แบบจำลองเชิงรูปนัยเพื่อการทวนสอบการกำหนดรายการเวลาของระบบยูเอวีด้วยสปิน

นายพันธ์เวสส์ สุขวนิช

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2558

Formal Model for Verifying UAV System's Time Scheduling using SPIN

Mr. Punwess Sukvanich

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Software Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2015

หัวข้อวิทยานิพนธ์	แบบจำลองเชิงรูปนัยเพื่อการทวนสอบการกำหนดรายการเวลาของระบบยูเอวีด้วยสปิน
โดย	นายพันธ์เวสส์ สุขวนิช
สาขาวิชา	วิศวกรรมซอฟต์แวร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร. วิวัฒน์ วัฒนาวุฒิ
อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม	ผู้ช่วยศาสตราจารย์ ดร. อาทิตย์ ทองทักษ์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร. บัณฑิต เอื้ออาภรณ์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ
(รองศาสตราจารย์ ดร. ธาราทิพย์ สุวรรณศาสตร์)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร. วิวัฒน์ วัฒนาวุฒิ)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม
(ผู้ช่วยศาสตราจารย์ ดร. อาทิตย์ ทองทักษ์)

.....กรรมการภายนอกมหาวิทยาลัย
(ดร. เด่นดวง ประดับสุวรรณ)

พันธ์เวสส์ สุขวนิช : แบบจำลองเชิงรูปนัยเพื่อการทวนสอบการกำหนดรายการเวลาของระบบยูเอวีด้วยสปิน (Formal Model for Verifying UAV System's Time Scheduling using SPIN) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: รศ. ดร. วิวัฒน์ วัฒนาวุฒิ, อ.ที่ปรึกษาวิทยานิพนธ์ร่วม: ผศ. ดร. อาทิตย์ ทองทักษ์, 91 หน้า.

งานวิทยานิพนธ์นี้เสนอแบบจำลองเชิงรูปนัยด้วยภาษาโปรเมลา เพื่อการทวนสอบการกำหนดรายการเวลาของระบบยูเอวีด้วยเครื่องตรวจสอบแบบจำลองสปิน แบบจำลองเชิงรูปนัยที่เป็นผลลัพธ์ประกอบด้วย 5 แบบจำลองย่อย คือ แบบจำลองย่อยสัญญาณนาฬิกา แบบจำลองย่อยภารกิจ แบบจำลองย่อยภารกิจไม่อิสระ แบบจำลองย่อยการจัดกำหนดรายการเวลา และแบบจำลองย่อยเครื่องประมวลผลเอมทีแอล ทั้งนี้เพื่อให้สามารถแสดงขีดความสามารถของระบบเวลาจริงของระบบยูเอวีด้วยเช่นกัน แบบจำลองเชิงรูปนัยผลลัพธ์นี้ไม่เพียงแต่สามารถใช้เพื่อการทวนสอบการจัดกำหนดรายการเวลาการทำงานที่สอดคล้องตามเงื่อนไขของระบบเวลาจริงได้เท่านั้น แต่ยังรองรับภารกิจแบบจวบกันและรวมถึงภารกิจที่มีการประสานเวลาด้วยระบบนับเวลาภายในแบบจำลอง งานวิทยานิพนธ์นี้ยังเสนอวิธีการแปลสัญลักษณ์ตัวดำเนินการเชิงปริมาณพื้นฐานของภาษาเอมทีแอลมาเป็นสูตรภาษาแอลทีแอลเพื่อการทวนสอบระบบเวลาจริงด้วยเครื่องตรวจแบบจำลองสปินได้อย่างอัตโนมัติ

ภาควิชา	วิศวกรรมคอมพิวเตอร์	ลายมือชื่อนิสิต .....
สาขาวิชา	วิศวกรรมซอฟต์แวร์	ลายมือชื่อ อ.ที่ปรึกษาหลัก .....
ปีการศึกษา	2558	ลายมือชื่อ อ.ที่ปรึกษาร่วม .....

5570988421 : MAJOR SOFTWARE ENGINEERING

KEYWORDS: PROMELA / REAL-TIME SYSTEM / TIMED AUTOMATA / SPIN

PUNWESS SUKVANICH: Formal Model for Verifying UAV System's Time Scheduling using SPIN. ADVISOR: ASSOC. PROF. WIWAT VATANAWOOD, Ph.D., CO-ADVISOR: ASST. PROF. ARTHIT THONGTAK, Ph.D., 91 pp.

A formal model for verifying UAV system's time scheduling is proposed in this thesis. The formal model is written in Promela and verified in SPIN model checker. Our resulting formal model consists of five submodels - Ticking submodel, Task submodel, Dependent task submodel, Scheduler submodel and MTL engine submodel, in order to cope with the real-time behavioral capability of the UAV system. The resulting formal model satisfactorily provides the real-time behavioral specification of not only the time scheduling, but also the concurrency and inner clock signal synchronization of the tasks. This thesis also proposes the automatic translating scheme of the basic MTL's quantitative operators into LTL formula as to suit the verification of the real-time system using SPIN model checker.

Department:	Computer Engineering	Student's Signature .....
Field of Study:	Software Engineering	Advisor's Signature .....
Academic Year:	2015	Co-Advisor's Signature .....

