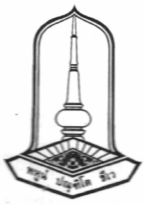
เทคนิคการจัดเวลาการทำงานแบบใหม่โดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์สำหรับสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงาน

สัญญา ควรคิด

เสนอต่อมหาวิทยาลัยมหาสารคาม เพื่อเป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาปรัชญาดุษฎีบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์
ตุลาคม 2558



คณะกรรมการสอบวิทยานิพนธ์ ได้พิจารณาวิทยานิพนธ์ของนายสัญญา ควรคิด แล้วเห็นสมควรรับเป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาปรัชญาดุษฎีบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์ ของมหาวิทยาลัยมหาสารคาม

คณะกรรมการสอบวิทยานิพนธ์

.....	ประธานกรรมการ
(รศ.ดร.สิงห์ทอง พัฒนเศรษฐานนท์)	(อาจารย์บัณฑิตศึกษาประจำคณะ)
.....	กรรมการ
(รศ.ดร.อภินันท์ อุรโสภณ)	(อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก)
.....	กรรมการ
(ผศ.ดร.วรวัฒน์ เสงี่ยมวิบูล)	(อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม)
.....	กรรมการ
(อาจารย์ ดร.ณัฐวุฒิ สุวรรณหา)	(อาจารย์บัณฑิตศึกษาประจำคณะ)
.....	กรรมการ
(อาจารย์ ดร.มนตรี โพธิโสโนทัย)	(ผู้ทรงคุณวุฒิ)

มหาวิทยาลัยอนุมัติให้รับวิทยานิพนธ์ฉบับนี้ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาปรัชญาดุษฎีบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์ ของมหาวิทยาลัยมหาสารคาม

.....	.....
(ศ.ดร.สัมพันธ์ ฤทธิเดช)	(ศ.ดร.ประดิษฐ์ เทอดทูล)
คณบดีคณะวิศวกรรมศาสตร์	คณบดีบัณฑิตวิทยาลัย
	วันที่ 13 เดือน ก.ค. พ.ศ. 2558

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ได้รับทุนรัฐบาลที่จัดสรรให้กระทรวงวิทยาศาสตร์และเทคโนโลยี เพื่อศึกษาระดับปริญญาเอก ด้านเทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์

วิทยานิพนธ์ฉบับนี้สำเร็จสมบูรณ์ได้ด้วยความกรุณาและความช่วยเหลืออย่างสูงยิ่งจาก รองศาสตราจารย์ ดร.อภินันท์ อุรโสภณ ประธานกรรมการควบคุมวิทยานิพนธ์ ผู้ช่วยศาสตราจารย์ ดร.วรวัฒน์ เสงี่ยมวิบูล อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม รองศาสตราจารย์ ดร.สิงห์ทอง พัฒนเศรษฐานนท์ ประธานกรรมการสอบ อาจารย์ ดร.ณัฐวุฒิ สุวรรณา กรรมการสอบ และ อาจารย์ ดร. มนตรี โพธิโสโนทัย ผู้ทรงคุณวุฒิภายนอก

ขอขอบพระคุณนักศึกษาและเจ้าหน้าที่ทุกท่านที่เอื้อเฟื้อสถานที่ทำงาน เครื่องมือและอุปกรณ์ในการทำงานวิจัย ณ ห้องปฏิบัติการวิจัยวิศวกรรมอิเล็กทรอนิกส์และคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยมหาสารคาม

สัญญา ควรคิด

ชื่อเรื่อง เทคนิคการจัดเวลาการทำงานแบบใหม่โดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์สำหรับสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงาน
ผู้วิจัย นายสัญญา ควรคิด
ปริญญา ปรัชญาดุษฎีบัณฑิต **สาขาวิชา** วิศวกรรมไฟฟ้าและคอมพิวเตอร์
กรรมการควบคุม รองศาสตราจารย์ ดร.อภินันท์ อุรโสภณ
ผู้ช่วยศาสตราจารย์ ดร.วรวัฒน์ เสงี่ยมวิบูล
มหาวิทยาลัย มหาวิทยาลัยมหาสารคาม **ปีที่พิมพ์** 2558

บทคัดย่อ

สถาปัตยกรรมกระตุ้นการทำงานด้วยเวลาแบบแบบไม่ขัดจังหวะการทำงานจัดเป็นระบบการจัดเวลาการทำงานที่ได้รับความนิยมเป็นอย่างมากสำหรับงานวิจัยด้านระบบสมองกลฝังตัว เนื่องจากเป็นสถาปัตยกรรมที่ไม่ซับซ้อนและมีพฤติกรรมด้านเวลาที่สามารถคาดการณ์ได้อย่างแม่นยำ อย่างไรก็ตามถึงแม้ว่าระบบการจัดเวลาดังกล่าวจะมีประโยชน์เป็นอย่างยิ่งแต่ถ้าหากมีการจัดเวลาการทำงานไม่เหมาะสมจะทำให้เกิดการสูญเสียด้านประสิทธิภาพของระบบอย่างมาก ซึ่งปัญหาที่สำคัญของระบบการจัดเวลาได้แก่ปัญหาความเปราะบางในการออกแบบระบบการจัดเวลาการทำงานและปัญหาความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน จากปัญหาดังกล่าว วิทยานิพนธ์นี้เสนอวิธีการออกแบบอัลกอริทึมและการจัดสร้างระบบการจัดเวลาการทำงานแบบใหม่โดยอาศัยเทคนิคการอินเตอร์รัพต์แบบหลายไทเมอร์เพื่อแยกการทำงานระหว่างส่วนควบคุมการจัดการเวลาและส่วนของงานแต่ละงานออกจากกันซึ่งสามารถลดผลกระทบการเกิดการโอเวอร์รันของงานทำให้เพิ่มประสิทธิภาพด้านการออกแบบระบบที่ดีขึ้น นอกจากนี้ได้เสนอเทคนิคแซนวิชดีเลย์มาประยุกต์ใช้งานเพื่อปรับปรุงให้คาบเวลาของแต่ละงานมีค่าเท่ากันเพื่อลดปัญหาด้านความผิดพลาดทางเวลาขณะเริ่มต้นทำงานของระบบ

ผลลัพธ์จากการประเมินและทดสอบประสิทธิภาพของระบบที่นำเสนอ ประสิทธิภาพของหน่วยความจำ ซีพียู และการทดสอบด้านความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน พบว่าระบบที่นำเสนอสามารถลดผลกระทบของการเกิดทาสก์โอเวอร์รันและปัญหาความผิดพลาดทางเวลาขณะเริ่มต้นทำงานซึ่งเป็นปัญหาจากงานวิจัยที่ผ่านมาได้ นอกจากนี้อัลกอริทึมที่นำเสนอยังมีประสิทธิภาพด้านเวลาในการจัดตารางการทำงานที่สูงกว่าวิธีการที่ผ่านมาโดยที่ความสามารถในการจัดตารางการทำงานใกล้เคียงกัน

คำสำคัญ: ระบบสมองกลฝังตัว; สถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงาน; เทคนิคการอินเตอร์รัพต์แบบหลายไทเมอร์; เทคนิคแซนวิชดีเลย์

TITLE A novel scheduling technique based on multiple timer interrupts for the time-triggered co-operative architecture
AUTHOR Mr. Sanya Kuankid
DEGREE Doctor of Philosophy **MAJOR** Electrical and Computer Engineering
ADVISORS Assoc. Prof. Apinan Aurasopon, Ph.D.
Asst. Prof. Worawat Sa-ngiamvibool, Ph.D.
UNIVERSITY Mahasarakham University **DATE** 2015

ABSTRACT

The importance of the field of research comes from the increased role of embedded systems in the world of today. When developing high predictable and reliable embedded system, the use of time-triggered co-operative architecture offers significant advantages over other alternative approaches because such scheduler provides a simple and helps to produce highly predictable behavior. Despite many advantages, there are failure modes that can greatly impair system performances. That are the fragility of scheduler and the problem of task release jitter.

To address the problems, this thesis proposed a novel scheduling technique based on the employment of multiple timer interrupts to avoid the impact of task overruns and the sandwich delay mechanism to fix the problem of jitter at the starting time of the tasks.

The results of the system performance show that the proposed system can achieve better performance to overcome the problems of task release jitter, task overruns, and the fragility of scheduler. In addition, the evaluating of proposed algorithm provides the effective schedulability and can help in a significant reduction of computation time as compared with a traditional scheduler.

Key Words: Embedded system; Co-operative architecture; Multiple timer interrupts; Sandwich delay mechanisms

สารบัญ

หน้า

กิตติกรรมประกาศ ก
บทคัดย่อภาษาไทย ข
บทคัดย่อภาษาอังกฤษ ค
สารบัญ ง
สารบัญตาราง ฉ
สารบัญภาพประกอบ ช
บทที่ 1 บทนำ 1
1.1 ภูมิหลัง 1
1.2 ความมุ่งหมายของการวิจัย 5
1.3 ขอบเขตของการวิจัย 5
1.4 นิยามศัพท์เฉพาะ 5
บทที่ 2 ปริทัศน์เอกสารข้อมูล 6
2.1 สถาปัตยกรรมในการพัฒนาระบบสมองกลฝังตัว 6
2.2 การจัดการตารางงานในระบบเวลาจริง 7
2.3 ระบบปฏิบัติการสำหรับระบบสมองกลฝังตัวที่มีทรัพยากรจำกัด 9
2.4 อัลกอริทึมการจัดตารางเวลางานสถาปัตยกรรมการกระตุ้นการทำงานด้วยเวลา (Scheduling algorithm in TT system) 10
2.5 การบริหารแบบวนรอบ (Cyclic executive) 12
2.6 ปัญหาที่สำคัญของระบบการจัดเวลาแบบ TTC 16
2.7 สรุป 20
บทที่ 3 วิธีดำเนินการวิจัย 21
3.1 แนวคิดในการการจัดเวลาการทำงานแบบใหม่สำหรับสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงาน 21
3.2 ระบบการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์ (Multiple Timer interrupts: MTIs) 24
3.3 อัลกอริทึมเพื่อทดสอบความสามารถในการจัดตารางการทำงานของระบบ TTC-MTIs 25
3.4 การลดผลกระทบของค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน 28
บทที่ 4 ผลการวิจัยและการอภิปราย 31
4.1 การพัฒนาระบบการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์ 31
4.2 การประเมินประสิทธิภาพอัลกอริทึม TTSA-MTI 37
4.3 การทดสอบประสิทธิภาพระบบ TTC-MTIs 41
4.4 สรุป 44

สารบัญ

หน้า

บทที่ 5 สรุปผล อภิปรายผล และข้อเสนอแนะ 46
5.1 การออกแบบสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะที่เหมาะสม 46
5.2 การปรับปรุงประสิทธิภาพสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะ 46
5.3 ข้อเสนอแนะ 47
เอกสารอ้างอิง 48
ภาคผนวก 52
ประวัติย่อผู้วิจัย 75

สารบัญตาราง

หน้า

ตาราง 2.1 คุณลักษณะของงานที่ต้องการจัดตารางการทำงาน 11
ตาราง 2.2 ตัวอย่างคุณลักษณะงานระบบการบริหารแบบวนรอบ 13
ตาราง 4.1 รีจิสเตอร์ที่เกี่ยวข้องกับการประยุกต์ใช้ไทเมอร์ในการสร้างสัญญาณคาบเวลา 33
ตาราง 4.2 มอดูลต้นฉบับภาษาซีและหน้าที่การทำงานของฟังก์ชันต่างๆ 36
ตาราง 4.3 ค่าเวลาการเกิดโอเวอร์เฮดของ Tick interrupt จำนวน 100 งาน 38
ตาราง 4.4 ค่าเวลาการเกิดโอเวอร์เฮดของ Task interrupt 39
ตาราง 4.5 เวลาที่ใช้ในการจัดตารางการทำงานของอัลกอริทึม TTC-Dispatch และ TTSA-MTI 40
ตาราง 4.6 ผลการทดลองเปรียบเทียบประสิทธิภาพหน่วยความจำของระบบ TTC-Dispatch และ TTC-MTIs 42
ตาราง 4.7 ผลการทดลองเปรียบเทียบประสิทธิภาพซีพียูของระบบ TTC-Dispatch และ TTC-MTIs 42
ตาราง 4.8 คุณลักษณะงานในการทดสอบความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน 43
ตาราง 4.9 ความผิดพลาดทางเวลาขณะเริ่มต้นทำงานของระบบ TTC-Dispatch และ TTC-MTIs 44

สารบัญภาพประกอบ

หน้า

ภาพประกอบ 1.1 รูปแบบการทำงานของการจัดเวลาแบบ TTC-Dispatch 2
ภาพประกอบ 1.2 ความผิดพลาดทางเวลาขณะเริ่มต้นทำงานสำหรับงานแบบรายคาบ 3
ภาพประกอบ 1.3 แนวคิดในการจัดตารางการทำงาน TTC แบบใหม่โดยอาศัยเทคนิค
การอินเตอร์รัพต์แบบหลายไทเมอร์ 4
ภาพประกอบ 2.1 การคาดการณ์ด้านการตลาดของเทคโนโลยีระบบสมองกลฝังตัว
ของ BCC Research ตั้งแต่ปี ค.ศ. 2009-2015 6
ภาพประกอบ 2.2 โมเดลของงานในระบบเรียลไทม์ 8
ภาพประกอบ 2.3 โมเดลงานแบบรายคาบ 8
ภาพประกอบ 2.4 โมเดลของงานแบบไม่เป็นรายคาบ 9
ภาพประกอบ 2.5 สถาปัตยกรรม (a) RTOS และ (b) sEOS 10
ภาพประกอบ 2.6 ตารางเวลาการทำงานของงานที่ต้องการจัดตารางการทำงาน 11
ภาพประกอบ 2.7 ตัวอย่างตารางเวลาการจัดตารางงานแบบความสำคัญคงที่ 12
ภาพประกอบ 2.8 ตัวอย่างตารางเวลาการจัดตารางงานแบบเปลี่ยนแปลงได้ 12
ภาพประกอบ 2.9 รูปแบบของการบริหารแบบวนรอบที่ได้จากคุณลักษณะงานจากตาราง 2.2 13
ภาพประกอบ 2.10 ตัวอย่างโปรแกรมระบบ TTC-SL 14
ภาพประกอบ 2.11 ตัวอย่างการทำงานของ TTC-ISR 15
ภาพประกอบ 2.12 ไดอะแกรมแสดงการทำงานของการจัดเวลาแบบ TTC-ISR 15
ภาพประกอบ 2.13 แสดงการทำงานของ TTC-Dispatch 16
ภาพประกอบ 2.14 ตัวอย่างการเกิดโอเวอร์รันของงานในการจัดเวลาแบบ TTC-Dispatch 17
ภาพประกอบ 2.15 ค่าความผิดพลาดทางเวลาของการจัดเวลาที่เกิดจากจากโอเวอร์เฮด 18
ภาพประกอบ 2.16 ค่าความผิดพลาดทางเวลาที่เกิดจากการจัดลำดับการดำเนินการของงาน 18
ภาพประกอบ 3.1 ฝังงานการทำงานของ TTC-Dispatch 22
ภาพประกอบ 3.2 การเรียกใช้งานฟังก์ชันต่างๆ ของ TTC-Dispatch 22
ภาพประกอบ 3.3 แนวคิดในการจัดเวลาการทำงานแบบใหม่สำหรับสถาปัตยกรรม TTC 23
ภาพประกอบ 3.4 เทคนิคการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์ 25
ภาพประกอบ 3.5 การเรียกใช้งานฟังก์ชันต่างๆ ของ TTC-MTIs 25
ภาพประกอบ 3.6 โฟลว์ชาร์ตของอัลกอริทึม TTSA-MTI สำหรับทดสอบความสามารถ
ในการจัดตารางการทำงานของระบบ TTC-MTIs 25
ภาพประกอบ 3.7 โอเวอร์เฮดของการจัดตารางงานของระบบ TTC-MTIs 28
ภาพประกอบ 3.8 (a) ค่าความผิดพลาดทางเวลาที่เกิดจากการจัดลำดับการดำเนินการของงาน,
(b) การลดผลกระทบของ Task period โดยใช้เทคนิคแซนวิชดีเลย์,
(c) การนำเทคนิคแซนวิชดีเลย์ไปประยุกต์ใช้กับระบบ MTIs 29

หน้า

ภาพประกอบ 4.1 บล็อกไดอะแกรมไมโครคอนโทรลเลอร์ ARM7TDMI เบอร์ LPC2129 32
ภาพประกอบ 4.2 โปรแกรม RealView MDK โดยเลือกใช้ไมโครคอนโทรลเลอร์เบอร์ LPC2129 35
ภาพประกอบ 4.3 โครงสร้างโปรเจคในรูปแบบภาษาซีของระบบ TTC-MTIs 35
ภาพประกอบ 4.4 สมการเส้นตรงของค่าเวลาโอเวอร์เฮดของ Tick interrupt 38
ภาพประกอบ 4.5 กราฟแสดงเวลาที่ใช้ในการคำนวณของอัลกอริทึม TTC-Dispatch และ TTSA-MTI 40
ภาพประกอบ 4.6 กราฟแสดงความสามารถในการจัดตารางการทำงานของอัลกอริทึม TTC- Dispatch และTTSA-MTI 41
ภาพประกอบ 4.7 ค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน 43
ภาพประกอบ 4.8 รูปแบบของงานแต่ละเซตเพื่อใช้ทดสอบความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน 43
ภาพประกอบ 4.9 กราฟแสดงการเปรียบเทียบค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานของระบบ TTC-Dispatch และ TTC-MTIs 44

บทที่ 1

บทนำ

1.1 ภูมิหลัง

ความสำคัญของงานวิจัยด้านระบบสมองกลฝังตัว (Embedded system: ES) มีบทบาทเพิ่มขึ้นมากในโลกปัจจุบัน เทคโนโลยีด้านระบบสมองกลฝังตัวถูกนำเสนอในรูปแบบของคอมพิวเตอร์ที่ออกแบบมาสำหรับงานเฉพาะด้าน ที่มีรูปลักษณ์เฉพาะในเรื่อง ความเร็วและมีความน่าเชื่อถือสูง โดยอาศัยระบบปฏิบัติการแบบฝังตัวควบคุมการทำงานของอุปกรณ์ประมวลผลซึ่งถูกติดตั้งไว้ภายในระบบฝังตัว Laplant [1] ได้อธิบายว่าการออกแบบระบบสมองกลฝังตัวส่วนใหญ่เป็นการทำงานร่วมกันระหว่างฮาร์ดแวร์และวิธีการทางซอฟแวร์ เพื่อประสิทธิภาพด้านความเร็ว การลดค่ากำลังงาน การลดค่าใช้จ่าย การเพิ่มประสิทธิภาพของระบบ หรือสำหรับความต้องการพิเศษด้านอื่นๆ

สำหรับงานด้านสถาปัตยกรรมในการพัฒนาซอฟแวร์ระบบสมองกลฝังตัว หากอาศัยวิธีการกระตุ้นการทำงาน (Triggering mechanism) ในการทำงาน สามารถแยกสถาปัตยกรรมซอฟแวร์ได้สองรูปแบบได้แก่ สถาปัตยกรรมการกระตุ้นการทำงานด้วยเหตุการณ์ (Event trigger: ET) และสถาปัตยกรรมการกระตุ้นการทำงานด้วยเวลา (Time trigger: TT) [2, 3] สถาปัตยกรรมแบบกระตุ้นการทำงานด้วยเหตุการณ์หรือแบบออนไลน์ (Online system) จะทำงานเมื่อมีเหตุการณ์ใดๆ เกิดขึ้นซึ่งจะเกิดขึ้น ณ เวลาใดก็ได้ สถาปัตยกรรมแบบนี้ส่วนใหญ่จะถูกนำไปประยุกต์ใช้สำหรับงานที่ต้องการการตอบสนองด้วยเวลาอันรวดเร็วและมีความยืดหยุ่นสูง อย่างไรก็ตามสถาปัตยกรรมนี้มีความต้องการทรัพยากรสูงทั้งด้านฮาร์ดแวร์ ได้แก่ หน่วยประมวลผล หน่วยความจำ ฯลฯ และซอฟแวร์ระบบปฏิบัติการที่ต้องสามารถรองรับการทำงานแบบเรียลไทม์ได้ (Real time operating system: RTOS) ในส่วนของสถาปัตยกรรมแบบกระตุ้นการทำงานด้วยเวลาหรือแบบออฟไลน์นั้น (Offline system) [2-4] ระบบจะถูกกระตุ้นให้ทำงานโดยอาศัยการทำงานตามตารางเป็นรายคาบ (Periodic time) การทำงานจะแบ่งเป็นสองส่วนได้แก่ การออกแบบตารางเวลา (Design time) และการดำเนินการจริง (Run time) โดยในการออกแบบตารางการทำงานนั้นจะต้องรู้การทำงานของระบบล่วงหน้าเพื่อจะได้ออกแบบระบบให้มีประสิทธิภาพและมีการทำงานอย่างแม่นยำ

อย่างไรก็ดี สำหรับการประยุกต์ใช้งานระบบสมองกลฝังตัวที่ต้องการการคาดการณ์และความน่าเชื่อถือสูงนั้น (High predictability and reliability) นักวิจัยส่วนมากได้นำสถาปัตยกรรมกระตุ้นการทำงานด้วยเวลาสำหรับประยุกต์ใช้งาน เนื่องจากสถาปัตยกรรมดังกล่าวมีข้อดีดังนี้

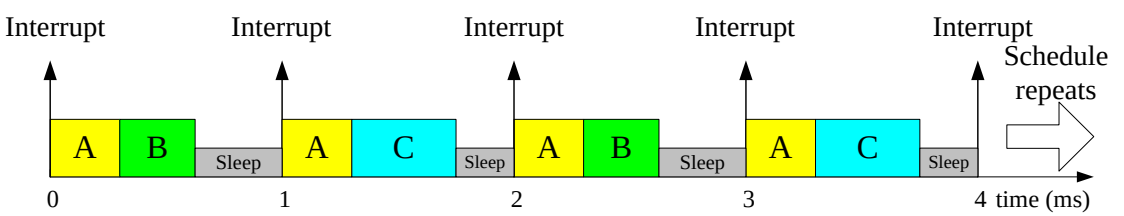
1. มีความเหมาะสมที่จะนำไปใช้กับระบบฝังตัวที่มีทรัพยากรจำกัด (Resource constrained embedded system) เนื่องจากระบบได้มีการออกแบบการทำงานมาก่อนทำให้สามารถจัดการทรัพยากรได้เหมาะสม และสามารถสร้างระบบปฏิบัติการฝังตัวแบบง่ายมาใช้งานได้ ซึ่งไม่ต้องมีค่าใช้จ่ายในการจัดหาระบบปฏิบัติการในเชิงพาณิชย์ ทำให้ลดต้นทุนด้านการพัฒนาซอฟแวร์

2. มีความเหมาะสมที่จะนำไปใช้สำหรับระบบที่ต้องการการคาดการณ์และความน่าเชื่อถือสูงเนื่องจากระบบได้มีการออกแบบการทำงานมาก่อน ทำให้สามารถตอบสนองการทำงานได้ทันเวลาและยังสามารถนำไปใช้งานกับระบบที่ต้องการความปลอดภัยสูง (Safety related applications)

3. ในการออกแบบสถาปัตยกรรมแบบกระตุ้นการทำงานด้วยเวลาอย่างเหมาะสม ระบบสามารถลดค่าการใช้พลังงานได้ เนื่องจากระบบจะเริ่มทำงาน (Active mode) ตามเวลาที่กำหนด เมื่อทำงานเสร็จเรียบร้อยแล้วก็ให้เข้าสู่โหมดประหยัดพลังงาน (Idle mode) เพื่อลดค่าพลังงานและรอให้ถึงเวลาในการทำงานต่อไปจึงจะเริ่มทำงานอีกครั้ง การทำงานในลักษณะนี้ทำให้สามารถลดพลังงานรวมในการใช้งานได้

ในการออกแบบระบบการกระตุ้นการทำงานด้วยเวลาเพื่อไปประยุกต์ใช้กับระบบฝังตัวที่มีทรัพยากรจำกัด เช่นไมโครคอนโทรลเลอร์ สถาปัตยกรรมซอฟแวร์ชนิดกระตุ้นการทำงานด้วยเวลาแบบโคโอเปอเรทีฟ (Time-Triggered Co-operative architecture: TTC) จัดเป็นระบบการจัดการเวลาการทำงานที่ได้รับความนิยมเป็นอย่างมาก อัลกอริทึมการจัดเวลาการทำงานแบบ TTC เป็นสถาปัตยกรรมที่ไม่ซับซ้อนและมีพฤติกรรมด้านเวลาที่สามารถคาดเดาได้อย่างแม่นยำ (Highly-predictable timing behavior) การจัดตารางเวลาการทำงานความสำคัญคงที่แบบ Co-operative หรือ Non-preemptive คือระบบการทำงานที่อนุญาตให้งานเพียงหนึ่งงานดำเนินการทำงานต่อเนื่องจนเสร็จในเวลาหนึ่งโดยไม่ถูกขัดจังหวะหรือรบกวน ระบบนี้จะปลอดจากการถูกแทรกแซงจากงานอื่น ซึ่งลำดับความสำคัญของแต่ละงานได้ถูกกำหนดไว้ก่อนที่ระบบจะเริ่มทำงานโดยที่ลำดับความสำคัญของงานจะไม่เปลี่ยนแปลงตลอดช่วงอายุการทำงานของระบบ การจัดตารางเวลาในลักษณะนี้สามารถประยุกต์ใช้ได้ทั้งการใช้งานที่เกี่ยวกับความปลอดภัยยานยนต์ ระบบการสื่อสารแบบไร้สาย ระบบการรวบรวมข้อมูล (Data acquisition system) ระบบการเฝ้าดู ระบบควบคุมแบบต่างๆ ฯลฯ [5-7]

ในการพัฒนารูปแบบการจัดเวลาการทำงานแบบ TTC นั้น นักพัฒนาส่วนใหญ่นิยมใช้ระบบการจัดเวลาชนิดกระตุ้นด้วยเวลาแบบโปรแกรมเลือกจ่ายงาน (TTC-Dispatch scheduler) หรือที่เรียกว่า “TTC-Dispatch” ซึ่งเป็นระบบการจัดตารางเวลาที่สามารถสร้างระบบได้ง่าย ไม่สิ้นเปลืองทรัพยากรและใช้พลังงานในการทำงานอย่างมีประสิทธิภาพ รูปแบบการทำงานในลักษณะนี้เหมาะสำหรับการนำไปประยุกต์ใช้งานสำหรับระบบฝังตัวที่มีราคาไม่แพง (Low-cost embedded applications) โดยการทำงานจะใช้เทคนิคการอินเตอร์รัพท์ในการสร้างสัญญาณรายคาบเพื่อกระตุ้นระบบการทำงาน ตัวอย่างการทำงานของการจัดเวลาแบบ TTC-Dispatch แสดงดังภาพประกอบ 1.1 จะเห็นว่าระบบการทำงานประกอบด้วย 3 งานได้แก่ Task A, Task B, และ Task C ซึ่ง Task A จะถูกกระตุ้นให้ทำงานทุกๆ 1 ms ในขณะที่ Task B และ Task C จะทำงานภายหลังจาก Task A ดำเนินการเสร็จสิ้นโดยทำงานทุกๆ 2 ms สลับกัน เมื่อทุกงานในระบบถูกดำเนินการเรียบร้อยแล้วก็จะเข้าสู่โหมดประหยัดพลังงานเพื่อลดค่าพลังงานและรอให้ถึงเวลาในการทำงานต่อไปจึงจะเริ่มทำงานอีกครั้ง



ภาพประกอบ 1.1 รูปแบบการทำงานของการจัดเวลาแบบ TTC-Dispatch

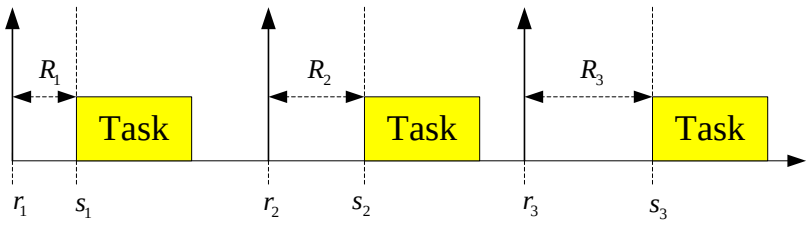
อย่างไรก็ตามถึงแม้ว่าระบบการจัดเวลาแบบ TTC-Dispatch จะมีประโยชน์เป็นอย่างยิ่งสำหรับระบบสมองกลฝังตัวที่เน้นการทำงานที่ต้องการการคาดการณ์และความน่าเชื่อถือสูง แต่ถ้าหากมีการจัดเวลาการทำงานไม่เหมาะสมจะทำให้เกิดการสูญเสียด้านประสิทธิภาพของระบบอย่างมาก ซึ่งปัญหาที่สำคัญของระบบการจัดเวลาแบบนี้ ได้แก่

1. ความเปราะบางในการออกแบบระบบการจัดเวลาการทำงาน (Fragility of scheduler)

การจัดเวลาการทำงานแบบ TTC จะมีความเปราะบางมากในสถานการณ์ที่เกิดสภาวะโอเวอร์รัน (Overruns) ซึ่งเกิดจากกรณีที่เวลาการทำงานจริงของงานใดๆ มากกว่าเวลาที่ทำนายไว้ โดยงานวิจัยของ Buttazzo [8] ได้กล่าวว่าสภาวะการเกิดโอเวอร์รันนี้สามารถสร้างผลกระทบแบบโดมิโน (Domino effect) กับงานที่ดำเนินการต่อเนื่องกันมา ซึ่งปัญหานี้อาจทำให้ระบบไม่สามารถดำเนินการใดๆ ต่อไปได้ (Hang indefinitely) โดยอาจส่งผลให้ไม่สามารถดำเนินการจัดตารางเวลาให้ทำงานได้ตามที่ออกแบบไว้และอาจเกิดการสูญเสียกับระบบงานได้

2. ความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน (Release jitter: *R*)

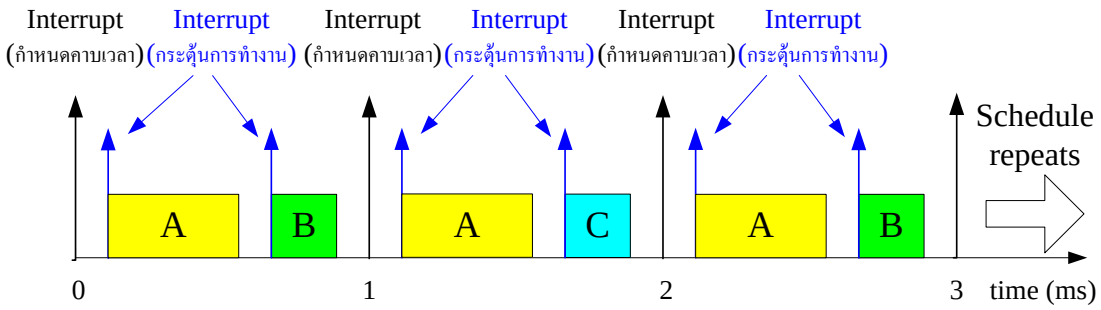
ค่า Release jitter คือค่าความผิดพลาดทางเวลาในการเริ่มต้นทำงานแสดงดังภาพประกอบ 1.2 จะเห็นว่าในการทำงานแบบอุดมคติ ค่าเวลาเริ่มต้นทำงาน (Task start time: s) ของงานใดๆ ควรเริ่มต้นทำงานหลังจากเกิดการกระตุ้นการทำงาน (Task release time: r) ณ เวลาเดียวกันในทุกคาบเวลา อย่างไรก็ตามในทางปฏิบัติค่าเวลาเริ่มต้นทำงานอาจมีเปลี่ยนแปลงได้ ซึ่งเกิดได้หลายสาเหตุ ได้แก่ความผิดพลาดทางเวลาที่เกิดจากพฤติกรรมการอินเตอร์รัพต์ (Interrupt timing behavior) โอเวอร์เฮดของการจัดตารางเวลา (Scheduler overhead) หรือจากการจัดลำดับการดำเนินการของงาน (Task placement) เป็นต้น [5, 9, 10] สิ่งเหล่านี้จะส่งผลให้เวลาในการเริ่มต้นทำงานในแต่ละคาบเวลาไม่เท่ากัน ซึ่งจากงานวิจัยของ Hughes และคณะกล่าวว่า Release jitter เป็นสิ่งสำคัญในการพิจารณาพฤติกรรมในการทำงานของระบบ ซึ่งระบบ TTC ที่มีพฤติกรรมการคาดการณ์ในการทำงานสูง (highly predictable behavior) ควรมีค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานน้อยที่สุด เพราะปัญหาของค่าความผิดพลาดทางเวลานี้อาจส่งผลกระทบต่อระบบงาน โดยเฉพาะอย่างยิ่งระบบงานแบบฮาร์ดเรียลไทม์ (Hard real-time) และอาจเกิดความสูญเสียอย่างรุนแรงในการทำงาน [8, 11-13]



ภาพประกอบ 1.2 ความผิดพลาดทางเวลาขณะเริ่มต้นทำงานสำหรับงานแบบรายคาบ

จากงานวิจัยที่ผ่านมาพบว่าได้มีการพัฒนาวิธีการเพื่อแก้ปัญหาดังกล่าวได้แก่ วิธี Dynamic Voltage scaling (DVS) [5] โดยใช้วิธีการปรับระดับแรงดันที่เหมาะสมในการทำงานเพื่อให้งานแต่ละงานเริ่มต้นทำงานในเวลาเดียวกันในแต่ละคาบเวลาเพื่อลดค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานของระบบ และวิธี Task Guardian (TG) โดยระบบจะใช้วิธีการสำรองข้อมูลการทำงานไว้เพื่อแก้ปัญหาการเกิดโอเวอร์รันของงาน อย่างไรก็ดีวิธีการแก้ปัญหาที่เกิดขึ้นในการจัดตารางการทำงานเหล่านี้มีความซับซ้อนและตัวโครงสร้างระบบมีความต้องการใช้ทรัพยากรที่สูง ซึ่งในทางปฏิบัติอาจไม่มีความเหมาะสมที่จะนำไปประยุกต์ใช้สำหรับระบบฝังตัวที่มีทรัพยากรจำกัดได้

จากข้อจำกัดดังกล่าว งานวิจัยนี้จึงสนใจพัฒนาเทคนิคการจัดเวลาการทำงานแบบใหม่ที่สามารถแก้ปัญหาเหล่านี้ได้ โดยมีแนวคิดให้ระบบทำการแยกการทำงานระหว่างส่วนควบคุมการกระตุ้นการทำงานและส่วนของงานแต่ละงานออกจากกัน ดังภาพประกอบ 1.3 ทั้งนี้ในการทำงานจะอาศัยเทคนิคการอินเตอร์รัพต์แบบหลายไทเมอร์เพื่อแยกการทำงานออกจากกันซึ่งสามารถลดผลกระทบการเกิดการโอเวอร์รันของงานในระบบ จากภาพประกอบจะเห็นว่าลูกศรสั้นแสดงสัญลักษณ์ของการอินเตอร์รัพต์เพื่อกำหนดสัญญาณรายคาบทุกๆ 1 ms ส่วนลูกศรยาวแสดงสัญลักษณ์ของการอินเตอร์รัพต์เพื่อกระตุ้นการทำงานในแต่ละงาน เนื่องจากระบบได้กำหนดความสำคัญของการอินเตอร์รัพต์ไว้ก่อนล่วงหน้าว่าแต่ละงานจะถูกดำเนินการในเวลาใด ดังนั้นเมื่อเกิดการโอเวอร์รันขึ้นในงานใดๆ ระบบดังกล่าวก็จะอนุญาตให้งานที่มีความสำคัญกว่าสามารถแทรกการทำงานของงานที่เกิดโอเวอร์รัน ทำให้ระบบการทำงานนี้สามารถกลับมาทำงานได้ตามเวลาที่กำหนดและไม่ส่งผลกระทบกับงานที่จะถูกดำเนินงานต่อมา นอกจากนี้ระบบยังสามารถลดค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานได้โดยการกำหนดค่าเวลาเริ่มต้นในการทำงานของแต่ละงานให้เกิดขึ้นในเวลาเดียวกันในแต่ละคาบเวลาที่งานนั้นถูกดำเนินการ



ภาพประกอบ 1.3 แนวคิดในการจัดตารางการทำงาน TTC แบบใหม่โดยอาศัยเทคนิคการอินเตอร์รัพต์แบบหลายไทเมอร์

1.2 ความมุ่งหมายของการวิจัย

1.2.1 เพื่อศึกษาสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงานสำหรับใช้งานกับระบบฝังตัวที่มีทรัพยากรจำกัด โดยเน้นการนำไปประยุกต์ใช้ในระบบอิเล็กทรอนิกส์เชิงเกษตรกรรม

1.2.2 เพื่อวิเคราะห์วิธีการแก้ปัญหาความเปราะบางในการออกแบบระบบและลดค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานของสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงาน

1.2.3 เพื่อนำเสนอเทคนิคการจัดเวลาการทำงานแบบใหม่ สำหรับสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงาน

1.3 ขอบเขตของการวิจัย

1.3.1 พัฒนาระบบปฏิบัติการฝังตัวโดยใช้เทคนิคการจัดเวลาการทำงานแบบใหม่เพื่อใช้งานกับสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะสำหรับประยุกต์ใช้กับไมโครคอนโทรลเลอร์

1.3.2 ประเมินและทดสอบประสิทธิภาพวิธีการที่นำเสนอ โดยการกำหนดคุณลักษณะงานชนิดตัวแปรสุ่มที่มีการกระจายแบบยูนิฟอร์ม

1.4 นิยามศัพท์เฉพาะ

ES	คือ	ระบบสมองกลฝังตัว (Embedded system)
ET	คือ	การกระตุ้นการทำงานด้วยเหตุการณ์ (Event trigger)
TT	คือ	การกระตุ้นการทำงานด้วยเวลา (Time trigger)
t_i	คือ	งาน (Task)
p_i	คือ	คาบเวลางาน (Task period)
c_i	คือ	ค่าเวลาในการดำเนินงาน (Task execution time)
d_i	คือ	เดดไลน์ของงาน (Task relative deadline)
o_i	คือ	ออฟเซตของงาน (Task offset time)
r	คือ	เวลาการกระตุ้นการทำงาน (Task release time)
s	คือ	เวลาเริ่มต้นการทำงาน (Task start time)
f	คือ	เวลาสิ้นสุดการทำงาน (Task finishing time)
U	คือ	ค่ายูทิไลเซชัน (Capacity utilization)
$WCET$	คือ	ค่าเวลาสูงสุดในการดำเนินงาน (Worst case execution time)
lcm	คือ	ค่าคูณร่วมน้อย (Least common multiplier)
gcd	คือ	ค่าหารร่วมมาก (Greatest common divisor)
$Overhead_{tick}$	คือ	โอเวอร์เฮดของ Tick interrupt
$Overhead_{task}$	คือ	โอเวอร์เฮดของ Task interrupt
t_r	คือ	ค่าเวลาเริ่มต้นการทำงาน (Task released time)

บทที่ 2

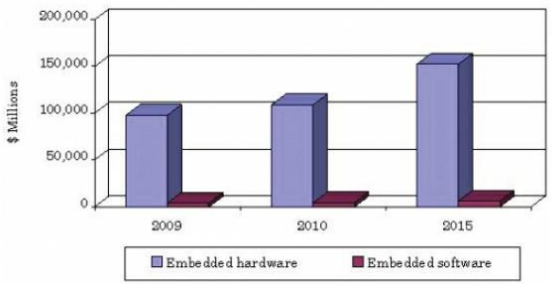
ปริทัศน์เอกสารข้อมูล

สถาปัตยกรรมการกระตุ้นการทำงานด้วยเวลาถูกนำมาประยุกต์ใช้สำหรับงานระบบสมองกลฝังตัวอย่างแพร่หลาย เนื่องจากสถาปัตยกรรมดังกล่าวสามารถทำงานในระบบเวลาจริงได้อย่างมีประสิทธิภาพ โดยเฉพาะอย่างยิ่งสำหรับระบบการทำงานที่เกี่ยวข้องกับด้านความปลอดภัย เช่น ระบบยานยนต์อัตโนมัติ อุปกรณ์การแพทย์ หรือระบบควบคุม เป็นต้น เนื้อหาในบทนี้ได้อธิบายโครงสร้างการทำงานของสถาปัตยกรรมดังกล่าวตลอดจนงานวิจัยต่างๆ ที่นำสถาปัตยกรรมนี้ไปประยุกต์สำหรับระบบสมองกลฝังตัว

2.1 สถาปัตยกรรมในการพัฒนาระบบสมองกลฝังตัว

ทุกวันนี้ระบบสมองกลฝังตัวถูกนำมาประยุกต์ใช้งานอย่างแพร่หลายในชีวิตประจำวัน ปริมาณของระบบสมองกลฝังตัวมีการเติบโตอย่างรวดเร็วและถือเป็นตลาดที่มีการเติบโตสูงมากเป็นระดับต้นๆ ของโลก ที่จริงแล้วระบบสมองกลฝังตัวก็คือคอมพิวเตอร์ขนาดเล็กที่ประกอบด้วย หน่วยประมวลผล อุปกรณ์ต่อพ่วง และโปรแกรมที่ใช้สำหรับจุดประสงค์เฉพาะอย่าง โดยระบบสมองกลฝังตัวจะเป็นการทำงานร่วมกันระหว่างฮาร์ดแวร์และซอฟแวร์ โดยมีหน้าที่ทำงานตามฟังก์ชันที่ถูกออกแบบ ระบบสมองกลฝังตัวมีความแตกต่างกับคอมพิวเตอร์ใช้งานทั่วไปคือ ระบบสมองกลฝังตัวถูกออกแบบมาสำหรับการใช้งานเพื่อวัตถุประสงค์ในการประยุกต์ใช้งานเฉพาะด้านอย่างใดอย่างหนึ่ง ปัจจุบันเราสามารถพบอุปกรณ์ฝังตัวได้ทั่วไป เช่น อุปกรณ์เครื่องใช้ไฟฟ้าภายในบ้าน โทรศัพท์มือถือ กล้องดิจิตอล เครื่องเล่นดีวีดี เครื่องนำทางรถยนต์ ฯลฯ

สำหรับตลาดด้านสมองกลฝังตัวในปัจจุบันมีการเติบโตอย่างสูงมาก จากผลการวิจัยการคาดการณ์ด้านการตลาดของ BCC (BCC Research Market Forecasting) [1] แสดงดังภาพประกอบ 2.1 คาดการณ์ว่าการเติบโตของตลาดโลกของระบบสมองกลฝังตัวด้านฮาร์ดแวร์และซอฟแวร์ ตั้งแต่ปี ค.ศ. 2009-2015 เพิ่มขึ้นปีละประมาณ 7% โดยในปี ค.ศ. 2015 คาดว่าการเติบโตของตลาดโลกทั้งหมดประมาณ 158.6 พันล้านดอลลาร์ ($158.6 billion)



ภาพประกอบ 2.1 การคาดการณ์ด้านการตลาดของเทคโนโลยีระบบสมองกลฝังตัวของ BCC Research ตั้งแต่ปี ค.ศ. 2009-2015

ในส่วนของการพัฒนาซอฟแวร์สำหรับงานด้านสมองกลฝังตัวนั้น โดยส่วนใหญ่โปรแกรมใช้งานจะออกแบบเพื่อรองรับการทำงานแบบเวลาจริง (Real-time) ซึ่งมีความสามารถในการจัดการทรัพยากรระบบเพื่อให้ตอบสนองตามเงื่อนไขเวลาที่กำหนด อย่างไรก็ตามสำหรับงานด้านสถาปัตยกรรมในการพัฒนาซอฟแวร์ระบบสมองกลฝังตัว หากอาศัยวิธีการกระตุ้นการทำงาน (Triggering mechanism) ในการทำงาน เราสามารถแยกสถาปัตยกรรมซอฟแวร์ได้สองวิธี ได้แก่

1. สถาปัตยกรรมการกระตุ้นการทำงานด้วยเหตุการณ์

สถาปัตยกรรมแบบ ET หรือแบบออนไลน์จะทำงานเมื่อมีเหตุการณ์ใดๆ เกิดขึ้น ซึ่งจะเกิดขึ้น ณ เวลาใดก็ได้ ส่วนใหญ่ซอฟแวร์จะตอบสนองการทำงานโดยใช้เทคนิคการอินเตอร์รัพท์ สถาปัตยกรรมแบบนี้ส่วนใหญ่จะถูกนำไปประยุกต์ใช้สำหรับงานที่ต้องการการตอบสนองด้วยเวลาอันรวดเร็ว มีความยืดหยุ่นสูง และสามารถทำงานกับระบบที่มีการเปลี่ยนแปลงได้ดี ซึ่งคอมพิวเตอร์ใช้งานทั่วไปส่วนใหญ่อาศัยสถาปัตยกรรมแบบ ET ในการทำงาน สาเหตุนี้เองที่ทำให้สถาปัตยกรรมนี้มีความต้องการทรัพยากรสูงทั้งด้านฮาร์ดแวร์ ได้แก่ หน่วยประมวลผล หน่วยความจำ ฯลฯ และซอฟแวร์ระบบปฏิบัติการที่ต้องรองรับการทำงานแบบเวลาจริง

อย่างไรก็ตามระบบสมองกลฝังตัวส่วนใหญ่จะถูกออกแบบมาทำงานเฉพาะด้านและใช้ทรัพยากรอย่างคุ้มค่าเพื่อตอบสนองความต้องการด้านราคาของผู้บริโภค ดังนั้นจะเห็นได้ว่าสถาปัตยกรรมดังกล่าวมีข้อจำกัดในการนำไปประยุกต์ใช้งานกับระบบสมองกลฝังตัวที่มีทรัพยากรจำกัดด้วยเหตุผลดังกล่าว งานวิจัยของ Pont [2] ได้สรุปว่าภายใต้สถาปัตยกรรม ET ที่นำไปใช้กับระบบสมองกลฝังตัวที่มีทรัพยากรจำกัด สถาปัตยกรรมดังกล่าวอาจลดความสามารถในการคาดการณ์และลดความน่าเชื่อถือของระบบได้

2. สถาปัตยกรรมการกระตุ้นการทำงานด้วยเวลา [3, 4]

สถาปัตยกรรมแบบ TT หรือแบบออฟไลน์ [3-5] ระบบจะถูกกระตุ้นให้ทำงานโดยอาศัยการทำงานตามตารางเป็นรายคาบซึ่งการทำงานจะแบ่งเป็นสองส่วนได้แก่ การออกแบบตารางเวลาและการดำเนินการจริง โดยในการออกแบบตารางการทำงานนั้นจะต้องรู้การทำงานของระบบล่วงหน้าเพื่อจะได้ออกแบบระบบให้มีประสิทธิภาพและมีการทำงานอย่างแม่นยำ จากงานวิจัยที่ผ่านมาพบว่าสถาปัตยกรรมดังกล่าวมีความเหมาะสมที่จะนำไปใช้สำหรับระบบที่ต้องการการคาดการณ์และความน่าเชื่อถือสูงและยังสามารถนำไปประยุกต์ใช้งานกับระบบที่ต้องการความปลอดภัยสูง เช่น ระบบยานยนต์อัตโนมัติ อุปกรณ์การแพทย์ หรือระบบควบคุม [6-8]

2.2 การจัดการตารางงานในระบบเวลาจริง

ในงานวิจัยนี้สนใจการออกแบบการจัดการตารางงานสำหรับซอฟแวร์สถาปัตยกรรมการกระตุ้นการทำงานด้วยเวลาซึ่งในการออกแบบระบบจัดตารางการทำงานสำหรับระบบสมองกลฝังตัวนั้นสิ่งสำคัญคือต้องเข้าใจคุณลักษณะของงานที่นำมาจัดตารางซึ่งมีลักษณะดังนี้

2.2.1 คุณลักษณะของงาน (Task characteristics)

จากงานวิจัยที่ผ่านมา ได้มีการให้ความหมายของงานในระบบการทำงานแบบเวลาจริงดังนี้ Liu [9] ให้ความหมายว่างาน (Task) คือกลุ่มของชิ้นงาน (Job) ที่มีความสัมพันธ์กันซึ่งถูกเตรียมไว้สำหรับทำหน้าที่บางอย่างภายในระบบ โดยชิ้นงานก็คือหน่วยของงานที่ถูกจัดเวลาและทำงานโดยระบบ

นั่นเอง นอกจากนี้ Cottet [5] ได้ให้ความหมายของงานแบบเวลาจริงว่าคือรูปแบบการทำงานเบื้องต้นที่ถูกจัดตารางงาน ซึ่งอาจจะเป็นแบบรายคาบ (Periodic task) หรือไม่เป็นรายคาบ (Aperiodic task) และอาจมีข้อจำกัดด้านฮาร์ดเรียลไทม์หรือซอฟต์เรียลไทม์

โดยทั่วไปโมเดลของงานจะแสดงในรูปพารามิเตอร์ด้านเวลาเป็นสำคัญ โดยงานในระบบเวลาจริงประกอบด้วยกลุ่มของพารามิเตอร์แสดงดังภาพประกอบ 2.2 จากไดอะแกรมเวลาดังกล่าวสามารถสรุปพารามิเตอร์ต่างๆ ได้ดังนี้

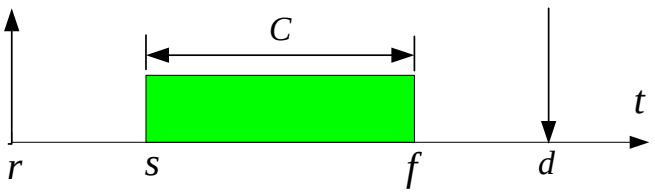
1) เวลาการกระตุ้นการทำงาน (r) แสดงสัญลักษณ์เป็นลูกศรชี้ขึ้น

2) เวลาเริ่มต้นการทำงาน (s)

3) เวลาสิ้นสุดการทำงาน (f)

4) เวลาในการดำเนินงาน (C)

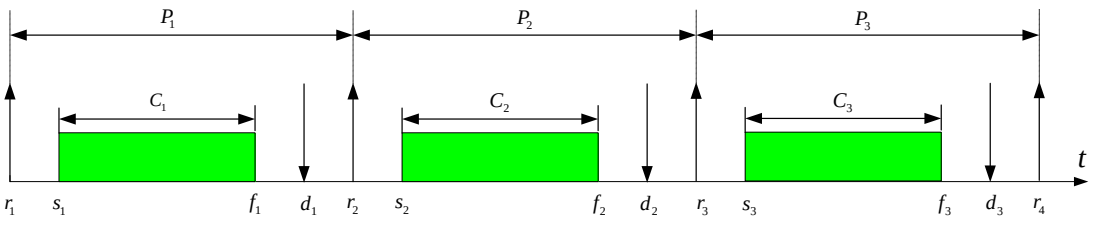
5) เวลามากที่สุดที่ยอมให้งานดำเนินงานจนเสร็จสิ้น (d) แสดงสัญลักษณ์เป็นลูกศรชี้ลง



ภาพประกอบ 2.2 โมเดลของงานในระบบเวลาจริง [10]

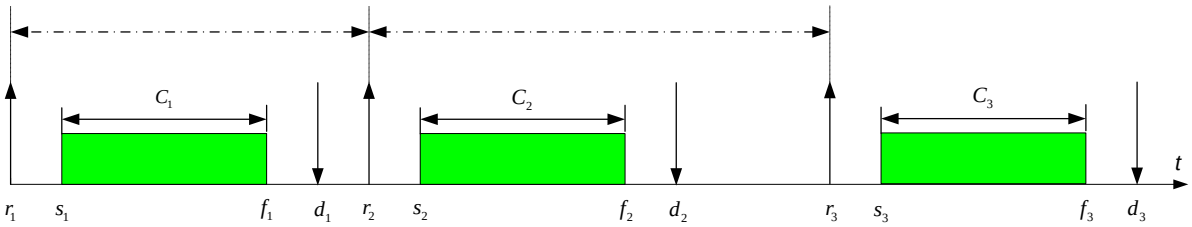
2.2.2 ชนิดของงาน

1) งานแบบรายคาบ โดยโมเดลของงานแบบรายคาบแสดงได้ดังภาพประกอบ 2.3 จะเห็นได้ว่างานแบบรายคาบสามารถกำหนดคาบเวลาได้ โดยจากภาพค่าคาบเวลา P_1, P_2 และ P_3 จะมีคาบเวลาเท่ากัน



ภาพประกอบ 2.3 โมเดลงานแบบรายคาบ [10]

2) งานแบบไม่เป็นรายคาบ โดยโมเดลของงานแบบไม่เป็นรายคาบ แสดงได้ดังภาพประกอบ 2.4 จะเห็นได้ว่างานแบบไม่เป็นรายคาบ จะมีคาบเวลาไม่คงที่และไม่สามารถคาดเดาพฤติกรรมด้านเวลาของงานที่เกิดขึ้นได้



ภาพประกอบ 2.4 โมเดลของงานแบบไม่เป็นรายคาบ [10]

2.2.3 ระบบการทำงานแบบเวลาจริง

ระบบการทำงานแบบเวลาจริงสามารถแบ่งได้เป็น 2 ประเภทหลักได้แก่ ฮาร์ดเรียลไทม์และซอฟต์เรียลไทม์ สามารถอธิบายได้ดังนี้

1) ระบบฮาร์ดเรียลไทม์

เป็นระบบที่สามารถทำงานใดงานหนึ่งให้เสร็จตามเวลาที่กำหนดได้ ซึ่งงานที่จะรับเข้ามาดำเนินงานนั้นจะมีความต้องการเวลาในการทำงาน ซึ่งหากการทำงานผิดเงื่อนไขเวลาแล้วจะส่งผลเสียหายอย่างร้ายแรงกับระบบและอาจเกิดการสูญเสียและเป็นอันตรายต่อชีวิตได้ ดังนั้นระบบสมองกลฝังตัวที่ใช้ในเครื่องมือหรืออุปกรณ์ประเภทที่ระบบเกิดความล้มเหลวแล้วก่อให้เกิดอันตรายได้ จำเป็นต้องใช้ระบบการทำงานแบบฮาร์ดเรียลไทม์ ตัวอย่างเช่น อุปกรณ์ที่ใช้ในอากาศยานหรืออวกาศ อุปกรณ์ทางทหาร เครื่องมือแพทย์ ระบบความปลอดภัยในรถยนต์ เป็นต้น

2) ระบบซอฟต์เรียลไทม์

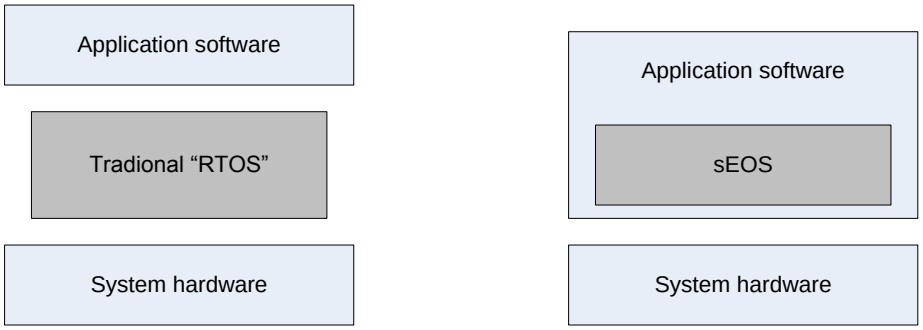
คือระบบที่ข้อจำกัดต่างๆ ไม่เข้มงวดเท่าระบบฮาร์ดเรียลไทม์ อาจเป็นระบบแบ่งเวลาที่มีการให้ระดับความสำคัญสำหรับงานบางประเภทหรืองานที่ถูกเลือกไว้ล่วงหน้าว่าเป็นงานเร่งด่วน ซึ่งอาจก่อให้เกิดปัญหาของการทำงานในระดับต่ำและไม่เกิดความเสียหายอย่างรุนแรงแม้ว่าระบบจะทำงานผิดพลาด

2.3 ระบบปฏิบัติการสำหรับระบบสมองกลฝังตัวที่มีทรัพยากรจำกัด

โดยทั่วไประบบปฏิบัติการแบบเวลาจริงจะประกอบด้วยเคอร์เนล (Kernel) ทำหน้าที่เป็นซอฟแวร์หลักในการจัดเตรียมการจัดตารางการทำงานและอัลกอริทึมในการจัดการทรัพยากรของระบบ โดยเคอร์เนลส่วนใหญ่จะประกอบด้วยส่วนสำคัญ 3 ส่วนได้แก่ ส่วนการจัดตารางการทำงาน ส่วนของออปเจ็ก (Objects) ที่ใช้สำหรับการสร้างแอปพลิเคชั่นและส่วนการให้บริการ [11]

ในส่วนของการใช้ระบบปฏิบัติการแบบเวลาจริงสำหรับระบบสมองกลฝังตัวที่มีทรัพยากรจำกัด มีบริษัทที่ผลิตซอฟแวร์ RTOS ในเชิงพาณิชย์ในท้องตลาดได้แก่ μITRON, μTKernal, μC-OS/II, EmbOS, Salvo, SharcOS, Ecos, KeilOS [12] โดยมีการนำ RTOS มาประยุกต์ใช้งานแสดงดังภาพประกอบ 2.5 (a) ระบบปฏิบัติการแบบเวลาจริงเหล่านี้มีข้อดีคือสามารถจัดการระบบงานแบบเวลาจริงได้ง่ายและมีประสิทธิภาพโดยที่ผู้ใช้งานไม่จำเป็นต้องมีความรู้ด้านระบบปฏิบัติการอย่างลึกซึ้ง อย่างไรก็ตามมีงานวิจัยจำนวนมาก [2, 13, 14] ไม่เห็นด้วยที่จะนำ RTOS เหล่านี้นำมาใช้สำหรับระบบสมองกลฝังตัวที่มีทรัพยากรจำกัดเนื่องจากระบบปฏิบัติการดังกล่าวต้องการทรัพยากร

ด้านหน่วยความจำและการคำนวณในปริมาณมาก นักวิจัยส่วนใหญ่เห็นด้วยกับการสร้างระบบปฏิบัติการแบบง่าย (In-house operating system) ซึ่งเหมาะสำหรับนำมาประยุกต์ใช้กับระบบฝังตัว ยกตัวอย่าง เช่น Pont [2] ได้นำเสนอระบบฝังตัวแบบง่าย (Simple Embedded Operating System: sEOS) แสดงดังภาพประกอบ 2.5 (b) โดยนำระบบดังกล่าวมาเป็นส่วนหนึ่งของซอฟแวร์ประยุกต์ นั่นคือ สถาปัตยกรรมแบบนี้ได้รวมระบบปฏิบัติการและส่วนการสร้างแอปพลิเคชั่นมาอยู่ร่วมกัน ซึ่งสถาปัตยกรรมแบบนี้มีข้อดีคือลดภาระโหลดของซีพียูทำให้เหมาะสำหรับระบบสมองกลฝังตัวที่มีทรัพยากรจำกัดและเป็นการง่ายในการสร้างระบบปฏิบัติการนี้ในอุปกรณ์ที่มีองค์ประกอบแตกต่างกัน



(a) (b)

ภาพประกอบ 2.5 สถาปัตยกรรม (a) RTOS และ (b) sEOS [2]

2.4 อัลกอริทึมการจัดตารางเวลางานสำหรับสถาปัตยกรรมการกระตุ้นการทำงานด้วยเวลา (Scheduling algorithm in TT system)

