหนดค่าเริ่มต้นให้กับการทำงานของ I/O และรอเข้าใช้งานเซมาฟอร์ เมื่อการทำงานของ I/O เสร็จสิ้น ISR หรือทาสก์อื่นจะส่งสัญญาณไปยังเซมาฟอร์และทาสก์ที่รอใช้งานเซมาฟอร์จะได้เข้ามาใช้งาน

ถ้าเคอร์เนลรองรับการทำงานของเคาท์ดาวน์เซมาฟอร์ เซมาฟอร์จะทำการเก็บเหตุการณ์ต่างๆที่ยังไม่ได้จัดการเอาไว้ ซึ่งถ้ามีมากกว่า 1 ทาสก์ที่รอการทำงาน เคอร์เนลจะต้องจัดการให้ดังนี้

- ทาสก์ที่มีความสำคัญสูงสุดที่รอ ได้ทำงาน และ
- ทาสก์ที่มารอก่อนได้ทำงาน

ซึ่งก็ขึ้นอยู่กับแอปพลิเคชันซึ่งเป็นไปได้ว่าจะมีทาสก์หรือ ISR มากกว่า 1 เหตุการณ์ที่เกิดขึ้นส่งสัญญาณไป ทาสก์ 2 ตัวสามารถชิงโครไนซ์งานของมันได้โดยการใช้ 2 เซมาฟอร์ซึ่งเป็นดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.19 ทาสก์ซิงโครไนซ์กิจกรรมร่วมกัน

โดยเรียกว่า bilateral rendezvous ซึ่งคล้ายกับ unilateral rendezvous เว้นแต่ว่าทั้ง 2 ทาสก์จะต้องซิงโครไนซ์กับอีกทาสก์หนึ่งก่อนจะทำงานต่อไป

```

Task1 ()
{
    for (;;) {
        Perform operation;
        Signal task #2;
        Wait for signal from task #2;
        Continue operation;
    }
}

Task2 ()
{
    for (;;) {
        Perform operation;
        Signal task #1;
        Wait for signal from task #1;
        Continue operation;
    }
}

```

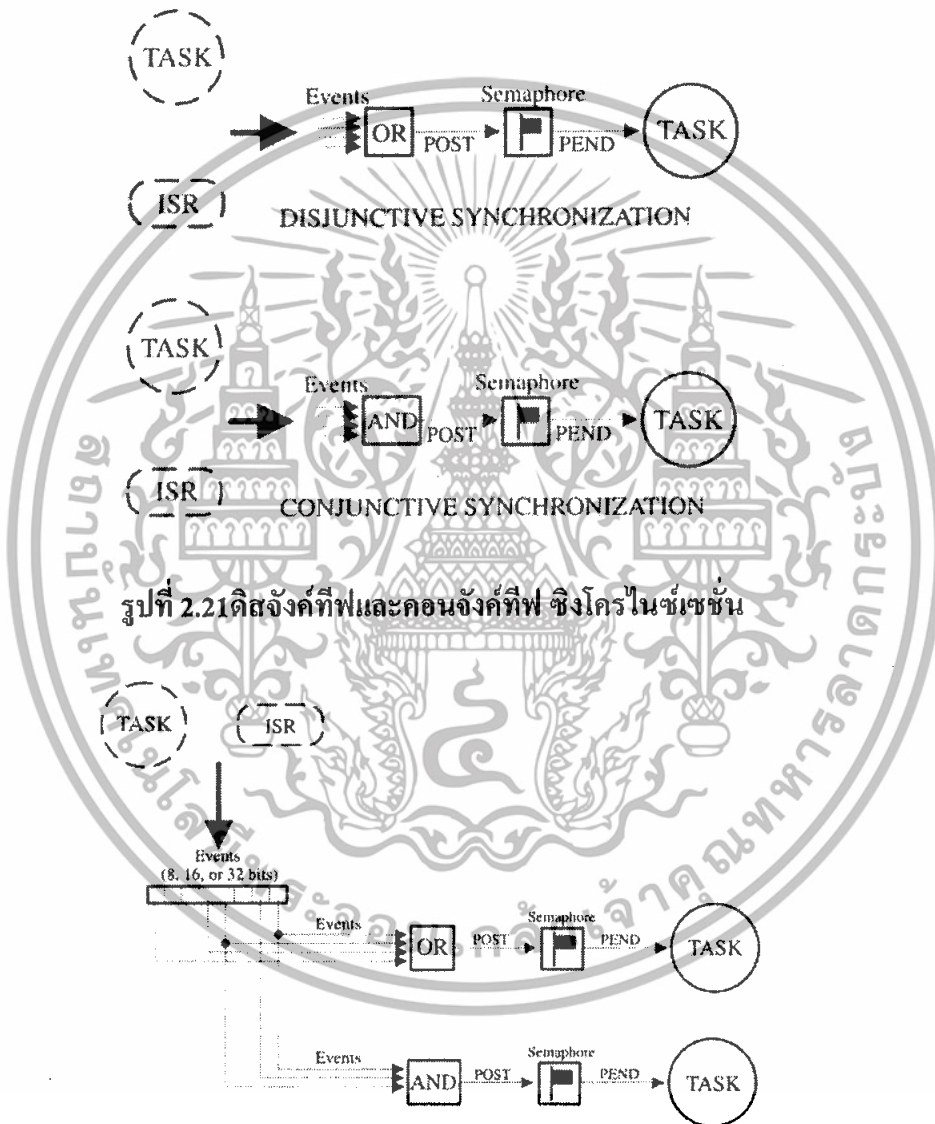
รูปที่ 2.20 code bilateral rendezvous

มีทาสก์ 2 ตัวทำงาน เมื่อทาสก์ตัวแรกทำงานไปถึงจุด ๆ หนึ่งก็จะส่งสัญญาณไปหาเซมาฟอร์ เพื่อให้ทาสก์ 2 ได้ใช้งาน และจะหยุดรอให้ทาสก์ 2 ส่งสัญญาณกลับมา และในลักษณะเดียวกันทาสก์ 2 ทำงานไปถึงจุด ๆ หนึ่งก็จะส่งสัญญาณไปเพื่อให้ ทาสก์ 1 ได้ทำงานและรอสัญญาณตอบกลับมา ซึ่งในลักษณะนี้สแตททั้งสองจะมีการซิงโครไนซ์กัน แต่ ISR ไม่สามารถทำงานในลักษณะ bilateral rendezvous ได้ เพราะ ISR ไม่สามารถหยุดรอเซมาฟอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.4.6 อีเวนท์แฟลก(Event Flags)

อีเวนท์แฟลก (Event Flag) จะใช้เมื่อทาสก์ต้องการซิงโครไนซ์กับอีเวนท์ที่เกิดขึ้นหลาย ๆ เหตุการณ์ โดยทาสก์สามารถ ซิงโครไนซ์เมื่อมีอีเวนท์ใดอีเวนท์หนึ่งจากทั้งหมดเกิดขึ้น ซึ่งจะเรียกว่า ดิสจังก์ทีฟ ซิงโครไนซ์เซชัน (disjunctive synchronization) ซึ่งเป็นลอจิก OR และทาสก์สามารถซิงโครไนซ์เมื่อทุกเหตุการณ์ได้เกิดขึ้นซึ่งเรียกว่าคอนจังก์ทีฟ ซิงโครไนซ์เซชัน (conjunctive synchronization) ซึ่งเป็นลอจิก AND ซึ่ง ดิสจังก์ทีฟและคอนจังก์ทีฟ ซิงโครไนซ์เซชัน แสดงได้ดังนี้



รูปที่ 2.22 อีเวนท์แฟลก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.4.7 การสื่อสารระหว่างทาสก์

ในบางครั้งทาสก์หรือ ISR จำเป็นที่จะต้องสื่อสารข้อมูลข่าวสาร (informaiton) กับทาสก์อื่น ๆ ซึ่งการแลกเปลี่ยนข้อมูลข่าวสารเรียกว่าอินเตอร์ทาสก์คอมมูนิเคชัน (intertask communication) โดยข่าวสารสามารถสื่อสารระหว่างทาสก์ได้ 2 วิธีคือ ผ่านข้อมูลที่เป็น โกลบอล (global data) หรือ โดยการส่งข้อความหรือแมสเสจ (message)

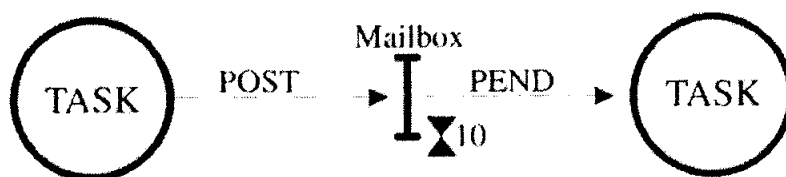
เมื่อใช้งานตัวแปร โกลบอลแต่ละทาสก์หรือ ISR จะต้องมั่นใจว่ามีผู้ใช้งานตัวแปรนั้นเพียงผู้เดียว โดยถ้าเป็น ISR วิธีที่จะทำให้มั่นใจคือใช้การดิสเอเบิลอินเตอร์รัพต์ (disable interrupt) ถ้าเป็นทาสก์ 2 ตัวใช้งานข้อมูลร่วมกัน สามารถใช้งานได้โดย ดิสเอเบิลและเอนาเบิลอินเตอร์รัพต์หรือใช้เซมาฟอร์ แต่ทาสก์สามารถสื่อสารกับ ISR ได้ทางเดียวคือใช้ตัวแปร โกลบอล ซึ่งทาสก์จะไม่รู้ว่าค่าของตัวแปร โกลบอลเปลี่ยนโดย ISR หรือ ไม่ เว้นแต่ว่า ISR จะส่งสัญญาณไปยังทาสก์ โดยใช้เซมาฟอร์หรือ เว้นแต่ว่าทาสก์จะคอยตรวจสอบค่าของตัวแปรเรื่อย ๆ ซึ่งวิธีแก้ไขในกรณีนี้สามารถใช้ แมสเสจเมลบ็อกซ์ (message mailbox) หรือแมสเสจคิว (message queue) ได้

2.4.7.1 แมสเสจ เมลล์บ็อกซ์(Message Mailbox)

แมสเสจ (message) จะสามารถส่งไปยังทาสก์โดยผ่านบริการของเคอร์เนลได้ โดยการแลกเปลี่ยนแมสเสจนี้จะเรียกว่า แมสเสจเมลบ็อกซ์ (Message Mailbox) ซึ่งมักจะเป็นตัวแปรใช้เก็บพอยน์เตอร์ (pointer-size variable) การใช้บริการผ่านเคอร์เนลนั้นทาสก์หรือ ISR สามารถสะสมแมสเสจ (พอยต์เตอร์) เข้าไปในเมลบ็อกซ์โดย 1 หรือหลาย ๆ ทาสก์สามารถรับแมสเสจได้ผ่านทางบริการของเคอร์เนล ซึ่งทั้งทาสก์ฝั่งส่งและทาสก์ฝั่งรับจะมีพอยน์เตอร์ชี้ไปที่เดียวกัน

Waiting List จะเกี่ยวข้องกับแต่ละเมลบ็อกซ์ ในกรณีที่มีทาสก์มากกว่า 1 ต้องการแมสเสจผ่านทางเมลบ็อกซ์ ทาสก์ที่ต้องการแมสเสจจากเมลบ็อกซ์ที่ว่างจะถูกหยุดการทำงานชั่วคราวและจะถูกใส่เข้าไปยัง Waiting List จนกว่าจะได้รับแมสเสจซึ่งทั่วไปแล้วเคอร์เนลจะมีไทม์เอาท์สำหรับทาสก์ที่รอแมสเสจ ถ้าทาสก์ไม่ได้รับแมสเสจก่อนเวลาไทม์เอาท์ที่กำหนด ทาสก์ตัวนั้นจะพร้อมที่จะทำงานและจะส่งโค้ดแสดงความคิดเห็น (error code) กลับไป เมื่อแมสเสจได้ถูกใส่เข้าไปในเมลบ็อกซ์แล้วทาสก์ที่จะได้รับแมสเสจคือ ทาสก์ที่มีค่าความสำคัญสูงที่สุด (คิดความสำคัญเป็นหลัก) หรือทาสก์ตัวแรกที่ร้องขอแมสเสจ (first-in-first-out)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.23 เมสเสจเมลบ็อกซ์

จากภาพแสดงการนำเมสเสจใส่เข้าไปในเมลบ็อกซ์ ซึ่งแสดงแทนเป็นไอ-บีม (I-beam) และไทม์เอาท์แสดงเป็น นาฬิกาทราย (hourglass) ตัวเลขเวลาที่อยู่ถัดไปจากนาฬิกาทราย คือจำนวนของคล็อกทริก (clock tick) เคอร์เนลจะจัดการการให้บริการเมลบ็อกซ์ ดังนี้

- ค่าเริ่มต้นของเมลบ็อกซ์ซึ่งเริ่มต้นนั้นอาจจะมีเมสเสจหรือไม่ก็ได้
- ใส่เมสเสจเข้าไปใน เมลบ็อกซ์ (POST)
- รอเมสเสจที่จะใส่เข้ามาใน เมลบ็อกซ์ (PEND)
- รับเมสเสจจากเมลบ็อกซ์เมื่อมีเมสเสจ (ACCEPT) ถ้าเมลบ็อกซ์มีเมสเสจก็จะส่งจากเมลบ็อกซ์ไป และรีเทิร์นโค้ด (return code) จะใช้ในการแจ้งเตือนตัวเรียกใช้งานเกี่ยวกับผลลัพธ์ของการเรียกใช้งาน

เมสเสจเมลบ็อกซ์สามารถทำหน้าที่คล้ายไบนารีเซมาฟออร์ (binary semaphore) โดยมีเมสเสจในเมลบ็อกซ์จะแสดงว่าสามารถใช้งานทรัพยากรได้และ ถ้าเมลบ็อกซ์ว่าง แสดงว่าทรัพยากรกำลังถูกใช้งานโดยทาสก์ตัวอื่น

2.4.7.2 เมสเสจคิว(Message Queue)

เมสเสจคิว (message queue) จะใช้ในการส่งเมสเสจโดยสามารถส่งได้มากกว่า 1 เมสเสจไปยังทาสก์ซึ่งเมสเสจคิวมักจะเป็นอาร์เรย์ (array) ของเมลบ็อกซ์ โดยทาสก์หรือ ISR สามารถใส่เมสเสจ (พอยท์เตอร์) เข้าไปในเมสเสจคิวโดยผ่านบริการของเคอร์เนลและเช่นเดียวกันทาสก์มากกว่า 1 สามารถรับเมสเสจผ่านบริการของเคอร์เนลได้ ซึ่งทั้งทาสก์ฝั่งส่งและฝั่งรับจะมีพอยท์เตอร์ชี้ไปที่เดียวกัน ซึ่งทั่วไปแล้วเมสเสจแรกๆที่ใส่เข้าไปจะเป็นเมสเสจแรกๆที่ออกมา (FIFO)

การทำงานคล้ายกับเมลบ็อกซ์ซึ่ง Waiting List จะเกี่ยวข้องกับแต่ละเมสเสจคิวในกรณีที่มีทาสก์มากกว่า 1 ต้องการรับเมสเสจจากคิว ทาสก์ที่ต้องการเมสเสจจากคิวที่ว่างจะหยุดการทำงานชั่วคราวและใส่เข้าไปใน Waiting List จนกว่าจะได้รับเมสเสจซึ่งเคอร์เนลจะมีการกำหนดไทม์เอาท์สำหรับทาสก์ที่รอเอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แมสเสจ ถ้าทาสก์ไม่ได้รับแมสเสจก่อนที่จะหมดเวลา ทาสก์นั้นจะพร้อมทำงานและจะส่งโค้ดแสดงคามผิดพลาด (error code) กลับไป เมื่อแมสเสจใส่เข้าไปในคิว จะมีทั้งทาสก์ที่มีค่าความสำคัญสูงสุดหรือทาสก์ที่รอเป็นลำดับแรก มารอรับแมสเสจ



รูปที่ 2.24แมสเสจคิว

แสดง ISR ซึ่งใส่แมสเสจเข้าไปในคิว ซึ่งคิวจะแสดงโดยภาพไอ-บีม (I-beam) สองอัน เลข 10 แสดงจำนวน แมสเสจที่สามารถสะสมอยู่ในคิวได้ และ เลข 0 แสดงว่าทาสก์สามารถรอแมสเสจได้ตลอดจนกว่าแมสเสจจะมาถึง เคอร์เนลจะจัดการบริการแมสเสจคิวได้ดังนี้

- เริ่มต้นคิวจะถูกกำหนดให้ว่าง
- การใส่แมสเสจเข้าไปในคิว (POST)
- การรอแมสเสจที่ถูกใส่เข้าไปในคิว (PEND)
- รับแมสเสจถ้าอยู่ในคิว (ACCEPT) ถ้าคิวมีแมสเสจอยู่ จะถูกนำออกจากคิวและรีเทิร์นโค้ด (return code) จะใช้ในการแจ้งเตือนตัวเรียกใช้งาน เกี่ยวกับผลลัพธ์ของการเรียกใช้งาน

2.5 อินเทอร์รัพต์(Interrupt)

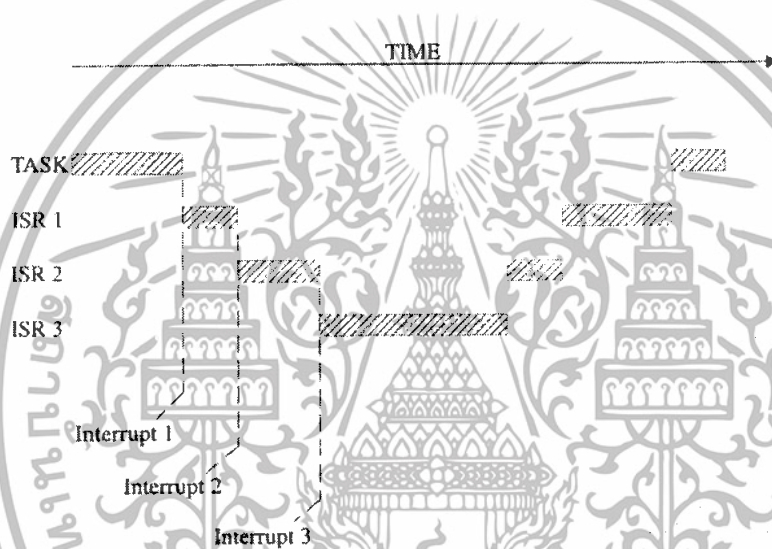
อินเทอร์รัพต์ (interrupt) เป็นกระบวนการทางฮาร์ดแวร์ (Hardware) ซึ่งใช้ในการแจ้งให้ซีพียูทราบว่าได้มีอซิงโครนัส อีเวนต์ (asynchronous event) เกิดขึ้น เมื่อแยกได้ว่าเกิดอะไรขึ้น ซีพียูจะทำการเก็บค่าคอนเท็กซ์ (context) บางส่วนหรือทั้งหมดเอาไว้และจะไปทำในกระบวนการย่อยที่เรียกว่า อินเทอร์รัพต์ เซอวิสรูทีน (Interrupt service routine) หรือ ISR โดยเมื่อทำ ISR เสร็จแล้วจะกลับไปยัง

- แแบ็กกราวด์ (background) สำหรับ ระบบฟอร์กราวด์/แบ็กกราวด์ (foreground/background system)
- ทาสก์ที่เกิดอินเทอร์รัพต์สำหรับนอนพรีเอมทีฟเคอร์เนล (non-preemptive kernel)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- ทาสก์ที่มีความสำคัญสูงสุด ซึ่งพร้อมที่จะทำงานในพรีเอมทีฟเคอร์เนล (preemptive kernel)

อินเทอร์รัพต์จะช่วยให้ไมโคร โปรเซสเซอร์จัดการกับเหตุการณ์ที่เกิดขึ้น โดยไมโคร โปรเซสเซอร์สามารถจะไม่สน อินเทอร์รัพต์ที่เกิดขึ้นได้โดยการใช้คำสั่งคิเสเบิลอินเทอร์รัพต์และเอนาเบิลอินเทอร์รัพต์ในระบบเรียลไทม์ อินเทอร์รัพต์จะต้องคิเสเบิลให้น้อยและระยะเวลาสั้นที่สุด ซึ่งการคิเสเบิลอินเทอร์รัพต์จะมีผลต่อ interrupt latency และอาจจะทำให้อินเทอร์รัพต์ทำงานผิดพลาดได้ โปรเซสเซอร์สามารถจะทำอินเทอร์รัพต์แบบซ้อนกันได้ (interrupt nest) ซึ่งหมายความว่าขณะที่กำลังทำงานในอินเทอร์รัพต์นั้น โปรเซสเซอร์สามารถจะให้บริการกับอินเทอร์รัพต์ตัวอื่นที่สำคัญกว่าได้



รูปที่ 2.25 อินเทอร์รัพต์แบบซ้อน

อินเทอร์รัพต์สามารถแบ่งออกได้เป็น 2 ชนิด คือ

- มาคส์เอเบิลอินเทอร์รัพต์ (Maskable Interrupt) การอินเทอร์รัพต์ที่ซีพียูสามารถปฏิเสธได้ ซึ่งขึ้นอยู่กับว่าได้เปิดใช้งานอินเทอร์รัพต์หรือไม่

- นอนมาคส์เอเบิลอินเทอร์รัพต์ (Non-Maskable Interrupt) การอินเทอร์รัพต์ที่ซีพียูไม่สามารถปฏิเสธ ซึ่งเมื่อมี อินเทอร์รัพต์ประเภทนี้เข้ามา ซีพียูจะต้องเข้ามาทำงานทันที

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.5.1 ความล่าช้าจากอินเทอร์รัพต์(Interrupt Latency)

สิ่งสำคัญที่สุดของเรียลไทม์เคอร์เนล คือ เวลาทั้งหมดที่อินเทอร์รัพต์ถูกคิสดิสเอเบิลซึ่งเรียลไทม์ซิสเต็มจะคิสดิสเอเบิลอินเทอร์รัพต์กับคริติคอลล โค้ด (critical code) และจะเปิดใช้งานอีกครั้งเมื่อคริติคอลล โค้ด ได้ทำงานเสร็จแล้ว โดยยังคงคิสดิสเอเบิลอินเทอร์รัพต์นานเท่าไรก็จะมี Interrupt latency มากยิ่งขึ้น

2.5.2 การตอบสนองต่ออินเทอร์รัพต์(Interrupt Response)

การตอบสนองของอินเทอร์รัพต์ (interrupt response) คือ ช่วงเวลาระหว่างที่รับอินเทอร์รัพต์เข้ามา และเริ่มทำงานในส่วน โค้ดของอินเทอร์รัพต์ ซึ่งซีพียูจะทำการเก็บค่าของคอนเท็กซ์ (context) ก่อนที่จะทำงานใน โค้ดของ ISR

- สำหรับระบบฟอร์กราวด์/แบ็กกราวด์ (foreground/background)และนอนพรีเอมทีฟเคอร์เนล (non-preemptive kernel) นั้นโค้ดใน ISR จะทำงานทันทีเมื่อเก็บค่าคอนเท็กซ์ของโปรเซสเซอร์เสร็จเรียบร้อยแล้ว

- สำหรับพรีเอมทีฟเคอร์เนล (preemptive kernel) จะมีการเรียกใช้งานฟังก์ชันพิเศษซึ่งจะต้องเรียกเพื่อแจ้งบอกให้เคอร์เนลรู้ว่า ISR กำลังทำงานอยู่และเคอร์เนลสามารถเก็บ การทำงานของอินเทอร์รัพต์แบบซ้อนกัน ได้ ซึ่งจะใช้เวลาานกว่าฟอร์กราวด์/แบ็กกราวด์และนอนพรีเอมทีฟเคอร์เนล เพราะมีขั้นตอนในการทำงานเยอะกว่า

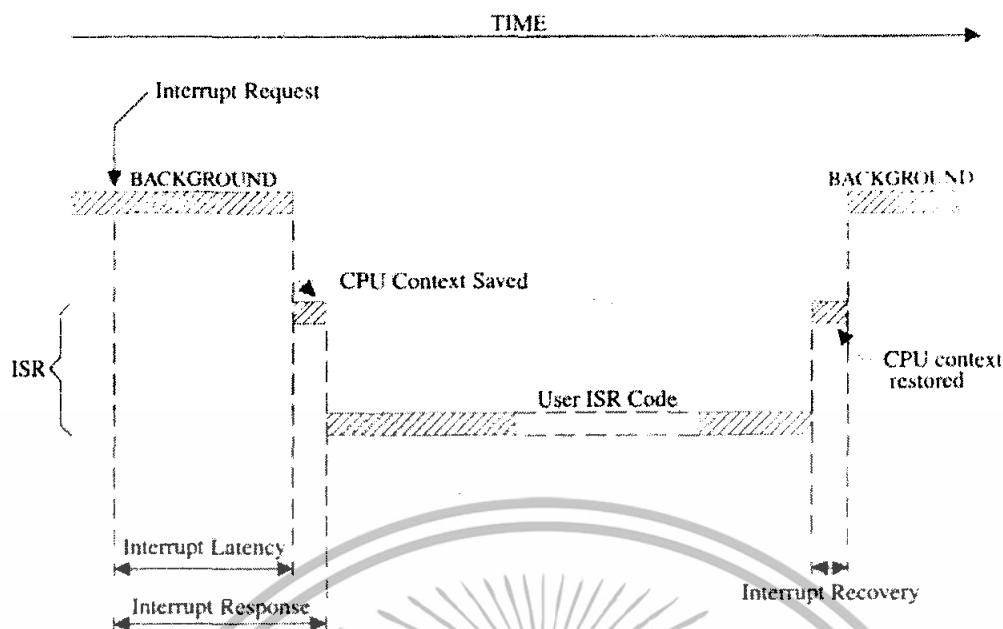
2.5.3 การออกจากอินเทอร์รัพต์(Interrupt Recovery)

อินเทอร์รัพต์รีคัฟเวอรี่ (interrupt recovery) คือเวลาที่โปรเซสเซอร์ต้องการที่จะคืนค่าไปยังโค้ดส่วนที่ถูกอินเทอร์รัพต์

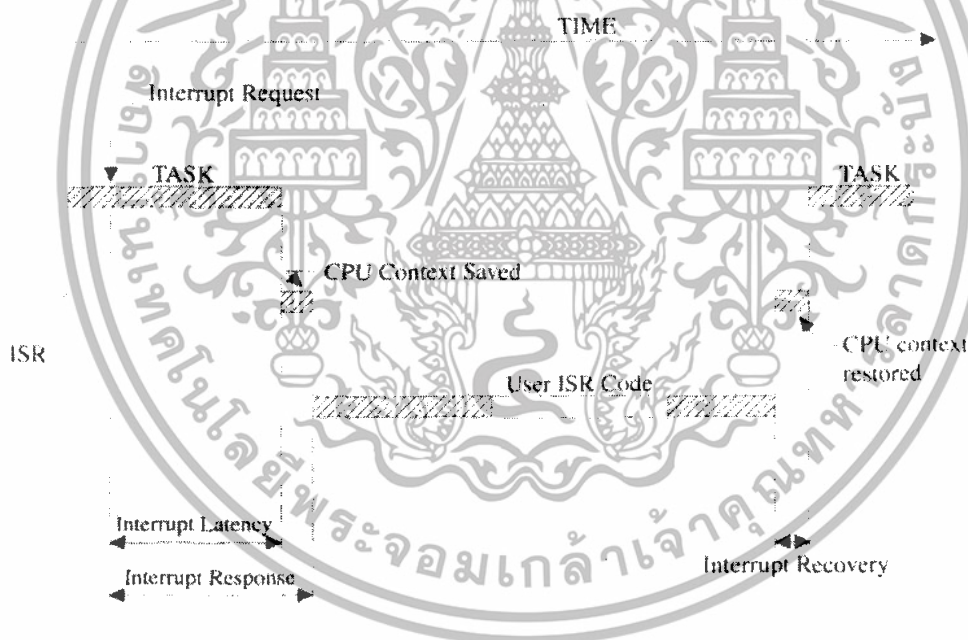
- ระบบฟอร์กราวด์/แบ็กกราวด์ (foreground/background system) และอินเทอร์รัพต์รีคัฟเวอรี่ของนอนพรีเอมทีฟเคอร์เนล (non-preemptive kernel) จะทำงานคล้าย ๆ กัน คือ ดึงค่าของคอนเท็กซ์คืนให้ซีพียูและคืนค่ากลับไปยังทาสก์ที่ถูกอินเทอร์รัพต์

- สำหรับพรีเอมทีฟเคอร์เนล (preemptive kernel) นั้นอินเทอร์รัพต์รีคัฟเวอรี่จะซับซ้อนกว่า โดยฟังก์ชันที่จัดเตรียมโดยเคอร์เนลจะถูกเรียกเมื่อเสร็จสิ้น ISR ซึ่งจะช่วยเคอร์เนลในการเลือกกรณีเมื่อมีอินเทอร์รัพต์ซ้อนกัน (interrupt nest) โดยจะเลือกทาสก์ที่มีความสำคัญสูงสุดที่และพร้อมจะทำงานมาทำต่อจาก ISR