กิตติกรรมประกาศ

ขอกราบขอบพระคุณ รศ.ดร. วิวัฒน์ วัฒนาวุฒิ และ ผศ.ดร.อาทิตย์ ทองทักษ์ ที่เมตตาให้คำแนะนำในการเขียนวิทยานิพนธ์และผลงานทางวิชาการ แม้ข้าพเจ้าจะติดภารกิจบ่อยครั้ง ท่านก็ยังสอบถามด้วยความเป็นห่วงอย่างสม่ำเสมอและยังกรุณาสละเวลาช่วยแก้ไขผลงานแม้เวลาจะกระชั้นชิด แต่ก็สามารถช่วยผลักดันให้วิทยานิพนธ์เล่มนี้สำเร็จได้

ขอกราบขอบพระคุณ รศ.ดร. ธาราทิพย์ สุวรรณศาสตร์ และ ดร.เด่นดวง ประดับสุวรรณ แม้กระผมจะติดขัดหลายประการ แต่อาจารย์ทั้งสองท่านยังมีความเมตตาสละเวลาให้คำแนะนำในการปรับปรุงวิทยานิพนธ์เสมอมา

ขอบคุณ ดร.วิทิต วรรณเลิศลักษณ์ และเพื่อนๆ นักวิจัยที่สถาบันเทคโนโลยีป้องกันประเทศ ที่ช่วยให้ความเห็นในการทำวิทยานิพนธ์ฉบับนี้

ขอบคุณ นายมนัชกร ดำริพัฒนโชติ และนางสาวชญานันท์ วัฒนดิเรก ที่คอยช่วยเหลือและเป็นธุระให้เสมอมา

สารบัญ

หน้า

หน้า

หน้า

บทที่ 1
บทนำ

1.1. ที่มาและความสำคัญของปัญหา

ระบบยูเอวี (UAV:Unmanned aerial vehicle system) เป็นระบบควบคุมอากาศยานที่ทำงานเป็นแบบเวลาจริง (Real-time system) ซึ่งต้องมีพฤติกรรมการทำงานทั้งความถูกต้องด้านตรรกะและเข้ากับข้อจำกัดด้านเวลา (Timing constraint) [1-5] โดยเฉพาะข้อจำกัดด้านเวลาที่มีความจำเป็นจะต้องทวนสอบอย่างเข้มงวด การทวนสอบการจัดกำหนดรายการเวลา (Schedule verification) เป็นการทวนสอบว่าภารกิจที่ทำงานอยู่บนระบบยูเอวี นั้นสอดคล้องกับข้อจำกัดด้านเวลาของระบบหรือไม่ โดยปกติแล้วการทวนสอบดังกล่าวจะใช้วิธีการวิเคราะห์เส้นทางวิกฤติ (Critical-path analysis) แยกตามกรณี [6] และมักจะได้คาบของเวลากระทำการ (Execution Time) ในแต่ละรอบเป็นค่าที่น้อยกว่าจริงเสมอ [7]

สาเหตุที่การทวนสอบโดยการวิเคราะห์เส้นทางวิกฤติไม่มีประสิทธิภาพเนื่องจากระบบยูเอวีมีลักษณะการทำงานที่ซับซ้อน มีอุปกรณ์หลายตัวทำงานร่วมกันในลักษณะการทำงานจวบกัน (Concurrent) และมีการเปลี่ยนเส้นทางการทำงานตามโทโพโลยีการจัดกำหนดรายการเวลา (Scheduling topology) ทำให้ลำดับการทำงานของแต่ละภารกิจสามารถที่จะเปลี่ยนแปลงได้ขณะทำงาน ส่งผลให้การประมาณเหตุการณ์ผิดพลาด และภารกิจต่างๆ ไม่สามารถที่จะทำได้ทันเส้นตาย

งานวิจัยนี้เสนอแบบจำลองเชิงรูปนัย (Formal model) ด้วยภาษาโปรเมลา (Promela) ในการทวนสอบการจัดกำหนดรายการเวลาของระบบยูเอวี ด้วยเครื่องตรวจสอบแบบจำลองสปิน ทำให้สามารถที่จะทวนสอบการจัดกำหนดรายการเวลาการทำงานของระบบว่าสามารถทำงานได้ตามข้อจำกัดด้านเวลาของระบบยูเอวีจากแบบจำลองที่สร้างขึ้นได้ และใช้สำหรับการทวนสอบแทนการวิเคราะห์เส้นทางวิกฤติเพื่อเพิ่มประสิทธิภาพและลดเวลาที่ใช้ในการทวนสอบการจัดกำหนดรายการเวลาของภารกิจบนระบบยูเอวี

1.2. วัตถุประสงค์งานวิจัย

เพื่อเสนอแบบจำลองรูปนัยสำหรับการทวนสอบการจัดเวลาของระบบยูเอวี ด้วยภาษาโปรเมลาและทวนสอบด้วยสปิน

1.3. ขอบเขตงานวิจัย

1) งานวิจัยนี้ทำการสร้างแบบจำลองเชิงรูปนัยด้วยภาษาโปรเมลา ในการทวนสอบการจัดเวลาการทำงานระบบยูเอวีด้วยโปรแกรมสปิน

2) งานวิจัยนี้แปลงข้อจำกัดด้านทรัพยากร และ ข้อจำกัดด้านเวลาให้อยู่ในรูปแบบของแบบจำลองภาษาโปรเมลา โดยมีจำนวนทรัพยากรอย่างน้อย 3 ประเภท และ มีจำนวนงานอย่างน้อย 6 งาน

1.4. ประโยชน์ที่ได้รับ

1) ได้แบบจำลองโปรเมลาสำหรับจำลองการจัดเวลาการทำงานระบบยูเอวีขนาดเล็กที่ใช้ระบบเวลาจริงบนระบบสมองกลฝังตัว
2) จะได้วิธีการแบบรูปนัยสำหรับการทวนสอบการจัดเวลาของระบบเวลาจริง สำหรับระบบที่คล้ายกับระบบตัวอย่างได้