Cottet [5] ได้กล่าวว่าอัลกอริทึมการจัดตารางเวลาทำหน้าที่กำหนดงานไปยังโปรเซสเซอร์และจัดเตรียมการจัดลำดับของงาน ดังนั้นอัลกอริทึมการจัดตารางเวลาถือเป็นองค์ประกอบหลักในการกำหนดเส้นทางที่งานจะถูกดำเนินการโดยตารางการทำงาน อย่างไรก็ดีอัลกอริทึมการจัดตารางการทำงานมีวัตถุประสงค์ที่เน้นการใช้ประโยชน์ซีพียูให้เต็มประสิทธิภาพ เพื่อให้ผู้ใช้พอใจในการตอบสนองของระบบโดยรวม โดยต้องคำนึงถึงความต้องการทั่วไปดังนี้ [15]

1) จัดสรรซีพียูให้กับโปรเซสต่างๆ ด้วยความยุติธรรม
2) ได้ปริมาณงานสูงสุด (Maximum throughput)
3) ช่วยลดค่าใช้จ่ายในการใช้งานทรัพยากร
4) ช่วยจัดสรรทรัพยากรให้กับระบบอย่างเหมาะสม
5) กำหนดลำดับความสำคัญของทรัพยากรที่จำเป็นต่อระบบปฏิบัติการได้
6) สามารถคาดเดาการสิ้นสุดของแต่ละโปรเซสได้
7) ให้บริการที่ดีที่สุด

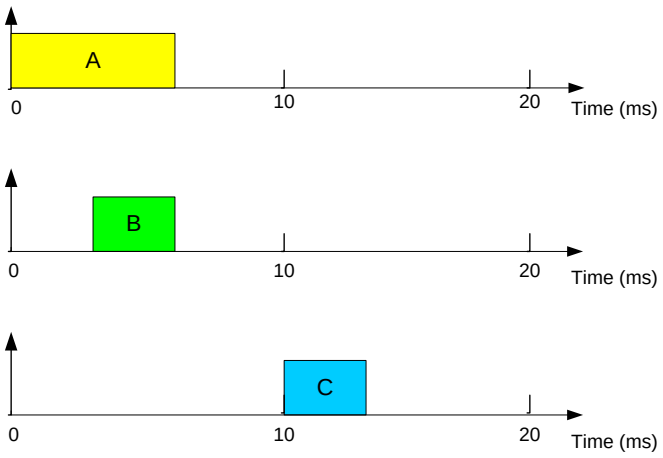
อย่างไรก็ตาม ในการกำหนดตารางเวลางานนั้น หากกำหนดลำดับการทำงานตามความสำคัญของงาน (Priority scheduling) สามารถกำหนดการจัดตารางการทำงานได้ดังนี้

2.4.1 การจัดตารางงานแบบความสำคัญคงที่ (Fixed priority scheduler)

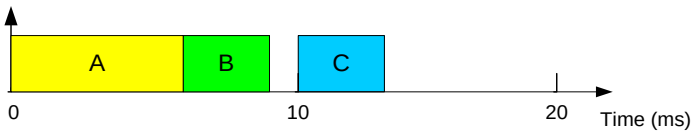
การจัดตารางงานแบบความสำคัญคงที่ถูกใช้มากที่สุดในระบบคอมพิวเตอร์ฝังตัวที่ทำงานแบบเวลาจริง [16] ลำดับความสำคัญของแต่ละงานได้ถูกกำหนดไว้ก่อนที่ระบบจะเริ่มทำงาน โดยที่ลำดับความสำคัญของงานจะไม่เปลี่ยนแปลงตลอดช่วงอายุการทำงานของระบบ ตัวอย่างของการจัดตารางงานแสดงดังตารางที่ 2.1 และภาพประกอบ 2.6 สมมุติว่าต้องการจัดตารางการทำงานทั้ง 3 งาน ได้แก่ Task A, Task B และ Task C ที่มีค่าเวลาสูงสุดในการดำเนินงาน (Worst case execution time: WCET) เท่ากับ 6 ms, 3 ms และ 3 ms ตามลำดับ โดยกำหนดให้ Task A มีลำดับความสำคัญสูงสุดและ Task C มีลำดับความสำคัญต่ำสุด จากการจัดตารางการทำงานโดยวิธีนี้จะเห็นได้ว่า Task A จะทำงานจนเสร็จก่อนจากนั้น Task B จึงสามารถทำงานได้ ซึ่งผลการจัดตารางแสดงได้ดังภาพประกอบที่ 2.7 ตัวอย่างของอัลกอริทึมตารางงานแบบความสำคัญคงที่ที่ได้รับความนิยมได้แก่ Rate monotonic (RM) และ Deadline monotonic (DM) [7]

ตาราง 2.1 คุณลักษณะของงานที่ต้องการจัดตารางการทำงาน

Tasks	*WCET* (ms)	Period (ms)
A	6	20
B	3	20
C	3	20



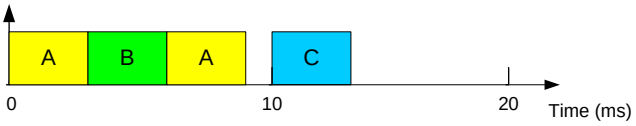
ภาพประกอบ 2.6 ตารางเวลาการทำงานของงานที่ต้องการจัดตารางการทำงาน



ภาพประกอบ 2.7 ตัวอย่างตารางเวลาการจัดตารางงานแบบความสำคัญคงที่

2.4.2 การจัดตารางงานแบบความสำคัญเปลี่ยนแปลงได้ (Dynamic priority scheduler)

สำหรับการจัดตารางงานแบบความสำคัญเปลี่ยนแปลงได้นั้น ลำดับความสำคัญของงานสามารถเปลี่ยนแปลงได้ในขณะที่ซีพียูกำลังดำเนินการ โดยงานที่มีความสำคัญสูงกว่า (Higher priority) สามารถแทรก (Pre-empt) และทำงานแทนงานที่มีความสำคัญต่ำกว่าขณะที่กำลังทำงานได้ โดยงานที่ถูกขัดจังหวะต้องรอที่จะกลับไปทำงานในโอกาสต่อไป ดังนั้นจากคุณลักษณะของงานจากตารางที่ 2.1 หาก Task B ถูกกำหนดให้มีความสำคัญสูงกว่า Task A เมื่อถึงเวลาที่ Task B พร้อมทำงานระบบจะอนุญาตให้ Task B สามารถแทรกการทำงานของ Task A และดำเนินการทำงานจนเสร็จสิ้นจากนั้น Task A จึงสามารถดำเนินการต่อไปได้ แสดงการทำงานดังภาพประกอบที่ 2.8 ตัวอย่างของอัลกอริทึมการจัดตารางงานแบบความสำคัญเปลี่ยนแปลงได้ที่ได้รับความนิยมได้แก่ Earliest-Deadline-First (EDF) และ Least-Laxity-First (LLF) [7]



ภาพประกอบ 2.8 ตัวอย่างตารางเวลาการจัดตารางงานแบบเปลี่ยนแปลงได้

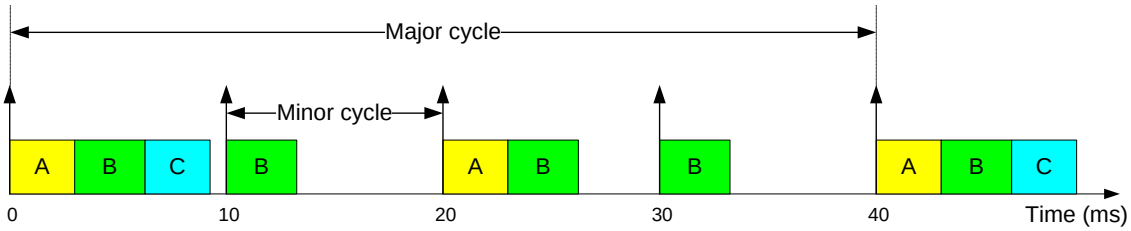
2.5 การบริหารแบบวนรอบ (Cyclic executive)

ในการออกแบบระบบการกระตุ้นการทำงานด้วยเวลาเพื่อไปประยุกต์ใช้กับระบบฝังตัวที่มีทรัพยากรจำกัด เช่นไมโครคอนโทรลเลอร์ สถาปัตยกรรมซอฟแวร์ชนิดกระตุ้นการทำงานด้วยเวลาแบบ TTC จัดเป็นระบบการจัดการเวลาการทำงานที่ได้รับความนิยมเป็นอย่างมาก ซึ่งระบบการจัดการเวลาดังกล่าวได้อาศัยอัลกอริทึมการบริหารแบบวนรอบ [17-20] เป็นแนวทางในการสร้างระบบ TTC ขึ้นมาใช้งานนั่นเอง การบริหารแบบวนรอบมีข้อดีคือสามารถสร้างระบบได้ง่ายโดยไม่ต้องอาศัยระบบปฏิบัติการแบบเวลาจริงและมีความแม่นยำในการทำงานสูง จากงานวิจัยของ Laplant [21] กล่าวว่าการบริหารงานแบบวนรอบเกี่ยวข้องกับการจัดตารางเวลาแบบลำดับที่สามารถกำหนดการทำงานได้โดยการดำเนินงานตามตารางเวลาที่ออกแบบไว้เรียบร้อยแล้ว เช่นเดียวกับงานวิจัยของ Barker และ Shaw [17] ได้ระบุว่าการบริหารแบบวนรอบเป็นการนำลำดับงานแบบมีคาบเวลามาดำเนินงานแบบวนรอบ โดยระบบงานจะถูกออกแบบไว้ก่อนที่การทำงานจริงจะเริ่มต้น นั่นคือระบบการบริหารแบบวนรอบจะมีรูปแบบการทำงานแบบออฟไลน์

โครงสร้างโดยทั่วไปของการบริหารแบบวนรอบจะประกอบด้วยรอบการทำงานที่สำคัญ 2 ส่วน ได้แก่ เมเจอร์ไซเคิล (Major cycle) และไมเนอร์ไซเคิล (Minor cycle) เมเจอร์ไซเคิลก็คือรอบหรือคาบเวลาที่เกิดซ้ำทุกช่วงเวลา Laplant [21] ให้ความหมายว่าเมเจอร์ไซเคิลคือเวลาที่น้อยที่สุดในการดำเนินการของงานไปยังส่วนประมวลผลเพื่อให้แน่ใจว่าค่าเดดไลน์ (Deadline) และคาบเวลา (Period) ของทุกโปรเซสสามารถดำเนินการได้ทันเวลา ค่าเวลาของเมเจอร์ไซเคิลก็คือค่าคูณร่วมน้อย (Least common multiplier: LCM) ของคาบเวลาของงานที่นำมาจัดตารางนั่นเอง ส่วนไมเนอร์ไซเคิลหรือ Tick interval คือค่าเวลาของเมเจอร์ไซเคิลที่ถูกหารให้มีขนาดคาบเวลาที่เล็กลง ซึ่งค่าเวลาของเมเจอร์ ไซเคิลสามารถคำนวณได้จากค่าหารร่วมมาก (Greatest common divisor: GCD) ของคาบเวลาของงานนั่นเอง ดังนั้นจากตัวอย่างข้อมูลสำหรับระบบการบริหารแบบวนรอบในตารางที่ 2.2 สามารถคำนวณหาค่าเมเจอร์ไซเคิลและไมเนอร์ไซเคิลของ Task A, Task B และ Task C มีค่าเท่ากับ 40 ms และ 10 ms ตามลำดับ ซึ่งหากกำหนดให้ Task A มีลำดับความสำคัญสูงสุดและ Task C มีลำดับความสำคัญต่ำสุดจะได้รูปแบบของการบริหารแบบวนรอบแสดงดังภาพประกอบ 2.9

ตาราง 2.2 ตัวอย่างคุณลักษณะงานระบบการบริหารแบบวนรอบ

Task	*WCET* (ms)	Deadline (ms)	Period (ms)
A	3	20	20
B	3	10	10
C	3	40	40



ภาพประกอบ 2.9 รูปแบบของการบริหารแบบวนรอบที่ได้จากคุณลักษณะงานจากตาราง 2.2

จากงานวิจัยที่ผ่านมาได้มีการนำอัลกอริทึมการบริหารเวลาแบบวนรอบมาประยุกต์ใช้งานการจัดสร้างระบบการจัดการเวลาการทำงานแบบ TTC บนระบบสมองกลฝังตัวที่มีทรัพยากรจำกัดดังนี้

2.5.1 การจัดเวลาการทำงานชนิดกระตุ้นด้วยเวลาแบบซุปเปอร์ลูป (Time-Triggered Co-operative-Super Loop: TTC-SL)

การจัดเวลาการทำงานชนิดนี้เป็นวิธีการสร้างระบบ TTC แบบไม่ซับซ้อน [2] โดยมีวิธีการดังนี้

1) กำหนดค่าพารามิเตอร์ของงานซึ่งประกอบด้วย Period และ *WCET*

2) เลือกฟังก์ชันดีเลย์ที่เหมาะสมและสอดคล้องกับคาบเวลาในการทำงาน

3) สร้างโปรแกรมแบบซุปเปอร์ลูปเพื่อให้มีการทำงานแบบวนรอบซึ่งจะประกอบด้วยงานและการเรียกใช้ฟังก์ชันดีเลย์ตามความต้องการ

ตัวอย่างการจัดสร้างระบบ TTC-SL แสดงได้ดังภาพประกอบในโปรแกรม 2.10 สมมุติว่าต้องการจัดตารางการทำงานที่ประกอบด้วย Task A, Task B และ Task C โดยทั้งสามงานมีค่า *WCET* และ Period เท่ากับ 4 ms และ 10 ms ตามลำดับ ดังนั้นในการจัดเวลาการทำงานชนิดนี้ค่าฟังก์ชันดีเลย์ที่เหมาะสมควรมีค่าเท่ากับ 6 ms จะเห็นได้ว่าการสร้างระบบ TTC-SL สามารถเข้าใจได้ง่ายและการจัดสร้างระบบไม่ยุ่งยากนอกจากนี้ยังต้องการทรัพยากรในการทำงานต่ำ อย่างไรก็ตามระบบนี้ยังมีข้อจำกัดเรื่องความน่าเชื่อถือในด้านเวลาที่ขาดความแม่นยำในการทำงานเนื่องจากหากมีงานหรือการหน่วงเวลาในส่วนใดเกิดทำงานผิดพลาดจะทำให้คาบเวลาในการดำเนินงานผิดพลาดทั้งระบบ นอกจากนี้จะเห็นได้ว่าซีพียูทำงานอยู่ตลอดเวลาทำให้กำลังงานรวมของระบบเพิ่มสูงขึ้นตามไปด้วย

```
Int main (void)
{
…
while (1)
        {
        Task_A ( ) ;
        Delay_6ms( );
        Task_B ( ) ;
        Delay_6ms( );
        Task_C ( ) ;
        Delay_6ms( );
        }
// Should never reach here
}
```

ภาพประกอบ 2.10 ตัวอย่างโปรแกรมระบบ TTC-SL

2.5.2 การจัดเวลาการทำงานชนิดกระตุ้นด้วยเวลาแบบไอเอสอาร์ (Time-Triggered Co-operative-Interrupt Service Routine: TTC-ISR) [2]

การจัดตารางเวลาแบบ TTC-ISR จะจัดเวลาการทำงานโดยใช้วิธีการให้บริการอินเตอร์รัพต์ ซึ่งจะถูกเรียกทำงานเมื่อเกิดโอเวอร์โฟลว์จากไทเมอร์ โดยเมื่อมีการอินเตอร์รัพต์เกิดขึ้นโปรแกรมก็จะเก็บค่าข้อมูลที่สำคัญของงานที่กำลังทำอยู่แล้วกระโดดไปทำงานตามตำแหน่งตามที่ระบุสำหรับการอินเตอร์รัพต์นั้น (Interrupt Vector Table: IVT) โดยโปรแกรมควบคุมจะเขียนโปรแกรมในลักษณะเป็นการประมวลผลด้านฟอร์กราวด์ (Forground processing) และแบ็คกราวด์ (Background processing) ตัวอย่างการทำงานของ TTC-ISR แสดงดังภาพประกอบโปรแกรมที่ 2.11 และไดอะแกรมการทำงานที่ 2.12 ตามลำดับ สมมุติว่า Task A, Task B และ Task C มีค่า WCET เท่ากันคือ 4 ms ค่าเมเจอร์ไซเคิลและไมเนอร์ไซเคิลเท่ากับ 10 ms และ 30 ms ตามลำดับ ดังนั้นระบบจะกำหนดให้ไทเมอร์สร้างสัญญาณอินเตอร์รัพต์ทุกๆ 10 ms นั่นหมายความว่าไทเมอร์จะถูกกำหนดให้เกิดโอเวอร์โฟลว์หรือเกิดการ Tick ที่มีค่าคาบเวลาเท่ากับ 10 ms โดยเมื่อเกิด

การอินเตอร์รัพต์ขึ้นระบบจะเรียกฟังก์ชัน Update ในส่วนฟอร์กราวด์เพื่อทำการดำเนินงาน Task A, Task B และ Task C ตามเงื่อนไขโปรแกรมและเมื่อทำงานเสร็จแล้วจะกลับเข้าไปสู่โหมดประหยัดพลังงาน (Idle mode) เพื่อลดค่าการใช้พลังงานของระบบในส่วนของแบ็คกราวด์ต่อไป

จะเห็นว่าการจัดตารางเวลาแบบนี้เป็นวิธีการที่ง่ายและสามารถจัดสร้างได้อย่างมีประสิทธิภาพเพราะว่าสามารถกำหนดคาบเวลาการทำงาน (Tick interval) ได้อย่างแม่นยำและยังสามารถลดค่าการใช้พลังงานจากโหมดประหยัดพลังงานได้อย่างมีประสิทธิภาพ อย่างไรก็ตาม เนื่องจากฟังก์ชัน Update ไม่ได้ทำการแยกส่วนการทำงานระหว่างส่วนควบคุมการจัดการเวลาและส่วนของงานแต่ละงานออกจากกัน ซึ่งหากระบบดำเนินงานจำนวนหลายงานที่มีคาบเวลาไม่เท่ากันจะเป็นการยากที่จะดีบักหรือตรวจสอบการทำงานของระบบ [22] นอกจากนี้ในกรณีเกิดปัญหาทากส์โอเวอร์รันขณะที่งานกำลังดำเนินการอาจทำให้ระบบละเลยการเกิดสัญญาณอินเตอร์รัพต์ซึ่งจะทำให้ระบบไม่สามารถดำเนินการต่อไปได้

```
BACKGROUND
PROCESSING

while (1)
{
Go_to_Sleep () ;
}
```

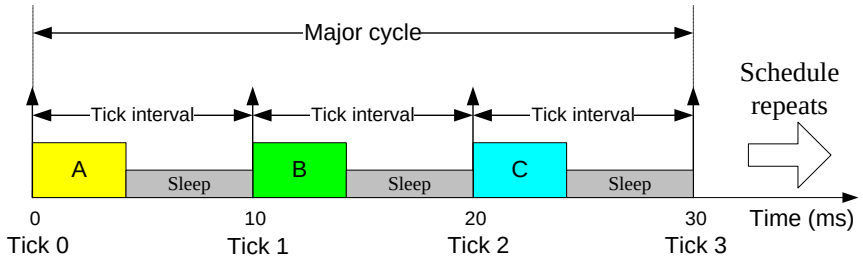
```
FOREGROUND
PROCESSING

                                  10 ms timer

void Update (void)
{
Tick_G++ ;

        switch (Tick_G)
                {
                case 1:
                  Task_A ( ) ;
                  break ;
                case 2:
                  Task_B ( ) ;
                  break ;
                case 3:
                  Task_C ( ) ;
                  Tick_G = 0 ;
                }
}
```

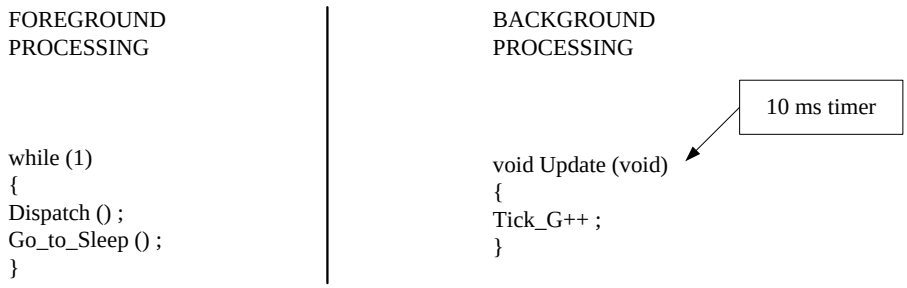
ภาพประกอบ 2.11 ตัวอย่างการทำงานของ TTC-ISR



ภาพประกอบ 2.12 ไดอะแกรมแสดงการทำงานของการจัดเวลาแบบ TTC-ISR

2.5.3 การจัดเวลาการทำงานชนิดกระตุ้นด้วยเวลาแบบโปรแกรมเลือกจ่ายงาน (TTC-Dispatch)

การจัดตารางเวลาแบบกระตุ้นด้วยเวลาแบบโปรแกรมเลือกจ่ายงานจะมีการทำงานคล้ายกับรูปแบบ TTC-ISR แต่ว่าได้ทำการแยกส่วนระหว่างส่วนควบคุมการจัดการเวลาและส่วนของโปรแกรมประยุกต์ออกจากกันเพื่อลดปัญหาของรูปแบบ TTC-ISR ที่กล่าวมาข้างต้น การทำงานของ TTC-Dispatch แสดงได้ดังตัวอย่างภาพประกอบที่ 2.13 การทำงานจะอาศัยการทำหน้าที่หลัก 2 ฟังก์ชัน ได้แก่ฟังก์ชัน Update และฟังก์ชัน Dispatch โดยฟังก์ชัน Update ทำหน้าที่อัพเดทตารางเวลาทุกๆ คาบเวลาและฟังก์ชัน Dispatch ใช้สำหรับเลือกจ่ายงานที่พร้อมสำหรับการดำเนินการ จะเห็นว่าเมื่อเกิดการอินเตอร์รัพต์ขึ้น ส่วนที่ให้บริการอินเตอร์รัพต์จะเรียกฟังก์ชัน Update ที่อยู่ในส่วนแบ็คกราวด์เพื่อทำการกำหนดค่าแฟลกแสดงสถานะว่ามีการเกิดอินเตอร์รัพต์ขึ้น หลังจากนั้นฟังก์ชัน Dispatch จะถูกเรียกใช้งาน โดยฟังก์ชันนี้จะอยู่ในส่วนฟอร์กราวด์ทำหน้าที่จัดเรียงลำดับของงานและดำเนินการทำงานแต่ละงานจนเสร็จจากนั้นระบบกลับเข้าไปสู่โหมดประหยัดพลังงานเพื่อลดค่าการใช้พลังงานของระบบและรอการเกิดอินเตอร์รัพต์เพื่อทำงานในคาบเวลาถัดไปต่อไป จะเห็นได้ว่า TTC-Dispatch มีการทำงานคล้ายกับ TTC-ISR แต่ว่าได้ทำการแยกส่วนระหว่างส่วนควบคุมการจัดการเวลาและส่วนของโปรแกรมประยุกต์ออกจากกันทำให้สามารถลดข้อจำกัดของวิธีการ TTC-ISR ดังนั้นวิธีนี้จึงได้รับความนิยมในการนำไปพัฒนาระบบสมองกลฝังตัวโดยการบริหารแบบวนรอบในปัจจุบัน [22]



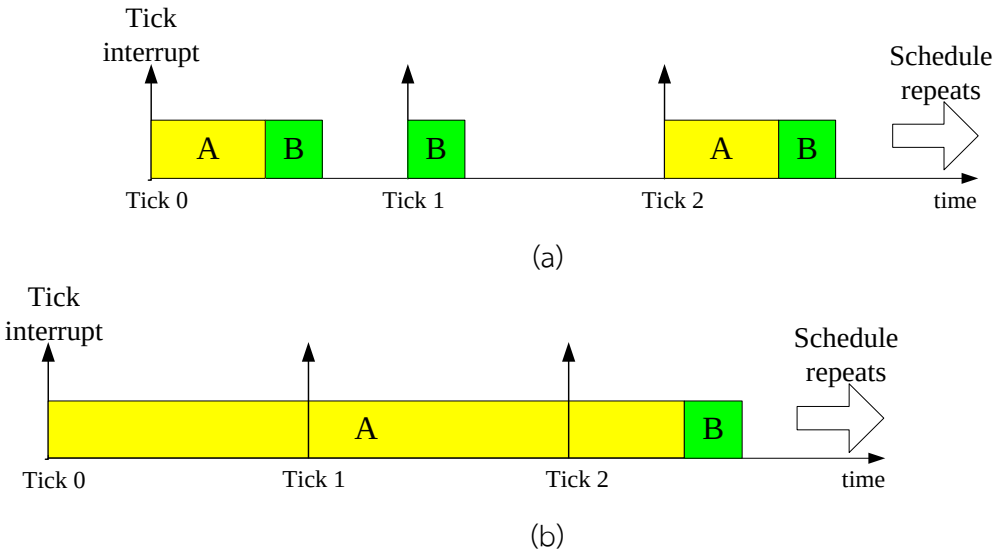
ภาพประกอบ 2.13 แสดงการทำงานของ TTC-Dispatch

2.6 ปัญหาที่สำคัญของระบบการจัดเวลาแบบ TTC

สำหรับระบบการกระตุ้นการทำงานด้วยเวลาแบบที่กล่าวมา นักพัฒนาด้านการออกแบบสถาปัตยกรรมจำนวนมาก [22-24] ได้กล่าวว่าการจัดเวลาการทำงานแบบ TTC สามารถออกแบบระบบปฏิบัติการได้ง่ายและมีความน่าเชื่อถือ ซึ่งหากมีการออกแบบและสร้างระบบได้อย่างเหมาะสม การจัดเวลาการทำงานรูปแบบนี้สามารถนำไปใช้งานกับระบบฮาร์ดเรียลไทม์ได้เป็นอย่างดี อย่างไรก็ตามถึงแม้ว่าระบบการจัดเวลาแบบ TTC จะมีประโยชน์เป็นอย่างยิ่ง แต่ถ้าหากมีการจัดเวลาการทำงานที่ไม่เหมาะสมจะทำให้เกิดการสูญเสียด้านประสิทธิภาพของระบบอย่างมาก ซึ่งปัญหาที่สำคัญของระบบการจัดเวลาแบบ TTC ได้แก่

2.6.1 ความเปราะบางในการออกแบบระบบการจัดเวลาการทำงาน

การจัดเวลาการทำงานนี้จะมีความเปราะบางมากในสถานการณ์ที่เกิดสภาวะโอเวอร์รันของงานซึ่งเกิดขึ้นในกรณีที่เวลาการทำงานจริงของงานใดๆ มากกว่าเวลาที่กำหนด เหตุการณ์ดังกล่าวสามารถสร้างผลกระทบแบบโดมิโนกับงานที่ดำเนินการต่อเนื่องกันมา ซึ่งปัญหานี้อาจทำให้ระบบไม่สามารถดำเนินการใดๆ ต่อไปได้ [24, 25] ตัวอย่างการเกิดโอเวอร์รันของงานในการจัดเวลาแบบ TTC-Dispatch แสดงดังภาพประกอบ 2.14 จะเห็นว่าภาพประกอบ 2.14 (a) จะเป็นการทำงานของระบบแบบปกติโดย Task A จะถูกดำเนินงานทุก ๆ 2 คาบ ในขณะที่ Task B จะดำเนินงานภายหลังจาก Task A ทำงานเสร็จสิ้นและจะถูกดำเนินงานทุก ๆ 1 คาบ อย่างไรก็ตามหาก Task A ใช้เวลาดำเนินงานมากกว่าค่าเวลาสูงสุดในการดำเนินงานของตัวเอง ลักษณะเช่นนี้จะเรียกว่าเกิดการโอเวอร์รันของ Task A ดังภาพประกอบ 2.14 (b) สถานการณ์นี้อาจทำให้ระบบไม่สามารถดำเนินการจัดตารางเวลาได้ตามที่ออกแบบไว้และอาจเกิดการสูญเสียกับระบบงานได้



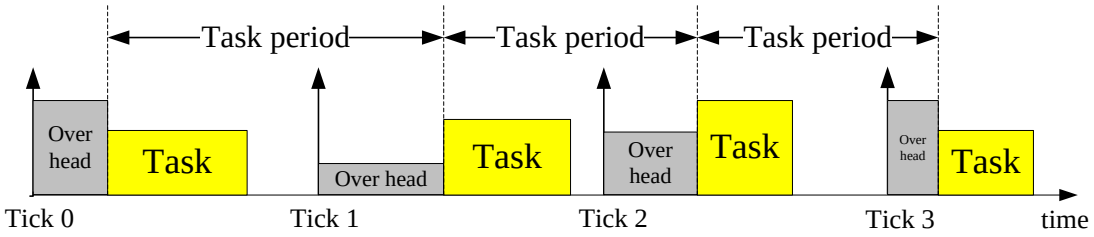
ภาพประกอบ 2.14 ตัวอย่างการเกิดโอเวอร์รันของงานในการจัดเวลาแบบ TTC-Dispatch

2.6.2 ค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน

ค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานของงานใดๆ อาจเกิดได้หลายสาเหตุ จากงานวิจัย [19, 25-27] ระบุว่าปัญหานี้อาจส่งผลกระทบโดยตรงต่อระบบงานที่เกี่ยวข้องด้านความปลอดภัย โดยเฉพาะอย่างยิ่งระบบงานแบบฮาร์ดเรียลไทม์และอาจเกิดความสูญเสียอย่างรุนแรงในการทำงานได้ สาเหตุสำคัญของความผิดพลาดทางเวลาเหล่านี้ได้แก่ความผิดพลาดที่เกิดจากโอเวอร์เฮดของการจัดตารางเวลา ความผิดพลาดที่เกิดจากจากการจัดลำดับการดำเนินการของงานหรือความผิดพลาดจากพฤติกรรมการอินเตอร์รัพต์ [24, 28, 29] ในที่นี้อธิบายสาเหตุหลักของค่าความผิดพลาดต่างๆ ดังนี้