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

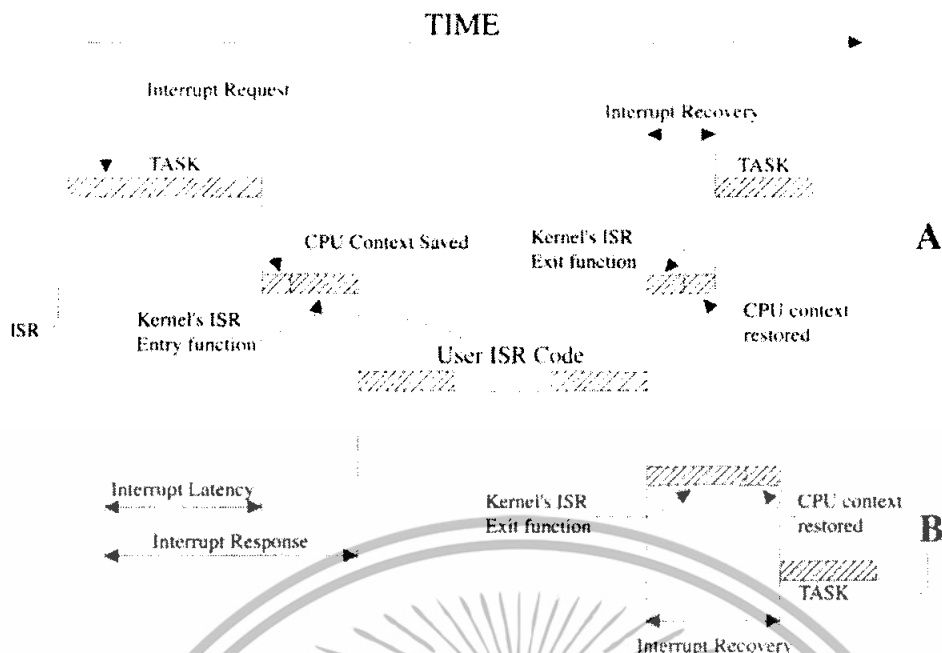


รูปที่ 2.26 Interrupt latency , response และ recovery (foreground/background)



รูปที่ 2.27 Interrupt latency , response และ recovery (non-preemptive kernel)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.28 Interrupt latency , response และ recovery (preemptive kernel)

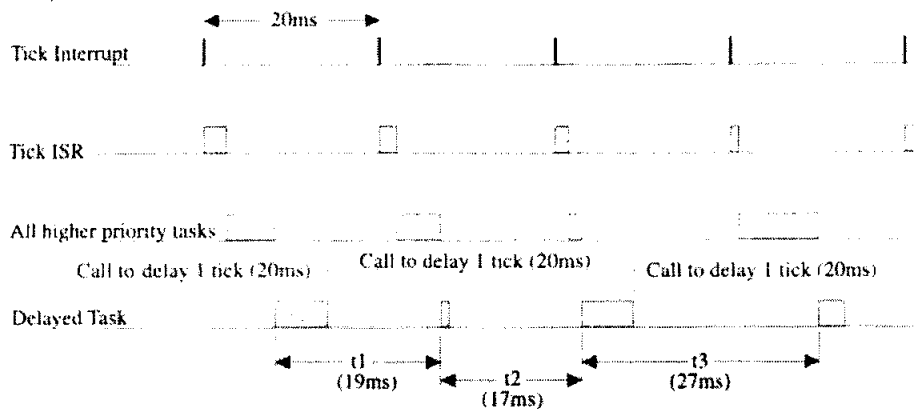
2.5.4 คลอกติก(Clock Tick)

clock tick คือ อินเทอร์เน็ตพิเศษซึ่งจะเกิดขึ้นเป็นรอบ ๆ ที่แน่นอน ซึ่งมีจะสร้างมาในช่วงระหว่าง 10 ถึง 200 ms ซึ่ง clock tick interrupt จะช่วยเคอร์เนลในการดีเลย์ทาสก์ (delay task) เพื่อหาจำนวน clock tick และช่วยจัดการ ไทม์เอาท์ เมื่อทาสก์รอให้อีเวนต์ให้เกิดขึ้น ถ้ามี tick rate ที่สูงจะทำให้ระบบทำงานหนักขึ้นได้

เคอร์เนลจะสามารถให้ทาสก์ดีเลย์ได้เพื่อหาให้จำนวนของ clock tick ที่แน่นอน ซึ่งเมื่อเกิดclock tick ครั้งหนึ่งก็จะมีดีเลย์แต่ก็ไม่แน่นอนไปว่าจะเกิดดีเลย์ทุกครั้งที่เกิด clock tick

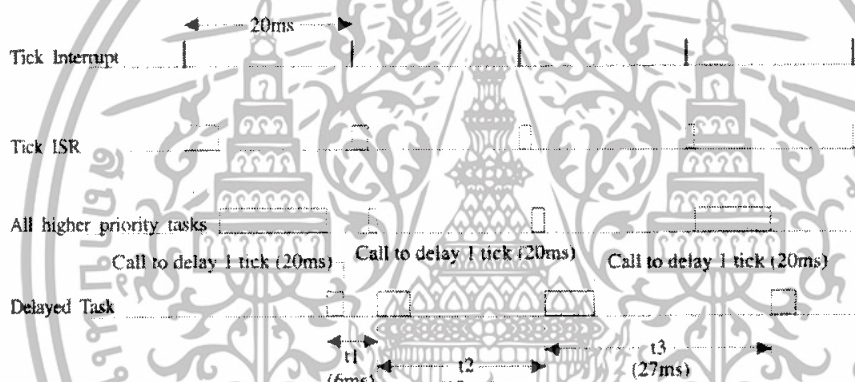
กรณีที่ 1 จะแสดงให้เห็นว่าทาสก์ที่มีความสำคัญสูงและการทำงานของISR ก่อนหน้าทาสก์จะมีการดีเลย์ในแต่ละ 1 ทิกค์ (tick) ซึ่งแต่ละทาสก์ต้องการจะดีเลย์ 20 ms แต่เพราะว่าค่าความสำคัญทำให้การทำงานจริงๆ จะไม่ตรง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.29 ดีเลย์ทาสก์ กรณีที่ 1

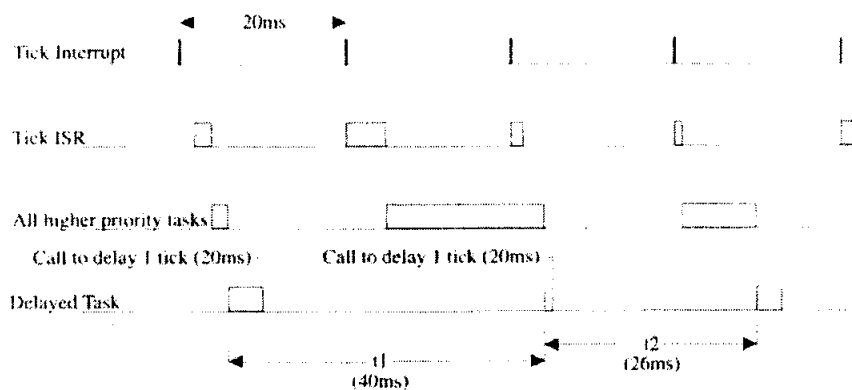
กรณีที่ 2 แสดงให้เห็นเมื่อเวลาในการทำงานของทาสก์ที่มีความสำคัญสูงและ ISR ใช้เวลาน้อยกว่า 1 ทิกต์ถ้าทาสก์มีดีเลย์ก่อนที่จะถึงคล็อกทิกต์ ทาสก์จะทำงานอีกครั้งในจะทันที



รูปที่ 2.30 ดีเลย์ทาสก์ กรณีที่ 2

กรณีที่ 3 แสดงให้เห็นเมื่อเวลาในการทำงานทั้งหมดของทาสก์ที่มีความสำคัญสูงและ ISR มีระยะเวลามากกว่า 1 ทิกต์ทำให้ทาสก์พยายามดีเลย์ 1 ทิกซึ่งจริง ๆ แล้วทำงานไป 2 ทิก ซึ่งกรณีนี้อาจจะยอมรับในบางแอปพลิเคชัน แต่โดยทั่วไปแล้วไม่ควร

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.31 ดีเลย์ทาสก์ กรณีที่ 3

กรณีต่าง ๆ จะปรากฏกับเรียลไทม์เคอร์เนลทั่วไป ซึ่งอาจจะทำให้ซีพียูทำงานหนักและเกิดข้อผิดพลาดได้ ซึ่งอาจจะแก้ปัญหานี้ได้โดยดังนี้

- เพิ่มคล็อกเรต (clock rate) ในไมโครคอนโทรลเลอร์ขึ้น
- เพิ่มระยะเวลาระหว่างทิกอินเตอร์รัพต์ (tick interrupt)
- ปรับปรุงค่าความสำคัญของทาสก์ (task priority)
- เขียนโค้ดในส่วนที่ต้องการความรวดเร็วด้วยการใช้ภาษาแอสเซมบลี (assembly)
- ใช้งานใช้งานไมโครโปรเซสเซอร์ที่มีความเร็ว

2.6 คริติคอลเซกชันของโค้ด

คริติคอลเซกชัน (critical section) ของโค้ดหรือเรียกว่าคริติคอลรีเจียน (critical region) คือ โค้ดซึ่งต้องการทำงานโดยไม่มีการจัดการทำงาน อย่างเช่นส่วนเริ่มการทำงานของโค้ดจะต้องไม่ถูกขัดจังหวะการทำงาน เพื่อป้องกันการเกิดการขัดจังหวะซึ่งส่วนใหญ่แล้วอินเตอร์รัพต์จะถูกดิสเอบิลเอาไว้ ก่อนที่คริติคอลโค้ดจะทำงานและจะเอนาเบิลเมื่อส่วนคริติคอลโค้ดทำงานเสร็จแล้ว

2.7 ความต้องการหน่วยความจำ

ถ้าทำการใช้ระบบฟอร์กราวด์/แบ็กกราวด์ (foreground/background) จำนวนเมมโมรี (memory) ที่ต้องใช้งานขึ้นอยู่กับตัวโค้ดของแอปพลิเคชัน ซึ่งมัลติทาสกิงเคอร์เนลจะแตกต่างออกไป โดยเคอร์เนลต้อง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การพื้นที่สำหรับ โค้ดพิเศษหรือ ROM ขนาดของ เคอร์เนลจะขึ้นอยู่กับหลายปัจจัย เช่น ฟีเจอร์(feature)ของ เคอร์เนล การทำคอนเท็กซ์ สวิตชิง (context switching) การจัดการเซมาฟอร์ ดีเลย์ไทม์เอ๊าท์ต่างๆ และอื่น ๆ

เนื่องจากแต่ละทาสก์ทำงานอิสระต่อกัน ซึ่งแต่ละทาสก์จะมีพื้นที่สแตกเป็นของตัวเอง (RAM) โดยในการออกแบบนั้นจะต้องคิดถึงสแตกที่ต้องใช้ในแต่ละทาสก์ซึ่งไม่เพียงแต่ใช้สำหรับใช้งานทั่วไปในทาสก์เท่านั้น แต่ยังบ่งบอกถึงจำนวนการทำอินเตอร์รัพต์ซ้อนกัน (interrupt nest) ซึ่งก็ขึ้นอยู่กับ โปรเซสเซอร์และเคอร์เนลที่ใช้

ถ้ามี RAM เพื่อใช้ในการทำงานมากพอ ก็จะต้องระมัดระวังในการใช้เพื่อลดความต้องการ RAM ในตัวแอปพลิเคชันทั้งหมด ซึ่งควรจะระมัดระวังในการใช้งานสำหรับ

- อาร์เรย์ (array) และ โครงสร้าง(structure)ที่มีขนาดใหญ่
- การทำฟังก์ชันซ้อนกัน (function nesting)
- การทำอินเตอร์รัพต์ซ้อนกัน (interrupt nesting)
- การใช้ไลบรารีฟังก์ชัน (library function) และฟังก์ชันซึ่งมีอาร์กิวเมนต์ (argument) มาก

โดยสรุปมิติทาสก์ซิสเต็มต้องการพื้นที่สำหรับ โค้ด (ROM) และพื้นที่สำหรับข้อมูล (RAM) มากกว่าระบบฟอร์กราวด์/แบ็กกราวด์ (foreground/background) โดยขนาดของ ROM ขึ้นอยู่กับขนาดของ เคอร์เนล และขนาดของ RAM ขึ้นอยู่กับจำนวนของทาสก์ในระบบ

2.8 ข้อดีและข้อเสียของเรียลไทม์เคอร์เนล

เรียลไทม์เคอร์เนล (real time kernel) หรือบางครั้งเรียกว่าเรียลไทม์ โอเปอเรติงซิสเต็ม (realtime operating system) หรือ RTOS จะใช้เพื่อให้ง่ายต่อการออกแบบและพัฒนากับเรียลไทม์แอปพลิเคชัน ซึ่งฟังก์ชันต่าง ๆ สามารถเพิ่มขึ้นได้โดยไม่ต้องเปลี่ยนซอฟต์แวร์ตัวหลัก การใช้งาน RTOS จะออกแบบให้ โค้ดของแอปพลิเคชันแยกย่อยออกเป็นทาสก์ และการทำงานจะสามารถตอบสนองได้ทันต่อเหตุการณ์และมีประสิทธิภาพ RTOS จะยังช่วยจัดการในเรื่องต่าง ๆ ไม่ว่าจะเป็นเซมาฟอร์, เมลบี็อกซ์, คิว, ไทม์ดีเลย์ เป็นต้น แต่ค่าใช้จ่ายในการพัฒนาให้รองรับกับงานหลาย ๆ ด้าน จะทำให้ต้นทุนในการพัฒนาและพื้นที่ของหน่วยความจำในระบบที่ต้องการอาจเพิ่มสูงได้

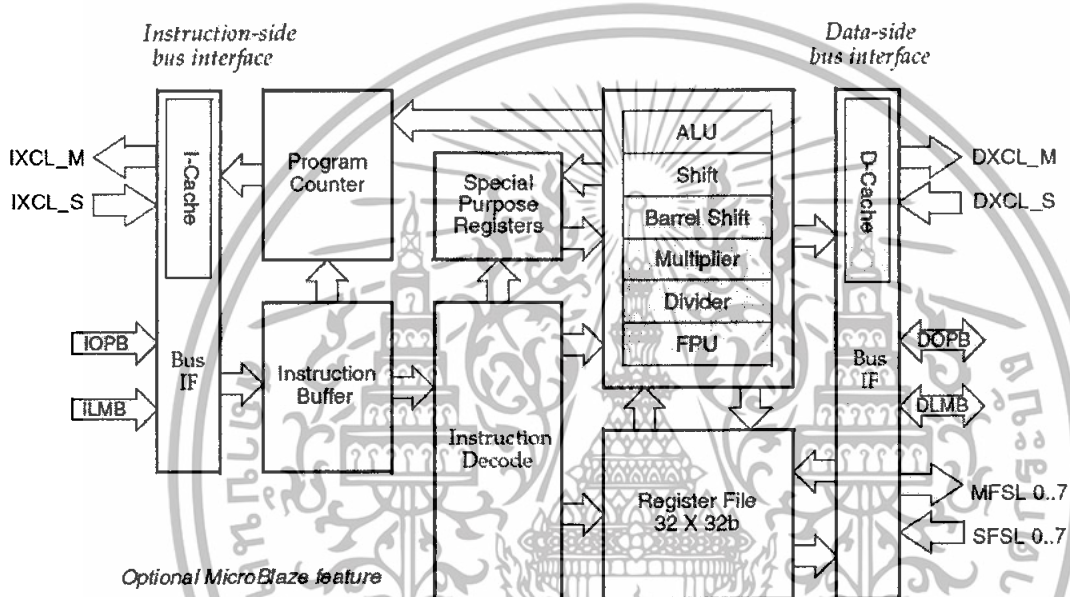
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3

สถาปัตยกรรมของไมโครเบลส

3.1 ภาพรวม

ไมโครเบลสเป็นซอฟต์แวร์โปรเซสเซอร์แบบ RISC (reduced instruction set computer) ซึ่งถูกออกแบบมาให้ใช้บนซอฟต์แวร์ของเซลลิ่ง รูปต่อไปนี้เป็นผังการทำงานโดยรวมของไมโครเบลสคอร์



รูปที่ 3.1 ผังการทำงานโดยรวมของ ไมโครเบลส คอร์

3.1.1 คุณสมบัติ

ไมโครเบลสเป็นซอฟต์แวร์โปรเซสเซอร์ที่อนุญาตให้เราปรับแต่งคุณสมบัติต่างๆ ให้เข้ากับงานของเราได้หลายที่ด้วยกัน

คุณสมบัติที่เปลี่ยนแปลงไม่ได้ได้แก่

- รีจิสเตอร์(register)ทั่วไปขนาด 32 บิตทั้งสิ้น 32 ตัว
- ชุดคำสั่งขนาด 32 บิตซึ่งประกอบไปด้วยโอเปอเรนด์(operand) 3 ตัวและวิธีการอ้างตำแหน่ง (addressing mode) 2 แบบ
- แอดเดรสบัส(address bus) ขนาด 32 บิต
- ไปป์ไลน์(pipeline)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ส่วนคุณสมบัติที่ปรับแต่งได้ของ ไมโครเบลสเป็นดังตารางนี้

Feature	MicroBlaze Versions				
	v2.10a	v3.00a	v4.00	v5.00	v6.00
Version Status	deprecated	deprecated	deprecated	deprecated	preferred
Processor pipeline depth	3	3	3	5	3/5
On-chip Peripheral Bus (OPB) data side interface	option	option	option	option	option
On-chip Peripheral Bus (OPB) instruction side interface	option	option	option	option	option
Local Memory Bus (LMB) data side interface	option	option	option	option	option
Local Memory Bus (LMB) instruction side interface	option	option	option	option	option
Hardware barrel shifter	option	option	option	option	option
Hardware divider	option	option	option	option	option
Hardware debug logic	option	option	option	option	option
Fast Simplex Link (FSL) interfaces	0-7	0-7	0-7	0-7	0-7
Machine status set and clear instructions	option	option	option	Yes	option
Instruction cache over IOPB interface	option	option	option	No	No
Data cache over IOPB interface	option	option	option	No	No
Instruction cache over CacheLink (DXCL) interface	-	option	option	option	option
Data cache over CacheLink (DXCL) interface	-	option	option	option	option
4 or 8-word cache line on XCL	-	4	4	option	option
Hardware exception support	-	option	option	option	option
Pattern compare instructions	-	-	option	Yes	option
Floating point unit (FPU)	-	-	option	option	option
Disable hardware multiplier ¹	-	-	option	option	option
Hardware debug readable ESR and EAR	-	-	Yes	Yes	Yes
Processor Version Register (PVR)	-	-	-	option	option
Area or speed optimized	-	-	-	-	option
Hardware multiplier 64-bit result	-	-	-	-	option
LUT cache memory	-	-	-	-	option

1. Used in Virtex™-II and subsequent families, for saving MUL18 and DSP48 primitives.

ตาราง 3.1 คุณสมบัติที่ปรับแต่งได้ของไมโครเบลส เรียงตามรุ่นของไมโครเบลส

3.2 ชนิดและการจัดเรียงตัวของข้อมูล

ไมโครเบลส ใช้การจัดเรียงตัวของข้อมูลกลับแบบบิกเอนเดียน (Big-Endian bit-reversed) ในการแสดงข้อมูล ซึ่งชนิดของข้อมูลที่รองรับ โดยฮาร์ดแวร์ของ ไมโครเบลส นั้น ได้แก่ เวิร์ด (word), ฮาล์ฟเวิร์ด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

(half word) และ ไบต์(byte) ซึ่งการจัดเรียงตัวของบิตและ ไบต์ของแต่ละชนิดของข้อมูลนั้นเป็นตามตารางต่อไปนี

Byte address	n	n+1	n+2	n+3
Byte label	0	1	2	3
Byte significance	MSByte			LSByte
Bit label	0			31
Bit significance	MSBit			LSBit

ตาราง 3.2 ข้อมูลชนิดเวิร์ด

Byte address	n	n+1
Byte label	0	1
Byte significance	MSByte	LSByte
Bit label	0	15
Bit significance	MSBit	LSBit

ตาราง 3.3 ข้อมูลชนิดฮาล์ฟเวิร์ด

Byte address	n
Bit label	0 7
Bit significance	MSBit LSBit

ตาราง 3.4 ข้อมูลชนิดไบต์

3.3 ชุดคำสั่ง

ชุดคำสั่งทั้งหมดของ ไมโครเบลส นั้นมีขนาด 32 บิต และแบ่งออกเป็น ชนิด A และ ชนิด B โดยชนิด A นั้นมีรีจิสเตอร์ที่เป็นซอส โอเปอเรนด์(source operand)ทั้งสิ้น 2 ตัวและรีจิสเตอร์ที่เป็นเดสดีเนชั่น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โอเปอเรนด์(destination operand)หนึ่งตัว ส่วน ชนิด B นั้นประกอบไปด้วยซอสตรีจิตเตอร์ 1 ตัวเดสทินเนชัน รีจิตเตอร์ 1 ตัว และอิมมีเดียคโอเปอเรนด์(immediate operand) อีก 1 ตัว

3.4 รีจิตเตอร์

ไมโครเบลส นั้นประกอบไปด้วย รีจิตเตอร์แบบทั่วไปที่มีขนาด 32 บิตทั้งสิ้น 32 ตัวและ แบบมีหน้าที่พิเศษ ที่มีขนาด 32 บิต สูงสุดถึง 18 ตัวซึ่งจะมีที่ตัวนั้นขึ้นอยู่กับค่าของ ไมโครเบลส

3.5 สถาปัตยกรรมของไปป์ไลน์

การทำงานของ ไมโครเบลส นั้นได้ถูกทำไปป์ไลน์ไว้แล้ว โดยในเกือบทุกๆ คำสั่ง แต่ละ สเตจ (stage) นั้นจะใช้ 1 คล็อกไซเคิล(clock cycle) ในการทำงาน ดังนั้นจำนวนคล็อกไซเคิลที่ต้องใช้ในการทำงานคำสั่งให้เสร็จสิ้นนั้นจะเท่ากับจำนวนสเตจของไปป์ไลน์และในทุกๆคล็อกไซเคิลนั้นจะมี 1 คำสั่งที่ทำงานเสร็จสิ้น แต่มีบางคำสั่งที่ต้องใช้มากกว่า 1 คล็อกไซเคิลในเอกซิวสเตจซึ่งก็ยังสามารถทำงานได้โดยการสตอล(stall)ไปป์ไลน์

3.5.1 ไปป์ไลน์แบบ 3 สเตจ

หากมีการเปิดตัวเลือกแอเรียออปติไมเซชัน (area optimization) ในขั้นตอนการสังเคราะห์ตัว ไมโครเบลส แล้วไปป์ไลน์จะถูกแบ่งออกเป็น 3 สเตจเพื่อลดขนาดของฮาร์ดแวร์ซึ่งได้แก่ เฟตช์(Fetch), ดีโค้ด(Decode) และ เอกซิว(Execute)

	cycle 1	cycle 2	cycle 3	cycle4	cycle5	cycle6	cycle7
instruction 1	Fetch	Decode	Execute				
instruction 2		Fetch	Decode	Execute	Execute	Execute	
instruction 3			Fetch	Decode	Stall	Stall	Execute

รูปที่ 3.2 ไปป์ไลน์แบบ 3 สเตจ

3.5.2 ไปป์ไลน์แบบ 5 สเตจ

หากไม่มีการเปิดตัวเลือกแอเรียออปติไมเซชันในขั้นตอนการสังเคราะห์ตัว ไมโครเบลส แล้วไปป์ไลน์จะถูกแบ่งออกเป็น 5 สเตจเพื่อเพิ่มประสิทธิภาพในการทำงาน ซึ่งได้แก่ Fetch(IF), Decode(OF), Execute(EX), Access Memory(MEM) และ Writeback(WB)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

	cycle 1	cycle 2	cycle 3	cycle 4	cycle 5	cycle 6	cycle 7	cycle 8	cycle 9
instruction 1	IF	OF	EX	MEM	WB				
instruction 2		IF	OF	EX	MEM	MEM	MEM	WB	
instruction 3			IF	OF	EX	Stall	Stall	MEM	WB

รูปที่ 3.3 ไปป์ไลน์แบบ 5 สเตจ

3.5.3 แบรินช์(Branches)

โดยปกติแล้ว คำสั่งที่อยู่ในเฟลชและดี โคดสเตจนั้นจะถูกล้างทิ้งทันทีที่การเอกซ์คิวชันนั้นพบกับคำสั่งประเภทแบรินช์จากนั้นไปป์ไลน์จะถูกโหลดด้วยคำสั่งใหม่ ทำให้การแบรินช์ใน ไมโครเบลส นั้นใช้ทั้งสิ้น 3 คล็อกไซเคิลก่อนที่จะทำการทำงานต่อไปได้ ซึ่ง 2 ใน 3 คล็อกไซเคิลที่เสียไปนี้ถูกใช้เพื่อเติมค่ากลับลงมาในไปป์ไลน์ดังนั้น เพื่อลดความล่าช้าที่เกิดขึ้นนี้ ไมโครเบลส จึงถูกออกแบบให้รองรับการแบรินช์แบบมีดีเลย์สล็อต

3.5.3.1 ดีเลย์สล็อต(Delay Slots)

ขณะที่ทำการแบรินช์โดยมีดีเลย์สล็อตนั้น มีเพียงแค่เฟลชสเตจในไปป์ไลน์เท่านั้นที่จะถูกล้างคำสั่งส่วนชุดคำสั่งที่อยู่ในดี โคดสเตจ(หรือเรียกว่าเป็นดีเลย์สล็อต) นั้นจะยอมให้ถูกทำงานต่อไป การใช้เทคนิคนี้ช่วยลดความสูญเสียเวลาจากการแบรินช์จาก 2 คล็อกไซเคิลเหลือเพียงคล็อกไซเคิลเดียวเท่านั้น

3.6 รีเซ็ต(Reset), อินเตอร์รัพต์(Interrupts), เอ็กเซพชัน(Exceptions) และเบรก(Break)

ไมโครเบลส นั้นรองรับสัญญาณ รีเซ็ต, อินเตอร์รัพต์, เอ็กเซพชันจากผู้ใช้, เบรกและ เอ็กเซพชันจากฮาร์ดแวร์โดยมีการเรียงลำดับความสำคัญจากมากไปหาน้อยดังนี้

- รีเซ็ต
- เอ็กเซพชันจากฮาร์ดแวร์
- นอนมาสก์เอเบิลเบรก(Non-maskable Break)
- เบรก
- อินเตอร์รัพต์
- เอ็กเซพชันจากผู้ใช้

ตารางต่อไปนี้เป็นตำแหน่งของเวกเตอร์แอดเดรส(Vector Address) และ รีจิสเตอร์ไฟล์รีเทิร์นแอดเดรส

(Register File Return Address) สำหรับแต่ละเหตุการณ์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Event	Vector Address	Register File Return Address
Reset	0x00000000 - 0x00000004	-
User Vector (Exception)	0x00000008 - 0x0000000C	Rx
Interrupt	0x00000010 - 0x00000014	R14
Break: Non-maskable hardware	0x00000018 - 0x0000001C	R16
Break: Hardware		
Break: Software		
Hardware Exception	0x00000020 - 0x00000024	R17 or BTR
Reserved by Xilinx for future use	0x00000028 - 0x0000004F	-

ตาราง 3.5 เวกเตอร์ และ ตำแหน่งของรีเทอร์นแอดเดรสไฟล์

3.6.1 รีเซต

เมื่อ ไมโครเบลส ได้รับสัญญาณ Reset หรือ Debug_Rst แล้ว จะทำการล้างไปป์ไลน์และเริ่มต้นการเฟตช์ชุดคำสั่งจากรีเซตเวกเตอร์(ที่ตำแหน่ง 0x0) ซึ่งสัญญาณรีเซตจากภายนอกทั้งสองแหล่งนี้ทำงานที่สถานะสูงและจะต้องมีความยาวไม่น้อยกว่า 16 ไซเคิล

3.6.1.1 คำสั่ง ซูโดโค้ด(Pseudocode) ที่อธิบายกระบวนการรีเซต

PC <- 0x00000000

MSR <- C_RESET_MSR

EAR <- 0

ESR <- 0

FSR <- 0

3.6.2 อินเทอร์รัพต์

ไมโครเบลส รองรับการเกิดสัญญาณ อินเทอร์รัพต์ จากภายนอก (ที่ต่อเข้ากับช่องรับสัญญาณอินเทอร์รัพต์) โดยตัวโปรเซสเซอร์จะตอบสนองต่ออินเทอร์รัพต์ก็ต่อเมื่อเปิด Interrupt Enable(IE) ใน Machine Status Register(MSR) นั้นถูกตั้งค่าเป็น 1 ไว้เท่านั้น เมื่อเกิดการอินเทอร์รัพต์คำสั่งที่อยู่ในเอกซิวทัศน์จะยังคงถูกทำงานต่อไปจนเสร็จสิ้น ในขณะที่คำสั่งที่อยู่ในดีโคคสแตจนั้นจะถูกเปลี่ยนเป็นคำสั่งเอกซิวทัศน์นี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แบริชไปยังอินเตอรร์พต์เวกเตอรร์(ที่ตำแหน่ง 0x10) ส่วนอินเตอรร์พต์รีเทอรร์นแอดเดรส(interrupt return address)นั้นจะถูกโหลดจาก PC ไปไว้ยัง R14 โดยอัตโนมัติ และโพเรสเซอรร์ยังทำการเคลียค่าในบิต IE ใน MSR ทิ้ง โดยอัตโนมัติ เพื่อปิดไม่ให้มีการเกิดอินเตอรร์พต์ขึ้นอีก และบิต IE นั้นจะถูกเซ็ทกลับดังเดิมทันทีที่มีการเรียกใช้คำสั่ง RTID