1.5. ขั้นตอนการดำเนินการ

1) เขียนโครงสร้างการทำงานของระบบควบคุมการบินที่ติดตั้งบนระบบยูเอวีขนาดเล็ก
2) รวบรวมทฤษฎีที่เกี่ยวข้อง และงานวิจัยที่เกี่ยวข้อง
3) ทำการวิเคราะห์ระบบและสร้างแผนภาพยูสเคสและแผนภาพกิจกรรม
4) สร้างแบบจำลองย่อยและแผนภาพกิจกรรม
5) สร้างคุณลักษณะของแบบจำลองย่อยเพื่อการทวนสอบ
6) ทวนสอบแบบจำลองย่อยเทียบกับคุณลักษณะของแบบจำลองย่อย
7) สร้างวิธีการแปลง การทำงานของระบบควบคุมการบินให้เป็นรูปนัย
8) แปลงคุณลักษณะที่ต้องการทวนสอบให้อยู่ในรูปแบบของแบบรูปคุณลักษณะ
9) ทวนสอบแบบจำลองระบบ เทียบกับแบบรูปคุณลักษณะของระบบ
10) สรุปผลการทวนสอบระบบ
11) จัดทำบทความวิชาการ และ นำเสนอผลงานวิจัย
12) เสนอรายงานวิจัยในรูปแบบวิทยานิพนธ์

1.6. บทความที่ตีพิมพ์จากงานวิจัย

ส่วนหนึ่งของวิทยานิพนธ์นี้ ได้รับการตีพิมพ์เป็นบทความวิชาการ ดังนี้

1) เรื่อง “Formalizing Real-Time Embedded System into Promela” โดย พันธ์เวสส์ สุขวนิช อาทิตย์ ทองทักษ์ และ วิวัฒน์ วัฒนาวุฒิ ในงานประชุมวิชาการ 4th International Conference on Mechanics and Control Engineering (ICMCE 2015) จัดโดย สมาคมวิจัยแห่งประเทศสหรัฐอเมริกา เมื่อวันที่ 22-24 พฤษจิกายน พ.ศ. 2558 ณ เมืองลิสบอน ประเทศโปรตุเกส

2) เรื่อง “Translating Basic Metric Temporal Logic Formulas into Promela” โดย พันธ์เวสส์ สุขวนิช อาทิตย์ ทองทักษ์ และ วิวัฒน์ วัฒนาวุฒิ ในงานประชุมวิชาการ 7th International Conference on Information Science and Applications เมื่อวันที่ 15-18 กุมภาพันธ์ พ.ศ. 2559 ณ เมืองโฮจิมิน ประเทศเวียดนาม

.

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ระบบเวลาจริง

ระบบเวลาจริง (Real-time system) คือระบบคอมพิวเตอร์ที่ให้ความสำคัญกับความถูกต้องของการคำนวณและเวลาที่ใช้ในการทำงาน พฤติกรรมการทำงานของระบบขึ้นอยู่กับข้อมูลที่ถูกต้องภายในขอบเขตเวลาที่กำหนดไว้ [8] นั่นคือมีชุดคำสั่งที่จำเป็นต้องทำภายในเวลาที่กำหนด หรือทำให้เสร็จในเวลาที่กำหนด การทำงานไม่ทันในเวลาที่กำหนดมีค่าเทียบเคียงได้กับการไม่ได้ทำงานนั้นๆ

ระบบเวลาจริงเป็นระบบที่มีเงื่อนไขความต้องการหลักสองอย่าง [9] คือ

- ความถูกต้องเชิงตรรกะ: ผลที่ได้จากการคำนวณจะต้องถูกต้อง
- ความถูกต้องเชิงเวลา: พฤติกรรมการทำงานของระบบถูกกำหนดด้วยเวลา

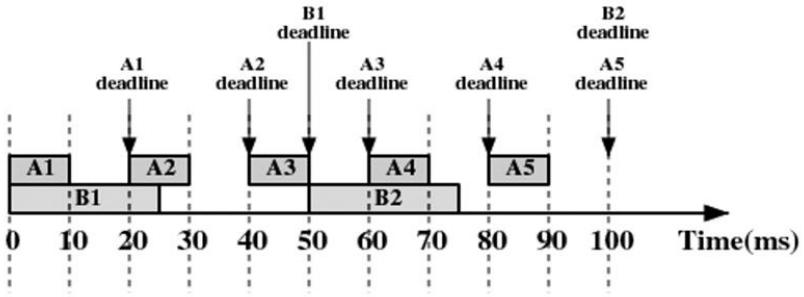
ความถูกต้องเชิงเวลาเวลามักจะถูกกำหนดไว้กับเส้นตาย เช่น ต้องทำภารกิจหนึ่งให้ทันก่อนเส้นตายที่กำหนดไว้ เป็นต้น ผลที่เกิดขึ้นแม้จะถูกต้องแต่จังหวะเวลาที่ได้ไม่ถูกต้องถือว่ามีพฤติกรรมการทำงานไม่ถูกต้องหรือกล่าวได้ว่า ความล่าช้าของข้อมูลถือเป็นความผิดพลาดที่จะทำให้เกิดผลเสียหายร้ายแรงต่อระบบ หากนำระบบเวลาจริงไปใช้ในระบบสมองกลฝังตัว (Embedded system) จะเรียกระบบดังกล่าวว่าระบบสมองกลฝังตัวแบบเวลาจริง (Real-time embedded system) การจำแนกชนิดของระบบเวลาจริง สามารถที่จะแยกได้เป็นสองชนิดดังนี้