1) ความผิดพลาดจากโอเวอร์เฮดของการจัดตารางเวลา (Scheduler overhead variation)

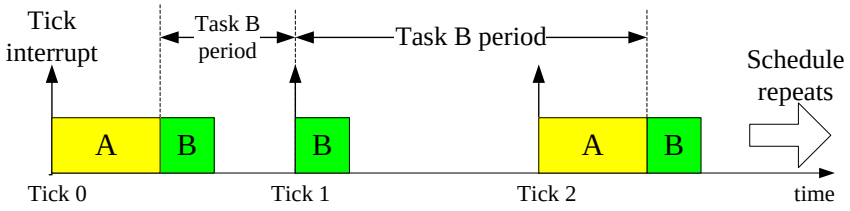
ค่าโอเวอร์เฮดสำหรับการจัดตารางการทำงานโดยทั่วไปจะเกิดขึ้นจากการสลับการทำงานของซีพียู (Context switching) สำหรับกรณีการจัดตารางการทำงานแบบ TTC-Dispatch ค่าโอเวอร์เฮดก็คือค่าเวลาที่เกิดขึ้นตั้งแต่ซีพียูให้บริการอินเตอร์รัพต์ การเรียกใช้งานฟังก์ชัน Update และตรวจสอบงานในตารางงาน จากงานวิจัยที่ผ่านมา [30, 31] จะพบว่ามีหลายปัจจัยที่ทำให้เกิดค่าความผิดพลาดจากโอเวอร์เฮดของการจัดตารางเวลาได้แก่เทคนิคการจัดการตารางงาน จำนวนของงานที่มีอยู่ในระบบหรือความเร็วของซีพียู เป็นต้น ตัวอย่างค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานจากโอเวอร์เฮดแสดงดังภาพประกอบ 2.15 จะเห็นว่าค่าโอเวอร์เฮดที่ความเร็วไม่เท่ากันจะส่งผลกระทบต่อคาบเวลาของงาน (Task period) ทำให้เกิดค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานได้



ภาพประกอบ 2.15 ค่าความผิดพลาดทางเวลาของการจัดเวลาที่เกิดจากจากโอเวอร์เฮด [28]

2) ความผิดพลาดจากการจัดลำดับการดำเนินการของงาน (Task placement variation)

การเกิดค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานของการจัดเวลาแบบ TTC ที่เกิดจากการจัดลำดับการทำงานแสดงดังตัวอย่างภาพประกอบ 2.16 จะเห็นได้ว่าที่เวลา Tick 0 และ Tick 1 เวลาเริ่มต้นการทำงานของ Task B ในแต่ละคาบเวลาไม่เท่ากัน ตัวอย่างเช่นที่เวลา Tick 0, Task B ต้องรอให้ Task A ดำเนินงานเสร็จสิ้นก่อนจึงสามารถดำเนินงานได้แต่ในขณะที่เวลา Tick 1, Task B สามารถดำเนินการได้ทันที ซึ่งกรณีเช่นนี้จะทำให้คาบเวลาของ Task B มีค่าไม่เท่ากัน



ภาพประกอบ 2.16 ค่าความผิดพลาดทางเวลาที่เกิดจากการจัดลำดับการดำเนินการของงาน

อย่างไรก็ตามจากงานวิจัยที่ผ่านมาได้มีการพัฒนาวิธีการเพื่อแก้ปัญหาดังกล่าว ดังนี้

1) วิธี Dynamic Voltage scaling (DVS) [28] ใช้วิธีการจัดเวลางานโดยอาศัยพื้นฐานการทำงานแบบ TTC-Dispatch มีวัตถุประสงค์ในการลดค่าพลังงานภายในระบบ โดยทำการปรับค่าแรงดันที่เหมาะสมในการทำงานเพื่อลดค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานของระบบ ถึงแม้ว่าวิธีการดังกล่าวช่วยลดค่าความผิดพลาดทางเวลาได้ดีกว่าแบบ TTC-Dispatch แต่เทคนิคที่ใช้มีความซับซ้อน ไม่ยืดหยุ่น และมีข้อจำกัดในการนำไปใช้งานกับระบบที่มีความต้องการการจัดตารางงานในปริมาณมาก

2) วิธี Task Guardian (TG) [24] ใช้วิธีการจัดเวลางานโดยอาศัยพื้นฐานการทำงานแบบ TTC-Dispatch เช่นเดียวกันกับวิธี DVS มีวัตถุประสงค์เพื่อแก้ปัญหาการเกิดโอเวอร์รันของงาน โดยระบบจะทำการสำรองข้อมูลการทำงานไว้ เมื่อใดก็ตามที่เกิดโอเวอร์รันของงานขึ้น วิธีการนี้จะทำการคืนค่าการทำงานเพื่อให้ระบบสามารถทำงานได้อย่างต่อเนื่อง อย่างไรก็ตามถึงแม้ว่าวิธีการดังกล่าวจะค่อนข้างยืดหยุ่นในการแก้ปัญหาการเกิดโอเวอร์รันของงาน แต่ยังคงส่งผลกระทบด้านความจุของโปรแกรมและหน่วยความจำ (Code and memory size) ซึ่งต้องใช้ทรัพยากรมากกว่าวิธีการ TTC-Dispatch ไม่น้อยกว่าสองเท่านอกจากนี้วิธีการนี้ยังเพิ่มเวลาในการจัดตารางงานของโหลดสูงขึ้น (Scheduler load) ซึ่งจะส่งผลกระทบต่อพลังงานของซีพียูรวมถึงค่าใช้จ่ายในการพัฒนาระบบที่เพิ่มขึ้นด้วย

จะเห็นได้ว่าจากงานวิจัยที่ผ่านมา วิธีการแก้ปัญหาที่เกิดขึ้นในการจัดตารางการทำงานแบบ TTC-Dispatch เหล่านี้ [24, 28] มีความซับซ้อนและตัวโครงสร้างระบบมีความต้องการใช้ทรัพยากรที่สูง ซึ่งในทางปฏิบัติอาจไม่มีความเหมาะสมที่จะนำไปประยุกต์ใช้สำหรับระบบฝังตัวที่มีทรัพยากรจำกัดได้ ดังนั้นงานวิจัยนี้จึงได้เสนอแนวคิดในการจัดการตารางการทำงานแบบใหม่ ซึ่งระบบการทำงานนี้สามารถดำเนินงานโดยการแยกการทำงานระหว่างส่วนควบคุมการจัดการเวลาและส่วนของงานแต่ละงานออกจากกัน โดยการทำงานอาศัยเทคนิคการอินเตอร์รัพต์แบบหลายไทเมอร์ [29] โดยเมื่อเกิดการโอเวอร์รันขึ้นในงานใดๆ ระบบดังกล่าวก็จะสามารถทำงานได้โดยไม่ส่งผลกระทบกับงานที่จะถูกดำเนินงานต่อมาเนื่องจากระบบได้กำหนดความสำคัญของการอินเตอร์รัพต์ไว้ก่อนล่วงหน้าว่าแต่ละงานจะถูกดำเนินการในเวลาใด ทำให้ระบบสามารถทำงานได้โดยปราศจากการเกิดทากส์โอเวอร์รัน นอกจากนี้ระบบยังสามารถลดค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานได้อีกด้วยหากมีการกำหนดค่าเวลาเริ่มต้นในการทำงานของแต่ละงานอย่างเหมาะสม ดังนั้นจากแนวคิดในการจัดการตารางการทำงานดังกล่าว งานวิจัยนี้จึงได้เสนอการจัดตารางการทำงานที่อาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์ซึ่งสามารถนำไปประยุกต์ใช้งานกับระบบสมองกลฝังตัวที่มีทรัพยากรจำกัดเพื่อให้มั่นใจว่าระบบสามารถลดผลกระทบของพฤติกรรมด้านเวลาในการทำงานจริงและทำงานได้อย่างมีประสิทธิภาพ

2.7 สรุป

การจัดเวลาการทำงานแบบ TTC สามารถออกแบบระบบปฏิบัติการได้ง่ายและมีความน่าเชื่อถือซึ่งเหมาะสำหรับประยุกต์ใช้งานกับระบบสมองกลฝังตัวที่มีทรัพยากรจำกัด ซึ่งในการออกแบบระบบดังกล่าวนักพัฒนาส่วนใหญ่นิยมใช้ระบบการจัดเวลาแบบ TTC-Dispatch อย่างไรก็ตามถึงแม้ว่าระบบดังกล่าวจะมีประโยชน์เป็นอย่างยิ่ง แต่ถ้าหากมีการจัดเวลาการทำงานไม่เหมาะสมจะทำให้เกิดการสูญเสียด้านประสิทธิภาพของระบบอย่างมากซึ่งปัญหาที่สำคัญนี้ได้แก่ความเปราะบางในการออกแบบระบบการจัดเวลาการทำงานและค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน เพื่อเป็นการแก้ปัญหาที่เกิดขึ้น งานวิจัยนี้ได้เสนอแนวคิดในการจัดการตารางการทำงานแบบใหม่โดยอาศัยเทคนิคการอินเตอร์รัพต์แบบหลายไทเมอร์ ซึ่งวิธีการแก้ปัญหาดังกล่าวจะได้อธิบายในบทถัดไป

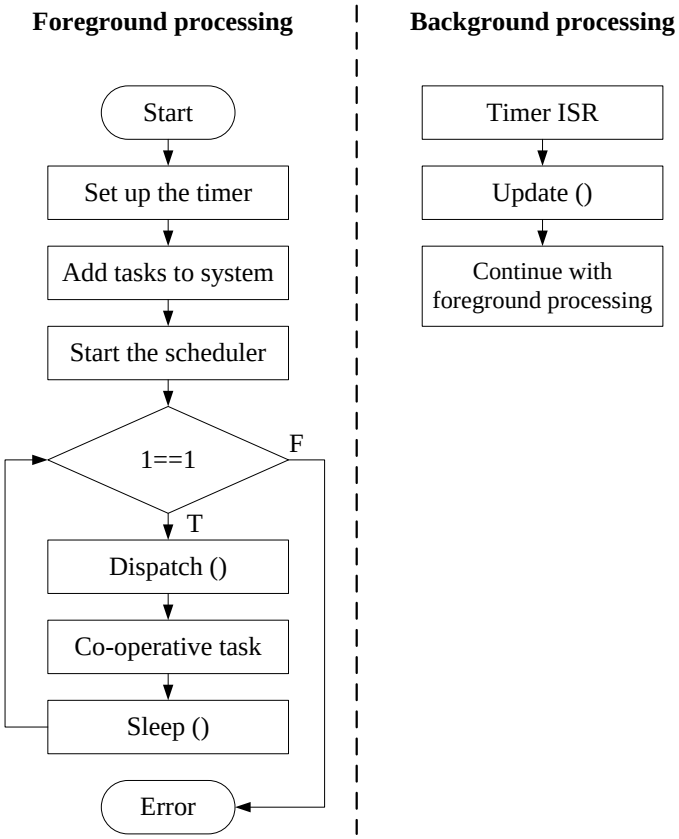
บทที่ 3

วิธีดำเนินการวิจัย

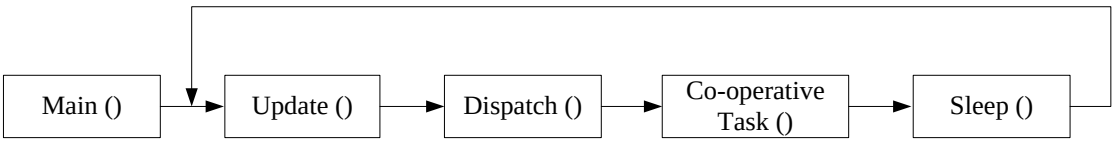
ในการออกแบบระบบการกระตุ้นการทำงานด้วยเวลาแบบ TTC เพื่อไปประยุกต์ใช้กับระบบฝังตัวที่มีทรัพยากรจำกัด นักพัฒนาส่วนใหญ่นิยมใช้ระบบการจัดเวลาชนิดกระตุ้นด้วยเวลาแบบโปรแกรมเลือกจ่ายงาน เพราะว่าสามารถออกแบบระบบได้ง่ายและมีความน่าเชื่อถือสูง อย่างไรก็ตาม หากมีการจัดเวลาการทำงานไม่เหมาะสมอาจทำให้เกิดการสูญเสียของระบบจนไม่สามารถทำงานได้อย่างมีประสิทธิภาพ เนื้อหาในบทนี้ได้นำเสนอวิธีการออกแบบอัลกอริทึมและการจัดสร้างระบบการจัดเวลาการทำงานแบบใหม่โดยการทำงานจะอาศัยเทคนิคการอินเตอร์รัพต์แบบหลายไทเมอร์เพื่อแยกการทำงานระหว่างส่วนควบคุมการจัดการเวลาและส่วนของงานแต่ละงานออกจากกัน ซึ่งสามารถลดผลกระทบการเกิดการโอเวอร์รันของงานและค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานได้

3.1 แนวคิดในการการจัดเวลาการทำงานแบบใหม่สำหรับสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงาน

การทำงานของสถาปัตยกรรมแบบ TTC เป็นระบบที่ไม่ซับซ้อนและมีพฤติกรรมด้านเวลาที่สามารถคาดเดาได้อย่างแม่นยำ ระบบจะอนุญาตให้งานเพียงหนึ่งงานดำเนินการทำงานต่อเนื่องจนเสร็จในเวลาหนึ่งโดยไม่ถูกขัดจังหวะหรือรบกวน ระบบนี้จะปลอดจากการถูกแทรกแซงจากงานอื่นซึ่งลำดับความสำคัญของแต่ละงานได้ถูกกำหนดไว้ก่อนที่ระบบจะเริ่มทำงานโดยที่ลำดับความสำคัญของงานจะไม่เปลี่ยนแปลงตลอดช่วงอายุการทำงานของระบบ พิจารณาโครงสร้างการจัดตารางเวลาแบบกระตุ้นด้วยเวลาแบบโปรแกรมเลือกจ่ายงานหรือแบบ TTC-Dispatch ดังโฟลว์ชาร์ตภาพประกอบ 3.1 โดยการทำงานเริ่มต้นระบบจะทำการกำหนดค่าเวลาในการกระตุ้นการทำงาน (Set up the timer) จากนั้นทำการกำหนดจำนวนงานและคุณสมบัติต่างๆ เพิ่มเข้าไปในระบบ เช่น ชนิดของงาน คาบเวลา ค่าเวลาในการดำเนินงาน เป็นต้น เมื่อระบบเริ่มทำงาน (Start the scheduler) ค่าเวลาที่ถูกกำหนดเป็นเวลาในการกระตุ้นการทำงานภายในไทเมอร์จะเริ่มต้นทำงาน โดยเมื่อถึงค่าเวลาที่กำหนดไว้ไทเมอร์ก็จะเกิดโอเวอร์โฟลว์ ทำให้เกิดการอินเตอร์รัพต์ขึ้นภายในระบบ เมื่อเกิดการอินเตอร์รัพต์ขึ้นส่วนการให้บริการอินเตอรัพต์จะเรียกฟังก์ชัน Update ในส่วนแบ็คกราวด์เพื่อทำการกำหนดค่าแฟลกแสดงสถานะว่าเกิดการกระตุ้นด้วยเวลาหรือเกิดอินเตอร์รัพต์ ซึ่งเมื่อฟังก์ชัน Update ดำเนินการเรียบร้อยแล้วจะออกจากการให้บริการอินเตอร์รัพต์เพื่อดำเนินการในส่วนของการประมวลผลด้านฟอร์กราวด์โดยระบบจะกระโดดไปยังฟังก์ชัน Dispatch สำหรับทำการจัดเรียงลำดับของงานและดำเนินการทำงานภายในคาบเวลาที่ถูกกระตุ้นการทำงานจนเสร็จ ต่อจากนั้นระบบกลับเข้าไปสู่โหมดประหยัดพลังงานเพื่อลดค่าการใช้พลังงานและรอให้เกิดการกระตุ้นการทำงานในคาบเวลาถัดไป จากการอธิบายที่ผ่านมาสามารถสรุปการเรียกใช้ฟังก์ชันต่างๆ ของ TTC-Dispatch ได้ดังภาพประกอบ 3.2



ภาพประกอบ 3.1 โฟลว์ชาร์ตการทำงานของ TTC-Dispatch



ภาพประกอบ 3.2 การเรียกใช้งานฟังก์ชันต่างๆ ของ TTC-Dispatch [1]

อย่างไรก็ตามถึงแม้ว่าสถาปัตยกรรมแบบ TTC-Dispatch จะเป็นระบบที่ไม่ซับซ้อน แต่ระบบดังกล่าวยังมีข้อจำกัดเรื่องความเปราะบางในสถานการณ์ที่เกิดสภาวะโอเวอร์รัน เพราะขณะที่งานใดๆ กำลังดำเนินงาน ระบบจะไม่ยอมให้ถูกแทรกแซงจากงานอื่นจนกว่างานที่กำลังดำเนินงานจะแล้วเสร็จ ดังนั้นหากมีงานที่กำลังดำเนินงานในฟังก์ชัน Dispatch แล้วยังดำเนินงานไม่เสร็จสิ้น ถึงแม้จะเกิดการอินเตอร์รัพต์เพื่อเป็นเวลาในการกระตุ้นการทำงานของคาบถัดไป ก็ไม่สามารถแทรกแซงงานที่กำลังดำเนินงานอยู่ได้ ลักษณะเช่นนี้จะเรียกว่าเกิดการโอเวอร์รันของงาน นั้นทำให้ระบบไม่สามารถดำเนินการจัดตารางเวลาได้ตามที่ออกแบบไว้และอาจเกิดการสูญเสียกับระบบงานได้

เพื่อเป็นการแก้ปัญหาดังกล่าว งานวิจัยนี้จึงเสนอการจัดเวลาการทำงานแบบใหม่โดยมีแนวคิดในการทำงานดังภาพประกอบ 3.3 สรุปได้ดังนี้

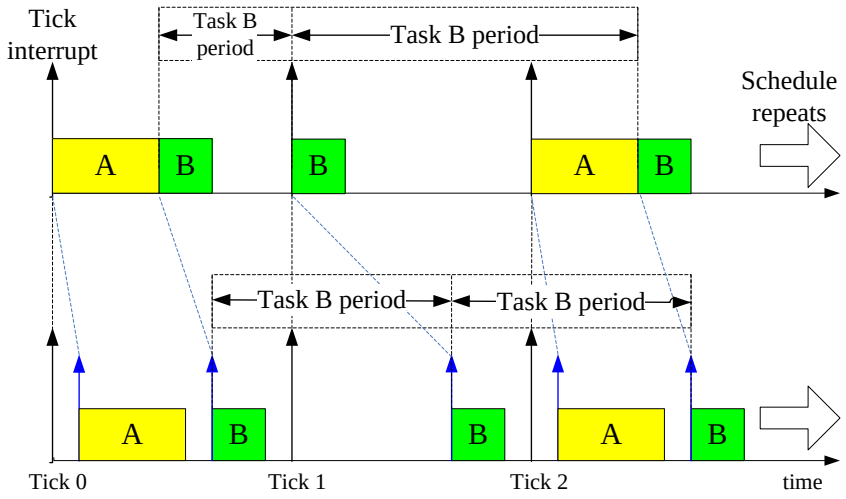
1. การแก้ปัญหาการเกิดการโอเวอร์รันของงาน

การแก้ปัญหาสามารถทำได้โดยให้ระบบทำการแยกส่วนการทำงานระหว่างส่วนควบคุมการกระตุ้นการทำงานและส่วนของงานแต่ละงานออกจากกัน ทั้งนี้ในการทำงานจะอาศัยเทคนิคการอินเตอร์รัพต์แบบหลายไทเมอร์เพื่อแยกการทำงาน จากภาพประกอบ 3.3 จะเห็นว่าลูกศรสีดำแสดงสัญลักษณ์ของการเกิดอินเตอร์รัพต์ (Tick interrupt) ซึ่งทำหน้าที่กำหนดสัญญาณรายคาบเพื่อกระตุ้นการทำงานของระบบ TTC ส่วนลูกศรสีน้ำเงินแสดงสัญลักษณ์ของการอินเตอร์รัพต์เพื่อกระตุ้นการทำงานในแต่ละงาน ซึ่งจากภาพประกอบจะเห็นว่า Task A และ Task B จะทำงานภายหลังจากการเกิดอินเตอร์รัพต์นี้และเนื่องจากระบบได้กำหนดความสำคัญของการอินเตอร์รัพต์ไว้ก่อนล่วงหน้าว่าแต่ละงานจะถูกดำเนินการในเวลาใด ดังนั้นเมื่อเกิดการโอเวอร์รันขึ้นในงานใดๆ ระบบดังกล่าวก็จะอนุญาตให้งานที่มีความสำคัญกว่าสามารถแทรกการทำงานของงานที่เกิดโอเวอร์รันทำให้ระบบการทำงานนี้สามารถกลับมาทำงานได้ตามเวลาที่กำหนดและไม่ส่งผลกระทบกับงานที่จะถูกดำเนินงานต่อมา

2. การแก้ปัญหาค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน

การแก้ปัญหาความผิดพลาดสามารถทำได้โดยการกำหนดค่าเวลาก่อนดำเนินงานของแต่ละงานให้มีค่าเท่ากัน [2-5] เพื่อลดผลกระทบของคาบเวลาที่ไม่เท่ากันของแต่ละงาน จากภาพประกอบ 3.3 หากกำหนดให้ Task B ทำงานทุกคาบเวลา จะเห็นว่าเวลาการทำงานเริ่มต้นในแต่ละคาบของ Task B จะเริ่มทำงานไม่พร้อมกันอันเนื่องมาจากผลกระทบของการดำเนินงานก่อนหน้า (ผลกระทบจาก Task A) ซึ่งจะส่งผลให้เกิดความผิดพลาดทางเวลาขณะเริ่มต้นทำงานใน Task B

อย่างไรก็ตาม เพื่อแก้ปัญหาดังกล่าวเราจะกำหนดค่าเวลาการเริ่มต้นการทำงานของ Task B ให้มีค่าเท่ากับค่าเวลาในการดำเนินการที่มากที่สุดของ Task A บวกกับค่าโอเวอร์เฮดของการจัดตารางเวลาซึ่งจะทำให้คาบเวลาของ Task B เริ่มต้นการทำงานพร้อมกันทุกคาบเวลา ทำให้สามารถลดค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานของ Task B ลงได้



ภาพประกอบ 3.3 แนวคิดในการจัดเวลาการทำงานแบบใหม่สำหรับสถาปัตยกรรม TTC

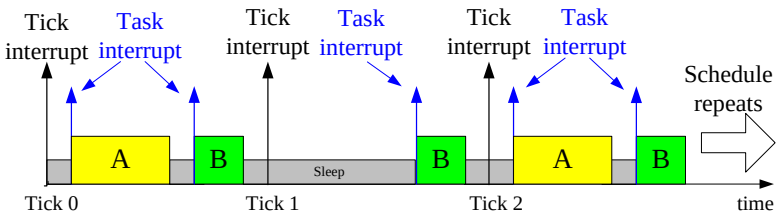
3.2 ระบบการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์ (Multiple Timer Interrupts: MTIs)

จากการเสนอระบบให้ทำการแยกการทำงานระหว่างส่วนควบคุมการกระตุ้นการทำงานและส่วนของงานแต่ละงานออกจากกัน สามารถทำได้โดยอาศัยเทคนิคการอินเตอร์รัพต์แบบหลายไทเมอร์แสดงดังภาพประกอบ 3.4 เทคนิคนี้ถูกออกแบบเพื่อใช้แก้ผลกระทบของการเกิดโอเวอร์รันของงาน การจัดลำดับของงานและปัญหาของค่าความผิดพลาดทางเวลาขณะที่ระบบเริ่มต้นทำงาน [1, 6] ระบบการทำงานนี้สามารถดำเนินงานโดยการแยกงานออกจากกันเพื่อลดผลกระทบของงานที่ถูกดำเนินงานก่อนหน้า โดยการทำงานลักษณะนี้จะใช้ไทเมอร์อินเตอร์รัพต์จำนวน 2 ตัว ได้แก่ Tick interrupt และ Task interrupt [6] ทำหน้าที่ต่างๆ ดังนี้

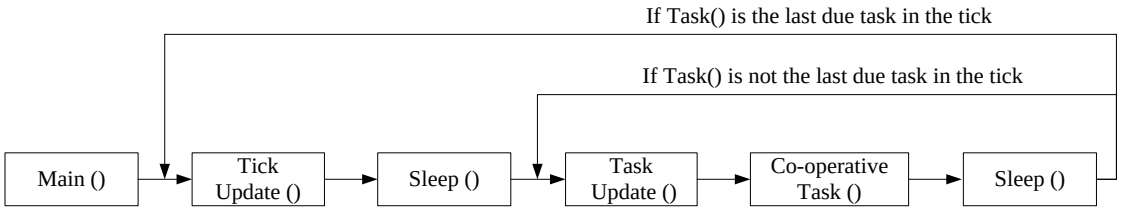
1. Tick interrupt ทำหน้าที่กำหนดสัญญาณรายคาบเช่นเดียวกับสถาปัตยกรรม TTC-Dispatch
2. Task interrupt ทำหน้าที่เป็นตัวกระตุ้นการดำเนินงานของแต่ละงานภายในช่วงเวลาของ Tick interrupt

การจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์ ระบบจะทำการกำหนดจำนวนงานและคุณสมบัติต่างๆ เพิ่มเข้าไปในระบบเช่นเดียวกับระบบ TTC-Dispatch โดยการจัดตารางเวลาจะทำการคำนวณค่าเมเจอร์ไซเคิลและค่าเวลาการเริ่มต้นการทำงานของแต่ละงานโดยการเรียกใช้ฟังก์ชันต่างๆ ของ TTC-MTIs แสดงดังภาพประกอบ 3.5 จะเห็นว่าการทำงานของ TTC-MTIs จะเรียกใช้ฟังก์ชัน Update จำนวน 2 ฟังก์ชัน ได้แก่ ฟังก์ชัน Tick Update และ Task Update

เมื่อเกิดการอินเตอร์รัพต์จาก Tick interrupt ฟังก์ชัน Tick Update จะถูกเรียกใช้งานโดยจะทำการตรวจสอบและจัดเรียงลำดับงานที่พร้อมทำงานพร้อมทั้งกำหนดค่าเวลาเริ่มต้นการทำงานของ แต่ละงานให้กับ Task interrupt เพื่อรอเวลาที่จะเกิดอินเตอร์รัพต์เพื่อกระตุ้นการทำงานของแต่ละงานภายในคาบเวลาที่กำลังดำเนินการ ต่อจากนั้นระบบเรียกใช้งานฟังก์ชัน Sleep เพื่อเข้าไปสู่โหมดประหยัดพลังงาน เมื่อถึงเวลาเริ่มต้นการทำงานของแต่ละงานจะเกิดการอินเตอร์รัพต์จาก Task interrupt เพื่อกระตุ้นเวลาเริ่มต้นในการทำงาน ในที่นี้ฟังก์ชัน Task Update จะถูกเรียกใช้งานโดยจะทำการดำเนินงานที่พร้อมทำงานจนเสร็จและตรวจสอบงานในลำดับถัดไปว่าพร้อมทำงานหรือไม่ หากมีงานที่พร้อมทำงานระบบจะรอการเกิดการอินเตอร์รัพต์จาก Task interrupt ตามเวลาเริ่มต้นการทำงานของแต่ละงานเพื่อทำการการดำเนินงานทุกงานที่พร้อมทำงานจนเสร็จ จากนั้นจะรอการเกิดอินเตอร์รัพต์จาก Tick interrupt เพื่อให้เกิดการกระตุ้นการทำงานในคาบเวลาถัดไป สังเกตว่าหลังจากงานแต่ละงานถูกดำเนินงานจนเสร็จระบบเรียกใช้งานฟังก์ชัน Sleep เพื่อลดค่าการใช้พลังงานรวมของระบบ

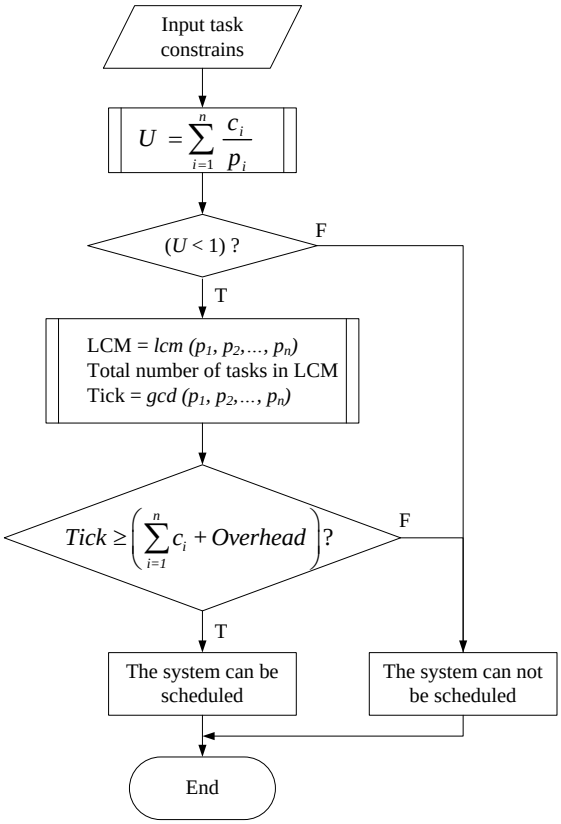


ภาพประกอบ 3.4 เทคนิคการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์



ภาพประกอบ 3.5 การเรียกใช้งานฟังก์ชันต่างๆ ของ TTC-MTIs [1]