อินเตอรร์พต์นั้นจะไม่มีการตอบสนองโดยโพเรสเซอรร์ถ้าหากบิต break in progress(BIP) หรือ exception in progress(EIP) ใน MSR นั้นถูกตั้งเป็น 1

3.6.2.1 ความล่าช้า

เวลาที่ถูกใช้โดย ไมโครเบลส เพื่อที่จะเข้าสู่ฟังก์ชันการตอบสนองอินเตอรร์พต์(ISR) โดยนับจากที่สัญญาณอินเตอรร์พต์เกิดขึ้นนั้นขึ้นอยู่กับค่าของตัวโพเรสเซอรร์และความล่าช้าจากตัวเมมโมรีคอนโทรลเลอร์ในการเก็บอินเตอรร์พต์เวกเตอรร์โดยถ้าหาก ไมโครเบลส ถูกตั้งค่าไว้ให้มีตัวหารซึ่งเป็นฮาร์ดแวร์ความล่าช้าสูงที่สุดจะเกิดขึ้นหากสัญญาณอินเตอรร์พต์เกิดในขณะที่กำลังทำงานคำสั่งหารอยู่

3.6.2.2 คำสั่งชุดโค๊ดที่อธิบายกระบวนการอินเตอรร์พต์

r14	<-	PC
PC	<-	0x00000010
MSR[IE]	<-	0

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 4

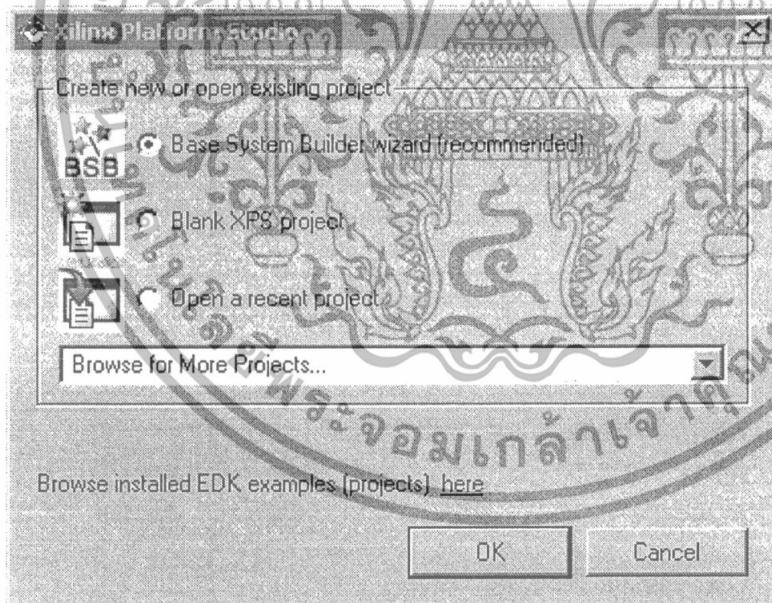
การสร้างระบบไมโครเบลสอย่างง่าย

4.1 ภาพรวม

เนื้อหาในบทนี้จะเป็นเอกสารประกอบการทดลองการสร้างระบบไมโครเบลสอย่างง่าย โดยภายในระบบจะประกอบไปด้วย ตัวคอร์ของไมโครเบลส และ ยูอาร์ที(UART) โมดูล โดยการทำงานของระบบคือ เมื่อระบบถูกทำการรีเซตจะพิมพ์ตัวหนังสือว่า “Hello MicroBlaze!” ออกมาทางยูอาร์ที โมดูล

4.2 การสร้างระบบไมโครเบลสด้วย เบสซิสเต็มบิลเดอร์วิซาร์ด (Base System Builder Wizard)

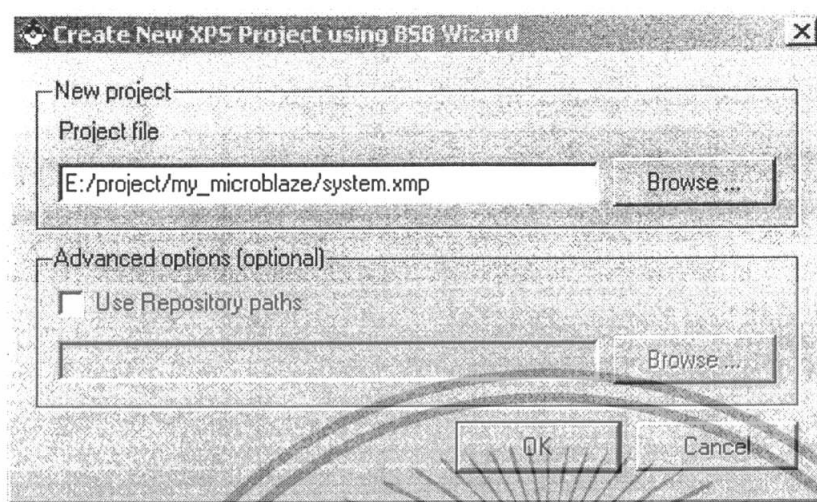
1. หลังจากเปิดโปรแกรมไซลิงก์แพลตฟอร์มสตูดิโอ(Xilinx Platform Studio) ขึ้นมาทำการเลือกไปที่ เบสซิสเต็มบิลเดอร์วิซาร์ด



รูปที่ 4.1 การเลือกเบสซิสเต็มบิลเดอร์วิซาร์ด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

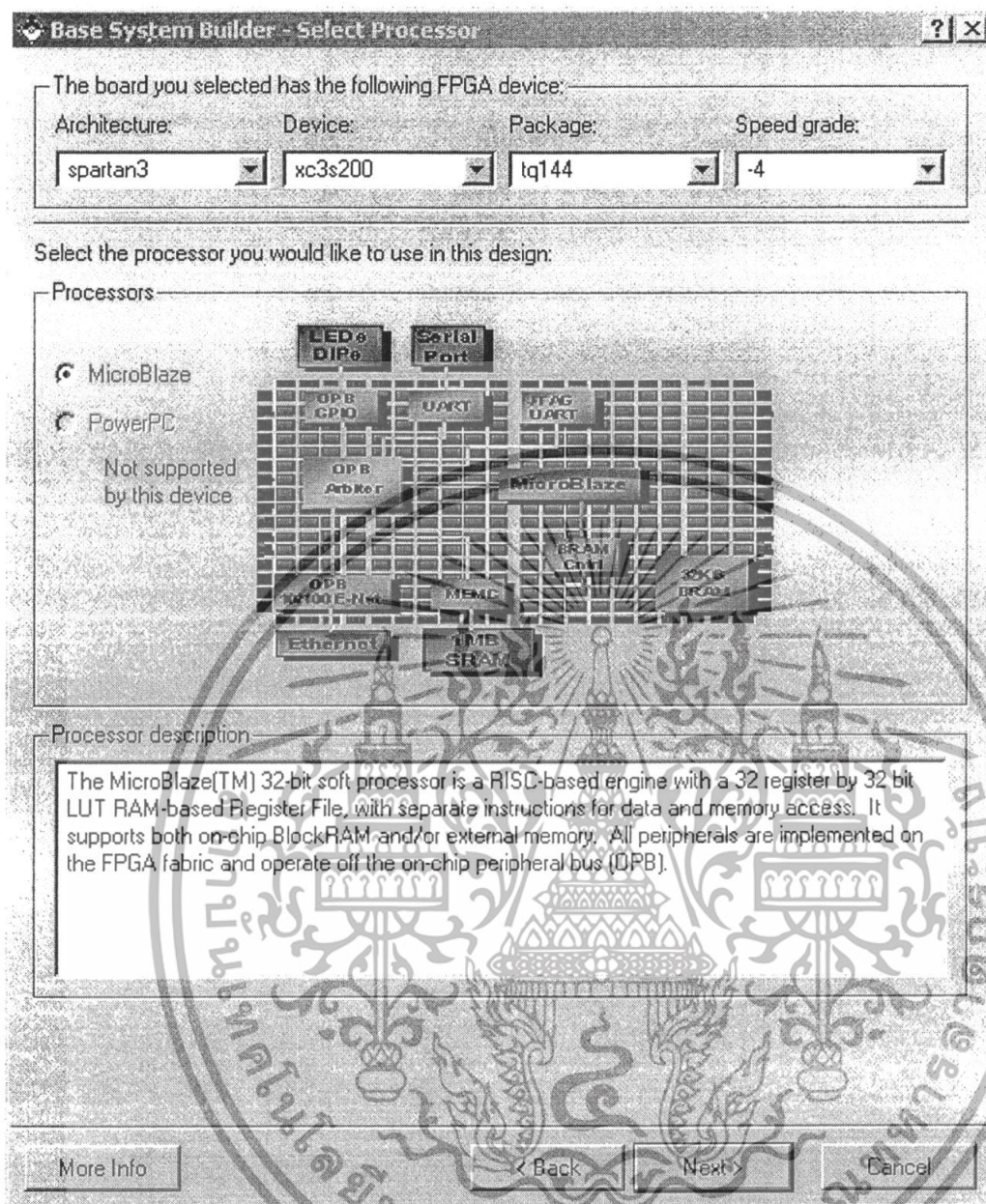
2. ทำการเลือกตำแหน่งสำหรับเซฟโปรเจกต์ที่ต้องการ



รูปที่ 4.2 การเลือกตำแหน่งในการเซฟโปรเจกต์

3. เลือกที่ “I would like to create a new design” กด Next
4. เลือกที่ “I would like to create a system for a custom board” เนื่องจากเราไม่ได้ใช้บอร์ดของไซดิง และไม่มีบอร์ดเดฟินิชั่นไฟล์ (Board definition file) มาจากผู้ผลิตบอร์ด
5. เลือกชนิดของเอฟพีจีเอให้ตรงกับบอร์ดที่เราใช้ซึ่งในการทดลองนี้เราใช้บอร์ดที่ใช้รีพอสิตอรีที่ 3 เบอร์ เอ็กซ์ซีซี3เอส200(xc3s200) ของบริษัทเอเพ็กซ์อินสตรูเมนต์ (Apex Instrument)
6. เลือกชนิดของโปรเซสเซอร์ให้เป็นไมโครเบลส เนื่องจากในชิพบางเบอร์นั้นจะมีพาวเวอร์พีซี ซึ่ง เป็นฮาร์ดโปรเซสเซอร์คอร์อยู่บนชิพ หลังจากตั้งค่าต่างๆ เสร็จแล้วจะได้ดังรูป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.3 การตั้งค่าเกี่ยวกับเอฟพีเจอี และโปรเซสเซอร์ที่จะใช้

7. กำหนดค่าเกี่ยวกับตัวไมโครเบลสที่จะสร้างดังนี้

- “Refernce clock frequency” คือความถี่ของออสซิลเลเตอร์บนบอร์ด ในที่นี้คือ 25 เมกะเฮิร์ต
- “Processor-Bus clock frequency” คือความถี่ที่เราจะใช้ในตัวโปรเซสเซอร์ โดยสามารถตั้งให้สูงกว่าออสซิลเลเตอร์ได้โดยระบบจะทำการใช้ดีซีเอ็ม โมดูลในการเพิ่มความถี่ให้
- “Reset polarity” คือลักษณะของสัญญาณรีเซ็ต ซึ่งปุ่มกดบนบอร์ดของเรานั้นเมื่อกดจะได้

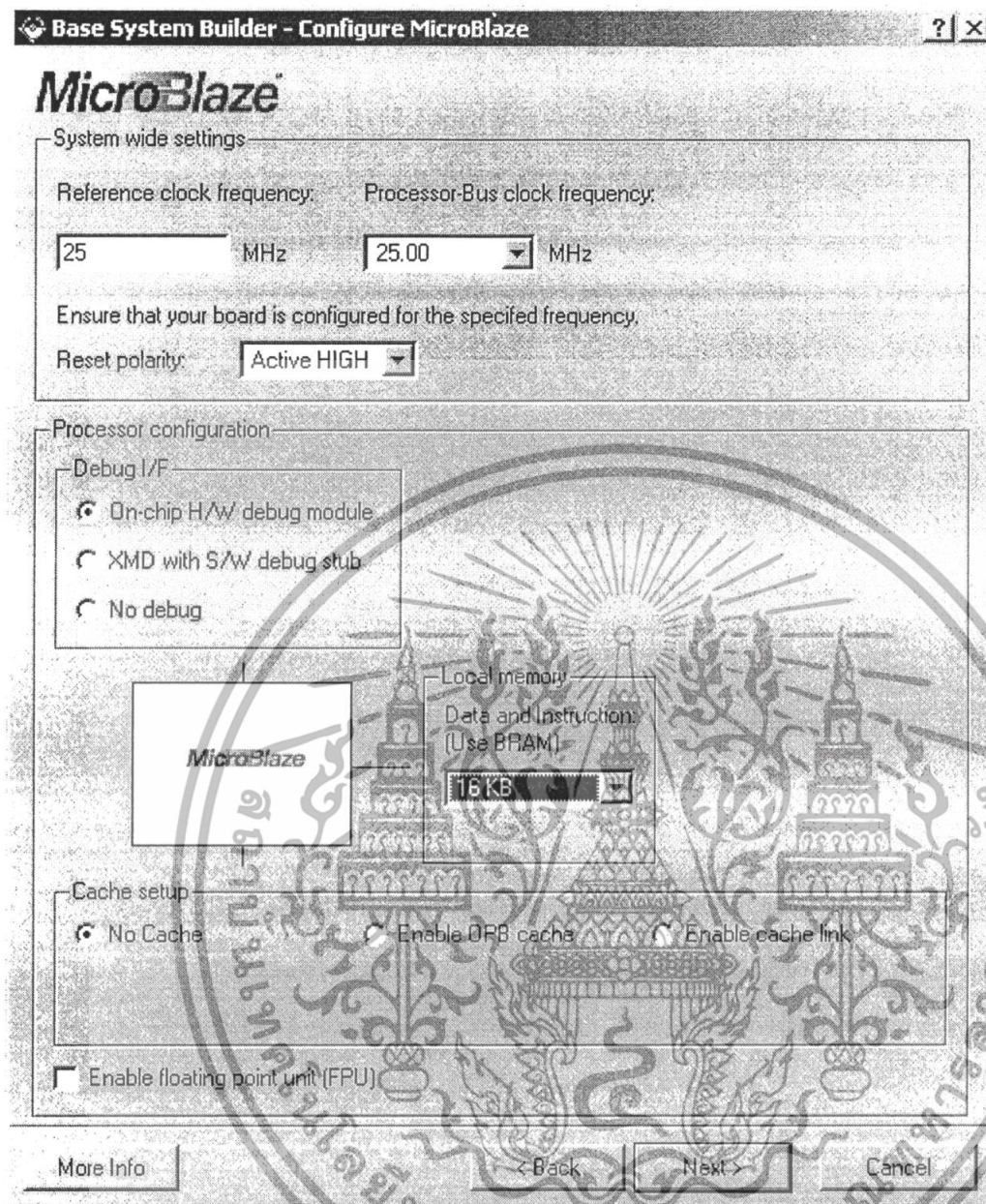
สัญญาณ ‘1’ ดังนั้นเราจึงตั้งเป็น “Active HIGH” ศึกษาเท่านั้น ไม่นอนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- d. “Debug I/F” เป็นการเลือกอินเตอร์เฟสในการดีบั๊กของระบบ
- e. “Local memory” เป็นการเลือกขนาดของบล็อกแรมที่จะใช้สำหรับเก็บค่าค่า และ อินสตรัคชัน ของระบบ
- f. “cache setup” เป็นการเลือกว่าจะให้ทำแคชขึ้นมาในระบบหรือไม่
- g. “Enable floating point unit” เป็นการเลือกว่าจะมีการใช้งานหน่วยประมวลผลเลขทศนิยมหรือไม่

หลังจากตั้งค่าต่างๆเสร็จเรียบร้อยแล้วจะมีหน้าต่างดังนี้



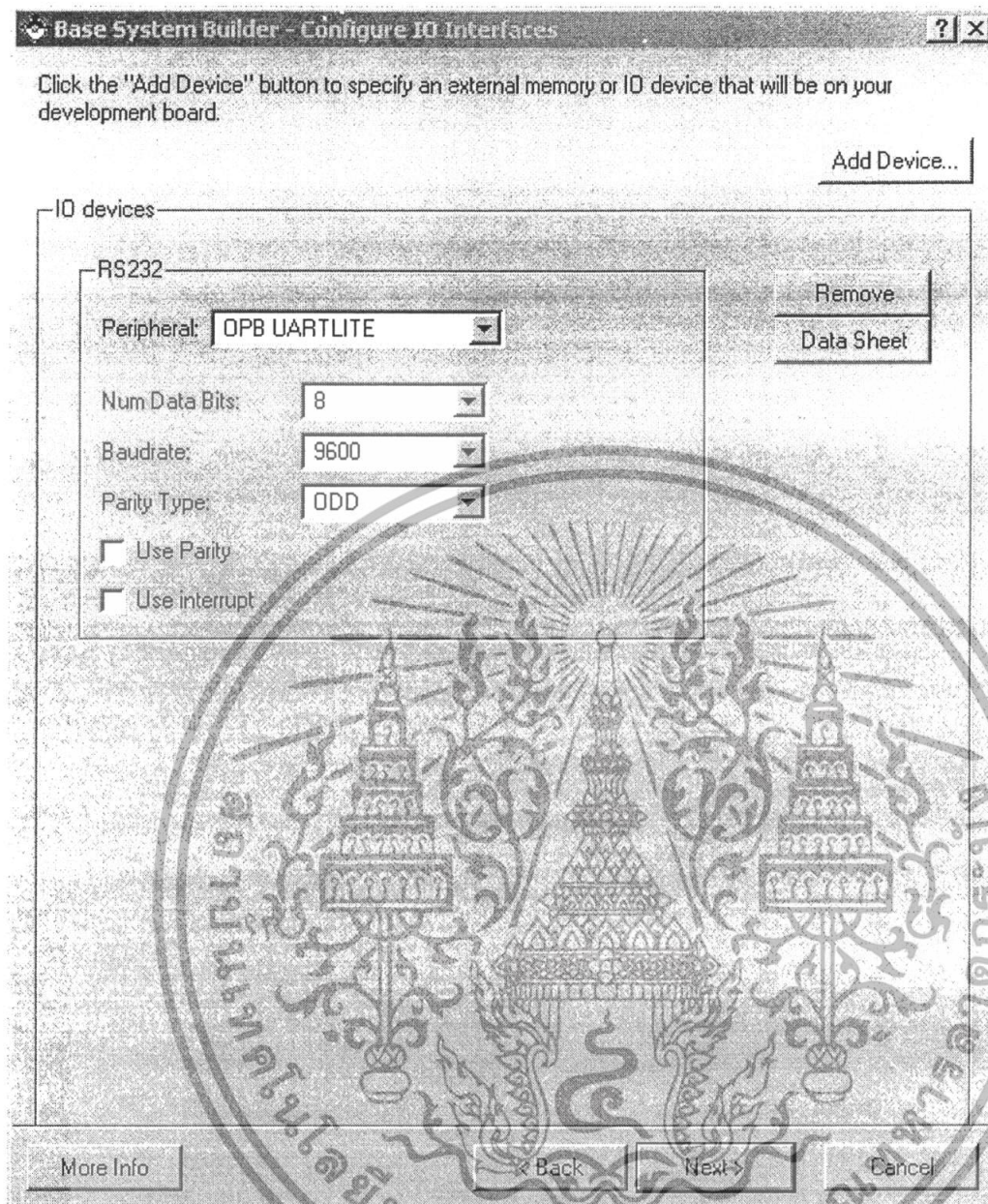
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.4 การตั้งค่าต่างๆของตัวไมโครเบลส

8. ทำการเพิ่มไอโอต่างๆ ของระบบ โดยการกดที่ปุ่ม “Add Device” โดยในที่นี้เราจะทำการเพิ่มยูอาร์ทโมดูลเข้ามาในระบบเพียงอย่างเดียว ซึ่งเราสามารถเพิ่ม โมดูลอื่นๆอีกในภายหลังได้ หลังจากที่เพิ่มยูอาร์ทโมดูลแล้วทำการตั้งค่าต่างๆดังรูป

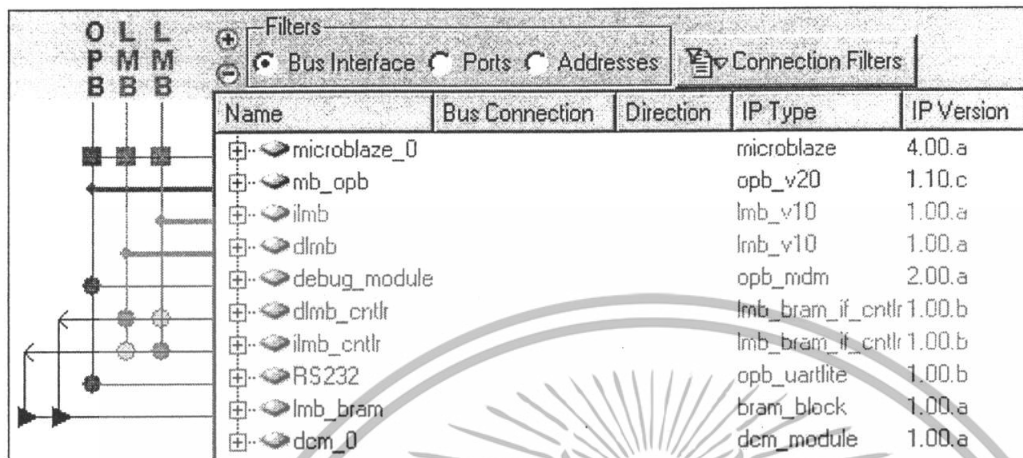
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.5 การเพิ่มไอโอของระบบ

9. สำหรับในหน้าต่อไปจะเป็นการเพิ่มเพอร์ipheral ของระบบ ซึ่งในที่นี้เราจะยังไม่เพิ่มอะไร ให้ทำการกด “Next” ข้ามไปก่อน
10. สำหรับหน้า “Software Setup” จะเป็นการตั้งค่าเกี่ยวกับซอฟต์แวร์ของระบบ โดยเราจะทำการแมพ STDIN และ STDOUT ไปยัง RS232 โมดูลทั้งคู่
11. หลังจากทำการตั้งค่าระบบทั้งหมดเสร็จสิ้นแล้วทำการกดปุ่ม “Generate” แล้วโปรแกรมจะทำการสร้างระบบออกมาให้ตามที่ตั้งค่าไว้
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.3 ภาพของระบบที่ถูกสร้างขึ้น



รูปที่ 4.6 ภาพรวมของระบบที่ถูกสร้างขึ้น

หลังจากที่เราทำการตั้งค่าต่างๆ ในเบสซิสเต็มบิลด์คอร์วิซาร์ดเรียบร้อยแล้ว โปรแกรมจะทำการสร้างระบบให้เราดังภาพ โดยจะมีตัวซีพียูคือ “microblaze_0” เชื่อมต่อเป็นมาสเตอร์บนบัสทั้งสามบัสได้แก่ “mb_opb” ซึ่งเป็น โอพีบี หรือออนชิพเพอริเฟอรัลบัส (On chip Peripheral Bus) “lmb” และ “dlmb” ซึ่งเป็น แอลเอ็มบี หรือ โลคอลเมมโมรีบัส (Local Memory Bus) สำหรับอินสตรัคชันและดาต้าตามลำดับ โดยจุดสีเหลี่ยมสีเขียวและน้ำเงินแสดงถึงความเป็นมาสเตอร์บนบัส โอพีบีและแอลเอ็มบีตามลำดับ

มีอุปกรณ์สองตัวที่เป็นสลาฟบน โอพีบี ได้แก่ “debug_module” ซึ่งเป็นฮาร์ดแวร์ที่ใช้ในการดีบั๊กโปรแกรมของระบบและ “RS232” หรือยูอาร์ที โมดูลซึ่งใช้ในการรับส่งข้อมูลผ่านทางโปรโตคอลยูอาร์ที โดยจุดกลมสีเขียวแสดงถึงความเป็นสลาฟบนบัส โอพีบี

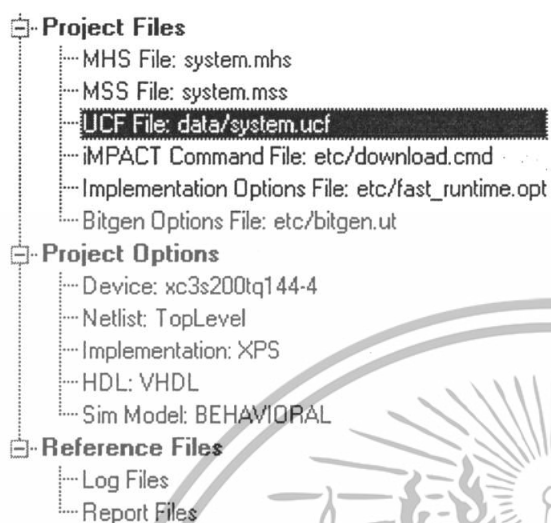
อุปกรณ์อีกสองตัวได้แก่ “dlm_b_ctrl” และ “ilmb_ctrl” เป็นสลาฟคอนโทรลเลอร์บนบัสแอลเอ็มบี ซึ่งมีหน้าที่ในการติดต่อกับบล็อกแรมในตัวเอฟพีจีเอ โดยจุดกลมสีน้ำเงินแสดงถึงความเป็นสลาฟบนแอลเอ็มบี และเส้นสีม่วงแสดงถึงการเชื่อมต่อกับบล็อกแรม

สำหรับอุปกรณ์ที่ชื่อเป็นสีดำคืออุปกรณ์ที่ไม่ได้เชื่อมต่ออยู่บนบัสใดเลยได้แก่ “dcm_0” ซึ่งเป็นวงจรมัลติเพลกซ์ที่ใช้ในการเพิ่มและลดความถี่จากแหล่งกำเนิดความถี่ของระบบหรือออสซิลเลเตอร์ของบอร์ดนั่นเอง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.4 การนำระบบไมโครเบลสลงไปทำงานบนบอร์ดเอฟพีจีเอ

1. ทำการเปิดยูซีเอฟ(ucf) ไฟล์ของระบบ “system.ucf” ขึ้นมาแก้ไข



รูปที่ 4.7 การเปิดยูซีเอฟไฟล์

2. ทำการแก้ไขยูซีเอฟไฟล์ โดยทำการเอาคอมเมนต์ตรงหน้าเน็ต(net) เหล่านี้ ออก “sys_clk_pin”, “sys_rst_pin”, “fpga_0_RS232_RX_pin” และ “fpga_0_RS232_TX_pin” แล้วทำการใส่ตำแหน่งของขาเหล่านั้นตรง “LOC=” โดยตำแหน่งของขาสามารถเปิดดูได้จากคู่มือการใช้งานบอร์ด หลังจากทำการแก้ไขเสร็จแล้ว ไฟล์จะมีหน้าตา ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

1 #####
2 ## This system.ucf file is generated by Base System Builder based on the
3 ## settings in the selected Xilinx Board Definition file. Please add other
4 ## user constraints to this file based on customer design specifications.
5 #####
6
7 Net sys_clk_pin LOC=P127;
8 Net sys_rst_pin LOC=P11;
9 ## System level constraints
10 Net sys_clk_pin TNM_NET = sys_clk_pin;
11 TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 40000 ps;
12 Net sys_rst_pin TIG;
13
14 ## IO Devices constraints
15
16 ##### Module RS232 constraints
17
18 # Net fpga_0_RS232_req_to_send pin LOC=;
19 Net fpga_0_RS232_RX_pin LOC=P129;
20 Net fpga_0_RS232_TX_pin LOC=P128;
21
22

```

รูปที่ 4.8 ไฟล์ยูซีเอฟที่ถูกแก้ไขแล้ว

- เลือกเมนูไปที่ “Hardware > Generate Bitstream” แล้วรอดูที่คอนโซลด้านล่าง ซึ่งขั้นตอนนี้ค่อนข้างใช้เวลานานพอสมควร ขึ้นอยู่กับข้อบังคับของระบบที่เราตั้งไว้ อย่างเช่นถ้าหากตั้งมีทอร์ของไมโครเบลสทำงานที่ความถี่ 100 เมกะเฮิร์ตซ์ก็อาจจะใช้เวลาถึง 6 ชั่วโมงได้ หากตั้งค่าตามตัวอย่างนี้ทุกประการจะใช้เวลาประมาณ 5 นาที (ทำงานโดยเฟรมเวิร์ก 3.0 กิกะเฮิร์ตซ์ไฮเปอร์เทรคดิ่ง) โดยเมื่อทำงานเสร็จสิ้นที่คอนโซลจะมีหน้าต่างดังนี้

```

Saving bit stream in "system.bit".
Bitstream generation is complete.