ระบบเวลาจริงแบบอ่อน (Soft real-time system) เป็นระบบที่อนุญาตให้เกิดความผิดพลาดด้านเวลาได้ เพราะการผิดพลาดไม่ทำให้เกิดความเสียหายต่อระบบ เช่น ในระบบสื่อสาร หรือในระบบสื่อประสม (Multimedia) เป็นต้น [10]

ระบบเวลาจริงแบบแข็ง (Hard real-time system) เป็นระบบที่กำหนดให้ภารกิจทุกๆ ภารกิจต้องทำสำเร็จภายในเส้นตาย มิฉะนั้นจะเกิดผลที่ไม่สามารถยอมรับได้ [11] ผลจากการพลาดเส้นตายจะทำให้เกิดความเสียหายอย่างรุนแรง และเมื่อระบบดังกล่าวถูกนำไปใช้ในการดูแลระบบวิกฤติ ระบบดังกล่าวจะถูกเรียกว่า ระบบวิกฤติความปลอดภัย เช่น ในระบบในยานยนต์ ระบบควบคุมการบิน เป็นต้น ข้อแตกต่างกันของระบบเวลาทั้งสองแบบคือ หากผลของการคำนวณยังใช้ได้หลังเส้นตาย ระบบดังกล่าวจะถูกเรียกว่าระบบเวลาจริงแบบอ่อน แต่หากผลของการคำนวณไม่สามารถที่จะใช้งานได้หลังจากที่ผ่านเส้นตายแล้ว จะเรียกระบบดังกล่าวว่าระบบเวลาจริงแบบแข็ง

2.2 การจัดกำหนดรายการเวลาบนระบบเวลาจริง

การจัดกำหนดรายการเวลาบนระบบเวลาจริง (Real-time scheduling) เป็นอัลกอรึทึมหนึ่งที่ทำหน้าที่ในการจัดกำหนดรายการเวลาในระบบเวลาจริง เพื่อจะทำให้สามารถที่จะจัดสรรเวลาและทรัพยากรที่มีอยู่อย่างจำกัดในระบบให้เพียงพอกับความต้องการของแต่ละภารกิจ [12-14] เช่น การแบ่งสรรช่วงเวลากระทำการบนหน่วยประมวลผลให้กับภารกิจแต่ละภารกิจ เป็นต้น



รูปที่ 2.1 ตัวอย่างของภารกิจในคอมพิวเตอร์ระบบเวลาจริง [15]

จากรูปที่ 2.1 หากภารกิจ A1 และ A2 ไม่ได้มีการแย่งทรัพยากรกัน และสามารถที่จะทำงานได้พร้อมกันทั้งสองภารกิจ กำหนดให้ภารกิจ A1 มีคาบเวลาเท่ากับ 20 มิลลิวินาทีต่อเนื่อง และมีเส้นตายที่ต้องทำเสร็จภายใน 20 มิลลิวินาที ในขณะเดียวกันภารกิจ B1 มีคาบเวลาเท่ากับ 50 มิลลิวินาที โดยมีเส้นตายคือ 50 มิลลิวินาที หรือทำให้เสร็จก่อนภารกิจครั้งถัดไปจะมาถึง ความซับซ้อนของระบบเวลาจริงนั้น อยู่ในส่วนของการจัดกำหนดรายการเวลาให้ภารกิจแต่ละภารกิจ สามารถที่จะทำงานให้เสร็จทันเส้นตายได้

2.3 ตรรกศาสตร์เชิงกาลเวลาและเอมทีแอล

ในปี ค.ศ. 1977 การใช้ตรรกศาสตร์เชิงกาลเวลา (Temporal logic) เป็นกรอบงานที่นิยมสำหรับการบรรยายระบบที่มีลักษณะการทำงานเป็นระบบปฏิกรณ์ (Reactive system) และเป็นภาษาในการอธิบายคุณลักษณะ (Specification language) และพฤติกรรม (Behavior) ของระบบเวลาจริง โดยเริ่มจากที่คุณนิวลี (Pnueli) [16] ได้เสนอแนวคิดของการทวนสอบเชิงรูปนัยของระบบโดยการใช้ตรรกศาสตร์เชิงกาลมาบรรยายพฤติกรรม ต่อมาวิธีการดังกล่าวเป็นที่นิยมเป็นอย่างมากในวงการของวิทยาศาสตร์คอมพิวเตอร์

สำหรับระบบเวลาจริงนั้นมีลักษณะการทำงานคล้ายกับระบบที่เป็นระบบปฏิกรณ์ กล่าวคือเป็นระบบที่ทำงานต่อเนื่องโดยไม่มีสถานะสิ้นสุด การใช้ตรรกศาสตร์เชิงกาลเวลา (Temporal logic) เป็นวิธีการทางรูปนัยวิธีหนึ่งที่สามารถที่จะตรวจสอบข้อกำหนดดังกล่าวได้ เพราะตรรกศาสตร์ดังกล่าว

สามารถที่จะใช้ในการอ้างถึงพฤติกรรมที่ทำงานอย่างไม่สิ้นสุดด้วยการบรรยายโดยใช้ตรรกศาตร์และตัวแปรเวลา (Modal) [5] การบรรยายด้วยแอลทีแอลเป็นเพียงลำดับเวลาเท่านั้นไม่ได้มีการบรรยายถึงปริมาณเวลาของพฤติกรรมของระบบ ทำให้หากต้องการที่จะกำหนดว่าเมื่อกดเบรกแล้วระบบเบรกต้องทำงานภายใน 3 หน่วยเวลา นั้นไม่สามารถที่จะใช้ตรรกศาสตร์ดังกล่าวบรรยายได้