3.3 อัลกอริทึมสำหรับทดสอบความสามารถในการจัดตารางการทำงานของระบบ TTC-MTIs



ภาพประกอบ 3.6 โฟลว์ชาร์ตของอัลกอริทึม TTSA-MTI สำหรับทดสอบความสามารถในการจัดตารางการทำงานของระบบ TTC-MTIs

การทดสอบความสามารถในการจัดตารางการทำงานของสถาปัตยกรรมแบบกระตุ้นการทำงานด้วยเวลาเป็นสิ่งสำคัญในการพัฒนาระบบเป็นอย่างยิ่ง เพราะว่าโดยทั่วไประบบการทำงานแบบนี้จะแบ่งการทำงานเป็นสองส่วนได้แก่ส่วนการออกแบบตารางเวลางานและส่วนของการปฏิบัติการ โดยในการออกแบบตารางการทำงานนั้น ผู้ออกแบบหรือนักพัฒนาระบบจะต้องรู้รูปแบบทำงานของระบบล่วงหน้าเพื่อจะได้ออกแบบพร้อมทั้งทดสอบความสามารถในการจัดตารางการทำงานเพื่อให้ระบบที่จะนำระบบไปประยุกต์ใช้งานสามารถทำงานได้อย่างมีประสิทธิภาพและมีการทำงานอย่างแม่นยำ

ดังนั้นงานวิจัยนี้ได้เสนอการออกแบบอัลกอริทึมเพื่อทดสอบความสามารถในการจัดตารางการทำงานของระบบ TTC-MTIs ชื่อ “Time-Triggered Scheduling Algorithm with Multiple Timer Interrupts” (TTSA-MTI) สำหรับการจัดตารางการทำงานแบบกระตุ้นด้วยเวลาโดยอาศัยเทคนิคการอินเตอร์รัพต์แบบหลายไทเมอร์ ดังภาพประกอบ 3.6 จากโฟลว์ชาร์ตของอัลกอริทึม TTSA-MTI สามารถอธิบายการทำงานได้ดังนี้

1. อินพุตเซตของงาน (t_i) ที่มีคุณลักษณะและเงื่อนไขตามที่กำหนด โดยกำหนดพารามิเตอร์ต่างๆ ดังนี้

$$t_i = \{p_i, c_i, d_i, o_i\} \tag{3.1}$$

โดยที่ p_i คือ คาบเวลางาน
c_i คือ ค่าเวลาในการดำเนินงาน
d_i คือ เดดไลน์ของงาน
o_i คือ ออฟเซตของงาน

2. คำนวณและทดสอบค่าอรรถประโยชน์ของซีพียู (CPU utilization: U) จากเซตของงานที่รับเข้ามาจำนวน *n* งาน

$$U = \sum_{i=1}^{n} \frac{c_i}{p_i} \tag{3.2}$$

3. ในการทดสอบค่าอรรถประโยชน์ของซีพียู หากค่า $U \leq 1$ แสดงว่าซีพียูสามารถจัดตารางการทำงานของระบบ TTC ได้ อัลกอริทึมจะทำขั้นตอนต่อไปคือการคำนวณเพื่อหาค่าเมเจอร์ไซเคิล (LCM) จำนวนของงานทั้งหมดที่เกิดขึ้นในเมเจอร์ไซเคิล (LCM_{total}) และค่าคาบเวลาที่เหมาะสมที่สุด สำหรับการนำไปทดสอบการจัดตารางงานในขั้นตอนต่อไป ในทางตรงข้ามหากค่า $U > 1$ แสดงว่าซีพียูไม่สามารถจัดตารางการทำงานได้ อัลกอริทึมจะออกจากการทดสอบการจัดตารางการทำงาน

$$LCM = lcm\{p_1, p_2, ..., p_n\} \tag{3.3}$$

$$LCM_{total} = \sum_{i=1}^{n} \frac{LCM}{p_i} \quad (3.4)$$

$$Tick = gcd\{p_1, p_2, ..., p_n\} \quad (3.5)$$

4. อัลกอริทึมจะทำการทดสอบความสามารถในการจัดการตารางการทำงานของระบบ TTC-MTIs โดยการเปรียบเทียบระหว่างค่า $Tick$ ที่ได้จากการคำนวณในสมการ 3.5 กับค่าผลรวมของค่าเวลาในการดำเนินงานของแต่ละงานและค่าโอเวอร์เฮดของการจัดตารางเวลา หากค่า $Tick$ มีค่ามากกว่าหรือเท่ากับผลรวมดังกล่าวในทุกคาบเวลาแสดงว่าระบบ TTC-MTIs สามารถจัดการตารางการทำงานได้ตามเงื่อนไขดังอสมการ 3.6

$$Tick \geq \left(\sum_{i=1}^{n} c_i + Overhead \right) \quad (3.6)$$

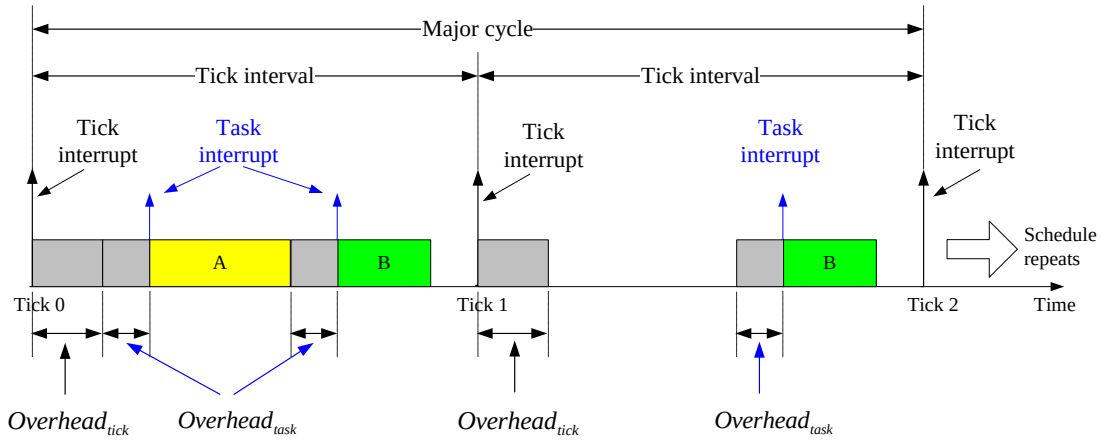
เมื่อพิจารณาค่าโอเวอร์เฮดการจัดตารางงานของระบบ TTC-MTIs ค่าโอเวอร์เฮดของระบบดังกล่าวจะประกอบด้วยโอเวอร์เฮดของ Tick interrupt ($Overhead_{tick}$) และโอเวอร์เฮดของ Task interrupt ($Overhead_{task}$)

1) โอเวอร์เฮดของ Tick interrupt คือค่าเวลาของสัญญาณรายคาบที่เกิดขึ้นตั้งแต่ซีพียูให้บริการอินเตอร์รัพต์จนถึงการตรวจสอบงานในตารางงาน

2) โอเวอร์เฮดของ Task interrupt คือค่าเวลาของสัญญาณที่เป็นตัวกระตุ้นการดำเนินงานของแต่ละงานตั้งแต่ซีพียูให้บริการอินเตอร์รัพต์จนกระทั่งงานแต่ละงานเริ่มดำเนินการ

ตัวอย่างค่าโอเวอร์เฮดของการจัดตารางงานแสดงดังภาพประกอบ 3.7 จะเห็นว่าในทุกๆ คาบการทำงานจะมีค่าโอเวอร์เฮดของ Tick interrupt และก่อนที่งานแต่ละงานจะเริ่มดำเนินการก็จะมีค่า โอเวอร์เฮดของ Task interrupt ซึ่งค่าโอเวอร์เฮดเหล่านี้ สามารถคำนวณได้ดังสมการ 3.7

$$Overhead = Overhead_{tick} + \sum_{i=1}^{n} Overhead_{task}(i) \quad (3.7)$$



ภาพประกอบ 3.7 โอเวอร์เฮดของการจัดตารางงานของระบบ TTC-MTIs

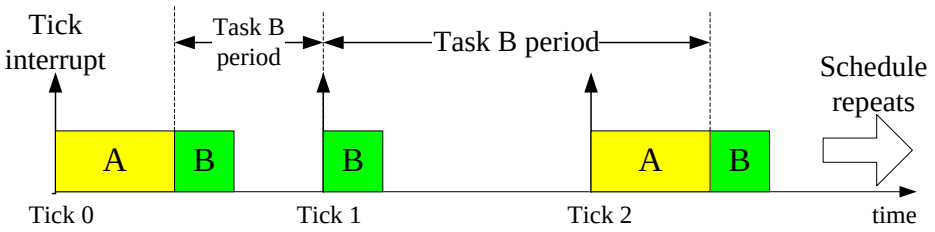
3.4 การลดผลกระทบของค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน

จากงานวิจัยที่ผ่านมา [1, 7, 8] พบว่าสาเหตุสำคัญของความผิดพลาดทางเวลาขณะเริ่มต้นทำงานได้แก่ความผิดพลาดที่เกิดจากโอเวอร์เฮดของการจัดตารางเวลา ความผิดพลาดที่เกิดจากการจัดลำดับการดำเนินการของงานและความผิดพลาดจากพฤติกรรมการอินเตอร์รัพต์ อย่างไรก็ตามในที่นี้สนใจการลดค่าความผิดพลาดที่เกิดจากการจัดลำดับการดำเนินการของงานอันเนื่องมาจากผลกระทบของการดำเนินงานก่อนหน้า เพื่อลดค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานที่ไม่เท่ากันของแต่ละงาน

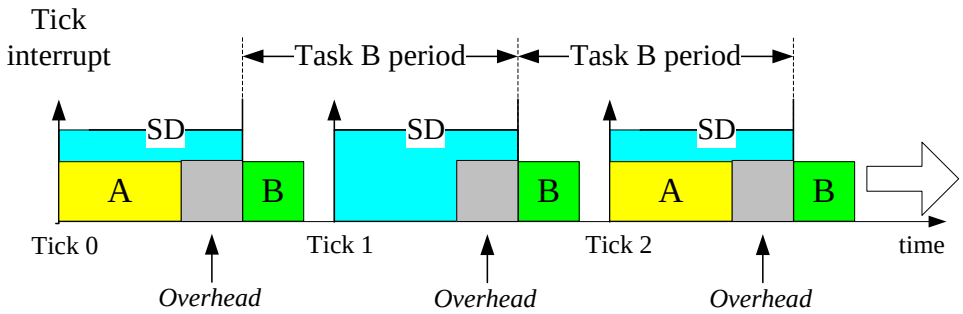
ตัวอย่างปัญหาค่าความผิดพลาดที่เกิดจากการจัดลำดับการดำเนินการของงานแสดงดังภาพประกอบ 3.8 (a) จะเห็นว่าคาบเวลาเริ่มต้นของ Task B มีขนาดไม่เท่ากันอันเนื่องมาจากผลกระทบของการดำเนินงานของ Task A ก่อนที่ Task B จะเริ่มดำเนินการซึ่งจะทำให้เกิดค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานใน Task B อย่างไรก็ดีจากงานวิจัยของ Lakhani [5] ได้แนะนำการประยุกต์ใช้เทคนิคแซนวิชดีเลย์ (Sandwich delay: SD) สำหรับการลดค่าความผิดพลาดทางเวลาของงานใดๆ ในระบบซึ่งสามารถทำโดยการวางค่า SD ไว้ก่อนหน้าที่งานจะถูกดำเนินการเพื่อกำหนดค่าเวลาก่อนดำเนินงานของแต่ละงานให้มีค่าเท่ากัน รูปแบบการประยุกต์ใช้งานเทคนิคแซนวิชดีเลย์แสดงดังภาพประกอบ 3.8 (b) จากภาพประกอบเราสามารถกำหนดค่าเวลาเริ่มต้นการทำงานของ Task B ได้โดยกำหนดให้มีค่าเท่ากับค่าเวลาในการดำเนินการที่มากที่สุดของ Task A บวกกับค่าโอเวอร์เฮดของการจัดตารางเวลา ซึ่งจะทำให้คาบเวลาของ Task B มีขนาดใกล้เคียงกันและสามารถลดค่าความผิดพลาดทางเวลาของ Task B ได้ อย่างไรก็ดีจากงานวิจัยของ Nahas [1] พบว่าถึงแม้เทคนิคแซนวิชดีเลย์จะสามารถลดค่าความผิดพลาดทางเวลาของงานได้อย่างมีประสิทธิภาพ แต่ว่าวิธีการดังกล่าวยังมีข้อจำกัดเรื่องการควบคุมเวลาในการทำงานที่ไม่แม่นยำ นอกจากนี้ยังเพิ่มอัตราการบริโภคพลังงานของซีพียู

เพื่อแก้ปัญหาดังกล่าว งานวิจัยนี้จึงได้นำเทคนิคแซนวิชดีเลย์ไปประยุกต์ใช้กับระบบ MTIs แสดงดังภาพประกอบ 3.8 (c) โดยในการทำงานของระบบนั้นตารางการจัดการเวลาจะคำนวณค่าเวลาเริ่มต้นการทำงานของงานต่างๆ (Required release time: t_r) ซึ่งค่าเวลาดังกล่าวสามารถคำนวณได้จากผลรวมของค่าเวลาในการดำเนินการที่มากที่สุดของงานที่ถูกดำเนินการก่อนหน้าบวกกับค่าโอเวอร์เฮดของการจัดการตารางดังสมการ 3.8

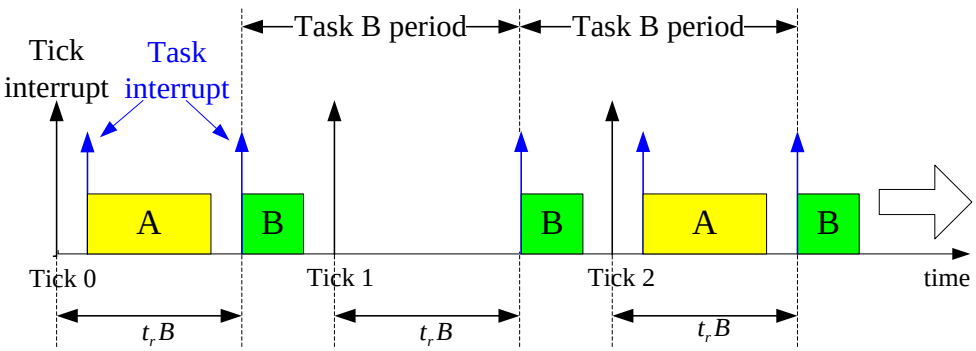
$$t_r(k) = \sum_{i=1}^{k-1} (wcet_i + Overhead_i) \qquad (3.8)$$



(a)



(b)



(c)

ภาพประกอบ 3.8 (a) ค่าความผิดพลาดทางเวลาที่เกิดจากการจัดลำดับการดำเนินการของงาน,
(b) การลดผลกระทบของ Task period โดยใช้เทคนิคแซนวิชดีเลย์,
(c) การนำเทคนิคแซนวิชดีเลย์ไปประยุกต์ใช้กับระบบ MTIs

เนื้อหาในบทนี้ได้นำเสนอระบบการจัดการเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์และวิธีการออกแบบอัลกอริทึมเพื่อทดสอบความสามารถในการจัดตารางการทำงานของระบบ TTC-MTIs เพื่อลดผลกระทบการเกิดการโอเวอร์รันของงานและค่าความผิดพลาดทางเวลา ขณะเริ่มต้นทำงานสำหรับประยุกต์ใช้งานกับสถาปัตยกรรมแบบ TTC โดยเนื้อหาในบทถัดไปได้ทำการพัฒนาระบบพร้อมทั้งประเมินและทดสอบประสิทธิภาพของระบบที่นำเสนอเพื่อเปรียบเทียบผลการทดลองกับวิธีการที่ผ่านมา

บทที่ 4

ผลการวิจัยและอภิปรายผล

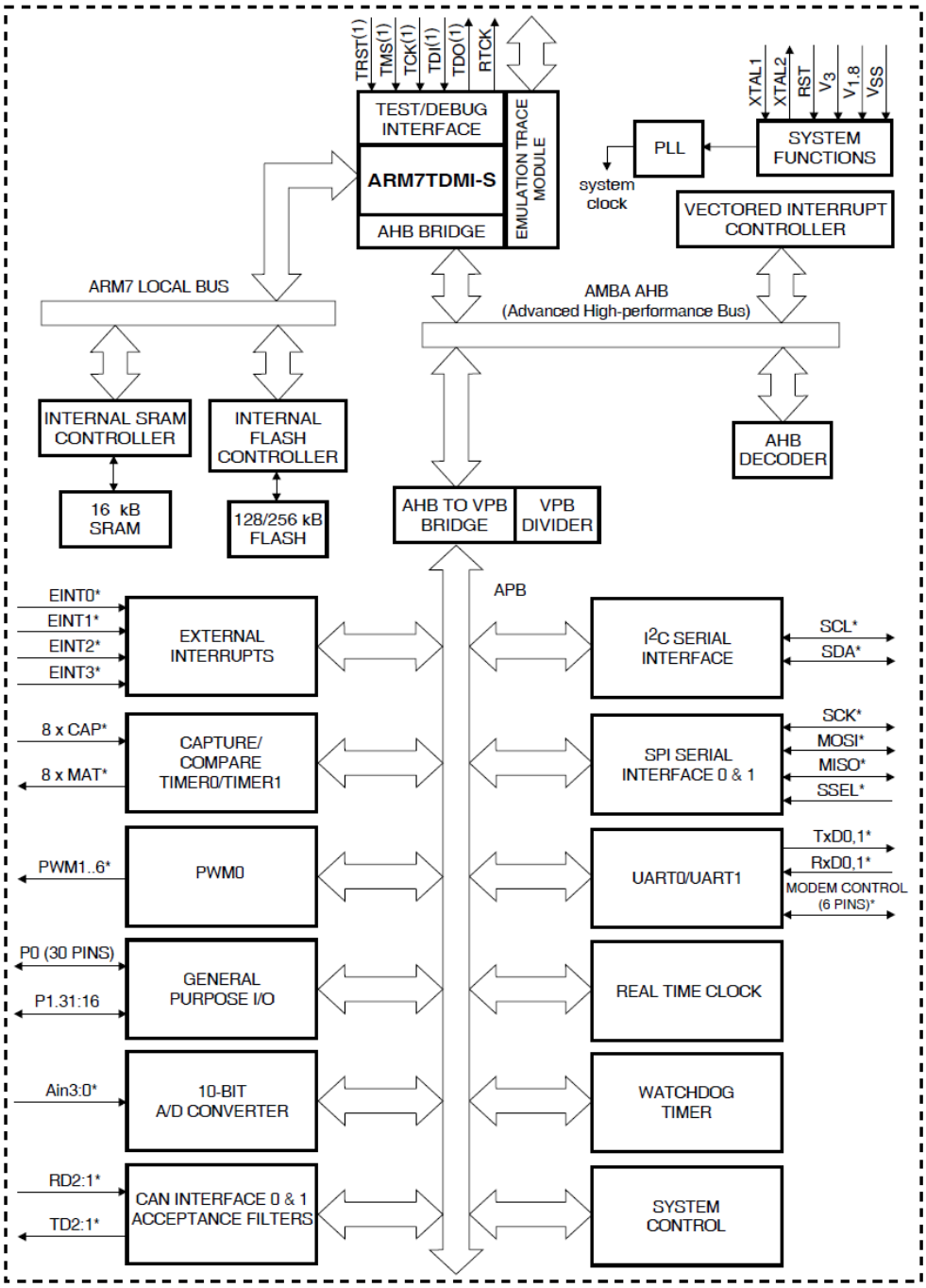
เนื้อหาในบทนี้ได้นำเสนอการพัฒนาระบบการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์สำหรับประยุกต์ใช้งานกับสถาปัตยกรรมแบบ TTC เพื่อลดผลกระทบของการเกิดทาสก์โอเวอร์รันและปัญหาของค่าความผิดพลาดทางเวลาขณะที่ระบบเริ่มต้นทำงาน พร้อมทั้งประเมินและทดสอบประสิทธิภาพของระบบที่นำเสนอ ประสิทธิภาพของหน่วยความจำ ซีพียู และการทดสอบด้านความผิดพลาดทางเวลาขณะเริ่มต้นทำงานเพื่อเปรียบเทียบผลการทดลองกับวิธีการที่ผ่านมา

4.1 การพัฒนาระบบการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์

4.1.1 ฮาร์ดแวร์

การพัฒนาระบบ TTC-MTIs เพื่อใช้ทดสอบประสิทธิภาพในการทดลองใช้ไมโครคอนโทรลเลอร์ตระกูล ARM ซึ่งเป็นไมโครคอนโทรลเลอร์ขนาด 32 บิต เบอร์ LPC2129 [1] โดยเลือกใช้แหล่งกำเนิดสัญญาณแบบ XTAL ค่า 12 MHz ซึ่งสามารถกำหนดการทำงานร่วมกับ Phase Lock Loop ให้ไมโครคอนโทรลเลอร์สามารถประมวลผลที่ความเร็ว 60 MHz ซีพียูชนิดนี้ประกอบด้วยมอดูลไทเมอร์ขนาด 32 บิต จำนวน 2 ตัว มอดูลไทเมอร์แต่ละตัวสามารถควบคุมการเกิดอินเตอรัพต์ได้ 4 ช่อง ซึ่งรองรับการพัฒนาระบบการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์ได้เป็นอย่างดี บล็อกไดอะแกรมของไมโครคอนโทรลเลอร์เบอร์ LPC2129 แสดงดังภาพประกอบ 4.1 มีคุณสมบัติทั่วไปดังนี้

1. หน่วยความจำ Flash สำหรับเขียนโปรแกรมขนาด 256 กิโลไบต์ และ RAM ขนาด 16 กิโลไบต์
2. มีพอร์ต I/O จำนวน 2 พอร์ตคือ P0 (32 บิต) และ P1 (16 บิต)
3. มีวงจรสื่อสารอนุกรม UART จำนวน 2 พอร์ต วงจรสื่อสาร SPI จำนวน 2 พอร์ต วงจรสื่อสาร I2C จำนวน 1 พอร์ตและวงจรสื่อสาร CAN จำนวน 2 ช่อง
4. มีมอดูล Timer/Counter ขนาด 32 บิต จำนวน 2 ชุด
5. มีวงจร ADC ขนาด 10 บิต จำนวน 4 ช่อง
6. มีวงจร Watchdog, PWM และ Real Time Clock



ภาพประกอบ 4.1 บล็อกไดอะแกรมไมโครคอนโทรลเลอร์ ARM7TDMI เบอร์ LPC2129

การพัฒนาระบบ time-triggered โดยทั่วไปจะอาศัยมอดูลไทเมอร์ภายในไมโครคอนโทรลเลอร์เพื่อสร้างสัญญาณคาบเวลาสำหรับไมโครคอนโทรลเลอร์เบอร์ LPC2129 อาศัยรีจิสเตอร์ Timers/Countersขนาด 32 บิต จำนวน 2 ตัว ได้แก่ Timer0 and Timer1 ในการสร้างสัญญาณคาบเวลา โดยในการประยุกต์ใช้งานมอดูลไทเมอร์จะประกอบด้วยรีจิสเตอร์ที่เกี่ยวข้องดังตาราง 4.1

ตาราง 4.1 รีจิสเตอร์ที่เกี่ยวข้องกับการประยุกต์ใช้มอดูลไทเมอร์ในการสร้างสัญญาณคาบเวลา

Register	หน้าที่
Prescale Register (PR)	PC เป็นรีจิสเตอร์ขนาด 32 bit ทำหน้าที่กำหนดค่าที่มากที่สุดสำหรับ Prescale Counter
Prescale Counter Register (PC)	PC ทำหน้าที่เป็นตัวนับซึ่งจะเพิ่มค่าทุกๆ PCLK จนถึงค่าที่เก็บไว้ในรีจิสเตอร์ PR เมื่อเพิ่มไปจนถึงค่า PR ค่ารีจิสเตอร์ TC จะถูกเพิ่มค่าขึ้นและค่า PC จะถูกเคลียร์
Timer Counter (TC)	TC เป็นรีจิสเตอร์ขนาด 32 bit จะถูกนับขึ้นทุกรอบ PR (PR + 1) ของ PCLK โดยรีจิสเตอร์ TC จะถูกควบคุมการนับผ่านรีจิสเตอร์ TCR
Timer Control Register (TCR)	TCR ทำหน้าที่ควบคุมการทำงานของ TC ซึ่งสามารถสั่งให้ disable หรือรีเซตได้ บิต 0 (Counter Enable): - เมื่อกำหนดค่าเป็น 1 รีจิสเตอร์ TC และ PC จะถูก enable เพื่อทำการนับ - เมื่อกำหนดค่าเป็น 0 การนับจะถูก disable บิต 1 (Counter Reset): - เมื่อกำหนดค่าเป็น 1 รีจิสเตอร์ TC และ PC จะถูกรีเซตใน PCLK ถัดไปและการนับจะถูกรีเซตจนกว่าค่าของบิตนี้จะกลายเป็น 0
Match Control Register (MCR)	MCR ใช้สำหรับควบคุมการทำงานใดๆ ที่ถูกกระทำเมื่อมีค่าใดค่าหนึ่งของ Match Register ตรงกับค่า TC
Interrupt Register (IR)	Interrupt Register มีขนาด 8 bit ประกอบด้วย 4 bit สำหรับ match interrupts และอีก 4 bit สำหรับ capture interrupts - ถ้าหากเกิดการ interrupt ค่าของ Interrupt Register ที่บิตนั้นจะถูก เซตเป็น high สามารถถูกกำหนดค่าเพื่อจะเคลียร์การ interrupt Bit 0 = MR0 interrupt Bit 1 = MR1 interrupt Bit 2 = MR2 interrupt Bit 3 = MR3 interrupt
Match Register 0 (MR0)	MR0 สามารถถูก enable ผ่านรีจิสเตอร์ MCR เพื่อทำการรีเซตรีจิสเตอร์ TC หยุดการนับของรีจิสเตอร์ TC และ PC, และสร้างสัญญาณอินเตอร์รัพต์ทุกครั้งที่ MR0 มีค่าเท่ากับ TC

ตาราง 4.1 รีจิสเตอร์ที่เกี่ยวข้องกับการประยุกต์ใช้มอดูลไทเมอร์ในการสร้างสัญญาณคาบเวลา (ต่อ)

Register	หน้าที่
Match Register 1 (MR1)	MR1 สามารถถูก enable ผ่านรีจิสเตอร์ MCR เพื่อทำการรีเซต รีจิสเตอร์ TC, หยุดการนับของรีจิสเตอร์ TC และ PC, และสร้างสัญญาณอินเตอร์รัพต์ทุกครั้งที่ MR1 มีค่าเท่ากับ TC
Match Register 2 (MR2)	MR2 สามารถถูก enable ผ่านรีจิสเตอร์ MCR เพื่อทำการรีเซต รีจิสเตอร์ TC หยุดการนับของรีจิสเตอร์ TC และ PC, และสร้างสัญญาณอินเตอร์รัพต์ทุกครั้งที่ MR2 มีค่าเท่ากับ TC
Match Register 3 (MR3)	MR3 สามารถถูก enable ผ่านรีจิสเตอร์ MCR เพื่อทำการรีเซต รีจิสเตอร์ TC หยุดการนับของรีจิสเตอร์ TC และ PC, และสร้างสัญญาณอินเตอร์รัพต์ทุกครั้งที่ MR3 มีค่าเท่ากับ TC

4.1.2 การพัฒนาโปรแกรม

เครื่องมือในการพัฒนาโปรแกรมสำหรับไมโครคอนโทรลเลอร์ ARM7 ที่ใช้ในปัจจุบันมีหลากหลายทั้งซอฟแวร์แบบมีลิขสิทธิ์ (Proprietary tools) เช่น KEIL, IAR Workbench, Windriver workbench, Codesourcery ฯลฯ หรือซอฟแวร์แบบโอเพนซอร์ส เช่น GNUARM, Yagarto ฯลฯ อย่างไรก็ดีสำหรับการพัฒนาโปรแกรมของระบบ TTC-MTIs ในการทดลองนี้ใช้เครื่องมือในการพัฒนาโปรแกรมชื่อ RealView MDK เวอร์ชัน 4.12 [2] ของบริษัท Keil ซึ่งมีข้อดีคือสามารถดาวน์โหลดโปรแกรมรุ่นทดลองใช้ได้ฟรีพร้อมทั้งมีเครื่องมือใช้งานซึ่งประกอบด้วย uvision IDE, คอมไพเลอร์ ดีบักเกอร์และเครื่องมือจำลองการทำงาน

การพัฒนาโปรแกรมของระบบ TTC-MTIs ทำโดยการเขียนโปรเจคในรูปแบบภาษาซีผ่านโปรแกรม RealView MDK โดยเลือกใช้ไมโครคอนโทรลเลอร์เบอร์ LPC2129 สำหรับการทดลองแสดงดังภาพประกอบ 4.2 และโครงสร้างของระบบ TTC-MTIs ที่พัฒนาขึ้นมีโครงสร้างเป็นมอดูลแสดงดังภาพประกอบ 4.3 โดยมอดูลต้นฉบับต่างๆ (Source module) ประกอบด้วยฟังก์ชันและหน้าที่การทำงานดังตาราง 4.2

จากภาพประกอบ 4.3 โปรเจค TTC-MTI.UVproj ประกอบด้วยมอดูลต้นฉบับต่างๆ ทำหน้าที่ดังนี้