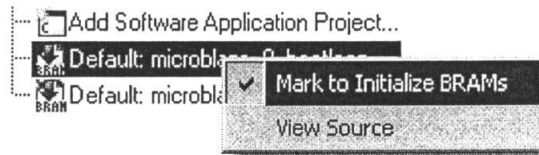
Done!

```

รูปที่ 4.9 คอนโซลหลังจากทำการสร้างบิตสตรีมแล้ว

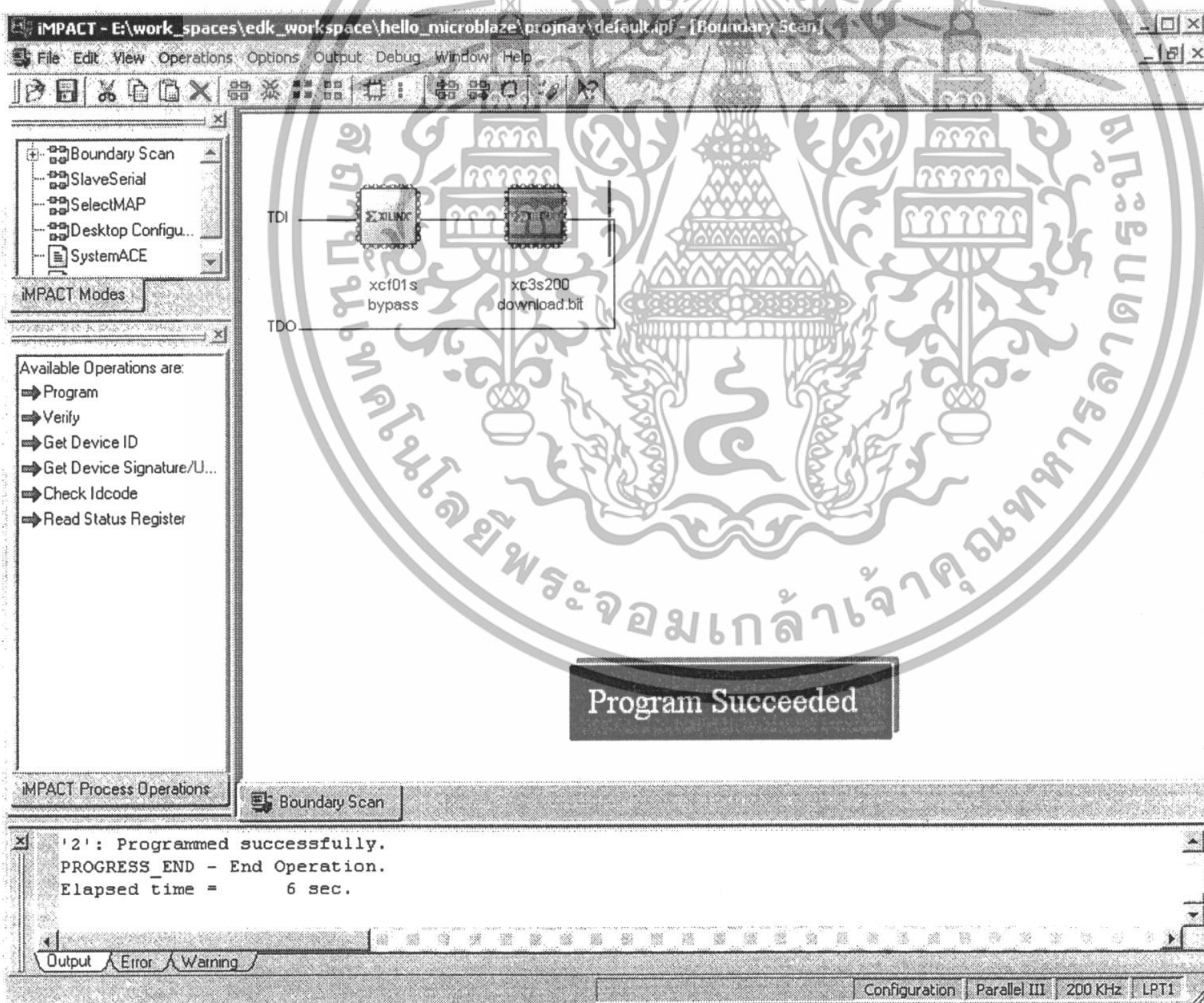
- ไปที่แถบ “Applications” แล้วทำการคลิกขวาที่ “microblaze_0_bootloop” แล้วเลือก “Mark to Initialize BRAMs” และหากมีตัวอื่นที่ถูกเลือก “Mark to Initialize BRAMs” อยู่ให้ทำการเลือกออกให้หมด โดยโปรแกรมบุทลูปนี้เป็น โปรแกรมที่จะทำการบรานซ์อยู่กับที่เพื่อไม่ให้ตัวไมโครเบลสไปทำหุคคำสั่งที่ไม่พึงประสงค์เนื่องจากยังไม่มีโปรแกรมถูกโหลดลงไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.10 การเลือกให้บูทลูปไปอยู่บนบล็อคแรม

5. ไปที่เมนู “Device Configuration > Update Bitstream” แล้วโปรแกรมจะทำการรวมชุดคำสั่งเริ่มต้นที่จะใส่ไปในบล็อคแรม (ในที่นี้คือ “microblaze_0_bootloop”) ไปไว้ในบิตสตรีมที่จะนำไปลงเฟิร์มแวร์ “system.bit” ได้เป็นไฟล์ใหม่ชื่อ “download.bit”
6. เชื่อมต่อสายเจแทค(JTAG) กับบอร์ดเฟิร์มแวร์จากนั้นทำการเปิดโปรแกรมอิมแพคแล้วทำการโหลดบิตสตรีมลงไปยังบอร์ดเฟิร์มแวร์



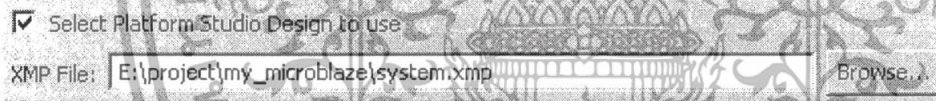
รูปที่ 4.11 ดาวโหลดบิตสตรีมด้วยโปรแกรมอิมแพค

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.5 การพัฒนาซอฟต์แวร์บนระบบไมโครเบลสที่สร้างขึ้น

สำหรับการเขียนซอฟต์แวร์บน ไมโครเบลสเราจะใช้โปรแกรมอีกตัวหนึ่งในชุดอีดีเค(EDK) ของไซลิ่ง ที่ชื่อว่า ไซลิ่งแพลตฟอร์มสตูดิโอเอสดีเค (Xilinx Platform Studio SDK) ซึ่งเป็นโปรแกรมที่ดัดแปลงมาจากโปรแกรมอีคลิปส์(Eclipse) ของไอบีเอ็ม

1. ทำการสร้างไฟล์ไครเวอร์ของอุปกรณ์ต่างๆในระบบ โดยไปยังเมนู “Software > Generate Libraries and BSPs” ในโปรแกรม ไซลิ่งแพลตฟอร์มสตูดิโอ โดยซอสไฟล์ของไครเวอร์ต่างๆจะอยู่ในโฟลเดอร์ “microblaze_0\libsrc” ในโฟลเดอร์โปรเจก และเฮเดอร์ไฟล์ที่จำเป็นต้องใช้ในการเขียนโปรแกรมประยุกต์จะอยู่ในโฟลเดอร์ “microblaze_0\include”
2. เปิดโปรแกรม ไซลิ่งแพลตฟอร์มสตูดิโอเอสดีเคขึ้นมาแล้วทำการสร้างโปรเจกใหม่โดยไปที่ “File > New > Project” แล้วเลือกที่ “Managed make C project”
3. กด “Browse” แล้วเลือกไปยังไฟล์ “system.xmp” ของโปรเจกที่เราสร้างไว้



รูปที่ 4.12 การตั้งค่าในเอ็กซ์พีเอสเอสดีเค

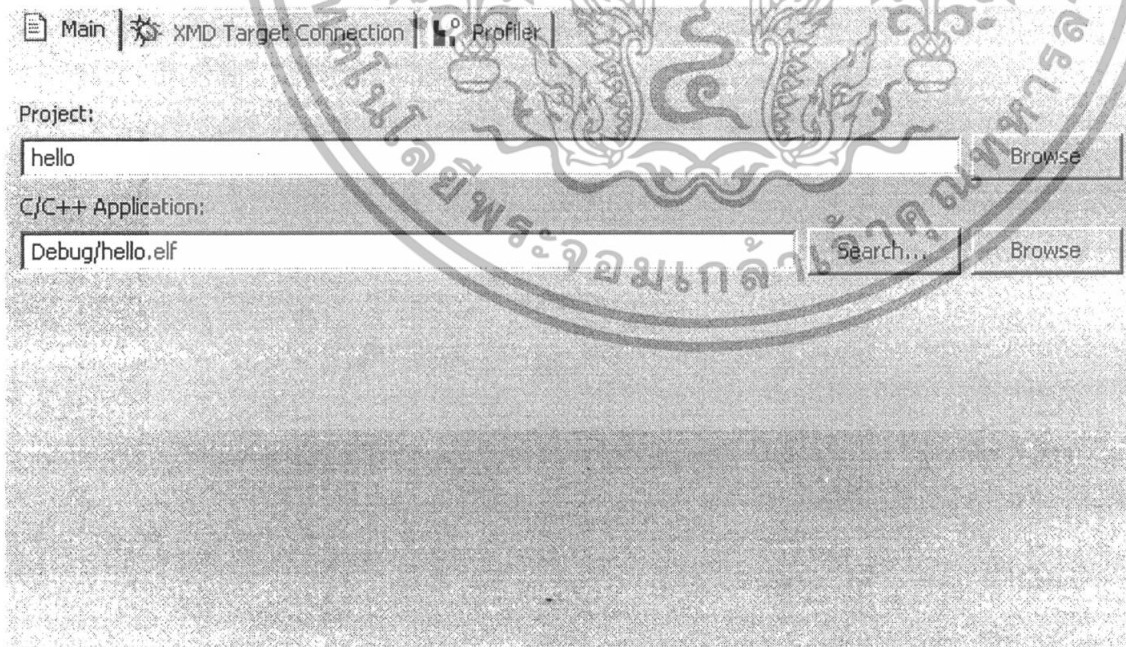
4. ตั้งชื่อโปรเจกว่า “hello” จากนั้นกด “Finish” โปรแกรมจะทำการสร้างโปรเจกขึ้นมาให้เราพร้อมไฟล์เริ่มต้นที่มีฟังก์ชัน “main” ไปด้วย
5. ทำการอินคลูดไฟล์ “stdio.h” เข้ามายังโปรเจกและสั่งให้โปรเจกปรี้น “Hello Microblaze!” ออกมาทาง STDOUT ซึ่งถูกแมพไปยังยูอาร์ท โมดูลแล้ว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
// Welcome to Xilinx Platform Studio SDK !
//
// This is an automatically created source file to help you get a
// To add more files, navigate to File -> New -> File
// You may delete this file if you want to use only other files f
//
#include <stdio.h>
int main()
{
    xil_printf("Hello Microblaze!\r\n");
    return 0;
}
```

รูปที่ 4.13 โปรแกรม "Hello Microblaze!"

6. ทำการเซฟ โปรแกรมจะถูกคอมไพล์และลิงค์เองโดยอัตโนมัติ
7. เชื่อมต่อพอร์ตอาร์เอส232 ของบอร์ดเข้ากับพีซี จากนั้นเปิด โปรแกรมไฮเปอร์เทอร์มินอลหรือโปรแกรมอื่นที่เทียบเท่าขึ้นมาแล้วทำการตั้งค่าบอดเราให้ตรงกับที่ตั้งไว้ในยูอาร์ทีโมดูล
8. ในเอ็กซ์พีเอสเอสดีเค กดที่เมนู "Run > Run" แล้วในแถบ "Main" ตรง "Project" กด "Browse" แล้วเลือกไปยังโปรเจกของเราและตรง "C/C++ Application" กด "Search" แล้วเลือกไปยังไฟล์ .elf ในโปรเจกของเรา กด "Apply" และ "Close"



รูปที่ 4.14 การตั้งค่าการรัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

9. กดปุ่ม “Run” สีเขียวตรงทูลบาร์ด้านบน โปรแกรมเอ็กซ์พีเอสเอสดีเคจะทำการส่งโปรแกรมของเราลงไปทำงานบนบอร์ดผ่านทางดีบั๊กโมดูล



รูปที่ 4.15ปุ่มรัน

10. คุณผลลัพธ์ในโปรแกรมเทอร์มินัลที่เปิดทิ้งไว้ ไมโครเบลสจจะส่งข้อความ “Hello Microblaze!” ออกมาทางพอร์ตอาร์เอส232ดังรูป



รูปที่ 4.16ผลการทำงานของโปรแกรม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5

การใช้งานจีพีไอโอบนระบบไมโครเบลต

5.1 ภาพรวม

เนื้อหาในบทนี้จะเป็นเอกสารประกอบการทดลองการใช้งานจีพีไอโอ (GPIO (General Purpose Input Output)) ของระบบ ไมโครเบลต โดยจะทำการส่งเอาต์พุตออกไปยังหลอดแอลอีดี และทำการอ่านอินพุตจากสวิตช์

5.2 ทำการเพิ่มไอพีคอร์ลงในระบบไมโครเบลต

สำหรับในการทดลองนี้เราจะใช้ระบบไมโครเบลตเดิมที่สร้างไว้จากการทดลองที่แล้วมาทำการเพิ่มไอพีคอร์ (IP Core (Intellectual Properties Core)) ที่เป็นจีพีไอโอเข้าไป โดยมีขั้นตอนการทำดังนี้

1. เปิดโปรแกรมไซลิ่งแพลตฟอร์มสตูดิโอขึ้นมาแล้วทำการเปิดโปรเจกต์ที่เคยทำไว้จากการทดลองที่แล้วขึ้นมา
2. ไปที่แถบ “IP Catalog” แล้วเลือกไปที่ “General Purpose IO” แล้วทำการดับเบิลคลิกที่ “opb_gpio” เพื่อทำการเพิ่มจีพีไอโอ ไอพีคอร์ ลงในระบบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

⊕ Analog
 ⊕ Bus
 ⊕ Bus Bridge
 ⊕ Clock Control
 ⊕ Communication High-Speed
 ⊕ Communication Low-Speed
 ⊕ Debug
 ⊕ DMA
 ⊖ General Purpose IO

- opb_gpio 3.01.a OPB General Purpose IO
- opb_gpio 3.01.b OPB General Purpose IO

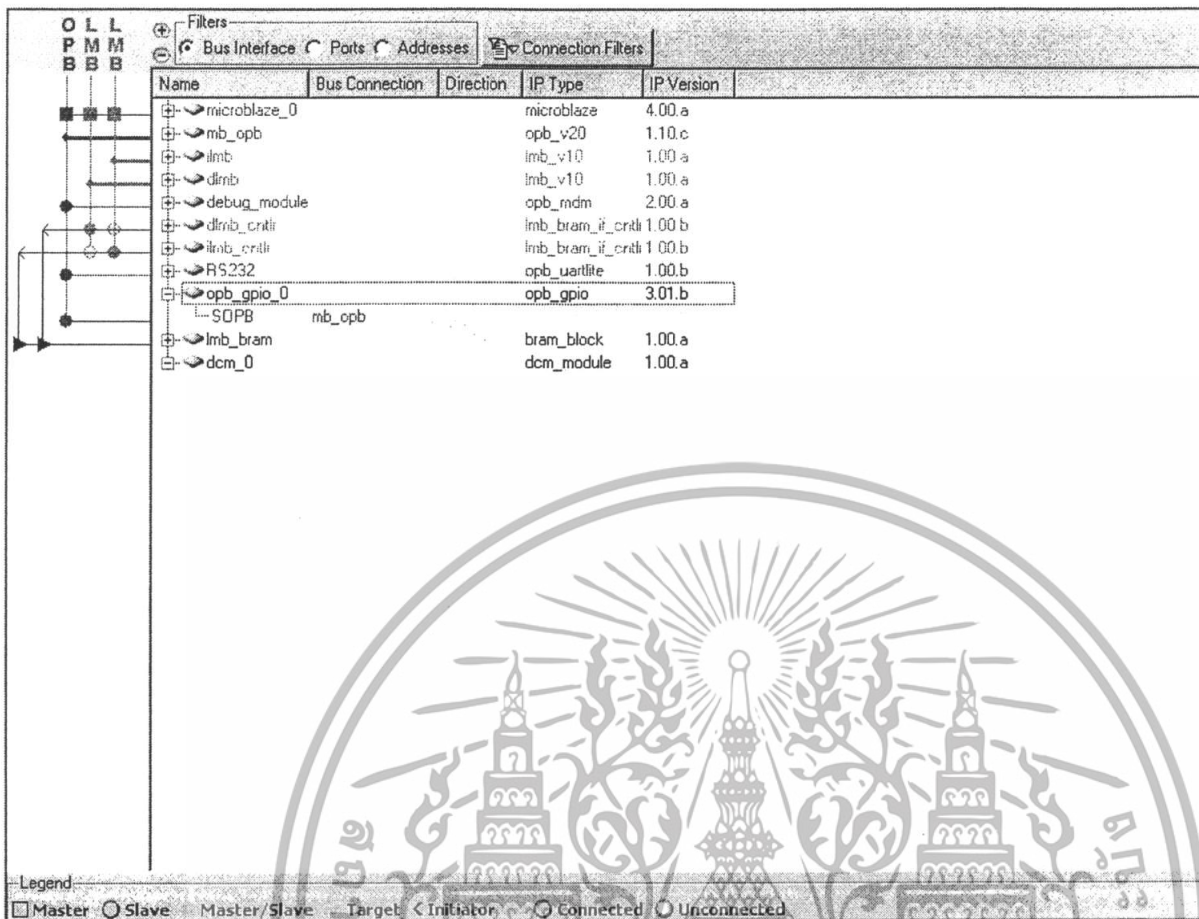
 ⊕ Interrupt Control
 ⊕ Memory Block
 ⊕ Memory Controller
 ⊕ PCI
 ⊕ Peripheral Controller
 ⊕ Processor
 ⊕ Reset Control
 ⊕ Timer
 ⊕ Utility

Processor Support: PowerPC & MicroBlaze
Status: Active
Early Access: False
Directory: C:
 \EDK\hw\XilinxProcessorIPLib\cores\opb_gpio_v3_01_b\

รูปที่ 5.1 การเพิ่มจีพีไอโอ

3. ทำการคลิกที่จุดสีเขียวเพื่อเชื่อมต่อ "opb_gpio_0" เข้ากับ ไอพีบี

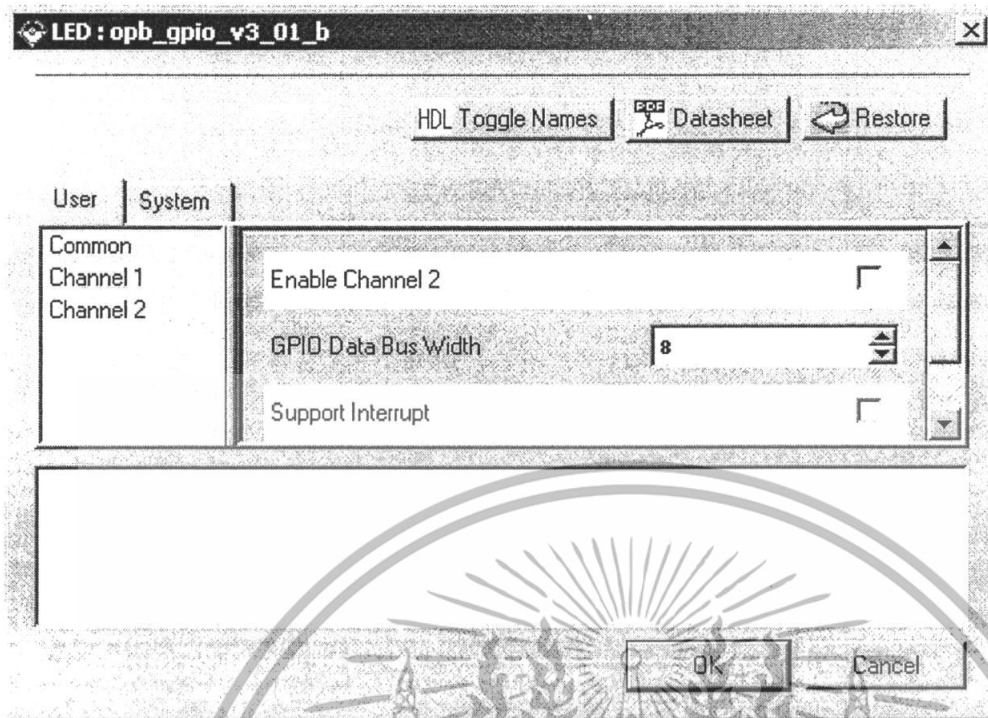
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.2 การเชื่อมต่อไฟไอโอเข้ากับไอพีบี

4. ทำการแก้ไขชื่อ "opb_gpio_0" เป็น "LED"
5. ดับเบิ้ลคลิกที่ "LED" แล้วทำการตั้งค่า "GPIO Data Bus Width" ให้เท่ากับจำนวนแอสลิตบนบอร์ด ซึ่งในที่นี้เป็นแปดดวง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.3 การแก้ความกว้างบัสของจีพีไอโอ

- กดตรง “Filters” ไปที่ “Ports” แล้วดูที่ “LED” ตรงแถบ “Net” ที่ “GPIO_IO” เปลี่ยนจาก “No Connection” เป็น “Make External” เพื่อให้ขาของจีพีไอโอเชื่อมต่อออกไปนอกตัวชิพดังกล่าว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Filters
 Bus Interface Ports Addresses Filters (Applied) Add External Port

Name	Net	Direction	Class	Sensitivity	Range	IP Type	IP Version
External Ports							
microblaze_0						microblaze	4.00.a
mb_opb						opb_v20	1.10.c
lmb						lmb_v10	1.00.a
dlmb						lmb_v10	1.00.a
debug_module						opb_mdm	2.00.a
dlmb_cntrl						lmb_bram_if_cntrl	1.00.b
ilmb_cntrl						lmb_bram_if_cntrl	1.00.b
RS232						opb_uartlite	1.00.b
LED						opb_gpio	3.01.b
IP2INTC_lrp	No Connection	0	INTE...	LEVEL_HI...			
GPIO_IO	No Connection	IO				[0:(C_G...	
GPIO_in	No Connection	I				[0:(C_G...	
GPIO_d_out	No Connection	O				[0:(C_G...	
GPIO_t_out	LED_GPIO_IO	O				[0:(C_G...	
GPIO2_IO	No Connection	IO				[0:(C_G...	
GPIO2_in	No Connection	I				[0:(C_G...	
GPIO2_d_out	No Connection	O				[0:(C_G...	
GPIO2_t_out	No Connection	O				[0:(C_G...	
lmb_bram						bram_block	1.00.a
dcm_0						dcm_module	1.00.a

รูปที่ 5.4 การทำขาพินให้เชื่อมต่อกับภายนอก

7. ไปดูที่ “External Ports” จะพบว่า มี “LED_GPIO_IO_pin” ขั้วมาซึ่งเชื่อมต่อกับเน็ตที่ชื่อว่า “LED_GPIO_IO”

Name	Net	Direction	Class	Sensitivity	Range	IP Type	IP Version
External Ports							
fpga_0_RS...	net_gnd	0					
fpga_0_RS...	fpga_0_RS232...	I					
fpga_0_RS...	fpga_0_RS232...	O					
sys_clk_pin	dcm_clk_s	I	DCM...				
sys_rst_pin	sys_rst_s	I					
LED_GPIO...	LED_GPIO_IO	IO			[0:7]		
microblaze_0						microblaze	4.00.a
mb_opb						opb_v20	1.10.c
lmb						lmb_v10	1.00.a
dlmb						lmb_v10	1.00.a
debug_module						opb_mdm	2.00.a
dlmb_cntrl						lmb_bram_if_cntrl	1.00.b
ilmb_cntrl						lmb_bram_if_cntrl	1.00.b
RS232						opb_uartlite	1.00.b
LED						opb_gpio	3.01.b
lmb_bram						bram_block	1.00.a
dcm_0						dcm_module	1.00.a

รูปที่ 5.5 พอร์ทของระบบหลังจากทำการเพิ่มขาภายนอก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่นิยมนำไปใช้ประโยชน์ด้านการค้า
 ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

8. เปลี่ยน “Filters” ไปที่ “Addresses” จะพบว่าตรง “LED” จะยังไม่มีแอดเดรสถูกแมพอยู่ที่หัวคอปุม
“Generate Addresses” เพื่อทำการสร้างแอดเดรสแมพใหม่

Filters										
Generate Addresses										
Name /	Address	Base Address	High Address	Size	Lock	Bus Connection	Direction	IP Type	IP Version	Instance
				U	<input type="checkbox"/>					mb_opb
SLMB		0x00000000	0x00003fff	16K	<input type="checkbox"/>	dlmb				dlmb_cntrl
SLMB		0x00000000	0x00003fff	16K	<input type="checkbox"/>	ilmb				ilmb_cntrl
SOPB				U	<input type="checkbox"/>	mb_opb				LED
SOPB		0x40600000	0x4060ffff	64K	<input type="checkbox"/>	mb_opb				RS232
SOPB		0x41400000	0x4140ffff	64K	<input type="checkbox"/>	mb_opb				debug_module

รูปที่ 5.6 การทำแอดเดรสแมพ

9. ทำซ้ำขั้นตอนเดิมทั้งหมดในการเพิ่ม “opb_gpio” เข้าสู่ระบบอีกครั้ง โดยครั้งนี้ให้ตั้งชื่อเป็น
“switch” สร้างพอร์ตภายนอก และทำแอดเดรสแมพให้เรียบร้อย



Filters				
Connection Filters				
Name	Bus Connection	Direction	IP Type	IP Version
microblaze_0	microblaze		microblaze	4.00.a
mb_opb	opb_v20		opb_v20	1.10.c
ilmb	lmb_v10		lmb_v10	1.00.a
dlmb	lmb_v10		lmb_v10	1.00.a
debug_module	opb_mdm		opb_mdm	2.00.a
dlmb_cntrl	lmb_bram_if_cntrl		lmb_bram_if_cntrl	1.00.b
ilmb_cntrl	lmb_bram_if_cntrl		lmb_bram_if_cntrl	1.00.b
RS232	opb_uartlite		opb_uartlite	1.00.b
LED	opb_gpio		opb_gpio	3.01.b
switch	opb_gpio		opb_gpio	3.01.b
lmb_bram	bram_block		bram_block	1.00.a
dcm_0	dcm module		dcm module	1.00.a

รูปที่ 5.7 ภาพรวมของระบบหลังจากเพิ่มสวิตช์แล้ว

10. ทำการแก้ไขยูซีเอฟไฟล์เพื่อแมพขาของสวิตช์และแอลอีดีบนบอร์ดจริง โดยใช้ชื่อเน็ทตามชื่อพอร์ต
ภายนอก (LED_GPIO_IO_pin และ switch_GPIO_IO_pin) โดยใส่ <0> ถึง <7> ต่อท้ายไว้เพื่อเป็น
การบอกว่าเป็นขาที่เท่าไร ทำการแมพขาทั้งหมดไปยังบอร์ด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