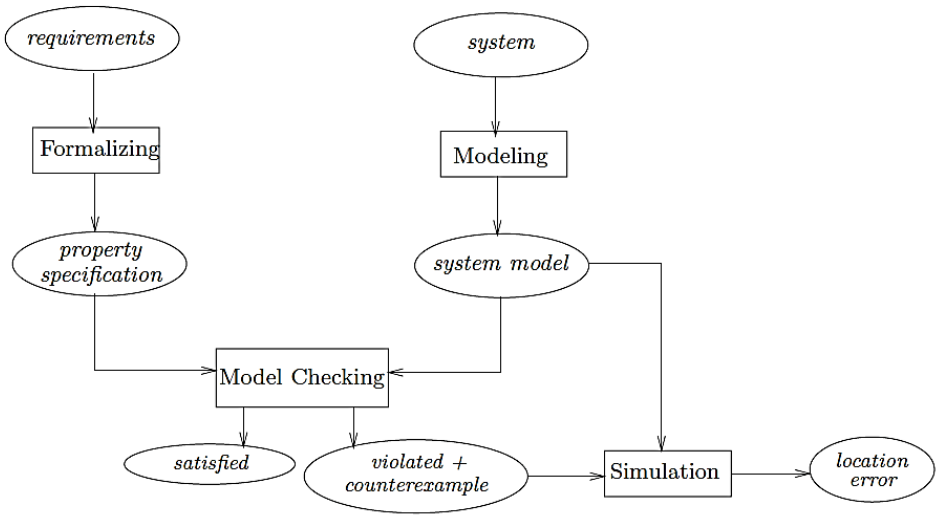
หลังจากนั้นได้มีความพยายามที่จะนำเอาตัวเลขที่สามารถอธิบายช่วงเวลาในการเกิดเหตุการณ์ต่างๆ โดยการสร้างเป็นแบบใช้ประพจน์กับแอลทีแอลเส้นหรือพีแอลทีแอล (Propositional linear time temporal logic : PLTL) ซึ่งเป็นการแบบเวลาออกเป็นช่วงๆ โดยกำหนดให้เวลาเป็นจำนวนธรรมชาติ ต่อมาตรรกศาสตร์เชิงกาลเวลาแบบที่มีเวลากำหนด (Timed temporal logic) ได้ถูกเสนอให้ใช้เพื่อที่ใช้ในการอธิบายข้อกำหนดเชิงเวลาของแอลทีแอล [16] เช่น เมตริกเทมพอรัลลอจิก หรือเอมทีแอล (Metric Temporal Logic :MTL) [17] เป็นต้น

การสร้างตัววัดสำหรับวัดเวลาเป็นการเพิ่มความสามารถที่จะใช้ในการบรรยายเงื่อนไขบังคับของสูตรตรรกศาสตร์ที่มีเวลาเกี่ยวข้องได้ การเพิ่มตัววัดด้านเวลาเข้าไปในระบบจะช่วยให้สามารถที่จะบรรยายและนิยามพฤติกรรมของเหตุการณ์ต่างๆได้อย่างสมบูรณ์มากขึ้น เช่น การบรรยายถึงระยะเวลาที่เกิดขึ้นก่อนหรือหลังเหตุการณ์ที่เกิดขึ้นในระบบเป็นหน่วยเวลา เป็นต้น ความสามารถในการอธิบายปริมาณของเวลานี้เป็นหัวใจหลักของการเขียนคุณลักษณะของระบบเวลาจริง [18]

เอมทีแอลเป็นส่วนขยายของแอลทีแอล โดยการเพิ่มเอาข้อจำกัดด้านเวลาลงไปโดยที่ไวยากรณ์ของเอมทีแอลนั้นเหมือนกับภาษาแอลทีแอลแต่มีการนำตัวดำเนินการมาร่วมกับขอบเขตเวลา ซึ่งทำให้สามารถที่จะกำหนดได้ว่าเมื่อใดที่จะทำให้สูตรนั้นสอดคล้อง แม้ว่าเอมทีแอลกลับได้รับความนิยมมากในการใช้อธิบายระบบเวลาจริง แต่อย่างไรก็ตาม ในปัจจุบันยังไม่มีเครื่องมือสำเร็จรูปใดๆ ที่นำเอาเอมทีแอลไปใช้ในการทวนสอบระบบ

2.4 การตรวจสอบแบบจำลอง

การตรวจสอบแบบจำลองเป็นวิธีการทวนสอบที่ทำการสำรวจทุกสถานะของระบบที่เป็นไปได้โดยใช้วิธีแบบบรูทฟอร์ซ (Brute-force) เพื่อที่จะใช้ในการตรวจหาว่าระบบนั้นสอดคล้องกับกับคุณสมบัติของระบบ (System property) หรือไม่ แผนภาพการแบบจำลองแสดงไว้บนรูปที่ 2.2



รูปที่ 2.2 ขั้นตอนการตรวจสอบแบบจำลอง [5]

การตรวจสอบแบบจำลอง (Model checking) ทวนสอบอัตโนมัติ โดยปกติแล้วคุณสมบัติจะถูกอธิบายด้วยตรรกศาสตร์ เช่น แอลทีแอล หรือ เอมทีแอล เป็นต้น เมื่อตรวจแบบจำลองแล้วพบปัญหา เครื่องมือในการตรวจสอบแบบจำลองจะทำการสร้างตัวอย่างค้าน (Counter example) เพื่อที่จะช่วยให้ผู้ตรวจสอบสามารถที่ค้นหาปัญหาที่เกิดขึ้นในระบบได้ การนำเอาวิธีการตรวจสอบแบบจำลองไปใช้ประกอบด้วยขั้นตอนหลายขั้น ดังนี้