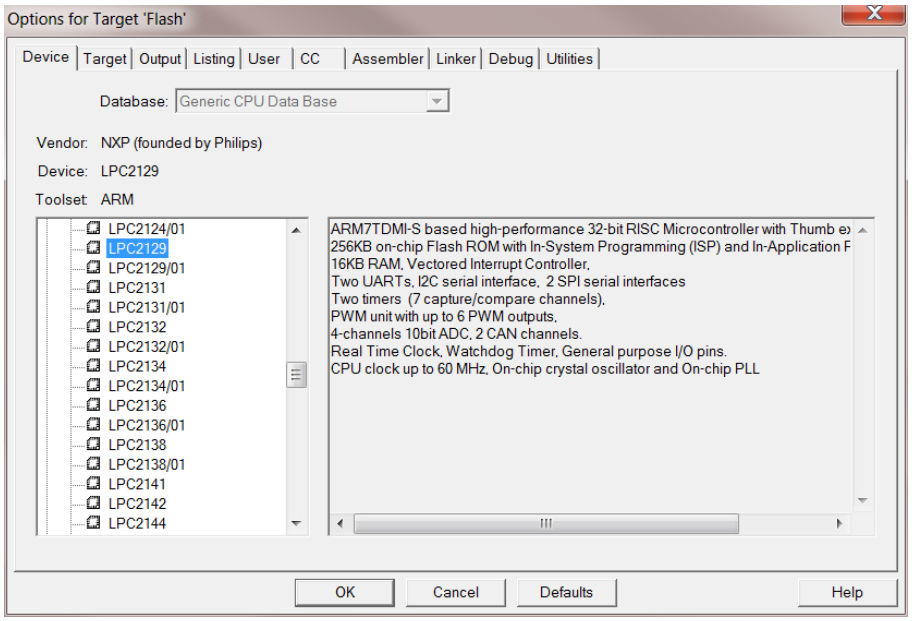
1) main.c ทำหน้าที่เป็นฟังก์ชันหลักของโปรแกรม อินพุตงานเข้าในระบบและเริ่มต้นการทำงานของระบบการจัดตารางงาน

2) system_init.c ทำหน้าที่กำหนดค่าเริ่มต้นและจัดเตรียมการจัดตารางการทำงานให้กับระบบ

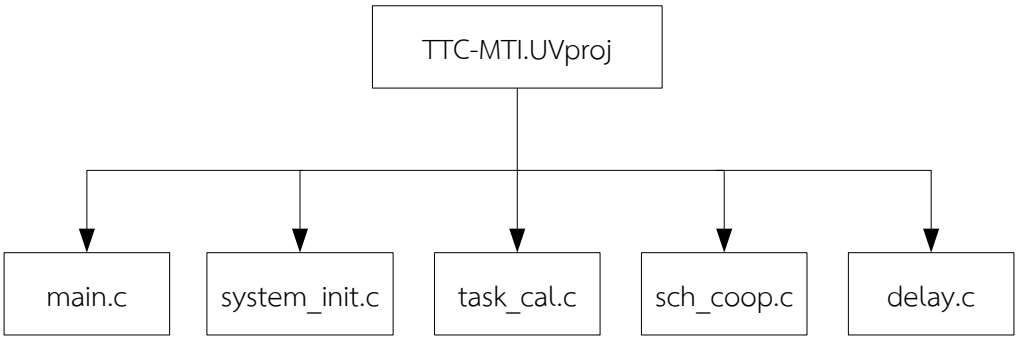
3) task_cal.c ทำหน้าที่คำนวณค่าต่างๆ ที่เกี่ยวข้องในการจัดตารางงานตามโครงสร้างของระบบ TTC-MTIs

4) sch_coop.c ทำหน้าที่กำหนดสัญญาณรายคาบผ่าน Tick interrupt และทำหน้าที่กระตุ้นการดำเนินงานของแต่ละงานผ่าน Task interrupt

5) delay.c ทำหน้าที่กำหนดค่าเวลาสูงสุดในการดำเนินงานของงานต่างๆ ในระบบ



ภาพประกอบ 4.2 โปรแกรม RealView MDK โดยเลือกใช้ไมโครคอนโทรลเลอร์เบอร์ LPC2129



ภาพประกอบ 4.3 โครงสร้างโปรเจคในรูปแบบภาษาซีของระบบ TTC-MTIs

ตาราง 4.2 มอดูลต้นฉบับภาษาซีและหน้าที่การทำงานของฟังก์ชันต่างๆ

ไฟล์	ฟังก์ชันย่อย	คำอธิบาย
main.c	main	- เริ่มต้นการทำงานด้วยฟังก์ชัน main
system_init.c	System_Init	- กำหนดค่ารีจิสเตอร์ PLL, MAP และ VPB
	PLL_Init	- เริ่มต้นกำหนดค่ารีจิสเตอร์ PLL (CCLK = 60 MHz)
	VPB_Init	- เริ่มต้นกำหนดค่ารีจิสเตอร์ VPB (PCLK = CCLK = 60 MHz)
	Set_Interrupt_Mapping	- กำหนดโหมด Memory mapping control (MAP= User flash mode)
task_cal.c	Calc_Schedule_MTI	- คำนวณค่าการจัดตารางงานโดยเทคนิค MTIs
	Reset_Delay	- รีเซตค่าดีเลย์ก่อนอินพุตงานในระบบ
	Calc_Sch_Major_Cycle	- คำนวณค่าเมเจอร์ไซเคิล
	GCD_New	- คำนวณค่าหารร่วมมากครั้งใหม่
	Calc_Total_Tasks	- คำนวณค่าจำนวนงานทั้งหมดในระบบ
	Calc_Tick_Interval	- คำนวณค่าสัญญาณรายคาบที่เหมาะสม
	Calc_Period_to_Tick	- เปลี่ยนค่าคาบเวลาเป็นค่าการเกิดสัญญาณ Tick
	Calc_Task_RT	- คำนวณค่าเวลาเริ่มต้นในการทำงานแต่ละงาน
	Calc_Max_Wcet	- คำนวณค่าเวลาสูงสุดในการดำเนินงาน
	Calc_Utilisation	- คำนวณค่าอรรถประโยชน์ซีพียู
	gcd	- นำข้อมูลสองค่ามาคำนวณค่าหารร่วมมาก (GCD)
	LCM_2_Numbers	- นำข้อมูลสองค่ามาคำนวณค่าคูณร่วมน้อย (LCM)
sch_coop.c	SCH_Add_Task	- อินพุตงานเข้าในระบบ
	SCH_Init	- เริ่มต้นกำหนดค่ารีจิสเตอร์สำหรับการจัดตารางเวลา - จัดเตรียมค่า match register (T1MRO) - Interrupt on match, and automatically restart counter (T1MCR)
	SCH_Start	- เริ่มต้นการทำงานของระบบด้วยรีจิสเตอร์ control Register Enable (T1TCR)
	SCH_Go_To_Sleep	- เรียกใช้งานฟังก์ชันนี้เพื่อเข้าไปสู่โหมดประหยัดพลังงาน
	SCH_Tick_Update	- ฟังก์ชันรองรับการเกิดอินเตอร์รัพต์จาก Tick interrupt ทำหน้าที่กำหนดสัญญาณรายคาบ

ตาราง 4.2 มอดูลต้นฉบับภาษาซีและหน้าที่การทำงานของฟังก์ชันต่างๆ (ต่อ)

ไฟล์	ฟังก์ชันย่อย	คำอธิบาย
	SCH_Task_Update	- ฟังก์ชันรองรับการเกิดอินเตอร์รัพต์จาก Task interrupt ทำหน้าที่กระตุ้นการดำเนินงานของแต่ละงาน
	Add _New_Task	- อินพุตงานเข้าในระบบ
	SCH_Delete_Task	- ลบงานออกจากระบบ
	Task_Set	- เซตของงานทั้งหมดในระบบ
delay.c	Hardware_Delay_T0	- กำหนดค่าเวลาสูงสุดในการดำเนินงานของงานต่างๆ ผ่าน ไทเมอร์ T0

4.2 การประเมินประสิทธิภาพอัลกอริทึม TTSA-MTI

ในบทที่ผ่านมาได้นำเสนออัลกอริทึมเพื่อทดสอบความสามารถในการจัดตารางการทำงานของระบบ TTC-MTIs โดยในการจัดตารางการทำงานจำเป็นต้องรู้ค่าโอเวอร์เฮดของระบบซึ่งประกอบด้วยโอเวอร์เฮดของ Tick interrupt และโอเวอร์เฮดของ Task interrupt ซึ่งค่าโอเวอร์เฮดเหล่านี้สามารถคำนวณได้ดังสมการ 3.7

1. โอเวอร์เฮดของ Tick interrupt

โอเวอร์เฮดของ Tick interrupt คือค่าเวลาของสัญญาณรายคาบที่เกิดขึ้นตั้งแต่ซีพียูให้บริการอินเตอร์รัพต์จนถึงการตรวจสอบงานในตารางงาน สำหรับการประมาณค่าโอเวอร์เฮดของ Tick interrupt ทำได้โดยการสุ่มค่าอินพุตงานเข้าไปในระบบโดยการกำหนดคุณลักษณะงานชนิดตัวแปรสุ่มที่มีการกระจายแบบยูนิฟอร์มครั้งละ 1 งานจำนวน 100 งาน จากนั้นทำการวัดค่าเวลาของการเกิดโอเวอร์เฮดของ Tick interrupt ในแต่ละงาน โดยใช้เครื่องมือจำลองการทำงาน (Performance and logic analyzer) ของโปรแกรม RealView MDK จากการทดลองได้ค่าเวลาการเกิดโอเวอร์เฮดของ Tick interrupt ของแต่ละงานแสดงดังตาราง 4.3

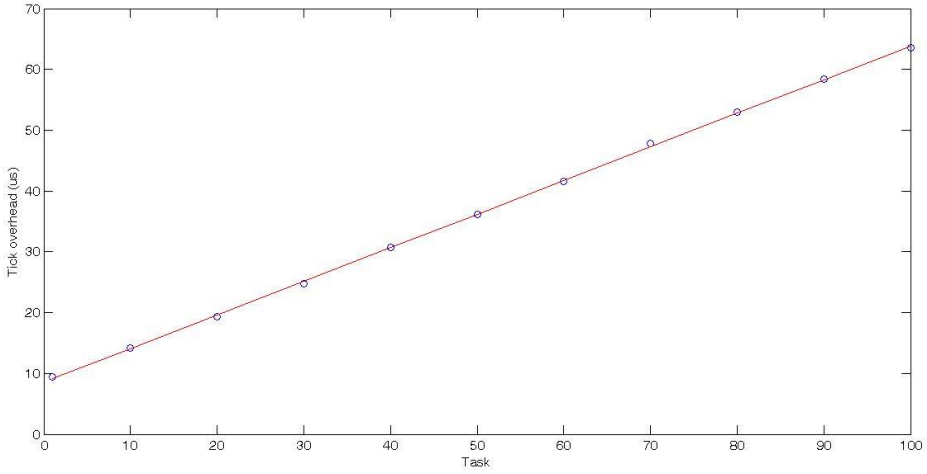
ตาราง 4.3 ค่าเวลาการเกิดโอเวอร์เฮดของ Tick interrupt จำนวน 100 งาน

งาน	โอเวอร์เฮดของ Tick interrupt (µs)		
	Mean	Maximum	Minimum
1	9.38	9.42	9.35
10	14.20	14.23	14.17
20	19.30	19.35	19.25
30	24.78	24.83	24.74
40	30.69	30.74	30.65
50	36.17	36.19	36.14
60	41.52	41.58	41.48
70	47.79	47.85	47.72
80	53.01	53.05	52.97
90	58.36	58.39	58.33
100	63.59	63.62	63.55

จากตาราง 4.3 จะเห็นว่าค่าเวลาโอเวอร์เฮดของ Tick interrupt มีลักษณะเพิ่มขึ้นแบบเชิงเส้นเมื่อจำนวนงานในระบบเพิ่มขึ้น ดังนั้นจึงใช้การวิเคราะห์การถดถอยเชิงเส้น (Linear regression analysis) ในการประมาณค่าเวลาโอเวอร์เฮดของ Tick interrupt จากการวิเคราะห์โดยโปรแกรม MATLAB ได้สมการเส้นตรงดังสมการ 4.1 และกราฟสมการถดถอยแสดงดังภาพประกอบ 4.4

$$Overhead_{tick} = (0.55 * n) + 8.55 \quad (4.1)$$

โดยที่ n คือจำนวนงานที่อินพุตในระบบ



ภาพประกอบ 4.4 สมการเส้นตรงของค่าเวลาโอเวอร์เฮดของ Tick interrupt

2. โอเวอร์เฮดของ Task interrupt

โอเวอร์เฮดของ Task interrupt คือค่าเวลาของสัญญาณที่เป็นตัวกระตุ้นการดำเนินงานของแต่ละงานตั้งแต่ซีพียูให้บริการอินเตอร์รัพต์จนกระทั่งงานแต่ละงานเริ่มดำเนินการ ในการประมาณค่าโอเวอร์เฮดของ Task interrupt ทำได้ในลักษณะเดียวกับการหาโอเวอร์เฮดของ Tick interrupt ตาราง 4.4 แสดงค่าเวลาการเกิดโอเวอร์เฮดของ Task interrupt ในแต่ละครั้ง

ตาราง 4.4 ค่าเวลาการเกิดโอเวอร์เฮดของ Task interrupt

โอเวอร์เฮดของ Task interrupt (µs)		
Mean	Maximum	Minimum
6.38	6.40	6.36

4.2.1 การประเมินประสิทธิภาพด้านเวลาในการประมวลผล

ในหัวข้อนี้ได้ทำการทดสอบอัลกอริทึมเพื่อประเมินประสิทธิภาพด้านเวลาในการจัดตารางการทำงาน (Scheduling time) ระหว่างวิธีการจัดตารางงานที่นำเสนอ (TTSA-MTI) เปรียบเทียบกับวิธีการที่ผ่านมา (TTC-Dispatch) โดยอินพุตในการทดลองใช้วิธีการสร้างเซตของงานที่ถูกกำหนดค่าพารามิเตอร์ตามสมการที่ 4.2 และ 4.3 ตามลำดับ

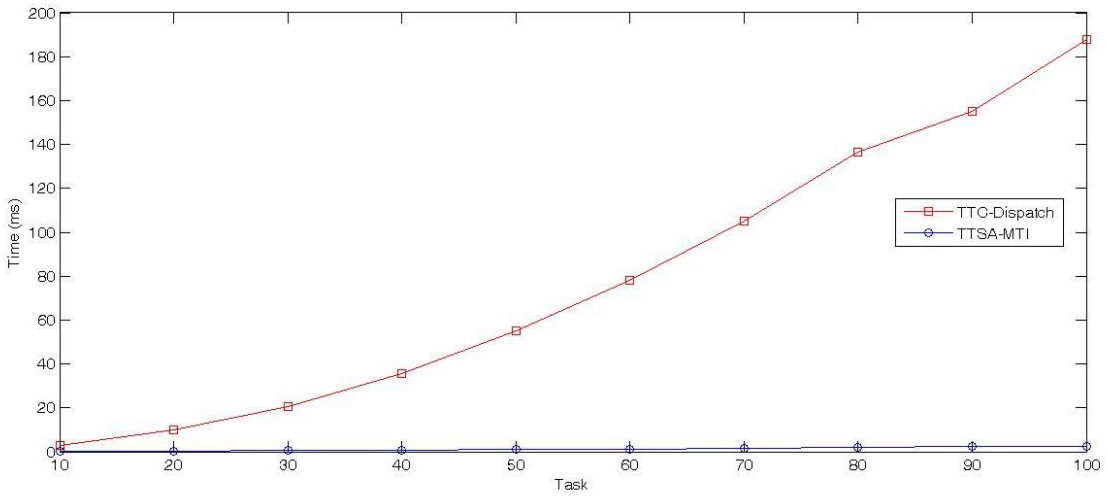
$$0 < WCET(i) < 1000\ us \tag{4.2}$$

$$WCET(i) < P(i) < 10000\ us \tag{4.3}$$

การทดสอบประสิทธิภาพด้านเวลาในการจัดตารางการทำงาน ทดลองโดยการสุ่มค่าเซตของงานเข้าไปในระบบโดยการกำหนดคุณลักษณะงานชนิดตัวแปรสุ่มที่มีการกระจายแบบยูนิฟอร์ม ครั้งละ 1 งานจำนวน 100 งาน จากนั้นทำการเปรียบเทียบค่าเวลาในการจัดตารางการทำงานของอัลกอริทึมที่นำเสนอกับวิธีการที่ผ่านมา ผลการทดลองแสดงดังตาราง 4.5 และภาพประกอบ 4.5 ตามลำดับ จากการทดลองพบว่าอัลกอริทึมที่นำเสนอใช้เวลาในการจัดตารางการทำงานต่ำกว่าวิธีการที่ผ่านมาประมาณ 94 เท่าที่งานจำนวน 100 งาน

ตาราง 4.5 เวลาที่ใช้ในการจัดตารางการทำงานของอัลกอริทึม TTC-Dispatch และ TTSA-MTI

Tasks (n)	TTC-Dispatch (ms)			TTSA-MTI (ms)		
	Mean ± SD	Maximum	Minimum	Mean ± SD	Maximum	Minimum
10	2.83 ± 0.014	2.85	2.81	0.144 ± 0.002	0.146	0.141
20	9.77 ± 0.015	9.80	9.75	0.317 ± 0.001	0.319	0.315
30	20.77 ± 0.015	20.80	20.76	0.513 ± 0.002	0.515	0.510
40	35.77 ± 0.017	35.80	35.75	0.733 ± 0.001	0.735	0.730
50	54.87 ± 0.015	54.89	54.85	0.979 ± 0.002	0.983	0.977
60	77.95 ± 0.018	77.98	77.92	1.248 ± 0.002	1.251	1.247
70	105.21 ± 0.014	105.23	105.19	1.547 ± 0.001	1.549	1.545
80	136.37 ± 0.011	136.39	136.35	1.865 ± 0.002	1.868	1.863
90	155.20 ± 0.018	155.23	155.17	2.352 ± 0.002	2.357	2.350
100	187.80 ± 0.020	187.83	187.78	2.622 ± 0.001	2.623	2.620

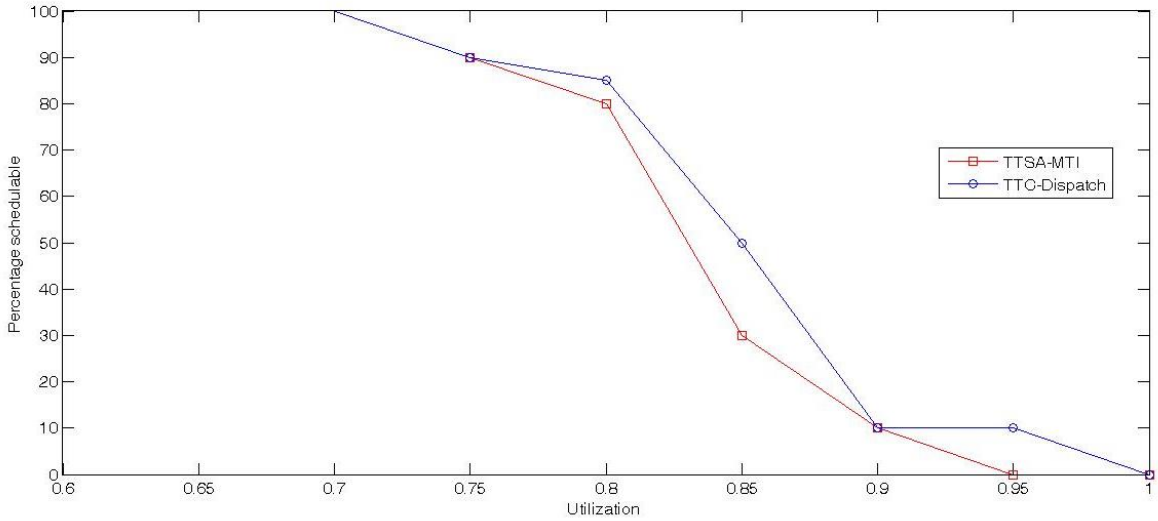


ภาพประกอบ 4.5 กราฟแสดงเวลาที่ใช้ในการคำนวณของอัลกอริทึม TTC-Dispatch และ TTSA-MTI

4.2.2 การประเมินความสามารถในการจัดตารางการทำงาน

การทดสอบความสามารถในการจัดตารางการทำงาน ทดลองโดยการสุ่มค่าเซตของงานจำนวนเซตละ 10 งานเข้าไปในระบบโดยการกำหนดคุณลักษณะงานตามค่าพารามิเตอร์ตามสมการที่ 4.1 และ 4.2 เพื่อนำมาเป็นอินพุตในการทดลอง โดยการทดสอบความสามารถในการจัดตารางการทำงานได้กำหนดค่าอรรถประโยชน์ของซีพียูตั้งแต่ 0-1 ผลการทดสอบแสดงดังภาพประกอบ 4.6 พบว่าอัลกอริทึมที่ TTSA-MTI มีความสามารถในการจัดตารางการทำงานได้ใกล้เคียงกับ TTC-Dispatch

อย่างไรก็ตามพบว่าที่ค่าอรรถประโยชน์ของซีพียูตั้งแต่ 0.75 ขึ้นไปความสามารถในการจัดตาราง การทำงานของวิธีการ TTC-Dispatch จะมีประสิทธิภาพสูงกว่าเล็กน้อย



ภาพประกอบ 4.6 กราฟแสดงความสามารถในการจัดตารางการทำงานของอัลกอริทึม TTC-Dispatch และ TTSA-MTI

4.3 การทดสอบประสิทธิภาพระบบ TTC-MTIs

การทดสอบประสิทธิภาพของระบบ ได้แก่ การทดสอบประสิทธิภาพของหน่วยความจำ ซีพียู และความผิดพลาดทางเวลาขณะเริ่มต้นทำงานโดยทำการเปรียบเทียบระหว่างระบบที่นำเสนอกับวิธีการที่ผ่านมา

4.3.1 การประเมินประสิทธิภาพหน่วยความจำในการประมวลผล

การทดสอบประสิทธิภาพหน่วยความจำที่ต้องใช้ในการประมวลผล (Space complexity) ทำได้โดยการวัดความต้องการการใช้หน่วยความจำของระบบ TTC-MTIs เปรียบเทียบกับระบบ TTC-Dispatch ผลการทดลองแสดงได้ดังตาราง 4.6 จะเห็นว่าระบบที่นำเสนอใช้หน่วยความจำโปรแกรม (ROM) มากกว่าระบบ TTC-Dispatch ประมาณ 2.3 กิโลไบต์ แต่ใช้หน่วยความจำข้อมูล (RAM) น้อยกว่าประมาณ 400 ไบต์ โดยเมื่อนำระบบที่นำเสนอเปรียบเทียบกับซีพียูที่นำมาใช้งาน (LPC2129) พบว่าระบบที่นำเสนอใช้หน่วยความจำ ROM และ RAM ประมาณ 5.57 เปอร์เซ็นต์ และ 20.88 เปอร์เซ็นต์ของหน่วยความจำของซีพียูที่ใช้งานจริงตามลำดับ ซึ่งชี้ให้เห็นว่าระบบที่นำเสนอสามารถรองรับการทำงานบนซีพียู LPC2129 ได้อย่างมีประสิทธิภาพ

ตาราง 4.6 ผลการทดลองเปรียบเทียบประสิทธิภาพหน่วยความจำของระบบ TTC-Dispatch และ TTC-MTIs

Scheduler	TTC-Dispatch	TTC-MTIs
ROM (bytes)	11,912	14,272
RAM (bytes)	3,732	3,340

4.3.2 การทดสอบประสิทธิภาพของซีพียู

การทดสอบทำได้โดยการทดสอบประสิทธิภาพระบบ TTC-MTIs เปรียบเทียบกับ TTC-Dispatch โดยการทดลองได้ทำการทดสอบระบบเป็นเวลา 25 วินาที จากนั้นทำการวัดประสิทธิภาพในการจัดตารางเวลาซึ่งได้ผลการทดลองดังตาราง 4.7 จากการทดลองพบว่าค่าโอเวอร์เฮดในการจัดตารางการทำงานของทั้งสองระบบมีค่าใกล้เคียงกันที่ประมาณ 30 เปอร์เซ็นต์

ตาราง 4.7 ผลการทดลองเปรียบเทียบประสิทธิภาพซีพียูของระบบ TTC-Dispatch และ TTC-MTIs

Scheduler	TTC-Dispatch	TTC-MTIs
Scheduler time (วินาที)	7.45	7.40
Total time (วินาที)	25	25
Scheduler overhead (%)	29.80	29.60

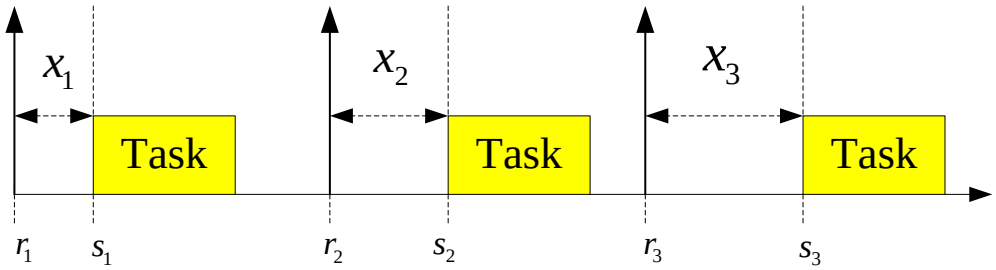
4.3.3 การทดสอบด้านความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน

ค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน แสดงดังภาพประกอบ 4.7 โดยกำหนดให้ s คือค่าเวลาเริ่มต้นทำงาน r คือค่าเวลาในการกระตุ้นการทำงาน และ x คือค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน ซึ่งโดยทั่วไปค่าความผิดพลาดทางเวลาสามารถแสดงได้ในรูปของ Relative release jitter และ Absolute release jitter

Relative release jitter คือค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานที่สูงสุดของงานใดๆ ที่ต่อเนื่องกันสองงาน ส่วน Absolute release jitter คือค่าความผิดพลาดสัมบูรณ์ทางเวลาขณะเริ่มต้นทำงานแสดงดังสมการ 4.4 และ 4.5 ตามลำดับ

$$RRj_i = max_k \left| (s_{i,k} - r_{i,k}) - (s_{i,k-1} - r_{i,k-1}) \right| \quad (4.4)$$

$$ARj_i = max_k (s_{i,k} - r_{i,k}) - min_k (s_{i,k} - r_{i,k}) \quad (4.5)$$

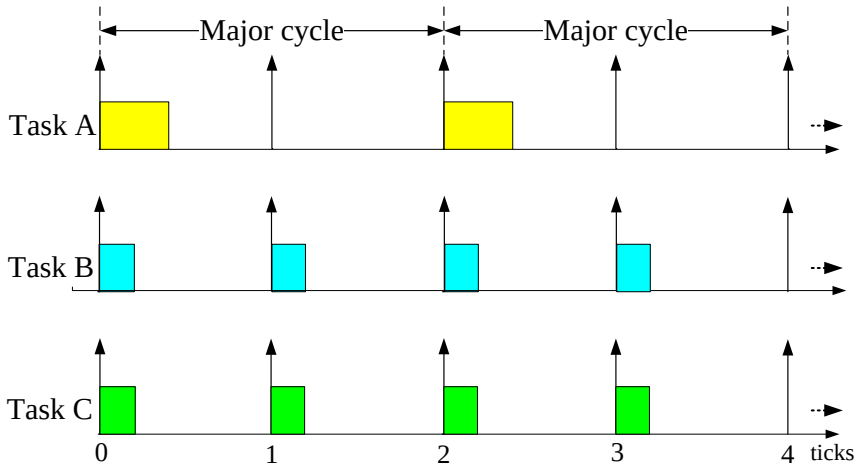


ภาพประกอบ 4.7 ค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน

สำหรับการทดสอบความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน ทดลองโดยการสุ่มค่าเซตของงานเข้าไปในระบบโดยแต่ละเซตประกอบด้วยคุณลักษณะงานตามตาราง 4.8 และภาพประกอบ 4.8 ตามลำดับ โดยแต่ละเซตจะประกอบด้วย 3 งานได้แก่ Task A, Task B และ Task C ซึ่งในการทดลองกำหนดให้ Task A มีความสำคัญสูงสุดในขณะที่ Task C มีความสำคัญต่ำสุด

ตาราง 4.8 คุณลักษณะงานในการทดสอบความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน

Task name	WCET (µs)	Periods (µs)	Priority
Task A	0 < WCET < 300	2,000	Highest
Task B	0 < WCET < 100	1,000	Lower
Task C	0 < WCET < 100	1,000	Lowest



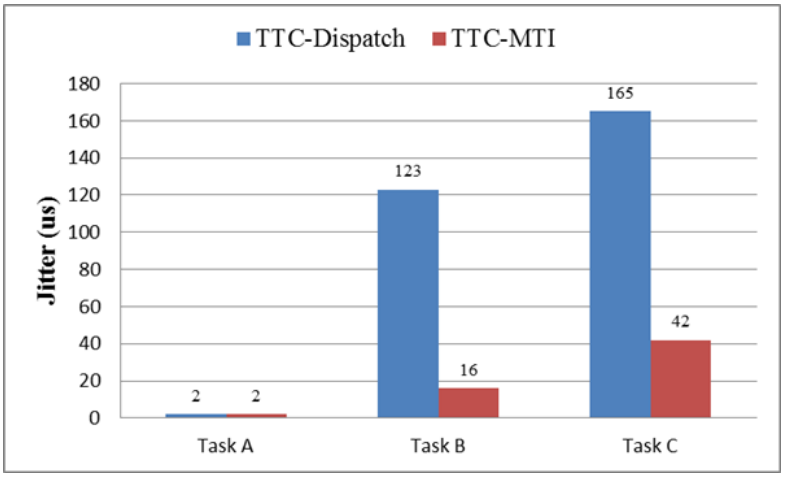
ภาพประกอบ 4.8 รูปแบบของงานแต่ละเซตเพื่อใช้ทดสอบความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน

การทดสอบความผิดพลาดทางเวลาขณะเริ่มต้นทำงานทดลองโดยใช้เครื่องมือจำลองการทำงานของโปรแกรม RealView MDK วัดค่า Relative release jitter ของแต่ละงานจำนวน 10,000 คาบเวลาต่อเนื่องกัน จากนั้นทำการเปรียบเทียบค่าความผิดพลาดทางเวลาขณะเริ่มต้น

ทำงานในการจัดตารางการทำงานของอัลกอริทึมที่นำเสนอกับวิธีการที่ผ่านมา ผลการทดลองแสดงดังตาราง 4.9 และภาพประกอบ 4.9 ตามลำดับ จากการทดลองพบว่าวิธีการที่นำเสนอสามารถลดค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานที่เกิดจากการจัดลำดับการดำเนินการของงานอันเนื่องมาจากผลกระทบของการดำเนินงานก่อนหน้าได้อย่างมีประสิทธิภาพ

ตาราง 4.9 ความผิดพลาดทางเวลาขณะเริ่มต้นทำงานของระบบ TTC-DISPATCH และ TTC-MTIS

Scheduler	TTC-Dispatch			TTC-MTIs		
Tasks	Task A	Task B	Task C	Task A	Task B	Task C
Min Period (μs)	1,999	934	930	1,999	989	973
Max period (μs)	2,001	1,057	1,095	2,001	1005	1,015
Average Period (μs)	2,000	995.5	1012.5	2,000	997	992.5
Absolute jitter (μs)	2	123	165	2	16	42



ภาพประกอบ 4.9 กราฟแสดงการเปรียบเทียบค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานของระบบ TTC-DISPATCH และ TTC-MTIS

4.4 สรุป

ในบทนี้ได้เสนอผลการวิจัยและการอภิปรายผลของการพัฒนาระบบการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์สำหรับประยุกต์ใช้งานกับสถาปัตยกรรมแบบ TTC เพื่อแก้ปัญหาความเปราะบางในการออกแบบระบบการจัดเวลาการทำงานและค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน

จากการประเมินและทดสอบประสิทธิภาพของระบบที่นำเสนอ ประสิทธิภาพของหน่วยความจำ ซีพียู และการทดสอบด้านความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน พบว่าระบบที่นำเสนอสามารถลดปัญหาความผิดพลาดทางเวลาขณะเริ่มต้นทำงานซึ่งเป็นปัญหาจากงานวิจัย

ที่ผ่านมาได้ นอกจากนี้อัลกอริทึมที่นำเสนอยังมีประสิทธิภาพด้านเวลาในการจัดตารางการทำงาน ที่สูงกว่าวิธีการที่ผ่านมาโดยที่ความสามารถในการจัดตารางการทำงานใกล้เคียงกัน

บทที่ 5

สรุปผล อภิปรายผล และข้อเสนอแนะ

การพัฒนาระบบการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์สำหรับประยุกต์ใช้งานกับสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงาน ออกแบบมาเพื่อแก้ปัญหาดังนี้

1. ปัญหาความเปราะบางในการออกแบบระบบการจัดเวลาการทำงาน
2. ปัญหาค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน

5.1 การออกแบบสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะที่เหมาะสม

ในการออกแบบสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงานเพื่อไปประยุกต์ใช้กับระบบฝังตัวที่มีทรัพยากรจำกัด นักพัฒนาส่วนใหญ่นิยมใช้ระบบการจัดเวลาชนิดกระตุ้นด้วยเวลาแบบโปรแกรมเลือกจ่ายงานหรือแบบ TTC-Dispatch เพราะว่าสามารถออกแบบระบบได้ง่ายและมีความน่าเชื่อถือสูง อย่างไรก็ตามถึงแม้ว่าสถาปัตยกรรมแบบ TTC-Dispatch จะเป็นระบบที่ไม่ซับซ้อน แต่ระบบดังกล่าวยังมีข้อจำกัดเรื่องความเปราะบางในสถานการณ์ที่เกิดสภาวะโอเวอร์รัน ซึ่งอาจทำให้ระบบไม่สามารถดำเนินการจัดตารางเวลาได้ตามที่ออกแบบไว้และอาจเกิดการสูญเสียกับระบบงานได้

งานวิจัยนี้จึงได้นำเสนอวิธีการออกแบบอัลกอริทึมและการจัดสร้างระบบการจัดเวลาการทำงานแบบใหม่โดยการทำงานจะอาศัยเทคนิคการอินเตอร์รัพต์แบบหลายไทเมอร์เพื่อแยกการทำงานระหว่างส่วนควบคุมการจัดการเวลาและส่วนของงานแต่ละงานออกจากกันซึ่งสามารถลดผลกระทบการเกิดการโอเวอร์รันของงานและค่าความผิดพลาดทางเวลาขณะเริ่มต้นทำงานได้

5.2 การปรับปรุงประสิทธิภาพสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะ

การออกแบบระบบการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์มีวัตถุประสงค์เพื่อปรับปรุงประสิทธิภาพในด้านต่างๆ ได้แก่

1. การปรับปรุงประสิทธิภาพด้านความเปราะบางในการออกแบบระบบการจัดเวลาการทำงาน

การปรับปรุงประสิทธิภาพทำได้โดยออกแบบสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงานทำการแยกส่วนการทำงานระหว่างส่วนควบคุมการกระตุ้นการทำงานและส่วนของงานแต่ละงานออกจากกัน เนื่องจากสถาปัตยกรรมดังกล่าวได้กำหนดความสำคัญของงานไว้ก่อนล่วงหน้าว่าแต่ละงานจะถูกดำเนินการในเวลาใด ดังนั้นเมื่อเกิดการโอเวอร์รันขึ้นในงานใดๆ ระบบดังกล่าวก็จะอนุญาตให้งานที่มีความสำคัญกว่าสามารถแทรกการทำงานของงานที่เกิดโอเวอร์รันทำให้ระบบการทำงานนี้สามารถกลับมาทำงานได้ตามเวลาที่กำหนดและไม่ส่งผลกระทบกับงานที่จะถูกดำเนินงานต่อมา ซึ่งทำให้สามารถลดผลกระทบด้านความเปราะบางในการออกแบบระบบได้อย่างมีประสิทธิภาพ

2. การปรับปรุงประสิทธิภาพด้านความผิดพลาดทางเวลาขณะเริ่มต้นทำงาน

การปรับปรุงประสิทธิภาพทำได้โดยการกำหนดค่าเวลาก่อนดำเนินงานของแต่ละงานให้มีค่าเท่ากันเพื่อลดผลกระทบของคาบเวลาไม่เท่ากันของแต่ละงาน โดยงานวิจัยนี้ได้เสนอเทคนิคแซนวิชดีเลย์มาประยุกต์ใช้งาน ซึ่งสามารถทำโดยการกำหนดค่าเวลาของแซนวิชดีเลย์ไว้ก่อนหน้าที่งานจะถูกดำเนินการซึ่งทำให้คาบเวลาของแต่ละงานให้มีค่าเท่ากัน

5.3 ข้อเสนอแนะ

ระบบการจัดเวลาการทำงานโดยอาศัยการอินเตอร์รัพต์แบบหลายไทเมอร์สำหรับประยุกต์ใช้งานกับสถาปัตยกรรมกระตุ้นเวลาแบบไม่ขัดจังหวะการทำงานมีความเหมาะสมที่จะนำไปใช้สำหรับระบบฝังตัวที่มีทรัพยากรจำกัดที่ต้องการการคาดการณ์และความน่าเชื่อถือสูง โดยในงานวิจัยนี้ได้เน้นการนำไปประยุกต์ใช้ในระบบอิเล็กทรอนิกส์เชิงเกษตรกรรม ได้แก่ การประยุกต์ใช้เซนเซอร์ความเร่งในการคัดแยกพฤติกรรมโคหรือตรวจสอบความเป็นสัดของโค [1, 2] ซึ่งระบบดังกล่าวสามารถรองรับการนำไปประยุกต์ใช้งานได้อย่างมีประสิทธิภาพ

เอกสารอ้างอิง

เอกสารอ้างอิง

1. Laplante PA. Real-Time Systems Design and Analysis: Institute of Electrical and Electronics Engineers; 2004.
2. Albert A, editor. Comparison of Event-triggered and Time-triggered Concepts with Regard to Distributed Control Systems. Proceedings of Embedded World; 2004; Nurnberg, Germany.
3. Kopetz H. Real-Time Systems: Design Principles for Distributed Embedded Applications: Kluwer Academic; 1997.
4. Cottet F, Delacroix, J., Kaiser, C., Mammeri, Z. Scheduling in real-time systems. 2002.
5. Phatrapornnant T, Pont MJ. Reducing Jitter in Embedded Systems Employing a Time-Triggered Software Architecture and Dynamic Voltage Scaling. IEEE. 2006.
6. Pont MJ. Applying time-triggered architectures in reliable embedded systems: challenges and solutions. e & i Elektrotechnik und Informationstechnik. 2008;125(11):401-5.
7. Pont MJ. Reducing the time taken to test your next embedded system. TTE System Ltd; 2010; Available from: http://www.tte-systems.com/downloads.
8. Buttazzo G. Rate Monotonic vs. EDF: Judgment Day. Real-Time Systems. 2005 2005/01/01;29(1):5-26.
9. Nahas M. Employing Two 'Sandwich Delay' Mechanisms to Enhance Predictability of Embedded Systems Which Use Time-Triggered Co-Operative Architectures. Journal of Software Engineering and Applications. 2011a;04(07):417-25.
10. Hughes ZM, Pont MJ. Reducing the Impact of Task Overruns in Resource-Constrained Embedded Systems in which a Time-Triggered Software Architecture is Employed. Transactions of the Institute of Measurement and Control. 2008;30(5):427-50.
11. Locke CD. Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives. Real-Time Syst. 1992;4(1):37-53.
12. Park M, Cho Y. Feasibility analysis of hard real-time periodic tasks. Journal of Systems and Software. 2004;73(1):89-100.
13. Liu CL, Layland JW. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. J ACM. 1973;20(1):46-61.
14. Joshi A. Embedded Systems: Technologies and Markets. USA: BCC Research; 2012; Available from: http://www.bccresearch.com/report/embedded-systems-technologies-markets-ift016d.html.

15. Pont MJ. Patterns For Time-triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers: ACM Press Books; 2001.
16. Kopetz H, Bauer G. The time-triggered architecture. Proceedings of the IEEE. 2003;91(1):112-26.
17. Maaita A. Techniques for Enhancing the Temporal Predictability of Real-Time Embedded Systems Employing a Time-Triggered Software Architecture. Leicester: University of Leicester; 2008.
18. Scarlett JJ, Brennan RW, editors. Re-evaluating Event-Triggered and Time-Triggered Systems. Emerging Technologies and Factory Automation, 2006 ETFA '06 IEEE Conference on; 2006 20-22 Sept. 2006.
19. Liu JWS. Real-time system: Prentice Hall; 2000.
20. Buttazzo GC. Hard Real Time Computing Systems Predictable Scheduling Algorithms and Applications. Third ed: Springer; 2011.
21. Li Q, Yao C. Real Time Concepts for Embedded Systems USA: CMP Books; 2003.
22. Nguyen T, Anh B, Tan S. Real-Time Operating Systems for Small Microcontrollers. IEEE Computer Society. 2009.
23. Nahas M. Implementation of highly-predictable time-triggered cooperative scheduler using simple super loop architecture. International Journal of Electrical & Computer Sciences IJECS-IJENS. 2011.
24. Pont MJ. Embedded C: Pearson Education; 2002.
25. หมุนสนิท พ, พงศาสกุลชัย ส, เลี้ยงอยู่ อ, ภักดีวัฒนะกุล ก. ระบบปฏิบัติการ: เคทีพี; 2010.
26. ตู้จินดา ว. วิศวกรรมไฟฟ้าและคอมพิวเตอร์ประยุกต์: สำนักพิมพ์มหาวิทยาลัยเกษตรศาสตร์; 2011.
27. Baker TP, Shaw A, editors. The cyclic executive model and Ada. Real-Time Systems Symposium, 1988, Proceedings; 1988 6-8 Dec 1988.
28. Burns A, Hayes N, Richardson MF. Generating feasible cyclic schedules. Control Engineering Practice. 1995;3(2):151-62.
29. Zamorano J, Alonso A, de la Puente JA. Building Safety-critical Real-time Systems with Reusable Cycle Executives. Control Engineering Practice. 1997;5(7):999-1005.
30. Gendy AK, Pont MJ. Automatically Configuring Time-Triggered Schedulers for Use With Resource-Constrained, Single-Processor Embedded Systems. IEEE. 2008a.
31. Gendy AK, Dong L, Pont MJ. Improvoing the performance of time-triggered embedded system by means of a scheduler agent. 2007.

32. Gendy AK. Techniques for scheduling time-triggered resource-constrained embedded systems Leicester: University of Leicester; 2009.
33. Pont MJ, Kurian S, Bautista-Quintero R. Meeting Real-Tme Constraints Using Sandwich Delays. In: James N, Ralph J, editors. Transactions on Pattern Languages of Programming I: Springer-Verlag; 2009. p. 94-102.
34. Kurian S, Pont MJ. Restructuring a pattern language which supports time-triggered co-operative software architectures in resource-constrained embedded systems. EuroPLoP 20062006.
35. Lakhani F, Wang H, Pont M, J. Supporting the migration between 'event triggered' and 'time triggered' software architectures: a small pattern collection intended for use by the developers of reliable embedded systems. 2012.
36. Lakhani F, Das A, Pont MJ. Improving the reliability of embedded systems as complexity increases: supporting the migration between event-triggered and time-triggered software architectures. Proceedings of the 15th European Conference on Pattern Languages of Programs; Irsee, Germany. 2328937: ACM; 2010. p. 1-17.
37. Nahas M. Bridging the gap between scheduling algorithms and scheduler implementations in time-triggered embedded systems. Leicester: University of Leicester; 2008.
38. NXP. LPC21xx and LPC22xx User manual 2012. Available from: www.nxp.com/documents/user_manual/UM10114.pdf.
39. Keil. Embedded Development Tools. 2012; Available from: www.keil.com.
40. Kuankid S, Rattanawong T, Aurasopon A, editors. Classification of the cattle's behaviors by using accelerometer data with simple behavioral technique. Asia-Pacific Signal and Information Processing Association, 2014 Annual Summit and Conference (APSIPA); 2014: IEEE.
41. Dulyala R, Kuankid S, Rattanawong T, Aurasopon A, editors. Classification system for estrus behavior of cow using an Accelerometer. Asia-Pacific Signal and Information Processing Association, 2014 Annual Summit and Conference (APSIPA); 2014: IEEE.

ภาคผนวก

เซตของงานที่ใช้สำหรับการทดลอง

1. เซตของงานที่ใช้สำหรับการทดลองหัวข้อ 4.2.1

1) Task set #1

Task	1	2	3	4	5	6	7	8	9	10
WCET (µs)	23	27	23	9	5	25	24	8	7	10
Period (µs)	1000	5000	5000	5000	1000	10000	1000	2000	10000	1000
Task	11	12	13	14	15	16	17	18	19	20
WCET (µs)	25	25	17	17	9	21	24	13	13	14
Period (µs)	1000	10000	1000	1000	2000	1000	2000	5000	2000	2000
Task	21	22	23	24	25	26	27	28	29	30
WCET (µs)	8	20	27	27	22	8	21	4	4	6
Period (µs)	5000	1000	2000	1000	2000	5000	10000	1000	1000	2000
Task	31	32	33	34	35	36	37	38	39	40
WCET (µs)	4	18	2	25	22	28	15	20	27	16
Period (µs)	10000	2000	10000	1000	1000	1000	1000	1000	1000	2000
Task	41	42	43	44	45	46	47	48	49	50
WCET (µs)	8	29	1	10	29	11	9	4	27	4
Period (µs)	1000	1000	1000	1000	1000	2000	1000	1000	1000	1000
Task	51	52	53	54	55	56	57	58	59	60
WCET (µs)	8	29	1	10	29	11	9	4	27	4
Period (µs)	1000	1000	1000	1000	1000	2000	1000	1000	1000	1000
Task	61	62	63	64	65	66	67	68	69	70
WCET (µs)	10	25	7	17	28	1	2	1	20	18
Period (µs)	1000	1000	2000	2000	1000	2000	10000	1000	2000	1000
Task	71	72	73	74	75	76	77	78	79	80
WCET (µs)	3	24	19	2	2	4	24	3	7	7
Period (µs)	1000	1000	5000	1000	1000	2000	1000	5000	2000	5000
Task	81	82	83	84	85	86	87	88	89	90
WCET (µs)	3	26	21	22	20	15	10	20	4	4
Period (µs)	5000	1000	1000	5000	2000	1000	5000	2000	2000	2000
Task	91	92	93	94	95	96	97	98	99	100
WCET (µs)	1	29	29	4	14	20	9	23	17	13
Period (µs)	5000	5000	1000	1000	5000	1000	5000	1000	5000	1000

2) Task set #2

Task	1	2	3	4	5	6	7	8	9	10
WCET (μs)	4	26	1	21	22	13	11	29	12	13
Period (μs)	10000	1000	1000	5000	1000	1000	2000	5000	1000	2000
Task	11	12	13	14	15	16	17	18	19	20
WCET (μs)	5	10	9	27	7	9	12	21	4	26
Period (μs)	2000	1000	1000	10000	1000	1000	1000	2000	5000	1000
Task	21	22	23	24	25	26	27	28	29	30
WCET (μs)	2	14	1	23	21	6	20	17	26	17
Period (μs)	1000	5000	1000	5000	1000	1000	1000	1000	1000	1000
Task	31	32	33	34	35	36	37	38	39	40
WCET (μs)	27	13	11	15	8	28	14	8	13	21
Period (μs)	5000	1000	5000	5000	5000	10000	1000	1000	10000	2000
Task	41	42	43	44	45	46	47	48	49	50
WCET (μs)	12	5	26	18	11	7	7	16	13	22
Period (μs)	5000	1000	1000	2000	1000	2000	5000	2000	2000	1000
Task	51	52	53	54	55	56	57	58	59	60
WCET (μs)	2	25	20	4	26	6	18	16	5	5
Period (μs)	1000	1000	1000	2000	10000	1000	1000	5000	1000	1000
Task	61	62	63	64	65	66	67	68	69	70
WCET (μs)	23	23	13	2	18	5	22	16	8	28
Period (μs)	5000	1000	5000	1000	1000	2000	2000	2000	2000	1000
Task	71	72	73	74	75	76	77	78	79	80
WCET (μs)	23	27	2	6	22	21	23	15	13	18
Period (μs)	1000	5000	1000	2000	5000	1000	10000	1000	5000	5000
Task	81	82	83	84	85	86	87	88	89	90
WCET (μs)	26	20	16	9	21	11	17	27	25	27
Period (μs)	1000	2000	10000	1000	5000	2000	5000	5000	1000	1000
Task	91	92	93	94	95	96	97	98	99	100
WCET (μs)	28	24	0	5	3	8	1	13	10	16
Period (μs)	2000	1000	10000	5000	2000	1000	1000	5000	1000	5000

3) Task set #3

Task	1	2	3	4	5	6	7	8	9	10
WCET (μs)	3	26	1	27	27	16	4	5	21	25
Period (μs)	5000	5000	1000	1000	1000	5000	1000	1000	2000	1000
Task	11	12	13	14	15	16	17	18	19	20
WCET (μs)	1	23	29	10	19	10	6	24	22	8
Period (μs)	1000	5000	2000	1000	5000	1000	1000	5000	1000	1000
Task	21	22	23	24	25	26	27	28	29	30
WCET (μs)	17	13	3	1	15	8	10	9	5	12
Period (μs)	2000	5000	5000	1000	1000	1000	5000	1000	1000	2000
Task	31	32	33	34	35	36	37	38	39	40
WCET (μs)	21	6	20	13	13	5	6	18	8	17
Period (μs)	1000	5000	10000	5000	5000	5000	5000	1000	1000	1000
Task	41	42	43	44	45	46	47	48	49	50
WCET (μs)	28	21	20	29	23	18	28	2	8	30
Period (μs)	5000	2000	1000	1000	2000	1000	1000	1000	1000	1000
Task	51	52	53	54	55	56	57	58	59	60
WCET (μs)	23	14	20	13	11	6	11	1	14	10
Period (μs)	1000	1000	5000	1000	1000	2000	5000	1000	1000	1000
Task	61	62	63	64	65	66	67	68	69	70
WCET (μs)	29	17	25	12	14	25	30	16	28	22
Period (μs)	1000	5000	1000	2000	2000	2000	2000	5000	2000	5000
Task	71	72	73	74	75	76	77	78	79	80
WCET (μs)	17	29	25	29	19	11	14	27	5	5
Period (μs)	1000	1000	1000	5000	1000	2000	2000	2000	5000	2000
Task	81	82	83	84	85	86	87	88	89	90
WCET (μs)	14	16	2	20	27	3	13	8	30	18
Period (μs)	1000	1000	10000	1000	1000	1000	2000	5000	10000	1000
Task	91	92	93	94	95	96	97	98	99	100
WCET (μs)	8	4	16	25	25	25	6	16	26	4
Period (μs)	5000	2000	1000	2000	1000	2000	5000	1000	1000	5000

4) Task set #4

Task	1	2	3	4	5	6	7	8	9	10
WCET (μs)	12	23	5	3	27	26	30	26	1	16
Period (μs)	10000	5000	2000	1000	1000	2000	1000	1000	10000	5000
Task	11	12	13	14	15	16	17	18	19	20
WCET (μs)	30	15	26	2	30	28	17	13	10	22
Period (μs)	5000	10000	1000	2000	2000	10000	2000	1000	5000	5000
Task	21	22	23	24	25	26	27	28	29	30
WCET (μs)	5	11	28	16	14	15	9	29	29	15
Period (μs)	2000	1000	10000	1000	2000	1000	1000	5000	1000	5000
Task	31	32	33	34	35	36	37	38	39	40
WCET (μs)	30	14	13	6	6	25	22	16	25	15
Period (μs)	5000	1000	1000	1000	5000	1000	2000	2000	5000	10000
Task	41	42	43	44	45	46	47	48	49	50
WCET (μs)	17	6	18	4	2	20	18	7	12	19
Period (μs)	1000	1000	5000	5000	1000	1000	2000	1000	2000	1000
Task	51	52	53	54	55	56	57	58	59	60
WCET (μs)	17	4	5	5	13	15	5	20	1	4
Period (μs)	1000	5000	2000	5000	1000	2000	1000	5000	2000	10000
Task	61	62	63	64	65	66	67	68	69	70
WCET (μs)	29	29	1	15	26	7	25	24	19	5
Period (μs)	1000	10000	2000	2000	1000	1000	5000	1000	1000	2000
Task	71	72	73	74	75	76	77	78	79	80
WCET (μs)	11	27	20	11	19	7	17	29	25	9
Period (μs)	10000	2000	10000	1000	1000	2000	1000	5000	1000	5000
Task	81	82	83	84	85	86	87	88	89	90
WCET (μs)	20	11	30	3	8	24	3	16	24	22
Period (μs)	1000	1000	10000	5000	1000	1000	5000	2000	1000	1000
Task	91	92	93	94	95	96	97	98	99	100
WCET (μs)	4	13	11	14	18	4	27	19	5	17
Period (μs)	10000	1000	1000	1000	1000	5000	1000	2000	1000	1000

5) Task set #5

Task	1	2	3	4	5	6	7	8	9	10
WCET (μs)	6	10	3	11	10	10	25	16	16	23
Period (μs)	5000	10000	1000	5000	2000	1000	5000	10000	1000	5000
Task	11	12	13	14	15	16	17	18	19	20
WCET (μs)	4	19	10	10	17	26	6	20	27	6
Period (μs)	1000	1000	1000	5000	10000	2000	5000	10000	2000	2000
Task	21	22	23	24	25	26	27	28	29	30
WCET (μs)	9	15	27	15	8	16	17	12	0	21
Period (μs)	1000	2000	1000	1000	1000	1000	2000	10000	1000	1000
Task	31	32	33	34	35	36	37	38	39	40
WCET (μs)	15	11	2	11	7	6	24	12	2	11
Period (μs)	2000	1000	1000	2000	5000	1000	1000	1000	2000	2000
Task	41	42	43	44	45	46	47	48	49	50
WCET (μs)	23	5	27	10	10	6	23	2	29	5
Period (μs)	5000	1000	2000	1000	5000	2000	1000	1000	1000	10000
Task	51	52	53	54	55	56	57	58	59	60
WCET (μs)	9	21	4	15	22	28	22	22	12	7
Period (μs)	1000	1000	1000	5000	5000	5000	1000	1000	10000	1000
Task	61	62	63	64	65	66	67	68	69	70
WCET (μs)	16	7	25	20	24	24	14	9	21	30
Period (μs)	2000	1000	1000	1000	5000	10000	5000	1000	1000	1000
Task	71	72	73	74	75	76	77	78	79	80
WCET (μs)	23	25	21	18	23	15	26	2	29	3
Period (μs)	1000	10000	2000	5000	5000	1000	10000	1000	5000	5000
Task	81	82	83	84	85	86	87	88	89	90
WCET (μs)	16	12	3	22	18	23	17	24	17	28
Period (μs)	5000	1000	1000	2000	2000	5000	2000	5000	2000	10000
Task	91	92	93	94	95	96	97	98	99	100
WCET (μs)	26	15	24	14	25	10	29	8	2	23
Period (μs)	1000	2000	1000	1000	2000	1000	2000	1000	5000	1000

6) Task set #6

Task	1	2	3	4	5	6	7	8	9	10
WCET (μs)	29	19	5	25	20	16	8	1	7	11
Period (μs)	10000	5000	1000	5000	10000	10000	2000	2000	1000	2000
Task	11	12	13	14	15	16	17	18	19	20
WCET (μs)	19	30	6	23	27	14	5	24	14	3
Period (μs)	2000	1000	1000	2000	1000	2000	1000	1000	2000	1000
Task	21	22	23	24	25	26	27	28	29	30
WCET (μs)	26	19	3	27	1	1	30	21	11	15
Period (μs)	2000	1000	1000	10000	2000	2000	2000	5000	5000	2000
Task	31	32	33	34	35	36	37	38	39	40
WCET (μs)	23	1	22	21	14	17	10	5	12	28
Period (μs)	1000	1000	5000	1000	1000	10000	5000	1000	10000	1000
Task	41	42	43	44	45	46	47	48	49	50
WCET (μs)	7	11	10	3	15	25	27	22	11	9
Period (μs)	1000	1000	1000	1000	1000	2000	10000	1000	1000	1000
Task	51	52	53	54	55	56	57	58	59	60
WCET (μs)	21	26	28	2	14	4	8	27	6	4
Period (μs)	1000	1000	1000	1000	1000	5000	2000	10000	10000	2000
Task	61	62	63	64	65	66	67	68	69	70
WCET (μs)	16	9	11	24	25	20	13	19	6	19
Period (μs)	1000	10000	2000	5000	1000	2000	1000	10000	2000	1000
Task	71	72	73	74	75	76	77	78	79	80
WCET (μs)	20	18	10	11	5	24	15	11	23	7
Period (μs)	5000	1000	1000	1000	1000	1000	5000	5000	2000	2000
Task	81	82	83	84	85	86	87	88	89	90
WCET (μs)	25	24	25	11	11	26	14	17	21	29
Period (μs)	2000	10000	2000	2000	10000	2000	1000	10000	1000	2000
Task	91	92	93	94	95	96	97	98	99	100
WCET (μs)	16	19	17	28	26	5	5	7	23	6
Period (μs)	5000	2000	1000	1000	1000	5000	2000	1000	1000	5000

7) Task set #7

Task	1	2	3	4	5	6	7	8	9	10
WCET (μs)	20	20	5	13	15	11	14	10	23	12
Period (μs)	5000	1000	2000	1000	5000	10000	1000	5000	1000	1000
Task	11	12	13	14	15	16	17	18	19	20
WCET (μs)	21	14	22	28	16	27	7	26	2	14
Period (μs)	1000	5000	5000	1000	5000	10000	1000	1000	2000	5000
Task	21	22	23	24	25	26	27	28	29	30
WCET (μs)	1	24	5	5	5	21	6	9	23	15
Period (μs)	5000	1000	1000	1000	1000	5000	1000	5000	2000	1000
Task	31	32	33	34	35	36	37	38	39	40
WCET (μs)	27	2	13	17	17	25	4	9	0	29
Period (μs)	5000	10000	1000	5000	1000	5000	10000	5000	1000	1000
Task	41	42	43	44	45	46	47	48	49	50
WCET (μs)	23	23	4	10	5	15	24	19	21	19
Period (μs)	2000	1000	1000	10000	2000	10000	1000	1000	5000	5000
Task	51	52	53	54	55	56	57	58	59	60
WCET (μs)	22	26	19	5	17	5	27	2	10	17
Period (μs)	1000	1000	1000	5000	2000	1000	2000	5000	1000	1000
Task	61	62	63	64	65	66	67	68	69	70
WCET (μs)	14	24	16	7	21	4	20	19	7	5
Period (μs)	5000	1000	5000	10000	1000	1000	1000	2000	1000	1000
Task	71	72	73	74	75	76	77	78	79	80
WCET (μs)	5	4	6	29	0	29	1	29	9	16
Period (μs)	10000	10000	1000	2000	1000	1000	1000	1000	1000	10000
Task	81	82	83	84	85	86	87	88	89	90
WCET (μs)	26	27	6	20	28	29	3	5	22	1
Period (μs)	1000	10000	5000	2000	2000	5000	1000	5000	10000	1000
Task	91	92	93	94	95	96	97	98	99	100
WCET (μs)	2	15	25	30	0	16	26	27	25	26
Period (μs)	5000	5000	10000	1000	10000	2000	1000	1000	5000	5000