4  ## user constraints to this file based on customer desi
5  #####
6
7  Net sys_clk_pin LOC=P127;
8  Net sys_rst_pin LOC=P11;
9  ## System level constraints
10 Net sys_clk_pin TNM_NET = sys_clk_pin;
11 TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 40000 ps;
12 Net sys_rst_pin TIG;
13
14 ## IO Devices constraints
15
16 ##### Module RS232 constraints
17
18 # Net fpga_0_RS232_req_to_send_pin LOC=;
19 Net fpga_0_RS232_RX_pin LOC=P129;
20 Net fpga_0_RS232_TX_pin LOC=P128;
21
22 Net LED_GPIO_IO_pin<0> LOC=P35;
23 Net LED_GPIO_IO_pin<1> LOC=P33;
24 Net LED_GPIO_IO_pin<2> LOC=P32;
25 Net LED_GPIO_IO_pin<3> LOC=P31;
26 Net LED_GPIO_IO_pin<4> LOC=P30;
27 Net LED_GPIO_IO_pin<5> LOC=P28;
28 Net LED_GPIO_IO_pin<6> LOC=P27;
29 Net LED_GPIO_IO_pin<7> LOC=P26;
30
31 Net switch_GPIO_IO_pin<0> LOC=P25;
32 Net switch_GPIO_IO_pin<1> LOC=P24;
33 Net switch_GPIO_IO_pin<2> LOC=P23;
34 Net switch_GPIO_IO_pin<3> LOC=P21;
35 Net switch_GPIO_IO_pin<4> LOC=P20;
36 Net switch_GPIO_IO_pin<5> LOC=P18;
37 Net switch_GPIO_IO_pin<6> LOC=P17;
38 Net switch_GPIO_IO_pin<7> LOC=P15;
39

```

รูปที่ 5.8 การแมพขาบนยูซีเอฟไฟล์

11. ทำการสร้างและอัปเดตบิตสตรีม แล้วดาวน์โหลดบิตสตรีมลงบอร์ดตามขั้นตอนที่เคยทำในบทที่แล้ว จากนั้นทำการสร้างไลบรารีไฟล์ให้เรียบร้อยและเปิดตัวเอ็กซ์พีเอสเอสดีเคขึ้นมา

5.3 การเขียนโปรแกรมเพื่อควบคุมจีพีไอโอ

1. สร้างโปรเจกใหม่ในเอ็กซ์พีเอสเอสดีเคตามขั้นตอนเดิมในบทที่แล้ว หรือจะใช้โปรเจกเดิมก็ได้ ทำการอินคลูดไฟล์เพิ่มอีกสองไฟล์ดังนี้

a. “xparameters.h” เป็นไฟล์ที่ทำการประกาศแอดเดรสแมพของแต่ละอุปกรณ์ในระบบไว้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- b. “xgpio_1.h” เป็นไฟล์ที่ประกาศโปรโตไทป์ที่สำคัญๆ ของฟังก์ชันที่เกี่ยวกับการใช้งานจีพีไอโอเอาไว้

2. สำหรับการอ่านค่าจากจีพีไอโออินพุต ก่อนอื่นเราต้องใช้ฟังก์ชัน

XGpio_mSetDataDirection(BaseAddress, Channel, DirectionMask)

ในการเขียนค่าลงไปยังรีจิสเตอร์ทิศทางของจีพีไอโอก่อน โดยพารามิเตอร์ของฟังก์ชันนี้มีดังนี้

- BaseAddress เป็นค่าเบสแอดเดรสของโมดูลจีพีไอโอที่ต้องการใช้ สามารถดูได้จากไฟล์ “xparameters.h”
- Channel คือแชนเนลของจีพีไอโอที่ต้องการใช้งานซึ่งจากระบบที่เราสร้างแต่ละจีพีไอโอจะมีเพียงแชนเนลเดียวคือแชนเนล 1
- DirectionMask เป็นบิตแมสก์ที่บอกว่าขาใดของจีพีไอโอจะเป็นอินพุตและขาใดจะเป็นเอาต์พุต โดยถ้าตั้งเป็น 0 จะเป็นเอาต์พุตและ 1 จะเป็นอินพุต

XGpio_mGetDataReg(BaseAddress, Channel)

เป็นฟังก์ชันที่ใช้ในการอ่านค่าขึ้นมาจากรีจิสเตอร์ของจีพีไอโอ โดยพารามิเตอร์ของฟังก์ชันนี้ นั้นมีความหมายเหมือนกับของฟังก์ชันที่ผ่านมา และจะรีเทิร์นค่าที่อยู่ภายในรีจิสเตอร์ออกมา

3. ทำการเขียนโปรแกรมให้อ่านค่าจากสวิตช์แล้วทำการพิมพ์ออกทางยูอาร์ทีโมดูลดังต่อไปนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
// Welcome to Xilinx Platform Studio SDK !
//
// This is an automatically created source file to help you get s
// To add more files, navigate to File -> New -> File
// You may delete this file if you want to use only other files f
//
#include <stdio.h>
#include "xparameters.h"
#include "xgpio_1.h"

int main()
{
    unsigned char sw;
    XGpio_mSetDataDirection(XPAR_SWITCH_BASEADDR, 1, 0xFF);
    sw = XGpio_mGetDataReg(XPAR_SWITCH_BASEADDR, 1);
    xil_printf("0x%X\r\n", sw);
    return 0;
}
```

รูปที่ 5.9 โปรแกรมอ่านค่าจากสวิตช์

4. ทดลองรันโปรแกรม จากนั้นสอง โย๊กสวิตช์เปลี่ยนแล้วกดปุ่มรีเซตบนบอร์ดจะได้ผลลัพธ์ที่เทอร์มินอลดังนี้



รูปที่ 5.10 ค่าที่พิมพ์ออกมาทางเทอร์มินอลหลังจากเปลี่ยนสวิตช์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5. สำหรับการเขียนค่าออกไปแสดงผลทางจีพีไอโอ นั้นเราจำเป็นต้องใช้ฟังก์ชันเหล่านี้

```
XGpio_mSetDataDirection(BaseAddress, Channel, DirectionMask)
```

ใช้ในการเขียนค่าลงในรีจิสเตอร์ทิศทางของจีพีไอโอ ดังที่ได้กล่าวไปแล้ว

```
XGpio_mSetDataReg(BaseAddress, Channel, Data)
```

ใช้ในการเขียนค่าลงในค่ารีจิสเตอร์ของจีพีไอโอ โดยพารามิเตอร์ “Data” คือข้อมูลที่เรากำลังจะเขียนออกไปยังจีพีไอโอ

6. ทำการเขียนโปรแกรมให้อ่านค่าจากสวิตช์แล้วนำมาแสดงผลที่แอลอีดีดังต่อไปนี้

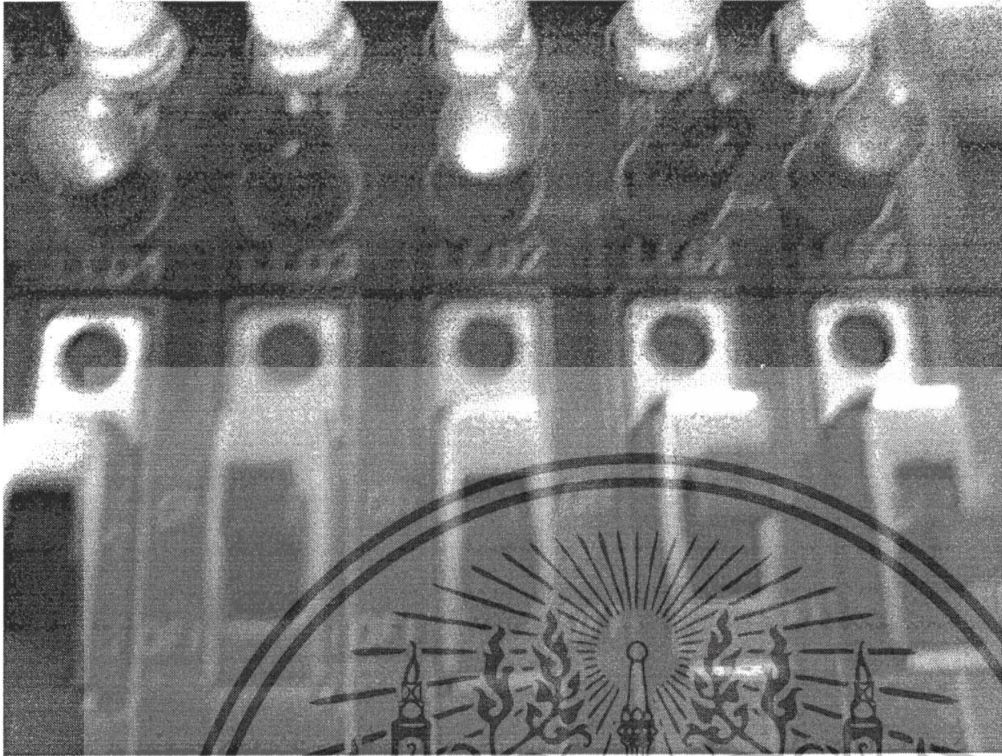
```
// Welcome to Xilinx Platform Studio SDK !
//
// This is an automatically created source file to help you get s
// To add more files, navigate to File -> New -> File
// You may delete this file if you want to use only other files f
//
#include <stdio.h>
#include "xparameters.h"
#include "xgpio_1.h"

int main()
{
    unsigned char sw;
    XGpio_mSetDataDirection(XPAR_SWITCH_BASEADDR, 1, 0xFF);
    XGpio_mSetDataDirection(XPAR_LED_BASEADDR, 1, 0x00);
    while(1)
    {
        sw = XGpio_mGetDataReg(XPAR_SWITCH_BASEADDR, 1);
        XGpio_mSetDataReg(XPAR_LED_BASEADDR, 1, sw);
    }
    return 0;
}
```

รูปที่ 5.11 โปรแกรมในการอ่านค่าสวิตช์และเขียนออกแอลอีดี

7. ทดลองนำโปรแกรมไปทำงานดูจะพบว่าแอลอีดีจะติดและดับตามสวิตช์ที่เปลี่ยนไปดังรูป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.12 การทำงานของสวิตช์และแอตอมีดี



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 6

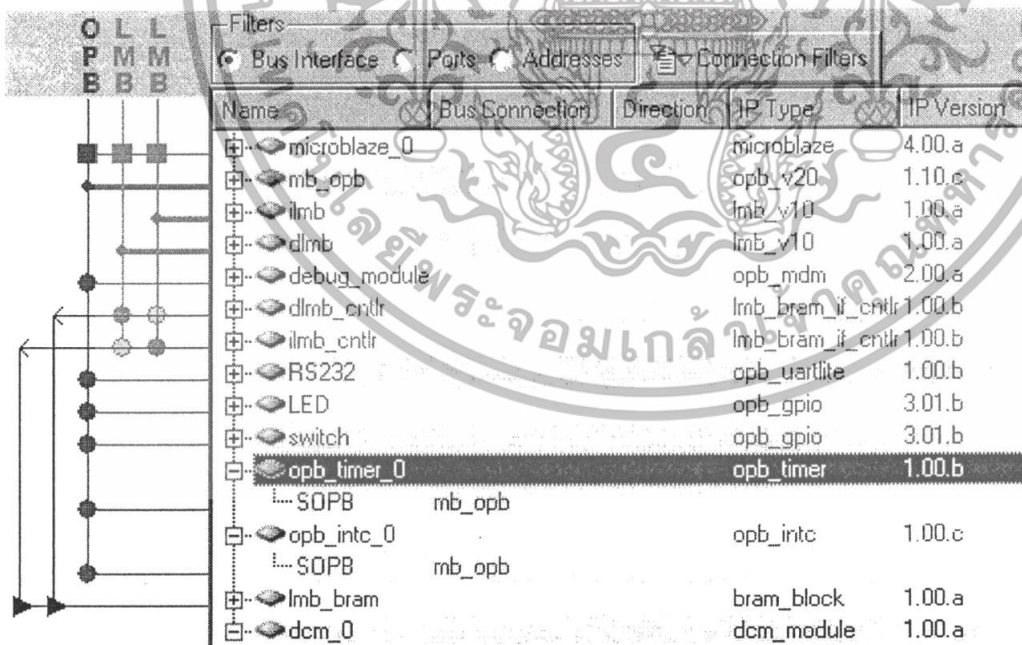
การใช้งานอินเทอร์พท์และไทม์เมอร์ในระบบไมโครเบลส

6.1 ภาพรวม

เนื้อหาในบทนี้จะเป็นเอกสารประกอบการทดลองการใช้งานไทม์เมอร์และอินเทอร์พท์คอนโทรลเลอร์ในระบบไมโครเบลส โดยจะทำการเพิ่มอินเทอร์พท์คอนโทรลเลอร์และไทม์เมอร์ลงในระบบที่ได้สร้างไว้ในบทที่แล้ว

6.2 การเพิ่มไทม์เมอร์และอินเทอร์พท์คอนโทรลเลอร์ลงในระบบ

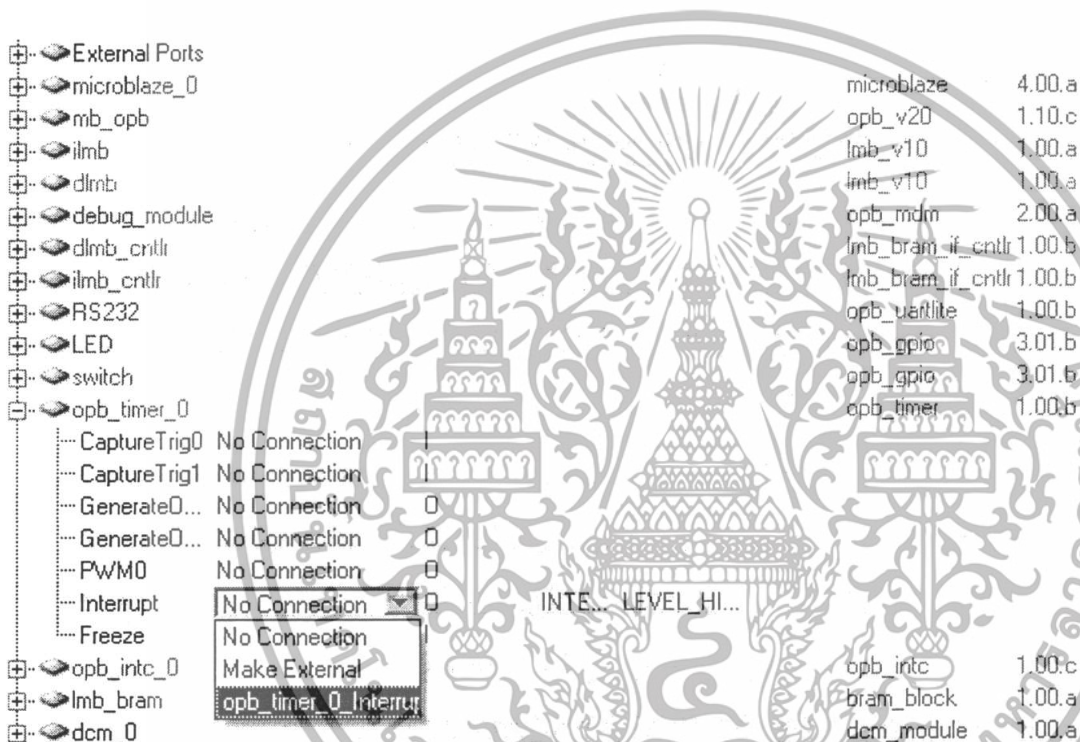
1. ทำการเปิดโปรแกรมเอ็กซ์พีเอสขึ้นมาแล้วทำการเปิดโปรเจกต์ที่ได้ทำไว้ในบทที่แล้วขึ้นมา
2. ในแถบ “IP Catalog” ทำการเพิ่ม “Timer > opb_timer” และ “Interrupt Control > opb_intc” ลงในระบบ
3. คลิกที่วงกลมสีเขียวเพื่อเชื่อมต่อไอพีคอร์ทั้งสองเข้ากับไอพีบี



รูปที่ 6.1 เพิ่มไทม์เมอร์และอินเทอร์พท์คอนโทรลเลอร์เข้าสู่ระบบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

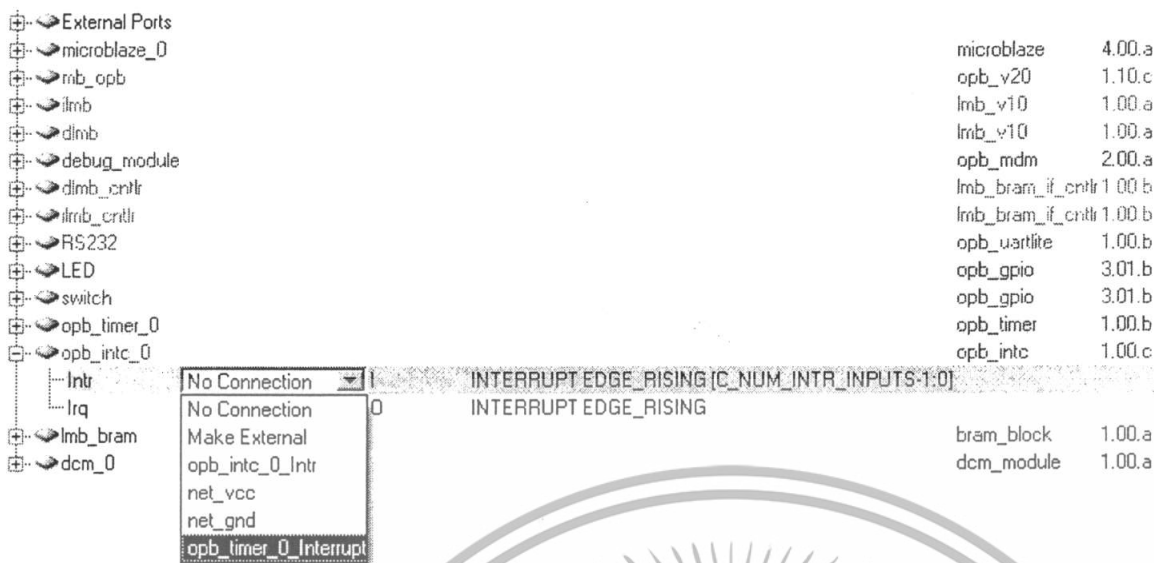
4. ในไทม์เมอร์หนึ่งชุดนั้นจะมีไทม์เมอร์ทั้งสิ้นสองตัว แต่ในระบบนี้เราจะใช้เพียงตัวเดียว ดังนั้นทำการดับเบิลคลิกที่ “opb_timer_0” และเลือก “Only One Timer is present”
5. ดับเบิลคลิกที่ “opb_intc_0” แล้วตั้งค่า “Number of Interrupt Inputs” ให้เป็น 1 เนื่องจากเราจะตอบสนองสัญญาณอินเทอร์รัพท์จากไทม์เมอร์เพียงแค่มอดุลเดียว
6. เปลี่ยน “Filters” ไปที่ “Ports” แล้วดูที่ “opb_timer_0” ตรง “Interrupt” ตั้งเน็ตไปที่ “opb_timer_0_Interrupt”



รูปที่ 6.2 การตั้งค่าเน็ตของไทม์เมอร์

7. ดูที่ opb_intc_0 ทำการแมพ Intr ไปยังเน็ต “opb_timer_0_Interrupt” เพื่อให้สัญญาณอินเทอร์รัพท์จากไทม์เมอร์วิ่งเข้าสู่อินเทอร์รัพท์คอนโทรลเลอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 6.3 การตั้งค่าเน็ท Intr ของอินเทอร์รัพท์คอนโทรล

8. ทำการแมพ "Irq" ของ "opb_intc_0" ไปยังเน็ทที่ชื่อ "opb_intc_0_Irq"



รูปที่ 6.4 การตั้งค่าเน็ท Irq ของอินเทอร์รัพท์คอนโทรล

9. คู่มือ "microblaze_0" ทำการแมพสัญญาณ INTERRUPT ไปที่เน็ท "opb_intc_0_Irq" เพื่อให้สัญญาณอินเทอร์รัพท์ที่ถูกสร้างจากอินเทอร์รัพท์คอนโทรลเลอร์วิ่งเข้าสู่ตัวไมโครเบลส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

The screenshot shows a hardware configuration interface. On the left, a tree view lists components under 'microblaze_0', including 'CLK', 'INTERRUPT', 'DBG_STOP', 'DBG_CLK', 'DBG_TDI', 'DBG_TDO', 'DBG_REG...', 'DBG_CAP...', 'DBG_UPD...', and 'MB_Halted'. The 'INTERRUPT' pin is connected to 'INTERRUPT_LEVEL_HIGH'. A dropdown menu is open for the 'INTERRUPT' pin, showing options: 'No Connection', 'Make External', 'microblaze_0_INTER...', 'net_vcc', 'net_gnd', 'opb_intc_0_irq', and 'opb_timer_0 Interrupt...'. The 'opb_intc_0_irq' option is selected. On the right, a list of components is shown with their versions: opb_v20 (1.10.c), lmb_v10 (1.00.a), dcm_module (1.00.a), lmb_bram_if_cntlr (1.00.b), opb_uartlite (1.00.b), opb_gpio (3.01.b), opb_timer (1.00.b), opb_intc (1.00.c), and dcm_module (1.00.a).

รูปที่ 6.5 การตั้งค่า pin INTERRUPT ของตัวไมโครเบลล์

10. เปลี่ยน “Filters” ไปที่ “Addresses” แล้วทำการเงินเนอเรทแอดเดรสแมพใหม่
11. เนื่องจากการทดลองนี้ไม่มีการเปลี่ยนแปลงภายนอกจึงไม่ต้องแก้ไขยูซีเอไฟล์ใหม่ ทำการสร้างและอัปเดตบิตสตรีมแล้วดาวน์โหลดบอร์ดได้เลย
12. ทำการสร้างไลบรารีไฟล์ต่างๆให้พร้อมแล้วเปิดเอ็กซ์พีเอสเอสดีเคขึ้นมา

6.3 การเขียนโปรแกรมเพื่อใช้งานไทม์เมอร์ และอินเทอร์รัพท์คอนโทรลเลอร์

6.3.1 การเขียนโปรแกรมเพื่อใช้งานไทม์เมอร์

1. สำหรับการเขียนโปรแกรมกับไทม์เมอร์ เราต้องทำการอินคลูดไฟล์ต่อไปนี้เข้ามาในโปรแกรม
 - a. “xparameters.h” เป็นไฟล์ที่ทำการประกาศแอดเดรสแมพของแต่ละอุปกรณ์ในระบบไว้
 - b. “xtmrctr_1.h” เป็นไฟล์ที่ทำการประกาศโปรโตไทป์ของฟังก์ชันที่เกี่ยวกับไทม์เมอร์
2. ในการเขียนโปรแกรมเพื่อควบคุมไทม์เมอร์นั้นมีฟังก์ชันสำคัญๆ ดังนี้

```
void XTmrCtr_mSetLoadReg(Xuint32 BaseAddress, Xuint8 TmrCtrNumber, Xuint32
```

RegisterValue)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ใช้ในการเขียนค่าลงในรีจิสเตอร์เริ่มต้นของตัวไทม์เมอร์

```
void XTmrCtr_mSetControlStatusReg(Xuint32 BaseAddress, Xuint8 TmrCtrNumber, Xuint32
RegisterValue)
```

ใช้ในการเขียนค่าลงในคอนโทรลรีจิสเตอร์ของไทม์เมอร์

```
Xuint32 XTmrCtr_mGetTimerCounterReg(Xuint32 BaseAddress, Xuint8 TmrCtrNumber)
```

ใช้ในการอ่านค่าจากรีจิสเตอร์เคาท์เตอร์ในตัวไทม์เมอร์

3. ทำการเขียนโปรแกรมให้ไทม์เมอร์ทำการนับลงและพิมพ์ค่าในรีจิสเตอร์ออกมาทางยูอาร์ทีโมดูล
ดังนี้

```
// Welcome to Xilinx Platform Studio SDK !
//
// This is an automatically created source file to help you get started.
// To add more files, navigate to File => New => File
// You may delete this file if you want to use only other files for your project.
//
#include <stdio.h>
#include "xparameters.h"
#include "xgpio_1.h"
#include "xtmrctr_1.h"

int main()
{
    unsigned long tmr_count;
    /* Set the number of cycles the timer counts*/
    XTmrCtr_mSetLoadReg(XPAR_OPB_TIMER_0_BASEADDR, 0, 0xFFFFFFFF);
    /* Reset the timers, and clear interrupts */
    XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_0_BASEADDR,
    0, XTC_CSR_INT_OCCURED_MASK | XTC_CSR_LOAD_MASK );
    /* Start the timers */
    XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_0_BASEADDR,
    0, XTC_CSR_ENABLE_TMR_MASK | XTC_CSR_AUTO_RELOAD_MASK |
    XTC_CSR_DOWN_COUNT_MASK);
    while(1)
    {
        tmr_count = XTmrCtr_mGetTimerCounterReg(XPAR_OPB_TIMER_0_BASEADDR, 0);
        xil_printf("0x%X\r\n", tmr_count);
    }
    return 0;
}
|
```

รูปที่ 6.6 โปรแกรมไทม์เมอร์นับลง

4. ทดลองนำโปรแกรมไปทำงานดูจะได้ผลลัพธ์ดังต่อไปนี้ซึ่งแสดงว่าไทม์เมอร์กำลังทำการนับลงอยู่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

Tera Term - COM3 VT
File Edit Setup Control Window Help
0xEB5F370B
0xEB5A780A
0xEB55B909
0xEB50FA08
0xEB4C3B07
0xEB477C06
0xEB42BD12
0xEB3DFE11
0xEB393F10
0xEB34800F
0xEB2FC10E
0xEB2B020D
0xEB26430C
0xEB21840B
0xEB1CC50A
0xEB180609
0xEB134708
0xEB0E8807
0xEB09C906
0xEB050A12
0xEB004B11
0xEAFB8C10
0xEAF6CD0F

```

รูปที่ 6.7 ผลลัพธ์ของโปรแกรมไทม์เมอร์นับลง

6.3.2 การเขียนโปรแกรมเพื่อใช้งานอินเทอร์รัพท์

1. สำหรับการเขียนโปรแกรมกับอินเทอร์รัพท์ เราต้องทำการอินคลูดไฟล์ต่อไปนี้เข้ามาในโปรแกรม
 - a. “xparameters.h” เป็นไฟล์ที่ทำการประกาศแอดเดรสเมมพ์ของแต่ละอุปกรณ์ในระบบไว้
 - b. “xintc_1.h” เป็นไฟล์ที่ทำการประกาศโปรโตไทป์ของฟังก์ชันที่เกี่ยวข้องกับอินเทอร์รัพท์คอนโทรลเลอร์

2. ฟังก์ชันที่สำคัญๆในการใช้งานอินเทอร์รัพท์มีดังนี้

`microblaze_enable_interrupts()`

ใช้ในการเปิดการทำงานอินเทอร์รัพท์ของไมโครเบลส

`void XIntc_RegisterHandler(Xuint32 BaseAddress, int InterruptId, XInterruptHandler Handler,`

`void *CallBackRef)`

ใช้ในการรีจิสเตอร์ฟังก์ชันตอบสนองการทำงานของอินเทอร์รัพท์

`void XIntc_mMasterEnable(Xuint32 BaseAddress)`

เป็นฟังก์ชันที่ใช้ในการเปิดการทำงานของอินเทอร์รัพท์คอนโทรลเลอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
void XIntc_mEnableIntr(Xuint32 BaseAddress, Xuint32 EnableMask)
```

ใช้ในการเปิดอินเทอร์รัพท์ที่เรีควสสำหรับสัญญาณจากแต่ละแหล่ง

3. สร้างฟังก์ชันตอบสนองการทำงานของอินเทอร์รัพท์ดังนี้

```
void timer_int_handler(void * baseaddr_p)
{
    unsigned int csr;
    csr = XTmrCtr_mGetControlStatusReg(XPAR_OPB_TIMER_0_BASEADDR, 0);
    xil_printf("Interrupted!\r\n");
    XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_0_BASEADDR, 0, csr);
}
```

รูปที่ 6.8 ฟังก์ชันตอบสนองการทำงานของอินเทอร์รัพท์

4. เขียนโปรแกรมเพื่อรีจิสเตอร์ฟังก์ชันตอบสนองการทำงานของอินเทอร์รัพท์และสั่งให้ไทม์เมอร์ทำงานดังนี้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