1) สร้างคุณลักษณะของระบบ (Specification) เมื่อผู้ตรวจสอบได้ทำการเลือกแล้วว่าจะตรวจสอบคุณสมบัติใดของระบบ ต่อมาจึงจำเป็นที่จะต้องสร้างคุณลักษณะเฉพาะของระบบ ที่สามารถอธิบายเชิงรูปนัยได้
2) การสร้างแบบจำลองระบบ (System model) เป็นขั้นตอนแรกในการสร้างแบบจำลอง โดยการแปลงระบบที่ต้องการวิเคราะห์ให้อยู่ในรูปแบบรูปนัย ผู้ออกแบบมักใช้การแปลงแบบจำลองให้เป็นแบบจำลองอย่างง่าย หรือการทำแบบคัดย่อ โดยเลือกเอาเฉพาะส่วนที่เป็นปัญหามาวิเคราะห์
3) การทวนสอบ (Verification) เป็นขั้นตอนสุดท้ายที่เครื่องมือสำหรับตรวจสอบแบบจำลองทำการทวนสอบว่าแบบจำลองที่สร้างขึ้นมาสอดคล้องกับคุณลักษณะที่กำหนดไว้หรือไม่ เวลาที่ใช้ในการทำงานในส่วนนี้นั้นใช้เวลามากหรือน้อยขึ้นกับขนาดของแบบจำลองที่สร้างขึ้น

2.5 โปรเมลาและสปิน

โปรเมลา (Promela) เป็นภาษารูปนัยที่ใช้ในการสร้างแบบจำลองของกระบวนการ [19-21] และตรวจสอบการทำงานของระบบที่ทำงานกันในลักษณะคู่ขนาน (Concurrent system) แต่ละกระบวนการเรียกว่า พร๊อกไทป์ (Proctype) แต่ละพร๊อกไทป์สามารถที่จะสื่อสารระหว่างกันโดยการแลกเปลี่ยนข้อมูลระหว่างกระบวนการแบบที่มีที่พักข้อมูลคั่นกลางหรือแบบไม่มีที่พักข้อมูลคั่นกลางก็ได้ แบบที่ไม่มีข้อมูลคั่นกลางเรียกได้อีกอย่างว่าเป็นแบบนัดพบ (Rendezvous) นอกจากนั้นแต่ละกระบวนการสามารถที่จะอ่านและเขียนตัวแปรร่วมกันผ่านตัวแปรแบบครอบคลุม (Global variable)

สปินเป็นเครื่องมือสำหรับตรวจสอบแบบจำลองที่เขียนขึ้นมาด้วยภาษาโปรเมลา สปินสามารถตรวจสอบความถูกต้องของข้อมูลโดยเขียนตรวจสอบด้วยแอลทีแอล นอกจากนั้น สปินยังสามารถที่จะแปลแบบจำลองให้เป็นโปรแกรมภาษาซีได้ เพื่อการทวนสอบที่มีความรวดเร็วยิ่งขึ้น สปินสามารถที่จะตรวจสอบภาวะติดตาย (Deadlock) หรือวงรอบที่ไม่ก้าวหน้า (Non-progressive cycle)

2.6 งานวิจัยที่เกี่ยวข้อง

ในปี ค.ศ. 2008 Wihelm และ คณะ ได้นำเสนองานวิจัยเรื่อง “The Worst Case Execution Time Problem” [7] โดยมีเนื้อหาเกี่ยวกับการนำเสนอวิธีการวิเคราะห์การหาค่าเวลากระทำการมากสุดด้วยวิธีการต่างๆ โดยได้ให้ความสำคัญในด้านที่เกี่ยวกับคอมพิวเตอร์แบบเวลาจริง และวิเคราะห์วิธีการที่ใช้ในการคำนวณ ค่าคาบเวลาสำคัญสองค่า คือ ค่าเวลากระทำการมากสุด และค่าเวลากระทำการน้อยสุด งานวิจัยนี้ได้นำเสนอวิธีการต่างๆ ที่ใช้ในการทวนสอบระบบ โดยเฉพาะระบบที่เป็นแบบ ประมวลผลแบบเวลาจริงที่ต้องการความมั่นใจสูง (Dependable hard real-time system) แต่งานดังกล่าวได้นำเสนอเฉพาะงานที่เป็นเรื่องของวิธีวิเคราะห์ค่าของค่าเวลากระทำการมากสุดของการประมวลผลแบบที่ใช้หน่วยประมวลผลเดี่ยว (Single-core processor) เท่านั้น มิได้นำเสนอวิธีการที่ทำงานเกี่ยวข้องการประสานงานของบัส หรืออุปกรณ์ต่อพ่วงโดยจากงานวิจัยนี้ผู้วิจัยสามารถ ที่จะนำวิธีการที่ได้นำเสนอมาประยุกต์ใช้กับงานวิจัยนี้ได้ มีงานวิจัยหลายชิ้นที่นำเอาออโตมาตาเวลามาสร้างแบบจำลองด้วยภาษาโปรเมลา และทวนสอบด้วยสปินเช่น [6, 22, 23] เป็นต้น ผู้วิจัยในงานวิจัยดังกล่าวเสนอแนวทางการแบ่งเวลาเป็นขนาดเท่าๆ กันโดยตั้งชื่อชิ้นของเวลาว่าติ๊ก โดยมีการตั้งให้ติ๊กเป็นจำนวนเต็มบวกที่เพิ่มขึ้นทุกๆ รอบการทำงานและเมื่อถึงจำนวนที่กำหนดไว้แล้วจึงทำการ ตั้งใหม่ (Reset) ให้เท่ากับศูนย์