8) Task set #8

Task	1	2	3	4	5	6	7	8	9	10
WCET (μs)	27	28	15	19	22	1	17	1	13	14
Period (μs)	1000	1000	5000	1000	2000	1000	1000	2000	1000	1000
Task	11	12	13	14	15	16	17	18	19	20
WCET (μs)	1	2	28	16	11	11	5	4	11	10
Period (μs)	2000	1000	5000	2000	5000	5000	5000	2000	5000	2000
Task	21	22	23	24	25	26	27	28	29	30
WCET (μs)	24	15	14	4	27	20	25	20	30	29
Period (μs)	1000	2000	5000	1000	2000	1000	10000	2000	10000	2000
Task	31	32	33	34	35	36	37	38	39	40
WCET (μs)	8	19	22	15	25	6	4	0	5	16
Period (μs)	5000	1000	2000	1000	2000	10000	2000	1000	5000	1000
Task	41	42	43	44	45	46	47	48	49	50
WCET (μs)	15	12	9	0	24	19	13	7	24	25
Period (μs)	5000	1000	2000	2000	2000	1000	1000	2000	10000	2000
Task	51	52	53	54	55	56	57	58	59	60
WCET (μs)	26	14	29	25	2	7	25	12	14	29
Period (μs)	10000	1000	2000	1000	2000	2000	1000	5000	1000	5000
Task	61	62	63	64	65	66	67	68	69	70
WCET (μs)	29	23	17	27	15	5	10	9	17	2
Period (μs)	1000	1000	1000	1000	10000	1000	1000	1000	1000	1000
Task	71	72	73	74	75	76	77	78	79	80
WCET (μs)	2	5	28	24	21	29	30	30	5	29
Period (μs)	2000	2000	1000	1000	1000	1000	1000	2000	1000	1000
Task	81	82	83	84	85	86	87	88	89	90
WCET (μs)	16	2	9	27	25	0	19	24	7	2
Period (μs)	5000	5000	1000	2000	1000	1000	5000	10000	2000	1000
Task	91	92	93	94	95	96	97	98	99	100
WCET (μs)	8	3	15	13	11	27	13	9	1	7
Period (μs)	1000	2000	1000	1000	2000	1000	2000	1000	1000	1000

9) Task set #9

Task	1	2	3	4	5	6	7	8	9	10
WCET (μs)	2	27	25	12	5	13	12	22	12	29
Period (μs)	1000	2000	2000	5000	2000	5000	10000	1000	1000	5000
Task	11	12	13	14	15	16	17	18	19	20
WCET (μs)	27	29	10	9	27	6	4	16	27	12
Period (μs)	1000	1000	1000	5000	1000	5000	5000	2000	1000	1000
Task	21	22	23	24	25	26	27	28	29	30
WCET (μs)	6	2	28	18	11	20	24	10	21	6
Period (μs)	2000	1000	1000	1000	1000	2000	5000	1000	1000	1000
Task	31	32	33	34	35	36	37	38	39	40
WCET (μs)	29	21	5	13	19	28	16	19	20	28
Period (μs)	2000	5000	1000	1000	5000	2000	2000	1000	1000	5000
Task	41	42	43	44	45	46	47	48	49	50
WCET (μs)	5	12	9	21	27	15	24	10	16	12
Period (μs)	1000	1000	1000	2000	2000	2000	1000	1000	1000	1000
Task	51	52	53	54	55	56	57	58	59	60
WCET (μs)	27	20	25	3	8	23	6	1	13	28
Period (μs)	1000	10000	5000	5000	5000	5000	5000	5000	10000	1000
Task	61	62	63	64	65	66	67	68	69	70
WCET (μs)	8	17	11	1	15	25	8	1	7	20
Period (μs)	1000	1000	1000	1000	10000	5000	1000	1000	5000	1000
Task	71	72	73	74	75	76	77	78	79	80
WCET (μs)	10	20	0	22	12	1	12	21	28	30
Period (μs)	1000	1000	1000	5000	10000	1000	2000	2000	1000	5000
Task	81	82	83	84	85	86	87	88	89	90
WCET (μs)	30	27	26	24	17	13	4	20	26	8
Period (μs)	1000	1000	1000	5000	5000	5000	2000	5000	1000	1000
Task	91	92	93	94	95	96	97	98	99	100
WCET (μs)	25	2	11	8	5	19	9	29	15	22
Period (μs)	1000	1000	5000	1000	2000	1000	1000	5000	1000	5000

10) Task set #10

Task	1	2	3	4	5	6	7	8	9	10
WCET (μs)	10	20	2	22	15	28	9	11	3	29
Period (μs)	5000	5000	10000	1000	5000	1000	2000	2000	1000	2000
Task	11	12	13	14	15	16	17	18	19	20
WCET (μs)	13	3	22	15	10	23	25	8	16	13
Period (μs)	5000	1000	1000	5000	1000	1000	1000	2000	2000	1000
Task	21	22	23	24	25	26	27	28	29	30
WCET (μs)	5	18	28	3	27	17	13	5	7	13
Period (μs)	2000	5000	1000	1000	1000	5000	1000	10000	1000	1000
Task	31	32	33	34	35	36	37	38	39	40
WCET (μs)	16	11	23	26	22	12	18	19	4	15
Period (μs)	5000	2000	1000	2000	1000	10000	5000	2000	2000	10000
Task	41	42	43	44	45	46	47	48	49	50
WCET (μs)	9	17	28	13	1	28	28	11	8	24
Period (μs)	1000	5000	5000	5000	1000	1000	1000	1000	1000	1000
Task	51	52	53	54	55	56	57	58	59	60
WCET (μs)	15	17	21	3	26	28	3	25	27	3
Period (μs)	2000	5000	1000	2000	10000	5000	2000	10000	1000	5000
Task	61	62	63	64	65	66	67	68	69	70
WCET (μs)	16	20	12	19	23	17	19	8	25	13
Period (μs)	1000	1000	5000	1000	1000	1000	10000	2000	5000	10000
Task	71	72	73	74	75	76	77	78	79	80
WCET (μs)	19	25	8	12	17	13	3	22	1	13
Period (μs)	2000	10000	1000	5000	10000	1000	1000	5000	1000	5000
Task	81	82	83	84	85	86	87	88	89	90
WCET (μs)	1	29	16	27	11	27	8	27	27	12
Period (μs)	1000	2000	1000	2000	2000	1000	1000	5000	1000	1000
Task	91	92	93	94	95	96	97	98	99	100
WCET (μs)	19	17	27	6	5	26	7	7	19	9
Period (μs)	2000	1000	10000	2000	1000	2000	5000	2000	5000	5000

2. เซตของงานที่ใช้สำหรับการทดลองหัวข้อ 4..2.2

1) μ = 0.3

Task set	Task	1	2	3	4	5	6	7	8	9	10
#1	WCET (μs)	45	18	11	10	5	26	16	14	9	28
	Period (μs)	5000	1000	1000	1000	10000	1000	1000	5000	5000	5000
#2	WCET (μs)	10	6	8	4	16	2	45	57	13	36
	Period (μs)	1000	1000	1000	1000	10000	1000	5000	1000	1000	5000
#3	WCET (μs)	5	9	15	5	51	9	37	32	11	46
	Period (μs)	2000	1000	1000	2000	1000	2000	10000	2000	2000	10000
#4	WCET (μs)	5	32	27	17	38	38	1	11	3	33
	Period (μs)	1000	2000	1000	10000	2000	2000	5000	1000	1000	5000
#5	WCET (μs)	21	11	31	38	5	1	17	3	5	4
	Period (μs)	1000	1000	2000	1000	5000	10000	5000	1000	5000	1000
#6	WCET (μs)	26	25	35	32	29	16	40	25	14	7
	Period (μs)	1000	1000	10000	5000	10000	10000	5000	1000	1000	2000
#7	WCET (μs)	19	26	32	34	28	11	12	7	24	31
	Period (μs)	1000	1000	2000	5000	2000	1000	1000	5000	10000	5000
#8	WCET (μs)	2	31	29	30	8	13	24	39	18	20
	Period (μs)	10000	5000	10000	1000	1000	1000	5000	5000	2000	1000
#9	WCET (μs)	31	11	11	19	15	19	18	10	6	11
	Period (μs)	1000	1000	1000	1000	10000	2000	1000	1000	5000	5000
#10	WCET (μs)	2	33	1	15	7	20	20	35	23	7
	Period (μs)	10000	2000	2000	5000	1000	5000	2000	1000	1000	1000

Schedμlability test

Task set	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
TTC-Dispatch	√	√	√	√	√	√	√	√	√	√
TTSA-MTI	√	√	√	√	√	√	√	√	√	√

2) μ = 0.4

Task set	Task	1	2	3	4	5	6	7	8	9	10
#1	WCET (μs)	50	34	46	19	6	42	45	24	21	3
	Period (μs)	1000	1000	1000	10000	2000	1000	2000	2000	10000	5000
#2	WCET (μs)	34	52	14	7	9	4	12	52	46	56
	Period (μs)	1000	1000	10000	2000	5000	1000	1000	1000	1000	10000
#3	WCET (μs)	13	53	18	15	54	7	40	20	20	50
	Period (μs)	2000	2000	5000	2000	1000	1000	2000	1000	10000	1000
#4	WCET (μs)	34	23	39	60	43	6	27	42	20	55
	Period (μs)	2000	10000	1000	1000	5000	1000	1000	10000	5000	1000
#5	WCET (μs)	23	15	58	40	30	12	6	53	22	27
	Period (μs)	1000	5000	1000	10000	1000	2000	2000	2000	1000	1000
#6	WCET (μs)	49	54	21	1	2	51	56	46	58	30
	Period (μs)	2000	2000	10000	10000	1000	2000	5000	1000	1000	2000
#7	WCET (μs)	6	35	7	15	25	43	52	49	14	21
	Period (μs)	10000	1000	1000	5000	1000	1000	1000	2000	1000	1000
#8	WCET (μs)	4	53	54	47	28	48	53	40	4	7
	Period (μs)	1000	5000	2000	1000	1000	1000	10000	2000	2000	1000
#9	WCET (μs)	60	48	19	2	44	28	24	52	3	23
	Period (μs)	1000	1000	2000	10000	1000	1000	10000	5000	5000	10000
#10	WCET (μs)	5	25	59	51	3	44	7	29	52	1
	Period (μs)	5000	5000	1000	10000	2000	5000	5000	1000	5000	1000

Schedμlability test

Task set	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
TTC-Dispatch	√	√	√	√	√	√	√	√	√	√
TTSA-MTI	√	√	√	√	√	√	√	√	√	√

3) μ = 0.5

Task set	Task	1	2	3	4	5	6	7	8	9	10
#1	WCET (μs)	15	15	13	35	50	51	39	59	59	41
	Period (μs)	5000	1000	5000	1000	1000	1000	1000	10000	1000	1000
#2	WCET (μs)	1	65	64	16	78	14	82	21	62	58
	Period (μs)	1000	1000	1000	10000	2000	10000	1000	2000	5000	1000
#3	WCET (μs)	75	15	18	31	22	31	89	56	54	52
	Period (μs)	1000	2000	5000	1000	1000	10000	1000	2000	5000	1000
#4	WCET (μs)	14	32	70	29	50	34	68	89	40	7
	Period (μs)	1000	1000	1000	10000	2000	2000	1000	1000	5000	2000
#5	WCET (μs)	12	44	76	52	25	33	48	84	25	58
	Period (μs)	1000	5000	1000	2000	2000	1000	1000	10000	1000	1000
#6	WCET (μs)	1	59	68	6	31	45	28	60	73	61
	Period (μs)	10000	2000	2000	10000	1000	1000	10000	1000	2000	1000
#7	WCET (μs)	64	86	17	7	8	86	87	84	62	12
	Period (μs)	2000	1000	5000	1000	10000	1000	5000	2000	1000	5000
#8	WCET (μs)	10	49	59	55	68	15	70	69	77	49
	Period (μs)	2000	1000	1000	10000	1000	1000	1000	5000	5000	5000
#9	WCET (μs)	60	86	83	33	51	72	81	83	37	18
	Period (μs)	1000	1000	10000	10000	5000	5000	2000	5000	1000	1000
#10	WCET (μs)	86	66	80	81	71	48	31	25	43	1
	Period (μs)	1000	5000	1000	2000	2000	10000	1000	1000	5000	5000

Schedμlability test

Task set	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
TTC-Dispatch	√	√	√	√	√	√	√	√	√	√
TTSA-MTI	√	√	√	√	√	√	√	√	√	√

4) μ = 0.6

Task set	Task	1	2	3	4	5	6	7	8	9	10
#1	WCET (μs)	35	84	98	28	98	60	19	46	12	70
	Period (μs)	5000	1000	2000	5000	1000	1000	1000	10000	5000	1000
#2	WCET (μs)	82	86	34	6	27	9	67	94	66	42
	Period (μs)	1000	1000	2000	1000	5000	1000	1000	5000	1000	1000
#3	WCET (μs)	95	47	71	49	37	61	65	21	56	96
	Period (μs)	5000	10000	2000	1000	1000	1000	1000	5000	2000	1000
#4	WCET (μs)	94	98	49	44	80	12	83	75	59	30
	Period (μs)	2000	2000	1000	1000	1000	10000	5000	1000	2000	1000
#5	WCET (μs)	20	11	24	61	60	54	65	76	31	29
	Period (μs)	2000	5000	10000	1000	1000	1000	1000	1000	1000	1000
#6	WCET (μs)	5	96	43	22	31	93	97	9	97	61
	Period (μs)	5000	10000	1000	10000	1000	1000	1000	2000	1000	5000
#7	WCET (μs)	84	90	78	22	63	71	0	60	94	9
	Period (μs)	1000	10000	1000	1000	2000	1000	1000	2000	1000	1000
#8	WCET (μs)	25	21	55	53	30	83	8	40	90	87
	Period (μs)	1000	1000	10000	2000	1000	1000	5000	2000	1000	1000
#9	WCET (μs)	15	92	73	15	21	49	29	66	10	77
	Period (μs)	1000	1000	1000	10000	1000	10000	1000	1000	10000	1000
#10	WCET (μs)	8	67	52	68	49	28	52	92	75	50
	Period (μs)	5000	10000	1000	1000	1000	1000	1000	1000	1000	10000

Schedμlability test

Task set	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
TTC-Dispatch	√	√	√	√	√	√	√	√	√	√
TTSA-MTI	√	√	√	√	√	√	√	√	√	√

5) μ = 0.7

Task set	Task	1	2	3	4	5	6	7	8	9	10
#1	WCET (μs)	23	111	9	13	61	2	24	137	76	128
	Period (μs)	10000	1000	1000	1000	10000	1000	1000	1000	1000	1000
#2	WCET (μs)	66	65	85	134	39	131	36	110	43	38
	Period (μs)	1000	10000	1000	5000	1000	2000	2000	1000	1000	1000
#3	WCET (μs)	90	77	58	113	22	105	68	59	95	139
	Period (μs)	10000	10000	2000	2000	5000	1000	2000	2000	1000	1000
#4	WCET (μs)	55	125	93	46	115	34	127	41	87	113
	Period (μs)	1000	5000	10000	2000	1000	5000	10000	1000	1000	1000
#5	WCET (μs)	26	123	86	106	76	28	106	24	121	116
	Period (μs)	5000	5000	2000	10000	1000	1000	1000	1000	2000	1000
#6	WCET (μs)	39	92	76	40	137	80	102	123	93	73
	Period (μs)	1000	2000	5000	1000	10000	1000	1000	10000	1000	1000
#7	WCET (μs)	29	104	70	10	55	36	106	123	122	30
	Period (μs)	1000	1000	5000	1000	2000	10000	1000	2000	1000	1000
#8	WCET (μs)	33	110	122	78	126	133	42	70	55	133
	Period (μs)	10000	5000	1000	1000	5000	5000	1000	1000	1000	2000
#9	WCET (μs)	100	129	78	116	50	43	94	8	46	124
	Period (μs)	1000	2000	1000	10000	1000	10000	1000	1000	1000	5000
#10	WCET (μs)	78	6	23	82	71	140	137	107	33	103
	Period (μs)	2000	2000	5000	2000	10000	1000	2000	1000	5000	1000

Schedμlability test

Task set	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
TTC-Dispatch	√	√	√	√	√	√	√	√	√	√
TTSA-MTI	√	√	√	√	√	√	√	√	√	√

6) μ = 0.75

Task set	Task	1	2	3	4	5	6	7	8	9	10
#1	WCET (μs)	76	119	76	168	128	171	165	33	120	17
	Period (μs)	10000	5000	1000	2000	1000	2000	2000	5000	2000	1000
#2	WCET (μs)	97	172	114	108	9	41	64	132	95	128
	Period (μs)	1000	5000	1000	10000	2000	1000	1000	1000	2000	10000
#3	WCET (μs)	90	8	191	97	17	11	126	61	148	113
	Period (μs)	5000	2000	1000	2000	5000	2000	10000	2000	1000	1000
#4	WCET (μs)	173	132	22	195	41	109	4	116	87	38
	Period (μs)	10000	5000	2000	1000	1000	1000	2000	2000	1000	2000
#5	WCET (μs)	32	169	43	0	192	195	159	37	30	52
	Period (μs)	2000	10000	1000	1000	1000	1000	5000	1000	5000	2000
#6	WCET (μs)	15	114	166	59	129	28	169	165	151	183
	Period (μs)	1000	5000	2000	1000	10000	5000	1000	2000	2000	10000
#7	WCET (μs)	114	70	9	181	50	7	27	50	76	188
	Period (μs)	1000	1000	1000	1000	5000	10000	2000	1000	10000	2000
#8	WCET (μs)	74	130	15	25	36	60	165	138	189	146
	Period (μs)	1000	10000	10000	1000	1000	1000	1000	2000	5000	2000
#9	WCET (μs)	104	112	184	193	11	59	197	58	15	72
	Period (μs)	1000	2000	1000	2000	1000	5000	10000	1000	2000	2000
#10	WCET (μs)	200	166	126	158	197	36	87	83	193	22
	Period (μs)	5000	1000	2000	5000	10000	10000	10000	10000	1000	5000

Schedμlability test

Task set	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
TTC-Dispatch	√	√	√	√	√	×	√	√	√	√
TTSA-MTI	√	√	√	√	√	×	√	×	√	√

7) μ = 0.8

Task set	Task	1	2	3	4	5	6	7	8	9	10
#1	WCET (μs)	71	55	54	124	102	107	68	57	122	35
	Period (μs)	1000	1000	5000	10000	1000	1000	2000	1000	1000	1000
#2	WCET (μs)	123	53	116	21	84	117	45	57	58	81
	Period (μs)	1000	1000	1000	1000	1000	2000	1000	1000	1000	10000
#3	WCET (μs)	135	85	93	129	63	89	30	78	15	41
	Period (μs)	2000	1000	1000	1000	2000	2000	2000	1000	1000	1000
#4	WCET (μs)	78	13	102	59	94	113	125	94	116	136
	Period (μs)	5000	2000	10000	1000	1000	10000	2000	1000	1000	1000
#5	WCET (μs)	90	116	99	25	81	109	106	35	48	102
	Period (μs)	1000	1000	5000	10000	2000	1000	1000	2000	1000	2000
#6	WCET (μs)	61	109	23	97	61	115	78	110	38	74
	Period (μs)	2000	1000	1000	1000	10000	1000	1000	1000	1000	5000
#7	WCET (μs)	113	112	123	56	117	88	29	53	99	4
	Period (μs)	1000	2000	1000	1000	2000	1000	5000	5000	1000	10000
#8	WCET (μs)	125	1	122	53	17	145	124	20	154	54
	Period (μs)	2000	2000	10000	10000	1000	1000	1000	5000	1000	1000
#9	WCET (μs)	110	58	12	147	16	120	125	57	3	3
	Period (μs)	1000	5000	10000	1000	1000	1000	1000	1000	1000	10000
#10	WCET (μs)	151	109	108	60	139	89	49	136	126	155
	Period (μs)	10000	2000	1000	1000	1000	1000	5000	2000	10000	2000

Schedμlability test

Task set	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
TTC-Dispatch	√	√	√	×	√	√	√	√	√	×
TTC-MTI	√	√	√	×	√	√	√	√	√	×

8) μ = 0.85

Task set	Task	1	2	3	4	5	6	7	8	9	10
#1	WCET (μs)	43	85	10	151	108	50	174	16	8	111
	Period (μs)	10000	1000	1000	1000	1000	1000	1000	5000	5000	2000
#2	WCET (μs)	101	157	40	146	26	103	69	117	98	69
	Period (μs)	1000	1000	1000	1000	1000	1000	1000	10000	10000	10000
#3	WCET (μs)	194	85	100	157	117	94	95	68	126	25
	Period (μs)	2000	10000	1000	1000	2000	2000	2000	10000	1000	1000
#4	WCET (μs)	99	148	94	177	72	152	15	2	117	171
	Period (μs)	2000	10000	1000	1000	5000	1000	10000	1000	1000	5000
#5	WCET (μs)	193	126	6	99	37	123	76	127	169	150
	Period (μs)	1000	2000	1000	5000	5000	1000	2000	10000	1000	10000
#6	WCET (μs)	130	157	16	96	13	87	109	171	78	166
	Period (μs)	1000	1000	10000	5000	2000	5000	1000	2000	2000	2000
#7	WCET (μs)	13	102	111	63	72	150	160	120	59	135
	Period (μs)	1000	2000	2000	2000	2000	1000	10000	1000	1000	1000
#8	WCET (μs)	30	23	137	171	29	89	3	18	136	189
	Period (μs)	1000	10000	1000	5000	2000	1000	1000	2000	1000	1000
#9	WCET (μs)	178	138	153	110	14	115	40	104	190	24
	Period (μs)	10000	5000	1000	5000	1000	1000	1000	2000	1000	2000
#10	WCET (μs)	64	32	11	196	139	142	47	157	160	34
	Period (μs)	1000	1000	5000	1000	2000	1000	10000	2000	5000	1000

Schedμlability test

Task set	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
TTC-Dispatch	√	√	×	×	×	×	√	√	×	√
TTC-MTI	√	√	×	×	×	×	×	√	×	×

9) μ = 0.9

Task set	Task	1	2	3	4	5	6	7	8	9	10
#1	WCET (μs)	135	140	122	33	55	57	180	141	126	150
	Period (μs)	1000	5000	5000	2000	1000	10000	5000	1000	1000	1000
#2	WCET (μs)	50	152	104	22	137	133	181	98	93	133
	Period (μs)	10000	1000	1000	5000	1000	2000	5000	2000	5000	1000
#3	WCET (μs)	79	94	53	155	99	125	102	169	160	99
	Period (μs)	5000	1000	10000	1000	1000	2000	1000	2000	2000	5000
#4	WCET (μs)	181	181	45	67	3	84	163	168	17	194
	Period (μs)	1000	5000	2000	5000	5000	10000	2000	1000	5000	1000
#5	WCET (μs)	84	37	177	195	171	10	40	166	65	178
	Period (μs)	1000	10000	1000	1000	2000	10000	5000	10000	1000	2000
#6	WCET (μs)	138	19	100	26	125	109	121	161	104	20
	Period (μs)	2000	5000	1000	1000	1000	1000	10000	1000	1000	1000
#7	WCET (μs)	101	101	141	107	181	190	120	145	85	2
	Period (μs)	2000	1000	1000	10000	5000	1000	1000	10000	2000	1000
#8	WCET (μs)	163	15	173	36	167	168	102	104	73	185
	Period (μs)	5000	10000	10000	1000	1000	2000	1000	5000	1000	1000
#9	WCET (μs)	200	144	175	85	93	27	18	96	186	89
	Period (μs)	1000	1000	10000	1000	5000	10000	1000	1000	2000	2000
#10	WCET (μs)	116	89	115	109	126	178	108	195	78	66
	Period (μs)	2000	5000	2000	2000	1000	10000	5000	1000	1000	1000

Schedμlability test

Task set	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
TTC-Dispatch	×	×	×	×	×	√	×	×	×	×
TTC-MTI	×	×	×	×	×	√	×	×	×	×

10) μ = 0.95

Task set	Task	1	2	3	4	5	6	7	8	9	10
#1	WCET (μs)	145	95	18	82	173	81	29	105	172	152
	Period (μs)	1000	5000	10000	10000	5000	1000	1000	1000	1000	1000
#2	WCET (μs)	156	123	162	58	52	12	144	89	25	142
	Period (μs)	10000	1000	1000	1000	1000	1000	1000	1000	1000	2000
#3	WCET (μs)	131	21	80	114	126	54	148	75	155	149
	Period (μs)	10000	1000	1000	1000	1000	10000	1000	1000	1000	5000
#4	WCET (μs)	79	32	169	33	171	169	97	155	169	80
	Period (μs)	2000	2000	1000	10000	1000	10000	1000	1000	2000	5000
#5	WCET (μs)	38	173	123	88	131	24	142	21	113	147
	Period (μs)	10000	5000	1000	5000	1000	1000	1000	2000	1000	1000
#6	WCET (μs)	104	114	45	74	101	91	185	143	134	33
	Period (μs)	1000	1000	1000	2000	1000	10000	5000	1000	1000	2000
#7	WCET (μs)	18	66	155	166	11	61	164	104	147	83
	Period (μs)	1000	1000	1000	1000	1000	1000	1000	1000	10000	10000
#8	WCET (μs)	147	76	200	41	30	160	193	184	154	87
	Period (μs)	1000	10000	1000	1000	5000	10000	5000	2000	1000	2000
#9	WCET (μs)	131	45	116	192	65	26	148	186	73	24
	Period (μs)	1000	5000	1000	1000	1000	5000	5000	1000	10000	2000
#10	WCET (μs)	4	55	198	76	198	52	130	143	143	147
	Period (μs)	1000	1000	2000	2000	2000	10000	1000	1000	5000	1000

Schedμlability test

Task set	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
TTC-Dispatch	×	×	×	×	×	×	√	×	×	×
TTC-MTI	×	×	×	×	×	×	×	×	×	×

11) μ = 1.0

Task set	Task	1	2	3	4	5	6	7	8	9	10
#1	WCET (μs)	29	143	55	117	157	26	156	155	135	120
	Period (μs)	10000	1000	1000	1000	5000	5000	5000	1000	1000	1000
#2	WCET (μs)	90	164	73	115	72	32	156	160	92	41
	Period (μs)	10000	1000	2000	1000	5000	1000	1000	1000	1000	10000
#3	WCET (μs)	33	168	119	125	145	68	45	59	176	44
	Period (μs)	1000	1000	5000	1000	1000	10000	2000	1000	1000	1000
#4	WCET (μs)	132	65	161	175	2	98	128	167	6	165
	Period (μs)	2000	1000	1000	1000	1000	1000	1000	10000	10000	2000
#5	WCET (μs)	59	104	162	160	55	48	120	55	114	101
	Period (μs)	1000	1000	2000	1000	1000	1000	1000	1000	5000	1000
#6	WCET (μs)	153	97	14	104	81	116	193	193	200	103
	Period (μs)	5000	1000	1000	1000	10000	1000	2000	1000	5000	1000
#7	WCET (μs)	142	171	58	124	196	102	162	138	7	23
	Period (μs)	10000	1000	1000	1000	1000	1000	5000	2000	1000	2000
#8	WCET (μs)	132	32	173	81	160	37	198	172	165	81
	Period (μs)	1000	2000	10000	1000	1000	5000	1000	1000	5000	10000
#9	WCET (μs)	144	144	110	120	21	138	165	158	20	183
	Period (μs)	1000	1000	10000	1000	5000	1000	1000	10000	2000	5000
#10	WCET (μs)	41	42	189	133	18	98	31	175	177	132
	Period (μs)	1000	10000	2000	1000	1000	5000	1000	1000	1000	1000

Schedμlability test

Task set	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
TTC-Dispatch	×	×	×	×	√	×	×	×	×	×
TTC-MTI	×	×	×	×	×	×	×	×	×	×

ประวัติย่อผู้วิจัย

ประวัติย่อผู้วิจัย

ชื่อ นามสกุล	นายสัญญา ควรคิด
วัน เดือน ปีเกิด	วันที่ 3 เดือนสิงหาคม พ.ศ. 2517
จังหวัด และประเทศที่เกิด	อำเภอเมือง จังหวัดราชบุรี
ประวัติการศึกษา	พ.ศ. 2558 ปริญญาปรัชญาดุษฎีบัณฑิต (ปร.ด.) สาขาวิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์ มหาวิทยาลัยมหาสารคาม พ.ศ. 2545 ปริญญาวิศวกรรมศาสตรมหาบัณฑิต (วศ.ม.) สาขาวิศวกรรมไฟฟ้า มหาวิทยาลัยพระจอมเกล้าธนบุรี พ.ศ. 2541 ปริญญาวิศวกรรมศาสตรบัณฑิต (วศ.บ.) สาขาวิศวกรรมอิเล็กทรอนิกส์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง พ.ศ. 2538 ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.) สาขาอิเล็กทรอนิกส์ วิทยาลัยเทคนิคราชบุรี พ.ศ. 2536 ประกาศนียบัตรวิชาชีพ (ปวช.) สาขาอิเล็กทรอนิกส์ วิทยาลัยเทคนิคราชบุรี
ตำแหน่ง สถานที่ทำงาน	อาจารย์
ที่อยู่ที่สามารถติดต่อได้	คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏนครปฐม อำเภอเมือง จังหวัดนครปฐม รหัสไปรษณีย์ 73000

รางวัลเรียนดี ทุนวิจัย และทุนการศึกษา

ได้รับทุนรัฐบาลที่จัดสรรให้กระทรวงวิทยาศาสตร์และเทคโนโลยี เพื่อศึกษาระดับปริญญาเอกด้านเทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์

ผลงานวิจัย

1. Kuankid S, Aurasopon A, Sa-Ngiamvibool W. Effective Scheduling Algorithm and Scheduler Implementation for use with Time-Triggered Co-operative Architecture. Elektronika ir Elektrotechnika. 2014;20(6):122-7.

2. Kuankid S, Rattanawong T, Aurasopon A, editors. Classification of the cattle's behaviors by using accelerometer data with simple behavioral technique. Asia-Pacific Signal and Information Processing Association, 2014 Annual Summit and Conference (APSIPA); 2014: IEEE.