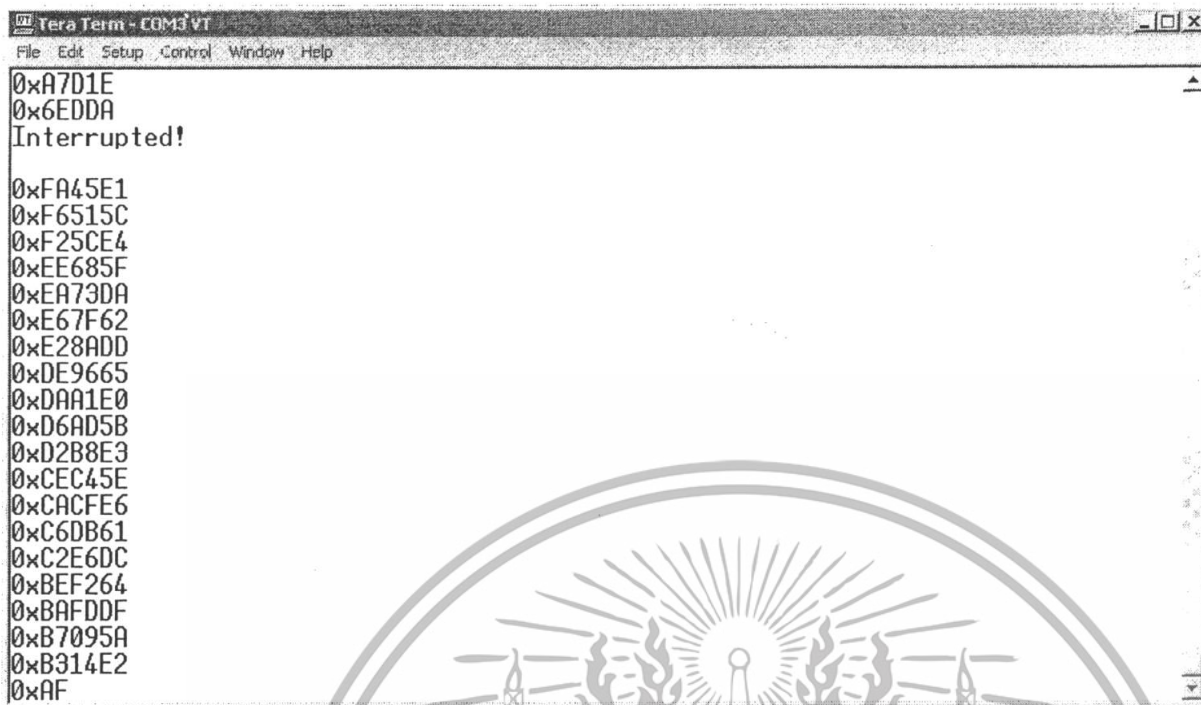
int main()
{
    unsigned long tmr_count;
    /* Enable MicroBlaze interrupts */
    microblaze_enable_interrupts();
    /* Register the timer interrupt handler in the vector table */
    XIntc_RegisterHandler(XPAR_OPB_INTC_O_BASEADDR,
    XPAR_OPB_INTC_O_OPB_TIMER_O_INTERRUPT_INTR,
    (XInterruptHandler)timer_int_handler, (void *)XPAR_OPB_TIMER_O_BASEADDR);
    /* Start the interrupt controller */
    XIntc_mMasterEnable(XPAR_OPB_INTC_O_BASEADDR);
    /* Enable timer interrupt requests in the interrupt controller */
    XIntc_mEnableIntr(XPAR_OPB_INTC_O_BASEADDR, XPAR_OPB_TIMER_O_INTERRUPT_MASK)
    /* Set the number of cycles the timer counts*/
    XTmrCtr_mSetLoadReg(XPAR_OPB_TIMER_O_BASEADDR, 0, 0xFFFFFFFF);
    /* Reset the timers, and clear interrupts */
    XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_O_BASEADDR,
    0, XTC_CSR_INT_OCCURED_MASK | XTC_CSR_LOAD_MASK );
    /* Start the timers */
    XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_O_BASEADDR,
    0, XTC_CSR_ENABLE_TMR_MASK | XTC_CSR_AUTO_RELOAD_MASK |
    XTC_CSR_ENABLE_INT_MASK | XTC_CSR_DOWN_COUNT_MASK);
    while(1)
    {
        tmr_count = XTmrCtr_mGetTimerCounterReg(XPAR_OPB_TIMER_O_BASEADDR, 0);
        xil_printf("0x%X\r\n", tmr_count);
    }
    return 0;
}

```

รูปที่ 6.9 โปรแกรมรีจิสเตอร์ฟังก์ชันตอบสนองอินเทอร์รัพท์

5. ทดลองนำโปรแกรมไปทำงานจะได้ผลลัพธ์ดังต่อไปนี้ พบว่าทุกครั้งที่ไทม์เมอร์นับลงจนถึง 0 จะเกิดการอินเทอร์รัพท์ขึ้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



```

Tera Term - COM3 VT
File Edit Setup Control Window Help
0xA7D1E
0x6EDDA
Interrupted!
0xFA45E1
0xF6515C
0xF25CE4
0xEE685F
0xEA73DA
0xE67F62
0xE28ADD
0xDE9665
0xDAA1E0
0xD6AD5B
0xD2B8E3
0xCEC45E
0xCACFE6
0xC6DB61
0xC2E6DC
0xBEF264
0xBAFDDF
0xB7095A
0xB314E2
0xAF

```

รูปที่ 6.10 ผลลัพธ์ของโปรแกรมตอบสนองอินเทอร์รัพท์



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 7

การสร้างไอพ็คเจอร์เพื่อใช้กับระบบไมโครเบลส

7.1 ภาพรวม

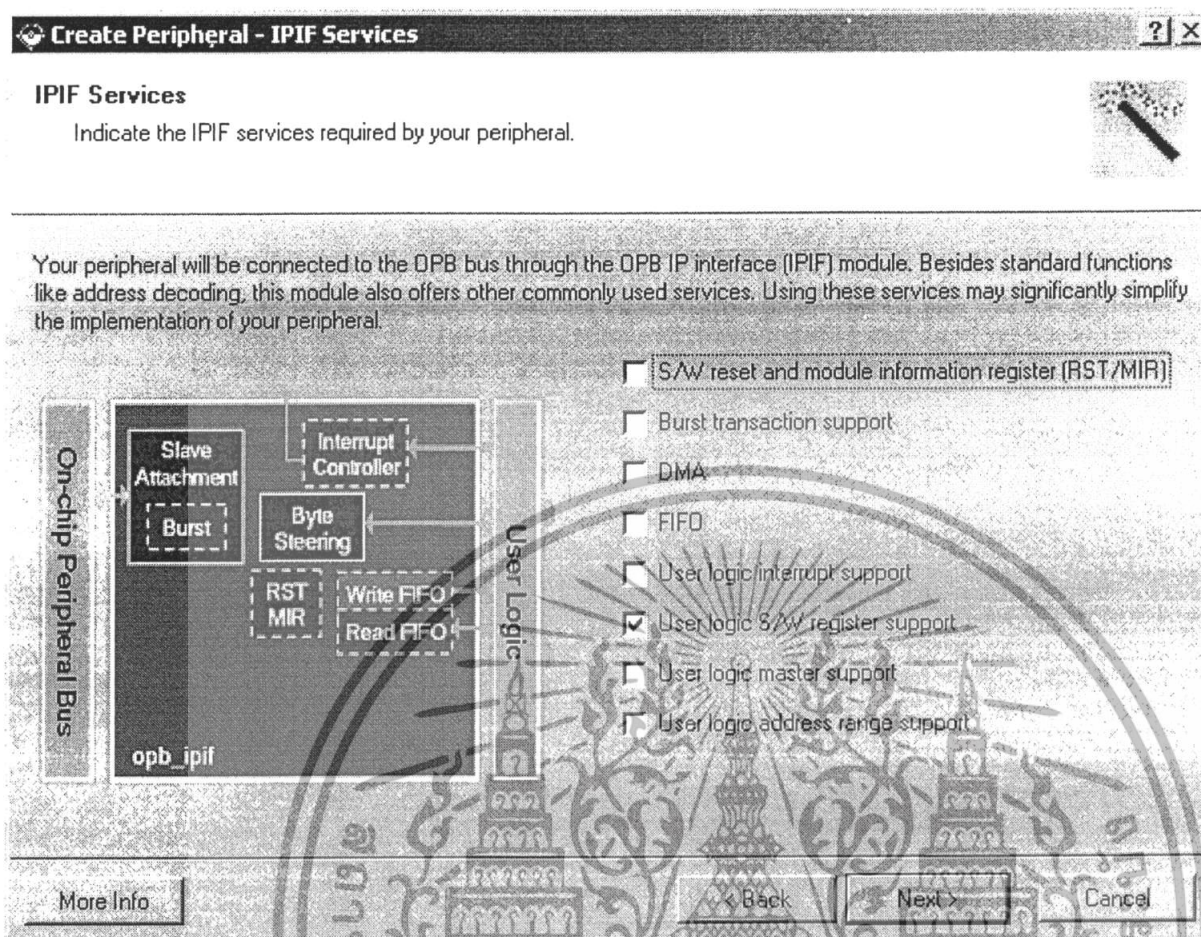
เนื้อหาในบทนี้จะเป็นเอกสารประกอบการทดลองสร้างไอพ็คเจอร์ของตนเอง เพื่อนำไปเชื่อมต่อกับไอพ็คเจอร์ของไมโครเบลส โดยไอพ็คเจอร์ที่จะสร้างเป็นตัวอย่างในบทนี้คือเซเวนเซกเมนต์คอนโทรลเลอร์ โดยผู้ใช้งานเพียงแค่เขียนตัวเลขที่เป็นบิตซีดีลงไปยังรีจิสเตอร์ของไมครูนีโมคูลจะจัดการนำตัวเลขนั้นออกไปแสดงผลยังเซเวนเซกเมนต์และทำการสแกนให้โดยอัตโนมัติ

7.2 ไอพ็คเจอร์อินเตอร์เฟส

การเชื่อมต่อไอพ็คเจอร์ของผู้ใช้งานเข้าสู่ระบบไมโครเบลสนั้นเราจะใช้คอร์ที่ชื่อว่า ไอพ็คเจอร์ไอเอฟเป็นตัวช่วยในการเชื่อมต่อ ซึ่ง ไอพ็คเจอร์ไอเอฟนี้เราสามารถสร้างได้โดยวิศวกรในโปรแกรมเอ็กซ์ทีเอส ดังนี้

1. เปิดโปรแกรมเอ็กซ์ทีเอสและเปิดโปรเจกต์เก่าขึ้นมา
2. ไปที่เมนู “Hardware > Create or Import Peripheral”
3. โปรแกรมจะแสดงหน้าวิศวกรขึ้นมา กด “Next”
4. เลือก “Create templates for a new peripheral”
5. เลือกที่เซฟไฟล์สำหรับเพอริเฟอรอลของเรา
6. ทำการตั้งชื่อเพอริเฟอรอล ซึ่งในที่นี้ขอตั้งชื่อว่า “seven_segment”
7. เลือกว่าจะให้เพอริเฟอรอลของเราเชื่อมต่อกับบัสใดในระบบ ในที่นี้เลือกเป็นไอพ็คเจอร์
8. ทำการเลือกคุณสมบัติที่จะให้ไอพ็คเจอร์ที่กำลังจะถูกสร้างขึ้นมารองรับ ในที่นี้เลือกให้มีซอฟต์แวร์รีจิสเตอร์ในฝั่งยูเซอร์ลอจิกดังรูป

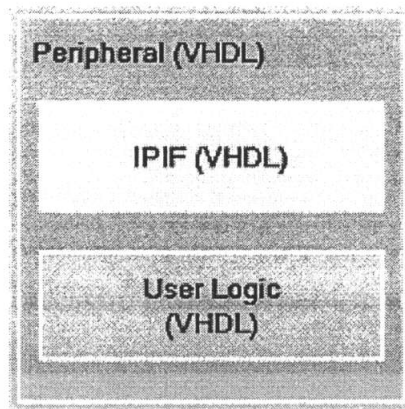
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7.1 การตั้งค่าไอพีไอเอฟ

9. เลือกจำนวนและความกว้างของรีจิสเตอร์ภายในส่วนยูเซอร์ลอจิก ในที่นี้ตั้งไว้ที่ 1 ตัวขนาด 32 บิต
10. ทำการเลือกสัญญาณไอพีอินเทอร์คอนเนคที่จะทำการส่งต่อไปยังส่วนยูเซอร์ลอจิก ซึ่งในหน้านี้เราไม่จำเป็นต้องทำการตั้งค่าเพิ่มเติม สามารถใช้ค่าเริ่มต้น ทำการกด “Next” ข้ามไปได้เลย
11. ในหน้าต่อไปจะเป็นเกี่ยวกับซิมูเลชันซัพพอร์ต กด “Next” ข้ามไปได้เลย
12. กด “Next” ไปจนสุดแล้วกด “Finish” โปรแกรมจะทำการสร้างโค้ดภาษาวีเอชดีแอลสำหรับเชื่อมต่อคอร์ของเรากับ ไอพีบีขึ้นมาให้ในลักษณะตามรูปต่อไปนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7.2 การเชื่อมต่อไอพีที่สร้างขึ้นกับไอพีบี

13. สิ่งที่เราจำเป็นต้องทำต่อไปคือเขียนโค้ดภาษาวีเอชดีแอลในส่วนยูเซอร์ลอจิกเพื่อให้คอร์ที่เราสร้างขึ้นมานั้นทำงานตามที่เราร้องการ

7.3 การสร้างยูเซอร์ไอพีคอร์

1. หลังจากทำการสร้างไอพีไอเอฟทางวีซาร์ดแล้ว ไปดูที่ไฟล์คอร์ "pcores" ในโปรเจกของเรา จะพบว่า "seven_segment_v1_00_a" อยู่
2. กดเข้าไปภายในโฟลเดอร์ จะมีโฟลเดอร์ที่ชื่อ "hdl/vhdl" อยู่ ด้านในจะมีไฟล์อยู่ทั้งสิ้นสองไฟล์ ได้แก่ "seven_segment.vhd" ซึ่งเป็นโค้ดภาษาวีเอชดีแอลที่เป็นทอปเลเวลของคอร์ของเราและไฟล์ "user_logic.vhd" ซึ่งเป็นโค้ดภาษาวีเอชดีแอลในส่วนที่เราต้องทำการแก้ไข
3. เปิดไฟล์ "user_logic.vhd" ขึ้นมา ด้านในจะมีโค้ดส่วนหนึ่งที่วีซาร์ดทำการสร้างให้เราเรียบร้อยแล้วทำการแก้ไขดังต่อไปนี้
4. ส่วน "entity" ในส่วน "port" ทำการเพิ่มขาสัญญาณต่อไปนี้ลงไป

```

97: port
98: (
99:     -- ADD USER PORTS BELOW THIS LINE -----
100:     --USER ports added here
101:     SevenSeg          : out   std_logic_vector (0 to 7);
102:     SevenSeg Dec      : out   std_logic_vector (0 to 3);
103:     -----
104:     -- ADD USER PORTS ABOVE THIS LINE -----
105: )

```

รูปที่ 7.3 การเพิ่มพอร์ตในยูเซอร์ลอจิก

5. ในส่วน "architecture" ทำการเพิ่ม "signal" ต่อไปนี้ลงไปศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

126:
127: architecture IMP of user_logic is
128:
129: --USER signal declarations added here, as needed for user logic
130: signal rCntDly          : std_logic_vector(31 downto 0);
131: signal r7Seg           : std_logic_vector(0 to 7);
132: signal rDec7Seg        : std_logic_vector(0 to 3);
133:

```

รูปที่ 7.4 "signal" ในส่วน "architecture"

- เพิ่มโค้ดการทำงานต่อไปนี้งไปด้านใต้ "begin" ในส่วนของ "architecture"



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

146: --USER Logic implementation added here
147: SevenSeg <= r7Seg;
148: SevenSeg_Dec <= rDec7Seg;
149:
150: u_rCntDly : Process (Bus2IP_Reset, Bus2IP_Clk) Is
151: Begin
152:   if ( Bus2IP_Reset = '1' ) then
153:     rCntDly <= (others => '0');
154:
155:   elsif ( rising_edge(Bus2IP_Clk) ) then
156:     if ( rCntDly = x"0002625A" ) then
157:       rCntDly <= (others => '0');
158:     else
159:       rCntDly <= rCntDly + x"00000001";
160:     end if;
161:   end if;
162: End Process u_rCntDly;
163:
164: u_r7Seg : Process (Bus2IP_Reset, Bus2IP_Clk) Is
165: Begin
166:   if ( Bus2IP_Reset='1' ) then
167:     r7Seg <= (others=>'0');
168:
169:   elsif ( rising_edge(Bus2IP_Clk) ) then
170:     case rDec7Seg is
171:       when "1110" =>
172:         r7Seg <= to_char(conv_integer(slv_reg0(26 to 31)));
173:       when "1101" =>
174:         r7Seg <= to_char(conv_integer(slv_reg0(25 to 30)));
175:       when "1011" =>
176:         r7Seg <= to_char(conv_integer(slv_reg0(24 to 29)));
177:       when "0111" =>
178:         r7Seg <= to_char(conv_integer(slv_reg0(17 to 23)));
179:       when others =>
180:         r7Seg <= (others=>'0');
181:     end case;
182:   end if;
183: End Process u_r7Seg;
184:
185: u_rDec7Seg : Process (Bus2IP_Reset, Bus2IP_Clk) Is
186: Begin
187:   if ( Bus2IP_Reset='1' ) then
188:     rDec7Seg <= "1110";
189:
190:   elsif ( rising_edge(Bus2IP_Clk) ) then
191:     if ( rCntDly=x"2625A" ) then
192:       rDec7Seg <= rDec7Seg(1 to 3) & rDec7Seg(0);
193:     else
194:       rDec7Seg <= rDec7Seg;
195:     end if;
196:   end if;
197: End Process u_rDec7Seg;

```

รูปที่ 7.5 โค้ดการทำงานของเซเวนเซกเมนต์คอนโทรลเลอร์

7. ทำการเพิ่มส่วนของการแปลงฟอนต์ต่อไปนีลงในส่วนของ “architecture”

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

123:
124: -- Architecture section
125:
126:
127: architecture IMP of user_logic is
128:
129:     -- convert a std_logic value to a character
130:
131:     type stdlogic_to_digit_fonts_t is array(0 to 9) of std_logic_vector(0 to 7);
132:     constant to_char : stdlogic_to_digit_fonts_t := (
133:         0 => "11111100",
134:         1 => "01100000",
135:         2 => "11011010",
136:         3 => "11110010",
137:         4 => "01100110",
138:         5 => "10110110",
139:         6 => "10111110",
140:         7 => "11100000",
141:         8 => "11111110",
142:         9 => "11110110"
143:     );
144:
145:     --USER signal declarations added here, as needed for user logic
146:     signal rCntDly      : std_logic_vector(31 downto 0);
147:     signal r7Seg        : std_logic_vector(0 to 7);
148:     signal rDec7Seg     : std_logic_vector(0 to 3);
149:

```

รูปที่ 7.6 การแปลงฟอนท์ของเซเว่นเซกเมนต์

8. ทำการเซฟและปิดไฟล์ จากนั้นทำการเปิดไฟล์ "seven_segment.vhd" ซึ่งเป็นทอปเลเวลขึ้นมาทำการแก้ไข
9. เพิ่มพอร์ทต่อไปนี้ในส่วนของ "entity"

```

116: port
117: (
118:     -- ADD USER PORTS BELOW THIS LINE
119:     --USER ports added here
120:     SevenSeg      : out std_logic_vector (0 to 7);
121:     SevenSeg_Dec  : out std_logic_vector (0 to 3);
122:
123:     -- ADD USER PORTS ABOVE THIS LINE

```

รูปที่ 7.7 พอร์ทที่เพิ่มในส่วนทอปเลเวล

10. เพิ่มพอร์ทแมพต่อไปนี้ในส่วนการสร้างอินสแตนซ์ของยูเซอร์ลอจิก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

375: -----
376: -- instantiate the User Logic
377: -----
378: USER_LOGIC_I : entity seven_segment_v1_00_a.user_logic
379: generic map
380: (
381:   -- MAP USER GENERICS BELOW THIS LINE -----
382:   --USER generics mapped here
383:   -- MAP USER GENERICS ABOVE THIS LINE -----
384:
385:   C_DWIDTH          => USER_DWIDTH,
386:   C_NUM_CE          => USER_NUM_CE
387: )
388: port map
389: (
390:   -- MAP USER PORTS BELOW THIS LINE -----
391:   --USER ports mapped here
392:   SevenSeg          => SevenSeg,
393:   SevenSeg_Dec      => SevenSeg_Dec,
394:   -- MAP USER PORTS ABOVE THIS LINE -----
395:

```

รูปที่ 7.8 การทำพอร์ตแมพที่ทอปเลเวล

11. ทำการเซฟและปิดไฟล์
12. หากเรานำคอร์ที่เราสร้างเสร็จนี้มาใช้ในโปรแกรมเอ็กซ์พีเอสเลย เราจะพบว่าจะยังไม่มีพอร์ตที่เราเพิ่งจะสร้างเพิ่มที่ทอปเลเวล เนื่องจากเรายังไม่ได้ทำการแก้ไขเอ็มพีดีไฟล์ (Microprocessor Peripheral Description file)
13. ไปยังโฟลเดอร์ “pcores\seven_segment_v1_00_a\data” ภายในโฟลเดอร์ของโปรเจกของเราแล้ว ทำการเปิดเอ็มพีดีไฟล์ขึ้นมา
14. ทำการเพิ่มพอร์ตทั้งสองที่เราสร้างขึ้นมาลงในเอ็มพีดีไฟล์ดังนี้ จากนั้นทำการเซฟ

```

43
44 PORT SevenSeg = "", DIR = 0 , VEC = [0:7]
45 PORT SevenSeg_Dec = "", DIR = 0 , VEC = [0:3]
46
47 END
48

```

รูปที่ 7.9 เพิ่มพอร์ตลงในเอ็มพีดีไฟล์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

15. คู่มือโปรแกรมเอ็กซ์พีเอสตรงแถบ “IP Catalog” ที่ “Project Repository” จะมีคอร์ “seven_segment” อยู่ทำการเพิ่มคอร์ดังกล่าวเข้าสู่ระบบ และเชื่อมต่อกับไอพีบีให้เรียบร้อย
16. เปลี่ยน “Filters” ไปที่ “Ports” แล้วดูที่ “seven_segment” ถ้าหากยังไม่มีพอร์ทใดๆ ให้ทำการปิด และเปิดโปรแกรมเอ็กซ์พีเอสอีกครั้งหนึ่ง
17. ทำการแมพพอร์ทที่ขั้วออกมาเป็นขาภายนอกและแก้ไขยูซีเอฟไฟล์ให้เรียบร้อย



รูปที่ 7.10 การแมพขาอุปกรณ์ออกมาเป็นขาภายนอก

18. ทำการแมพแอดเดรสใหม่ให้เรียบร้อยในหน้า “Filters” ที่เป็น “Addresses”
19. ทำการสร้างและอัปเดตบิตสตรีมแล้วทำการดาวน์โหลดลงบอร์ดให้เรียบร้อย
20. สั่งให้เอ็กซ์พีเอสทำการสร้างไฟล์ไลบรารีต่างๆ โดย “LibGen” ตามขั้นตอนที่เคยทำ จากนั้นเปิดโปรแกรมเอ็กซ์พีเอสอีกครั้งขึ้นมาเพื่อทำซอฟต์แวร์ต่อไป

7.4 การเขียนซอฟต์แวร์เพื่อสั่งงานไอพีคอร์ที่สร้างขึ้นเอง

1. หลังจากทำการรัน “LibGen” จากโปรแกรมเอ็กซ์พีเอสแล้วจะมีไฟล์เฮดเดอร์ใหม่ที่ประกาศฟังก์ชันโปรโตไทป์สำหรับการควบคุมไอพีคอร์ของเราขึ้นมา ชื่อว่า “seven_segment.h” ทำการอินคลูดไฟล์ดังกล่าวเข้ามาให้โปรแกรมของเรา

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. เราจะใช้ฟังก์ชันต่อไปนี้ในการเขียนค่าลงในรีจิสเตอร์ของไอพียูของเรา

```
SEVEN_SEGMENT_mWriteSlaveReg0(BaseAddress, Value)
```

3. ทดลองเขียนโปรแกรมให้ทำการเขียนตัวเลขบีซีดี “0x1234” ลงไปในรีจิสเตอร์ของเซเว่นเซกเมนต์โมดูลดังต่อไปนี้

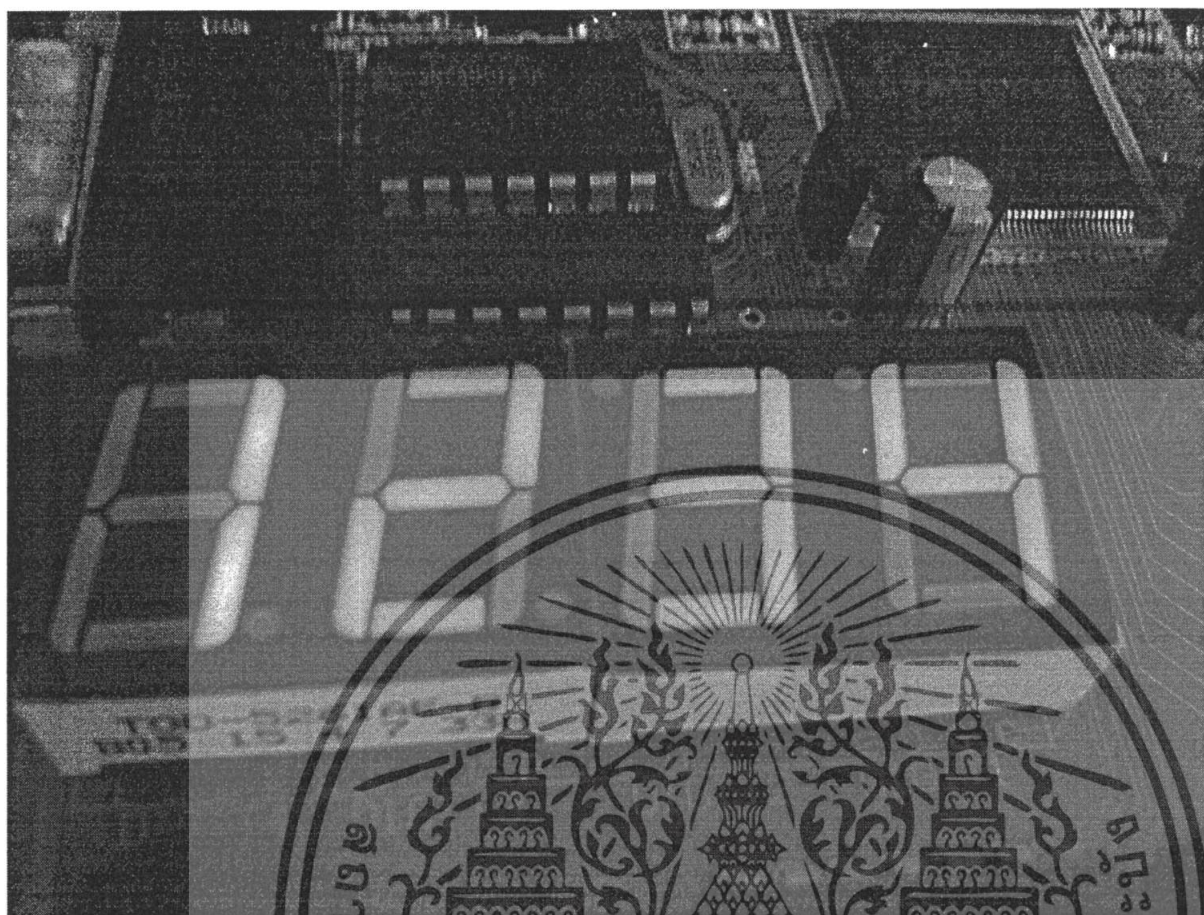
```
// Welcome to Xilinx Platform Studio SDK !
//
// This is an automatically created source file to help you get started.
// To add more files, navigate to File -> New -> File
// You may delete this file if you want to use only other files for your project
//
#include <stdio.h>
#include "xparameters.h"
#include "seven_segment.h"