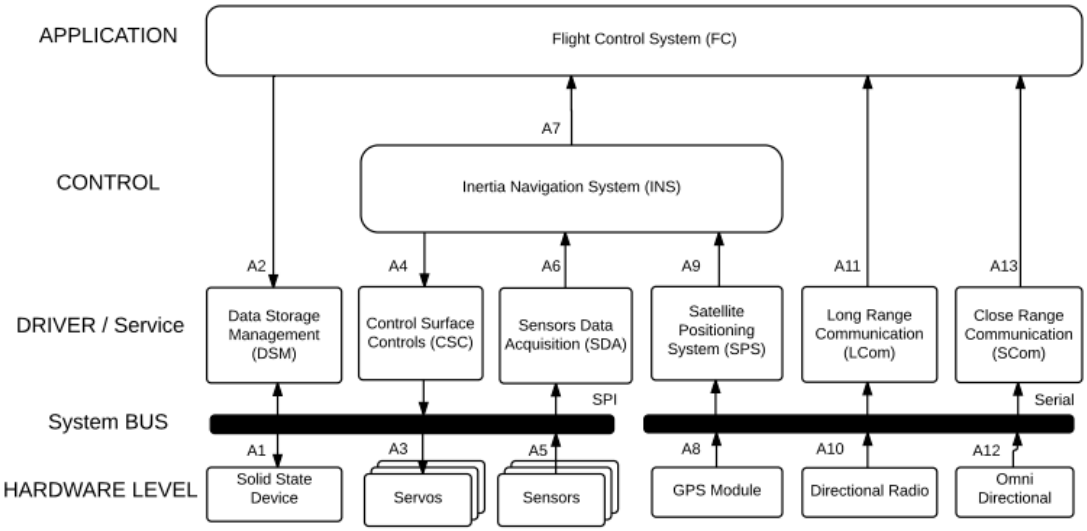
งานวิจัยของ Mahai Florian และคณะ ได้นำเสนองานวิจัยเรื่อง “Logic Model Checking of Time-periodic Real-Time System” [6] เป็นต้น งานวิจัยนี้ได้นำเสนอรายงานเกี่ยวกับการใช้เครื่องมือตรวจสอบแบบจำลองสปินในการทวนสอบโปรแกรมการทำงานของระบบสมองกลฝังตัวหนึ่ง ในผู้เขียนงานวิจัยนี้คือ Gerard ซึ่งเป็นผู้ออกแบบโปรแกรมตรวจสอบแบบจำลองสปิน ได้ทำการ

พัฒนาความสามารถของโปรแกรมสปิน ให้รองรับการทำงานของการทวนสอบระบบดังกล่าว ด้วยการกำหนดความสำคัญ (Priority) ของงานที่เกิดขึ้นพร้อมกัน ความสามารถนี้เพิ่มเข้ามาในเวอร์ชั่น 6.2.0 ทำให้การทำงานของโปรแกรมสปินสามารถที่จะจำลองระบบที่มีการทำงานแบบคาบเวลา และมีการทำงานของระบบงานอื่นที่แทรกเข้ามาขณะทำงานได้ แต่ผู้วิจัยพบว่าไม่สามารถที่จะนำเอาคุณสมบัตินี้มาใช้งานกับงานวิจัยนี้ เนื่องจากเมื่อเปิดใช้งานคุณสมบัตินี้ จะทำให้สปินทำการปิดคุณลักษณะของการ ลดลำดับบางส่วน (Partial order reduction) การใช้การลดลำดับบางส่วนเป็นคุณลักษณะหลักของเครื่องมือตรวจสอบแบบจำลองสปิน [24] โดยเฉพาะแบบจำลองเวลาที่มีปริมาณของจำนวนสถานะเป็นจำนวนมาก ในการทวนสอบระบบดังกล่าวจะไม่สามารถทำได้

นอกจากนั้นการทำงานของแต่ภารกิจในสปินจะเป็นแบบไม่ประสานเวลา (Asynchronous) และ กระบวนการแต่ละกระบวนการสามารถแทรกสลับ (Interleave) ได้ แต่การทำงานกับระบบเวลาจริงนั้นเป็นแบบประสานเวลา ทำให้ภารกิจที่จะต้องทำนั้นจะต้องถูกเขียนแบบจำลองให้ถูกต้อง มิฉะนั้นจะทำให้แต่ละภารกิจนั้นมีการทำงานก่อนที่จะถึงช่วงเวลาของตัวเอง ทำให้การทำงานของแบบจำลองการทำงานของระบบผิดได้ การที่จะป้องกันไม่ให้กระบวนการแต่ละกระบวนการทำงานก่อนช่วงเวลาของตัวเองนั้นมีงานวิจัย [23, 24] ได้เขียนวิธีการที่เหมาะสมไว้คือการใช้ข้อความสั่งเฉพาะของสปินที่เรียกว่า timeout ซึ่งข้อความสั่งเฉพาะนี้จะทำหน้าที่ในการขวางการทำงานไว้จนกว่าจะไม่มีกระบวนการใดๆสามารถทำงานได้อีก จึงเลิกการขัดขวาง

2.7 ระบบกรณีศึกษาที่จะใช้ในงานวิทยานิพนธ์นี้

โครงสร้างของระบบควบคุมยูเอวีขนาดเล็กที่เป็นกรณีศึกษาในงานวิจัยนี้ แสดงใน รูปที่ 2.3