int main()
{
    SEVEN_SEGMENT_mWriteSlaveReg0(XPAR_SEVEN_SEGMENT_0_BASEADDR, 0x1234);
    return 0;
}
```

รูปที่ 7.11 โปรแกรมใช้งานเซเว่นเซกเมนต์โมดูล

4. ทดลองนำโปรแกรมไปทำงานดูจะพบว่ามีตัวเลข “1234” ติดที่เซเว่นเซกเมนต์บนบอร์ดทดลองดังรูป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7.12 ผลลัพธ์ที่เขื่อนเข็กเหนือ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 8

การนำไมโครซีไอเอสทูมาใช้งานบนระบบไมโครเบลส

8.1 ภาพรวม

เนื้อหาในบทนี้จะอธิบายถึงวิธีการนำเรียดไทม์ไอเอสทีชื่อว่า ไมโครซีไอเอสทูมาทำงานบนระบบไมโครเบลส รวมถึงวิธีการเขียนโปรแกรมบนไมโครซีไอเอสทูเบื้องต้น

8.2 ไมโครซีไอเอสทู

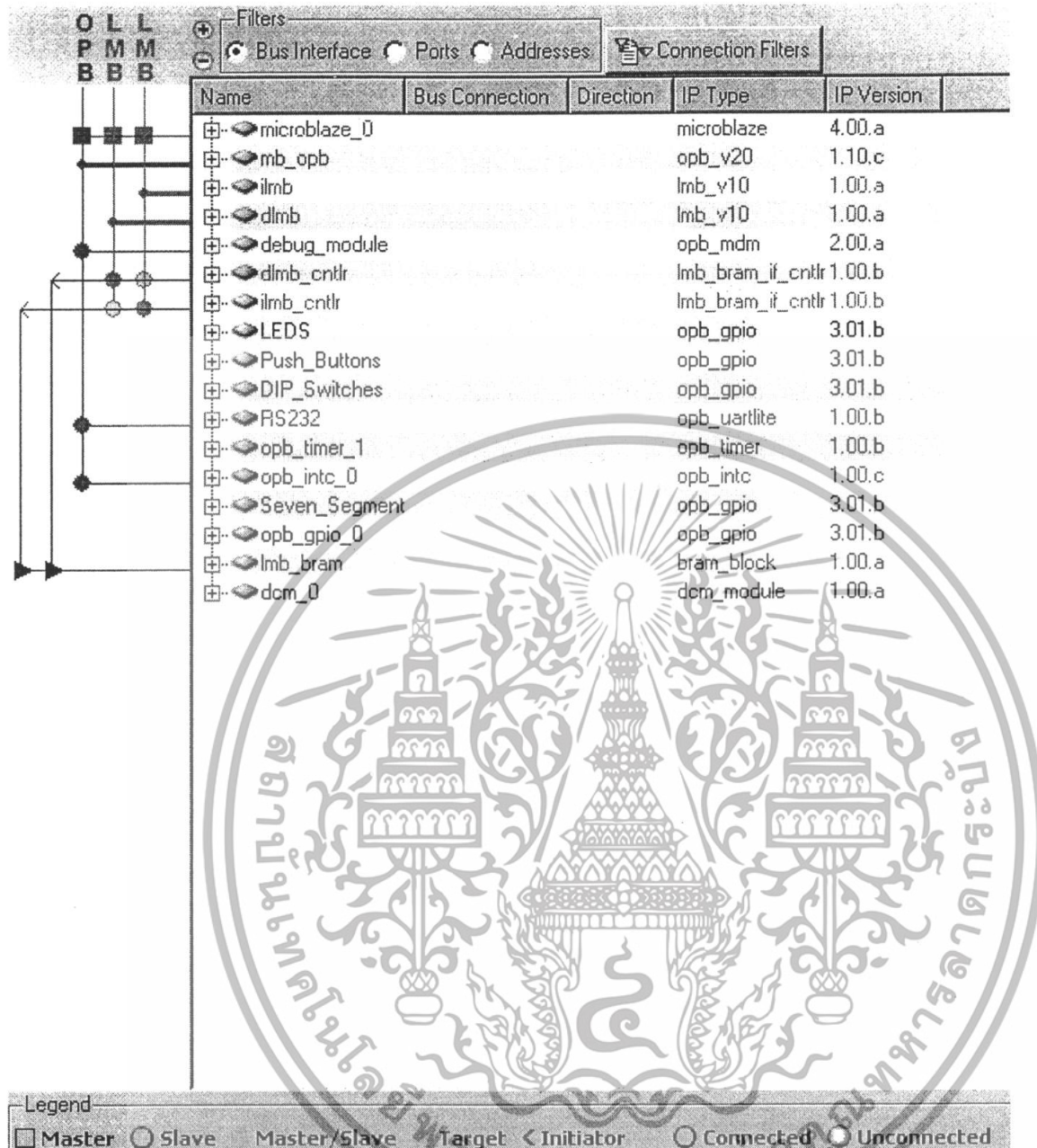
สำหรับซอสโค้ดของไมโครซีไอเอสทูนั้นถูกแบ่งออกเป็นสองส่วนได้แก่ ซอส และพอร์ท โดยซอสนั้นคือส่วนซอสโค้ดของตัวไอเอสทีไม่ขึ้นกับสถาปัตยกรรมของโปรเซสเซอร์ที่จะนำไปใช้งาน ส่วนพอร์ทนั้นคือซอสโค้ดส่วนที่จะเชื่อมต่อไอเอสเข้ากับสถาปัตยกรรมของไมโครโปรเซสเซอร์ตัวที่จะใช้งาน

8.2.1 เตรียมความพร้อมของระบบก่อนใช้งานไมโครซีไอเอสทู

สำหรับระบบไมโครเบลสที่จะใช้งานไมโครซีไอเอสทูนั้นจำเป็นที่จะต้องมีไทม์เมอร์อย่างน้อยหนึ่งตัวที่ซัพพอร์ทการเกิดอินเทอร์รัพท์ ซึ่งในการทดลองนี้เราจะเปลี่ยนบอร์ดทดลองไปใช้บอร์ดที่มีจีพียูเพื่อความสะดวกเนื่องจากขนาดของบล็อกแรมในเอฟพีจีเอของบอร์ดเดิมนั้นไม่เพียงพอต่อการลงไมโครซีไอเอสทู สำหรับขั้นตอนการเตรียมระบบนั้นมีดังนี้

1. ทำการดาวน์โหลดซอสโค้ดของตัวไมโครซีไอเอสทูและพอร์ทของไมโครเบลสจากเว็บไซต์ <http://www.micrium.com>
2. ทำการคลายการบีบอัดไฟล์ทั้งหมดไปไว้ที่ไดรฟ์ซี
3. เปิดโปรแกรมเอ็กซ์พีเอสขึ้นมาและทำการสร้างระบบไมโครเบลสโดยใช้เบสซิสเต็มบิลเดอร์วีซาร์ดให้ระบบมีลักษณะดังนี้

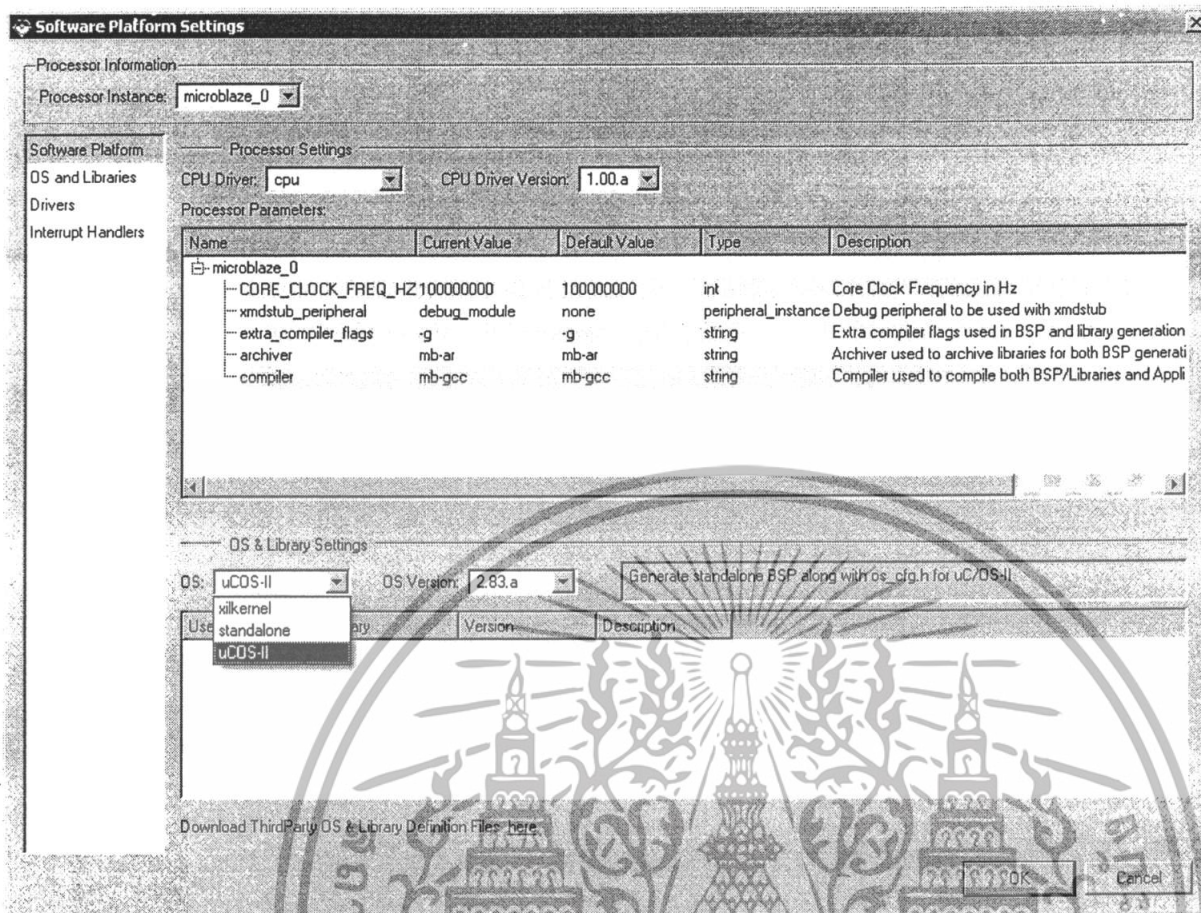
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 8.1 ระบบที่จะนำไมโครซีไอเอสทิวเข้ามาทำงาน

4. เปิดไปที่เมนู “Software > Software Platform Settings”
5. ในหน้าแรกด้านล่างตรง “OS” ให้เลือกเป็น “uCOS-II” ดังรูป

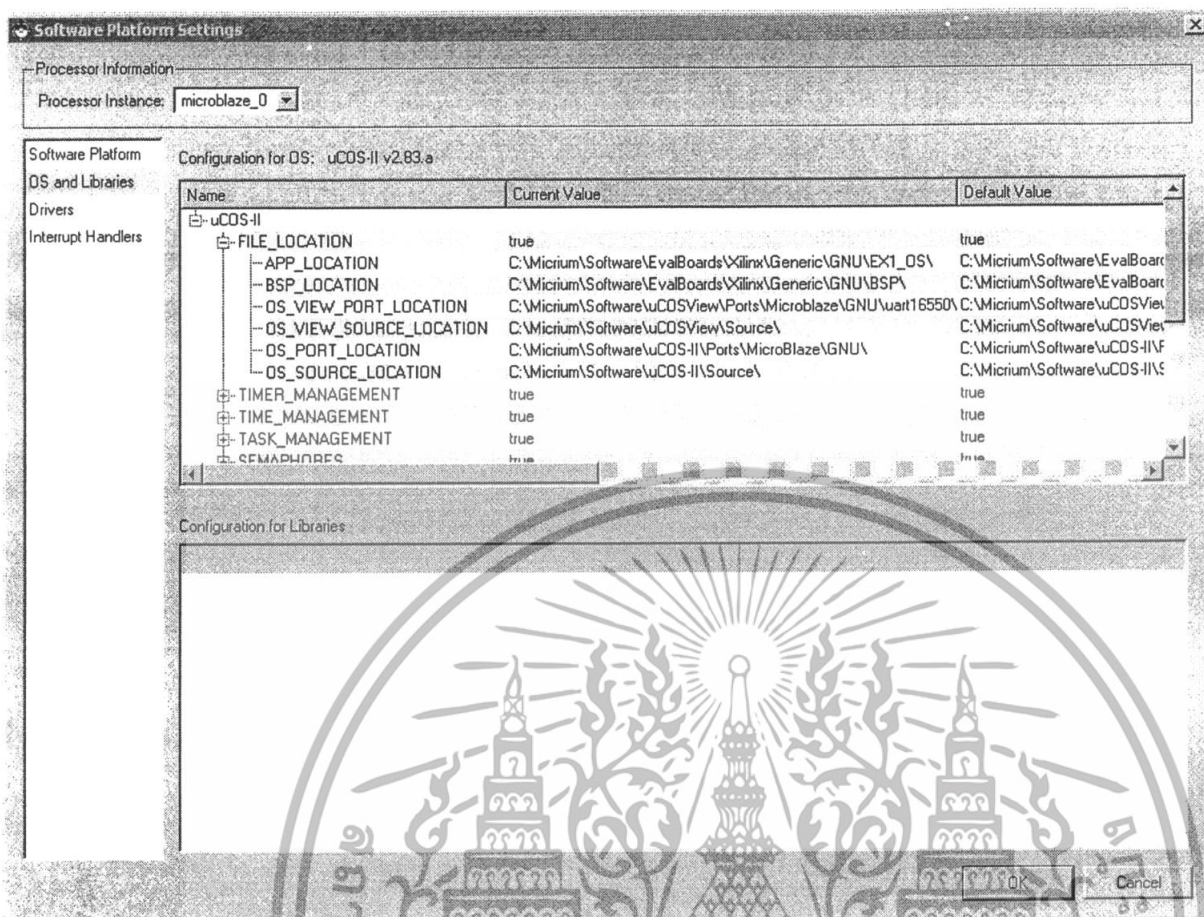
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 8.2 เลือก "OS" เป็น "uCOS-II"

6. ไปที่แถบ "OS and Libraries" ทำการแก้ "FILE_LOCATION" ให้ตรงตามภาพ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 8.3 การแก้ไข "FILE_LOCATION"

7. ที่ "TASK_MANAGEMENT" แก้ "OS_TASK_CREATE_EN" ให้เป็น 1 เพื่อเปิดการใช้งานฟังก์ชัน "OSTaskCreate"
8. ที่ "MISCELLANEOUS" แก้ "OS_TICKS_PER_SEC" ให้เป็น 1000 เพื่อให้ความละเอียดของโอเอสทิกสูงขึ้นเนื่องจากซีพียูของเราทำงานที่ความถี่ค่อนข้างสูง
9. แก้ "OS_MAX_TASKS" จาก 5 ให้เป็น 20 เพื่อที่จะได้สร้างทาสก์ได้มากกว่า 5 ทาสก์
10. แก้ "OS_LOWEST_PRIO" จาก 10 ให้เป็น 30 เนื่องจากปริมาณทาสก์ของเรามีมากขึ้น
11. แก้ "OS_MAX_EVENTS" จาก 2 ให้กลายเป็น 10 เพื่อที่เราจะจะสามารถสร้างอีเวนต์ได้มากขึ้น
12. ที่ "BSP_OPTIONS" ทำการแก้ "stdout" และ "stdin" ให้เป็น "debug_module" เพื่อให้ปริ้นข้อความผ่านทางดีบั๊กโมดูลได้
13. กด "OK" นี้เป็นเอกสารที่ส่งมอบไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

14. ทำการเปิดไฟล์ “C:\MICRIUM\Software\EvalBoards\Xilinx\Generic\GNU\BSP\bsp.c” ขึ้นมาทำการแก้ไข
15. ทำการแก้ไข “BSP_INTC_DEVICE_ID, BSP_INTC_TIMER1_ID, BSP_INTC_ADDR, BSP_TIMER1_ADDR” ให้ตรงกับอุปกรณ์ของเราที่ประกาศไว้ในไฟล์ “xparameters.h” ไม่เช่นนั้นไอเอสจะไม่สามารถทำงานได้

```

23  #/
24  *****
25  *          CONSTANTS
26  *****
27  #/
28
29  #define BSP_INTC_DEVICE_ID    XPAR_OPB_INTC_0_DEVICE_ID
30  #define BSP_INTC_TIMER1_ID   XPAR_OPB_INTC_0_OPB_TIMER_1_INTERRUPT_INTR
31
32  #define BSP_INTC_ADDR        XPAR_OPB_INTC_0_BASEADDR
33  #define BSP_TIMER1_ADDR     XPAR_OPB_TIMER_1_BASEADDR
34
35  #define BSP_GPIO_ADDR       XPAR_LEDS_BASEADDR
36
37  #define BSP_TMR_VAL         (XPAR_CPU_CORE_CLOCK_FREQ_HZ / OS_TICKS_PER_SEC)
38

```

รูปที่ 8.4 แก้ไขไฟล์ "bsp.c"

16. ทำการสร้างและอัปเดตบิตสตรีม และดาวน์โหลดบอร์ดพร้อมทั้งทำ “LibGen” ให้เรียบร้อย

8.2.2 การเขียนโปรแกรมให้ไมโครซีไอเอสทู้เริ่มทำงาน

1. เปิดโปรแกรมเอ็กซ์พีเอสดีเคขึ้นมาแล้วทำการสร้างโปรเจกใหม่ให้ขึ้นมาอีกซีไอเอสโปรเจกที่ทำได้ไว้แล้ว
2. ทำการอินคลูดไฟล์ “includes.h” เข้ามาใน โปรแกรมเพื่อให้อ่านเอพีไอของไมโครซีไอเอสทู้ได้
3. เรียกฟังก์ชัน “BSP_IntDisAll” เพื่อทำการปิดอินเทอร์รัพท์ทั้งหมด
4. เรียกฟังก์ชัน “OSInit” เพื่อเตรียมความพร้อมค่าต่างๆ ก่อนที่ไอเอสจะเริ่มทำงาน
5. สร้างทาสก์ “AppTaskFirst” ขึ้นมาเพื่อเป็นทาสก์แรกของไอเอส
6. เรียกฟังก์ชัน “OSStart” เพื่อย้ายการทำงานไปยังไอเอส
 เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

#include "includes.h"
#include "xparameters.h"
#include <stdio.h>

static OS_STK AppTaskFirstStk [APP_TASK_FIRST_STK_SIZE];
static void AppTaskFirst (void *p_arg);
// Main Program (OS Startup)
int main()
{
    INT8U err;
    // Make sure interrupts are disabled on interrupt controller
    BSP_IntDisAll();
    // Initialize uC/OS-II
    OSInit();
    // Create Startup Task
    OSTaskCreateExt (AppTaskFirst,
                    (void *)0,
                    &AppTaskFirstStk[APP_TASK_FIRST_STK_SIZE - 1],
                    APP_TASK_FIRST_PRIO,
                    APP_TASK_FIRST_ID,
                    &AppTaskFirstStk[0],
                    APP_TASK_FIRST_STK_SIZE,
                    (void *)0,
                    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

    OSTaskNameSet (APP_TASK_FIRST_PRIO, "App Task First", &err);
    // Start multitasking
    OSStart();
    return 0;
}

```

รูปที่ 8.5 โปรแกรมส่วน "main" ในการเริ่มการทำงานของโอเอส

7. ใน "AppTaskFirst" ทำการตั้งค่าเริ่มต้นของสเตตัสดีคทาสด์โดยการเรียกฟังก์ชัน "OSStatInit" จากนั้นโอเอสได้ถือว่าถูกเริ่มต้นอย่างสมบูรณ์แล้ว ทำการเรียกฟังก์ชัน "myMain" เพื่อย้ายการทำงานไปยังโปรแกรมประยุกต์ที่เราจะเขียนขึ้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

static void AppTaskFirst (void *p_arg)
{
    INT8U err;
    (void)p_arg; // Prevent compiler warning by doing something with argument
    BSP_InitIO(); // Initialize the I/Os
    #if OS_TASK_STAT_EN > 0
        OSStatInit(); // Initialize uC/OS-II's statistics
    #endif
    //end of the startup process
    myMain();
}

void myMain(void)
{
}

```

รูปที่ 8.6การทำงานภายในทาส์แรกของเรา

8. ทำการเขียน โปรแกรมให้พิมพ์ค่าตัวเลขออกมาในทุกๆวินาทีเพื่อทดสอบการทำงานของโอเอสของเราดังนี้

```

void myMain(void)
{
    unsigned int time = 0;
    while (1)
    {
        xil_printf("Time = %d\n", time);
        time++;
        OSTimeDlyHMSM(0,0,1,0);
    }
}

```

รูปที่ 8.7โปรแกรมพิมพ์ตัวเลขทุกวินาที

9. ตั้งค่าการรันโปรแกรมให้มารันที่โปรเจกต์ใหม่ของเรา แล้วนำโปรแกรมไปทดสอบการทำงานดู แล้วลองดูที่คอนโซลจะได้ผลดังภาพ

```

Problems Console Properties
test [D:\nix\C\C++\ELF] E:\work_spaces\edk_workspace\Demo_App\SDK_projects\UC052(Debug)\UC052.elf (2/14/51 4:22 AM)
Time = 20
Time = 21
Time = 22
Time = 23
Time = 24
Time = 25
Time = 26
Time = 27

```

เอกสารนี้เป็นเอกสารที่ 8.8 ผลการทำงานของโปรแกรมนับเลขในทุกวินาที อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

10. ทำการเขียนฟังก์ชันทาส์ก์เพิ่ม โดยให้ทาส์ก์นั้นพิมพ์คำว่า “Another Task” ออกมาในทุกๆ สองวินาทีดังนี้

```
static void AnotherTask (void *p_arg)
{
    (void)p_arg;
    while(1)
    {
        xil_printf("Another Task\n");
        OSTimeDlyHMSM(0,0,2,0);
    }
}
```

รูปที่ 8.9 ทาส์ก์ที่สองในโปรแกรม

11. ทำการสร้างทาส์ก์ที่ได้เขียนไว้โดยเพิ่มคำสั่ง “OSTaskCreate” ลงใน “myMain”

```
void myMain(void)
{
    unsigned int time = 0;
    OSTaskCreate(AnotherTask, (void*)0,
        &AnotherTaskStk[APP_TASK_FIRST_STK_SIZE - 1], 1);
    while(1)
    {
        xil_printf("Time = %d\n", time);
        time++;
        OSTimeDlyHMSM(0,0,1,0);
    }
}
```

รูปที่ 8.10 การสร้างทาส์ก์ที่สอง

12. นำโปรแกรมไปทดสอบการทำงานแล้วดูผลลัพธ์ที่คอนโซลจะได้ดังรูป

```
Another Task
Time = 0
Time = 1
Another Task
Time = 2
Time = 3
Another Task
Time = 4
```

รูปที่ 8.11 ผลลัพธ์ของโปรแกรมที่มีสองทาส์ก์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 9

ตัวอย่างโปรแกรมประยุกต์

9.1 ภาพรวม

เนื้อหาในบทนี้จะเป็นการออกแบบและการทดลองโปรแกรมประยุกต์ที่สร้างบนระบบไมโครเบลส โดยใช้รีเลย์ 1 ไมโครซีไอเอส 1 ไมโครซีไอเอสทูวี โดยระบบที่จะทำนั้นจะทำการอ่านค่าจากตัวตรวจจับอุณหภูมิ ทั้งสี่ในตัวในหลายๆวินาทีแล้ววาดออกมาบนจอกราฟิกแอลซีดีในลักษณะของกราฟ และสามารถติดต่อกับพีซีผ่านทางพอร์ตอนุกรมในการอ่านข้อมูลอุณหภูมิ 100 วินาทีล่าสุดของเซ็นเซอร์แต่ละตัวออกมาได้

9.2 อุปกรณ์ภายนอกที่ใช้ในการทดลองนี้

9.2.1 เซ็นเซอร์วัดอุณหภูมิ

สำหรับในการทดลองนี้เราใช้เซ็นเซอร์วัดอุณหภูมิเบอร์ดีเอส 1820 (DS1820) ของบริษัทดัลลาสเซมิคอนดักเตอร์ ซึ่งมีอินเตอร์เฟซการเชื่อมต่อแบบวันไวร์ (ONEWIRE) หรือใช้สายเส้นเดียวในการสื่อสาร

9.2.1.1 โปรโตคอลวันไวร์

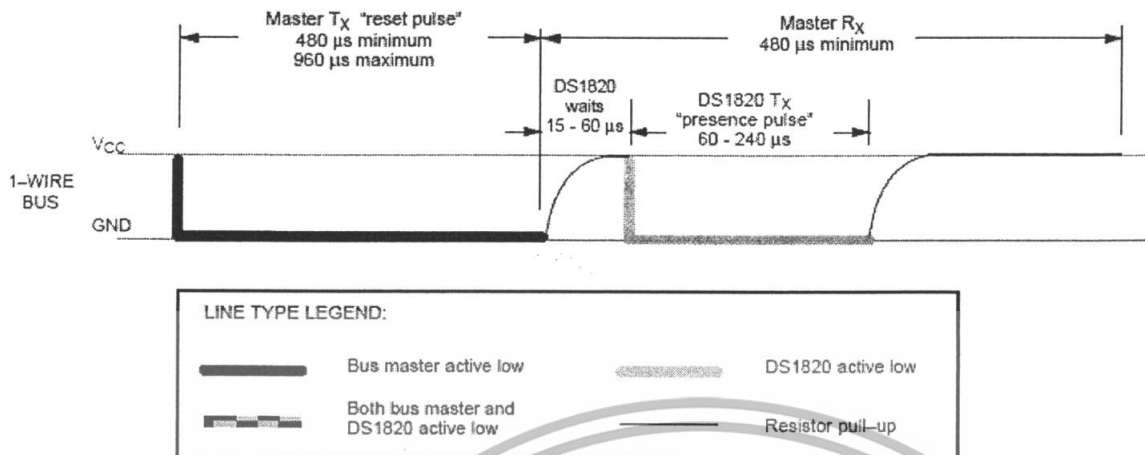
9.2.1.1.1 รีเซต

ในการสื่อสารบนโปรโตคอลวันไวร์นั้นจะเริ่มต้นด้วยการส่งสัญญาณรีเซตจากมาสเตอร์ซึ่งลักษณะของสัญญาณรีเซตนั้นเป็นดังนี้

มาสเตอร์ดึงขาสัญญาณให้เป็น 0 ประมาณ 480 – 960 ไมโครวินาที จากนั้นปล่อยให้เป็น 1 ประมาณ 15 – 60 ไมโครวินาที แล้วสลาฟจะตอบรับด้วยการดึงขาสัญญาณลงเป็น 0 ประมาณ 60 – 240 ไมโครวินาทีดังภาพ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

INITIALIZATION PROCEDURE “RESET AND PRESENCE PULSES” Figure 11



รูปที่ 9.1 สัญญาณรีเซตของโปรโตคอลวันไวร์

9.2.1.1.1 การเขียนและอ่านข้อมูล

9.2.1.1.1.1 การเขียนข้อมูล

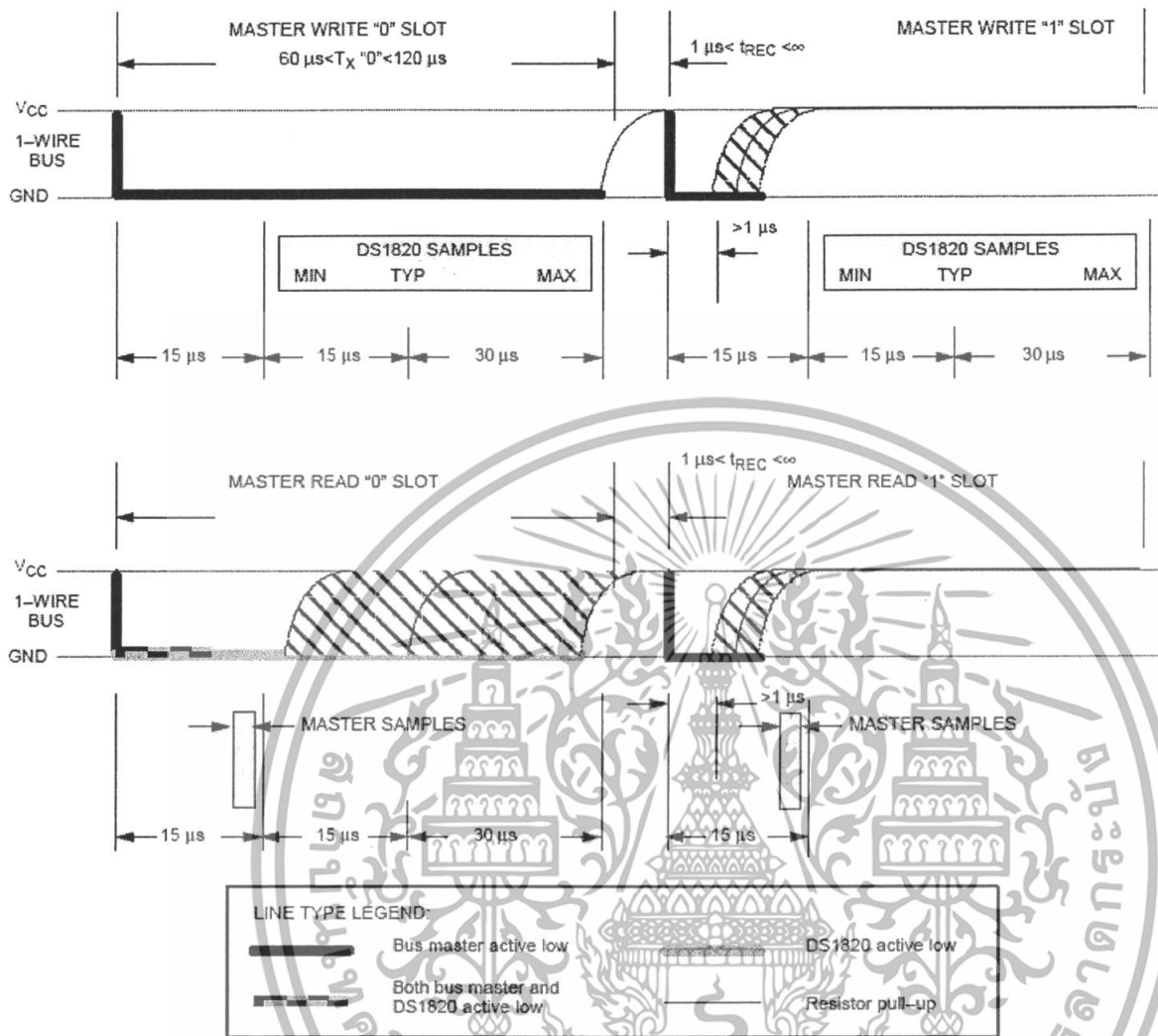
สำหรับการเขียนข้อมูลของมาสเตอร์บนโปรโตคอลวันไวร์นั้น จะเริ่มต้นด้วยการที่มาสเตอร์ดึงสายสัญญาณลงเป็น 0 อย่างน้อย 1 ไมโครวินาที จากนั้นภายใน 15 ไมโครวินาทีสลาฟจะทำการอ่านค่าภายใน 15 - 60 ไมโครวินาทีหลังจากนั้น ซึ่งมาสเตอร์จะต้องเปลี่ยนขาสัญญาณให้เป็นไปตามข้อมูลที่ต้องการจะส่ง (0 ถ้าจะส่ง 0 และ 1 ถ้าต้องการจะส่ง 1)

9.2.1.1.1.2 การอ่านข้อมูล

สำหรับการอ่านข้อมูลของมาสเตอร์บนโปรโตคอลวันไวร์นั้น เริ่มต้นด้วยการที่มาสเตอร์ดึงสายสัญญาณลงเป็น 0 อย่างน้อย 1 ไมโครวินาที จากนั้นอีก 15 ไมโครวินาทีมาสเตอร์จะทำการอ่านข้อมูลจากสายสัญญาณ ซึ่งสลาฟจะเปลี่ยนข้อมูลในสายสัญญาณในช่วงนี้ และทำการรอจนครบ 60 ไมโครวินาทีแล้วจึงเริ่มการสื่อสารในโหมดสล็อตถัดไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

READ/WRITE TIMING DIAGRAM Figure 12



รูปที่ 9.2 การอ่านและเขียนข้อมูลบนโปรโตคอลวันไวร์

9.2.1.2 การอ่านอุณหภูมิจากดีเอส1820

1. เริ่มต้นจากการที่มาสเตอร์ส่งสัญญาณรีเซตไปบนบัสวันไวร์เพื่อให้อุปกรณ์พร้อมที่จะทำงาน
2. มาสเตอร์ทำการส่งข้อมูล 0xCC ซึ่งเป็นคำสั่งข้ามรอมไปให้กับตัวเซ็นเซอร์เนื่องจากบนบัสมีเซ็นเซอร์อุณหภูมิต่ออยู่เพียงอย่างเดียว
3. มาสเตอร์ทำการส่งข้อมูล 0x44 ซึ่งเป็นคำสั่งให้เซ็นเซอร์ทำการแปลงค่าอุณหภูมิ จากนั้นรอนาน้อย 200 มิลลิวินาทีเพื่อรอให้เซ็นเซอร์ทำการแปลงอุณหภูมิให้เรียบร้อยแล้ว
4. มาสเตอร์ทำการส่งสัญญาณรีเซตบนบัสวันไวร์อีกครั้ง จากนั้นทำการส่ง 0xCC และ 0xBE เพื่อบอก

ให้เซ็นเซอร์ส่งข้อมูลอุณหภูมิกลับมา

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5. มาสเตอร์ทำการอ่านข้อมูลจากบัสซึ่งข้อมูลนี้จะมีขนาดแปดบิตและตัวเลขที่อ่านได้จะเป็น 2 เท่าของอุณหภูมิ (บิตสุดท้ายจะมีค่าเท่ากับ 0.5 องศา)

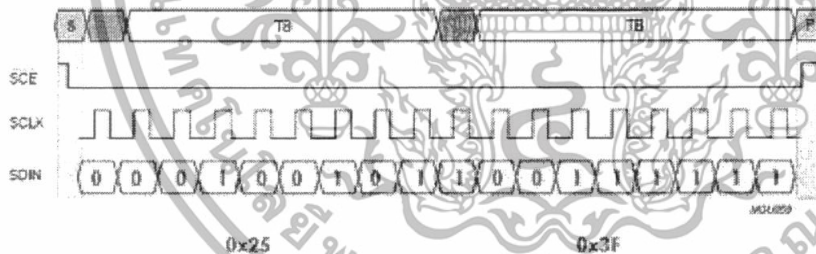
9.2.2 จอแอลซีดีแบบกราฟิก

จอแอลซีดีแบบกราฟิกที่เรานำมาใช้ในการทดลองนี้เป็นจอของโทรศัพท์มือถือโนเกียรุ่น 6100 ซึ่งใช้ชิพควบคุมของฟิลิปส์และมีการเชื่อมต่อโดยโปรโตคอลเอสพีไอ

9.2.2.1 โปรโตคอลเอสพีไอ

ในการติดต่อกับจอโนเกีย 6100 นั้นเราจะใช้โปรโตคอลเอสพีไอในการสื่อสารโดยลักษณะของโปรโตคอลเอสพีไอนั้นเป็นดังนี้

เมื่อต้องการเริ่มการสื่อสารมาสเตอร์จะทำการดึงสัญญาณซีพเอ็นเอเบิล(SCE) ลงเป็น 0 เพื่อให้อุปกรณ์รับฟังหลังจากนั้นมาสเตอร์จะทำการส่งข้อมูลบิตแรกซึ่งเป็นการบอกว่าสิ่งที่กำลังจะส่งนี้เป็นข้อมูลหรือคำสั่ง (0 คือคำสั่ง 1 คือข้อมูล) โดยมาสเตอร์จะทำการเปลี่ยนแปลงขาเอสดีเอ (SDA) ให้เป็นข้อมูลที่ต้องการจะส่ง (1 หรือ 0) จากนั้นทำการส่งสัญญาณคล็อกออกไปทางขาเอสซีแอล (SCL) หนึ่งลูก ภาาฟจะอ่านข้อมูลที่มาสเตอร์ส่งให้หลังจากได้รับสัญญาณทางเอสซีแอลแล้วเท่านั้น



รูปที่ 9.3 การสื่อสารบนโปรโตคอลเอสพีไอ

9.2.2.2 การสั่งงานจอโนเกีย 6100

9.2.2.2.1 การสั่งให้จอเริ่มทำงาน

ในการสั่งให้จอโนเกีย 6100 เริ่มต้นทำงานนั้นมีขั้นตอนดังต่อไปนี้

1. ทำการส่งสัญญาณรีเซตไปยังจอ
2. เริ่มส่งคำสั่ง “SLEEPOUT” (0x11) ออกไปทางโปรโตคอลเอสพีไอ

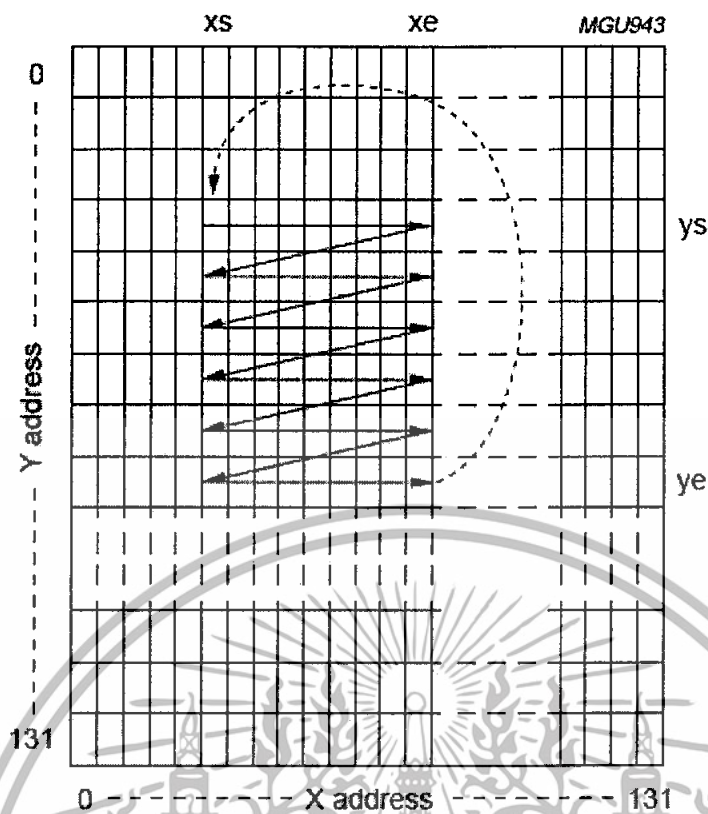
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. ส่งคำสั่ง “NORON” (0x13) ผ่าน โปรโตคอลเอสพีไอไปยังจอเพื่อบอกให้จอทำงานในโหมดปกติ
4. ส่งคำสั่ง “COLMOD” (0x3A) ไปยังจอเพื่อบอกว่าจะทำการตั้งค่าโหมดสี
5. ส่งข้อมูล 0x02 ไปยังจอเพื่อบอกให้จอทำงานที่โหมด 2 (สี 8 บิต)
6. ส่งคำสั่ง “MADCTL” (0x36) และข้อมูล 0x00 เพื่อตั้งค่าการเข้าถึงหน่วยความจำของจอ
7. ส่งคำสั่ง “SETCON” (0x25) และตามด้วยข้อมูลความสว่างของจอที่ต้องการ (ไม่เกิน) 0x3F ไปยังจอเพื่อปรับความสว่าง
8. ปิดการสื่อสารเอสพีไอ
9. เริ่มการสื่อสารเอสพีไอใหม่อีกครั้งและส่งคำสั่ง “DISPON” (0x29) ไปยังจอเพื่อเปิดจอภาพ
10. ปิดการสื่อสารเอสพีไอ และจอภาพพร้อมที่จะเริ่มทำงาน

9.2.2.2.2 การส่งภาพออกไปแสดงยังจอ

1. เริ่มต้นการสื่อสารเอสพีไอ จากนั้นทำการส่งคำสั่ง “CASET” (0x2A) ไปยังจอ เพื่อทำการตั้งค่าตำแหน่งคอลัมน์ที่ต้องการจะเริ่มเขียนภาพ
2. ส่งข้อมูล คอลัมน์เริ่มต้น(x1) และคอลัมน์สิ้นสุด(x2) เพื่อกำหนดขอบเขตในแนวแกนเอ็กซ์ที่จะทำการเขียนภาพ
3. ทำการส่งคำสั่ง “PASET” (0x2B) ไปยังจอภาพเพื่อตั้งค่าตำแหน่งแถวที่ต้องการจะเริ่มเขียนภาพ
4. ส่งข้อมูลแถวเริ่มต้น(y1) และตำแหน่งแถวสิ้นสุด(y2) เพื่อกำหนดขอบเขตในแนวแกนวายที่จะทำการเขียนภาพ
5. หลังจากที่ทำคำสั่ง “CASET” และ “PASET” เรียบร้อยแล้วข้อมูลที่เราส่งไปจะไปแสดงที่ตำแหน่งในขอบเขตที่ตั้งค่าไว้ทีละจุดคงภาพ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 9.4 การส่งภาพไปแสดงผลที่จอ

6. ทำการส่งคำสั่ง "RAMWR" (0x2C) เพื่อเริ่มการเขียนภาพ ในขอบเขตที่กำหนดไว้
7. ทำการส่งข้อมูลจุดภาพ โดยข้อมูล 1 ไบต์จะหมายถึงค่าสีแปดบิตของจุด 1 จุด

9.3 ทาสก์ในระบบ

ในระบบนี้ประกอบไปด้วยทาสก์ทั้งสิ้น 6 ทาสก์ด้วยกัน โดยแต่ละทาสก์มีหน้าที่ของตนเองดังนี้

9.3.1 AppTaskFirst

ทำหน้าที่ในการตั้งค่าเริ่มต้นของระบบ ทำการสร้างทาสก์อื่นๆขึ้นมา และทำการแสดงผลภาวะของ ซีพียูในหน่วยเปอร์เซ็นต์ออกทางเซเวนเซกเมนต์

9.3.2 TempLoggerTask1(2,3)

เป็นทาสก์สามทาสก์ที่ทำหน้าที่ในการอ่านค่าจากเซ็นเซอร์วัดอุณหภูมิแต่ละตัว (ทาสก์ละหนึ่งตัว) ในทุกๆ วินาที และดูแลการจัดเก็บอุณหภูมิใน 100 วินาทีล่าสุดในหน่วยความจำ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

9.3.3 DispTask

เป็นทาสก์ที่คอยควบคุมการแสดงผลของจอกกราฟิคแอลซีดีโนเกีย 6100 คอยทำการคำนวณตำแหน่งในการวาดกราฟบนจอ และควบคุมการลบและวาดภาพบนจอทั้งหมด

9.3.4 UARTCmdTask

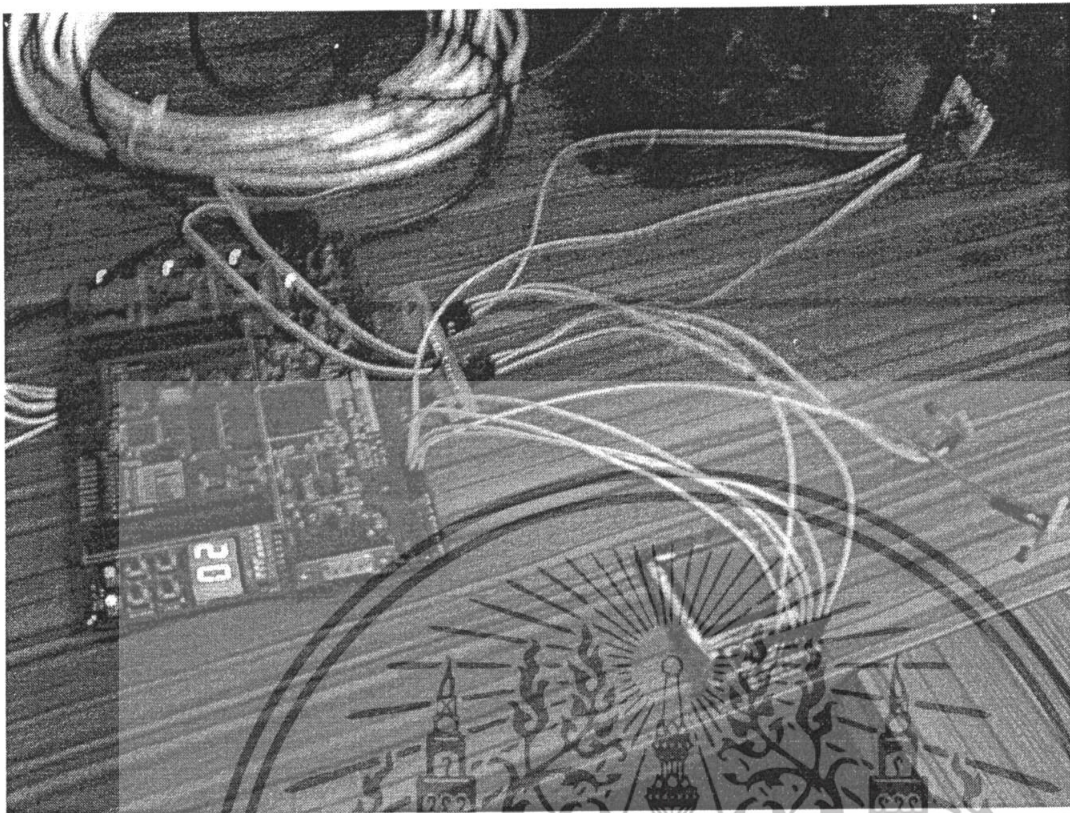
ทำหน้าที่ในการจัดการรับข้อมูลจากพอร์ทอนุกรมและทำการส่งเอค โคกลับ ไปยังคอน โซลฝั่งพีซี และประมวลผลคำสั่งที่ได้มาทางพอร์ทอนุกรม

9.4 ลำดับการทำงานของโปรแกรม

1. ทำการตั้งค่าเริ่มต้นต่างๆ และสร้างทาสก์ AppTaskFirst ขึ้นมา จากนั้นส่งการทำงานต่อให้กับ โอ เอส
2. โอเอสจะเรียกทาสก์ AppTaskFirst ขึ้นมาทำงาน AppTaskFirst จะทำการสร้างทาสก์ TempLoggerTask ทั้งสามทาสก์ขึ้นมา จากนั้นจะรออีเวนท์แฟลคที่บอกว่าทาสก์ทั้งสามนั้น ได้ทำการเตรียมการเรียบร้อยแล้ว
3. หลังจากได้รับอีเวนท์แฟลคครบทั้งสามตัวแล้ว AppTaskFirst จะทำการสร้าง DispTask ขึ้นมาเพื่อจัดการการแสดงผลที่จอภาพ จากนั้นทำการสร้าง UARTCmdTask เพื่อรอรับคำสั่งจากผู้ใช้งานระบบ
4. แต่ละทาสก์ทำหน้าที่ของตัวเองไปตามที่ได้กล่าวไว้แล้ว

9.5 ภาพรวมการทำงานของระบบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

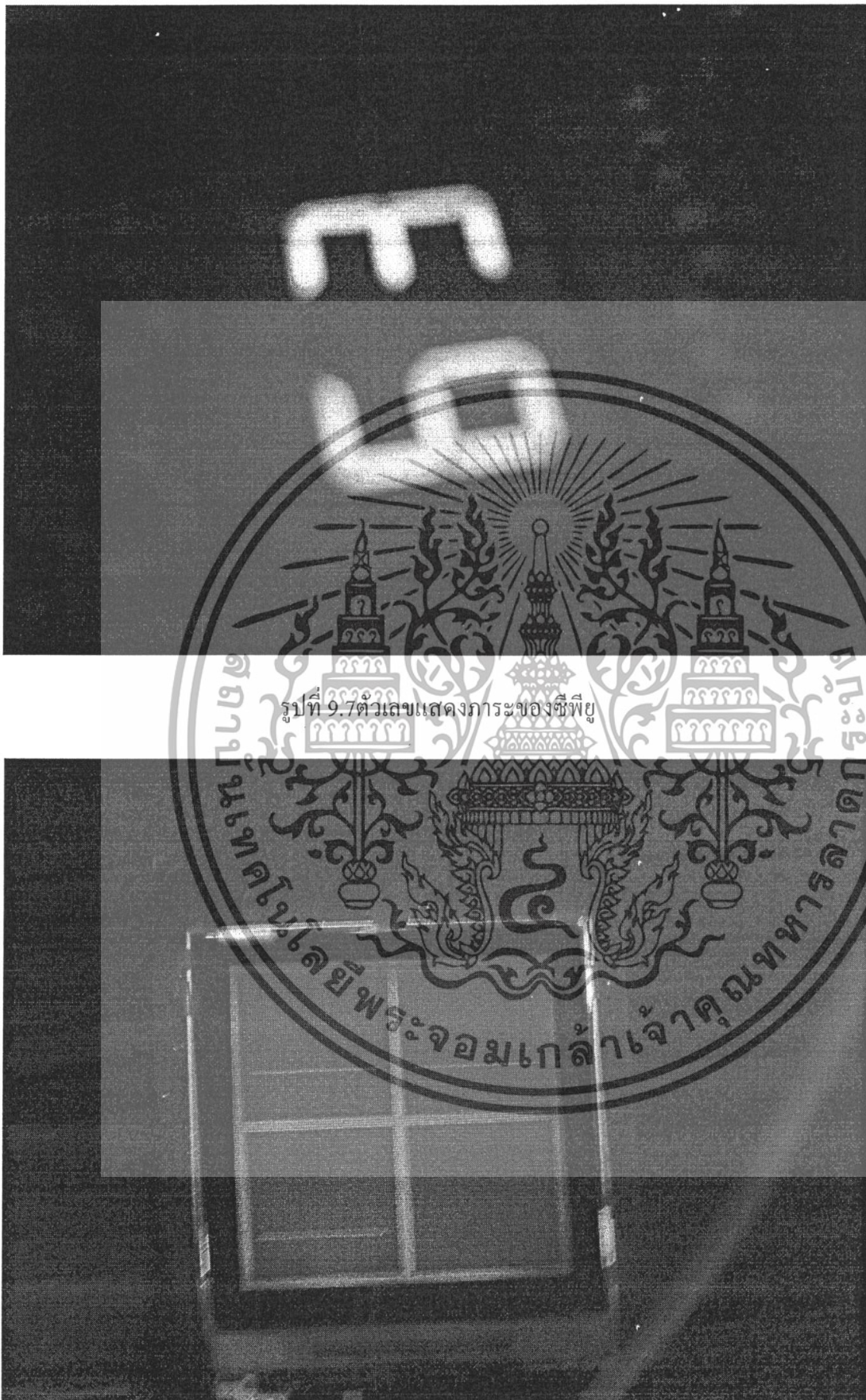


รูปที่ 9.5 ภาพรวมของระบบ



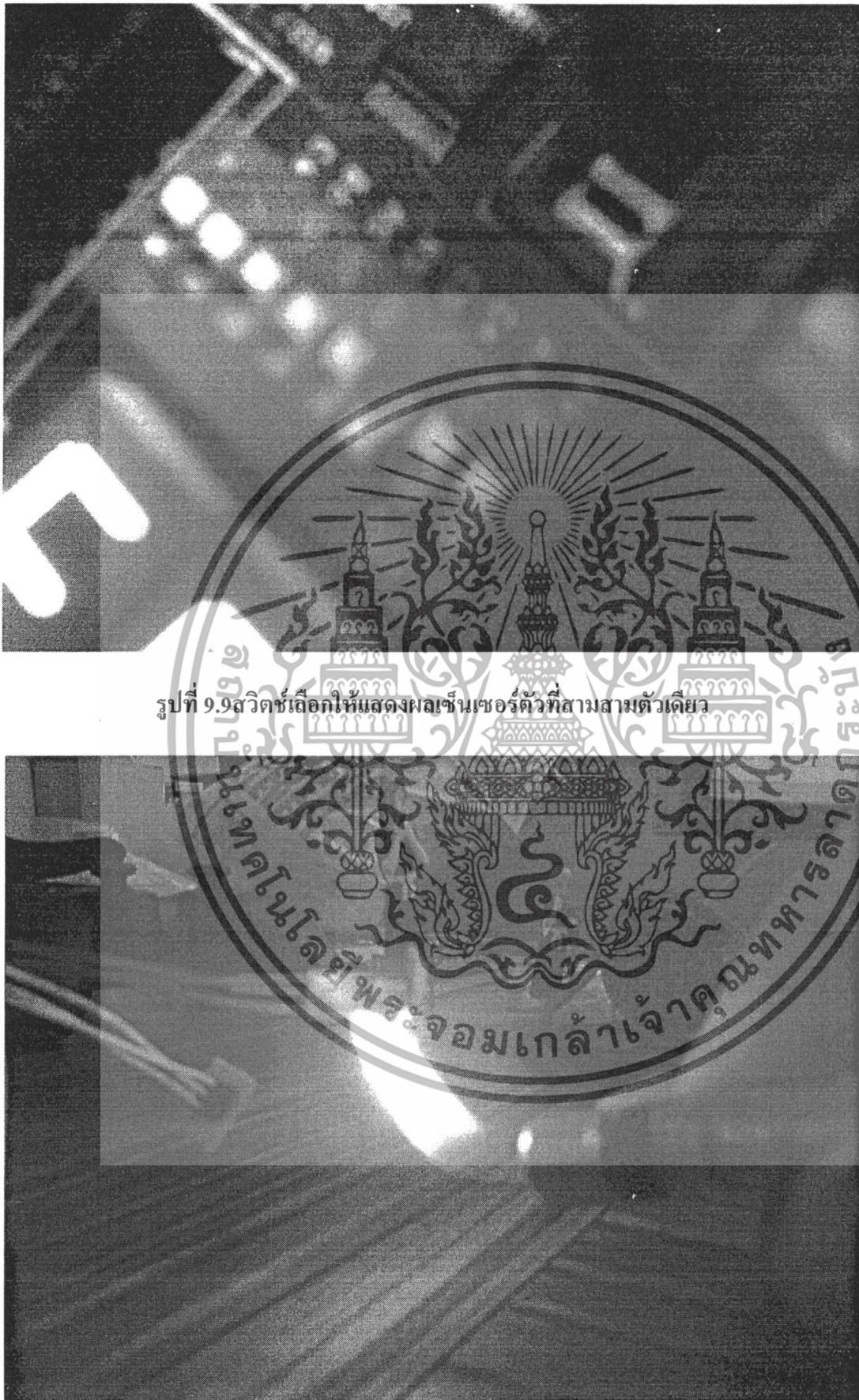
รูปที่ 9.6 เซ็นเซอร์วัดอุณหภูมิ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 9.7 ตัวเลขแสดงภาระของซีพียู

รูปที่ 9.8 กราฟแสดงอุณหภูมิของเซ็นเซอร์ทั้งสามตัว
 เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์และใช้ในเพียงอย่างเดียวเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
 ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 9.9 สวิตช์เด็กให้แสดงผลเช่นเซอร์ตัวที่สามสามตัวเดียว

เอกสารนี้เป็นเอกสารที่รูปที่ 9.10 เพิ่มอุณหภูมิที่เซ็นเซอร์ตัวที่สามนั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 10

บทวิจารณ์และบทสรุป

10.1 สรุปผลการทำงาน

1. สามารถสร้างระบบไมโครเบลสเพื่อใช้งานบนเอฟพีจีเอได้
2. สามารถเพิ่มและลดอุปกรณ์ รวมถึงปรับแต่งระบบไมโครเบลสที่สร้างขึ้นได้
3. สามารถสร้างอุปกรณ์ของตนเองบนเอฟพีจีเอและนำไปเชื่อมต่อกับระบบไมโครเบลสได้
4. สามารถเขียนโปรแกรมบนระบบไมโครเบลสได้
5. สามารถนำไมโครซีไอเอสทูวีไปใช้งานบนไมโครเบลสได้
6. สามารถเขียนโปรแกรมประยุกต์บนไมโครซีไอเอสทูวีได้
7. สร้างโปรแกรมประยุกต์ตัวอย่างพร้อมทั้งเอกสารประกอบการออกแบบ

10.2 ปัญหาและอุปสรรคในการทำงาน

1. ไมโครซีไอเอสทูวีมีขนาดใหญ่และไม่สามารถนำไปใช้งานบนเอฟพีจีเอสปาร์ทาน3 ได้เนื่องจากขนาดของบล็อกแรมไม่เพียงพอ
2. การเราท์วงจรภายในเอฟพีจีเอที่ต้องทำงานบนความถี่สูงๆ ระดับ 100 เมกะเฮิร์ตซ์ ใช้เวลาในการเราท์ที่นานมาก และหากทดสอบแล้วเกิดความผิดพลาด มีการแก้ไขวงจรจะต้องทำการเราท์ใหม่

10.3 แนวทางแก้ไข

1. เปลี่ยนไปใช้เอฟพีจีเอที่มีขนาดของบล็อกแรมสูงขึ้นเช่นเวอร์เท็กซ์2
2. ทำการเราท์วงจรที่ทำงานที่ความถี่ต่ำๆแล้วนำมาทดลองให้แน่ใจว่าทำงานได้ถูกต้องก่อนแล้วจึงเราท์วงจรที่ความถี่สูงๆครั้งเดียว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

10.4 บทวิจารณ์โครงการ

การออกแบบระบบเรียลไทม์โดยใช้ซอฟต์แวร์เซิร์ฟเวอร์บนเอฟพีจีเอนั้นสามารถทำได้ และนำไปใช้งานได้จริง อีกทั้งยังมีข้อดีในเรื่องของความยืดหยุ่นของระบบ เนื่องจากเราสามารถเพิ่ม ลด อุปกรณ์ภายในตัวชิพได้ตามต้องการ หรือแม้แต่จะสร้างอุปกรณ์ของตนเองเพิ่มเติมก็สามารถทำได้ อีกทั้งยังสามารถนำอุปกรณ์สำเร็จที่ผู้อื่นสร้างไว้เข้ามาเสริมได้อีกด้วย แต่ไม่เหมาะกับการนำไปใช้งานในระบบเล็กๆ เนื่องจากต้องอาศัยความรู้ความเข้าใจ และมีขั้นตอนในการพัฒนาที่ค่อนข้างจะซับซ้อนกว่าการใช้งานไมโครคอนโทรลเลอร์ทั่วไป และไม่เหมาะกับการนำไปใช้งานในระบบที่ต้องการประสิทธิภาพสูงเนื่องจากเอฟพีจีเอนั้นมีดีเลย์สูงกว่าวงจรรวมที่ออกแบบมาเฉพาะงาน

10.5 แนวทางการพัฒนา

ในงานอุตสาหกรรมจริงๆ โปรแกรมประยุกต์ที่เราทำการสร้างนั้นอาจมีขนาดใหญ่เกินกว่าที่จะเก็บชุดคำสั่งทั้งหมดไว้ในบล็อครวมได้ และในโครงการนี้ยังไม่ได้กล่าวถึงการนำหน่วยความจำแบบแฟลชมาเชื่อมต่อเป็นหน่วยความจำภายนอกของเอฟพีจีเอ แล้วนำชุดคำสั่งไปเก็บไว้ในหน่วยความจำแบบแฟลชนั้นๆ และเขียนโปรแกรมบูทโหลดเดอร์เพื่อให้ไมโครเบลสไปเรียกชุดคำสั่งจากแฟลชขึ้นมาทำงาน ซึ่งค่อนข้างที่จะมีขั้นตอนที่ต้องศึกษาอีกพอสมควร

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บรรณานุกรม

Jean J. Labrosse, 2002, **MicroC/OS-II The Real-Time Kernel Second Edition**,

CMPBooks, San Francisco, New York, Lawrence

“MicroBlaze Processor Reference Guide” [Online]

Available: http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf

“On-Chip Peripheral Bus Architecture Specifications” [Online]

Available: [http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/9A7AFA74DAD200D087256AB30005F0C8/\\$file/OpbBus.pdf](http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/9A7AFA74DAD200D087256AB30005F0C8/$file/OpbBus.pdf)

“Designing Custom OPB Slave Peripherals for MicroBlaze” [Online]

Available: http://www.xilinx.com/ipcenter/processor_central/microblaze/doc/opb_tutorial.pdf

“ภาคผนวก Sapatan3 (ปรับปรุงล่าสุด 14 ส.ค. 49)” [Online]

Available: http://www.kmitl.ac.th/~ksjirasa/Laboratory/DigitalLab/_Sapatan3BM.pdf

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้