รูปที่ 2.3 ส่วนประกอบของระบบยูเอวีที่เป็นกรณีศึกษาในงานวิจัยนี้

ระบบควบคุมยูเอวีขนาดเล็กที่เป็นกรณีศึกษา ทำงานบนระบบสมองกลฝังตัวขนาดเล็ก ทำงานภายใต้ระบบปฏิบัติการเวลาจริง (Real-time operating system) ที่มีการจัดสรรทรัพยากรโดยมีชุดคำสั่งการจัดกำหนดรายการเวลาที่สามารถปรับโทโพโลยีการเรียงลำดับความสำคัญของภารกิจได้ ชนิดของภารกิจแบ่งออกเป็น 4 ลำดับคือ ระดับฮาร์ดแวร์ ระดับไดร์เวอร์ ระดับควบคุม และ ระดับแอพพลิเคชัน รายละเอียดของแต่ละภารกิจที่ทำงานบนระบบควบคุมของยูเอวีขนาดเล็กมีดังนี้

ภารกิจควบคุมการบิน (Flight Control System Task) ทำหน้าที่ในการควบคุมและสั่งการ มีหน้าที่หลักคือการควบคุมให้เครื่องบิน บินในเพดานบิน ความเร็ว และตำแหน่งที่ถูกต้อง มีการคาดคะเนเส้นทางการบิน และการทำงานปกติ

ภารกิจภารกิจนำร่องด้วยความเฉื่อย (Inertia Navigation System Task) ทำหน้าที่ในการคำนวณตำแหน่ง คำนวนและระบุคุณลักษณะการบิน การควบคุมการบิน ส่งข้อมูลการบินให้กับส่วนจัดเก็บข้อมูล และข้อมูลการบินให้ ภารกิจควบคุมการบิน

ภารกิจควบคุมพื้นผิวควบคุมการบิน (Control Surface Controls Task) ทำหน้าที่ในการควบคุม เซอร์โวมอเตอร์ที่ควบคุมปีกต่างๆ เช่น แอเลียรอน (Ailerons), แอลลิเวเตอร์ (Elevator), รัดเดอร์ (Rudder) เป็นต้น

ภารกิจรับสัญญาณเซนเซอร์ (Sensor Data Acquisition Task) และ ภารกิจชุดเซนเซอร์ (Sensors Task) ทำหน้าที่ในการควบคุม รับสัญญาณ และประมวลผลค่าที่ได้จากเซนเซอร์ต่างๆ ที่ติดตั้งบนตัวเครื่องก่อนที่จะส่งให้กับ ภารกิจนำร่วงด้วยความเฉื่อย

ภารกิจชุดเซอร์โว (Servos Task) ทำหน้าที่ในการแปลงเอาสัญญาณควบคุมเซอร์โวให้เป็นแบบคำสั่งสำหรับสั่งการชุดเซอร์โวควบคุมปีก

ภารกิจจัดการจัดเก็บข้อมูล (Data Storage Management Task) และ ภารกิจเอสเอสดี (Solid State Device Task) ทำหน้าที่ในการบริหารจัดการการจัดเก็บข้อมูลจากข้อมูลการบินลงหน่วยความจำ ถูกกำหนดให้เขียนเมื่อเกิดเหตุการณ์ที่สำคัญต่างๆ เช่น ความเร็วของเครื่องสูงเกินไปหรือต่ำเกินไป เป็นต้น ก็จะเขียนข้อมูลดังกล่าว

ภารกิจชี้ตำแหน่งด้วยดาวเทียม (Satellite Positioning System Task) และ ภารกิจรับสัญญาณดาวเทียม (Global Positioning System Task) ทำหน้าที่ในการรับเอาข้อมูลจากเครื่องรับสัญญาณดาวเทียม และแปลความหมายเป็นตำแหน่งลงบนแผนที่ส่งต่อให้กับ ภารกิจควบคุมการบิน

ภารกิจสื่อสารระยะไกล (Long Range Communication Task) และ ภารกิจชุดวิทยุสื่อสารแบบมีทิศทาง (Directional Radio Task) ทำหน้าที่ในการแปลคำสั่งสื่อสารออกมาเป็นคำสั่งให้กับภารกิจ

ควบคุมการบิน ทำหน้าที่ในการสื่อสารติดต่อกับผู้บังคับเครื่องและส่งค่า สถานะของเครื่องบินต่างๆ กลับมาเมื่อผู้ควบคุมสั่ง ใช้เฉพาะขณะทำการบินปกติเท่านั้น

ภารกิจสื่อสารระยะใกล้ (Close Range Communication Task) และ ภารกิจชุดวิทยุสื่อสารแบบรอบทิศทาง ทำหน้าที่ในการติดต่อสื่อสารระยะใกล้กับนักบินภายนอก ใช้สำหรับเมื่อบินขึ้นและบินลงเท่านั้น เพื่อที่จะให้นักบินภายนอกสามารถที่จะควบคุมการบินของเครื่องบินได้โดยตรง

2.7.1 การทำงานเมื่อเครื่องยูเอวีทำการบินปกติ

เมื่อเครื่องทำการบินจะมีภารกิจที่ทำงานเป็นคาบเวลาคือ คือ ส่วนของภารกิจชุดเซนเซอร์ (Sensors task) และภารกิจชุดเซอร์โว (Servos task) ทั้งสองภารกิจทำหน้าที่ในการรักษาเพดานบิน ชุดเซนเซอร์จะทำหน้าที่ในการแปลงเอาท่าทางการบินของเครื่องบินให้เป็นสัญญาณไฟฟ้า และส่งข้อมูลผ่านชุดบัสแบบเอสพีไอและ บัสไอสแควร์ซี และส่งไปยังหน่วยประมวลผลสัญญาณเซนเซอร์ เพื่อที่จะแปลงและทำการปรับปรุงสัญญาณดังกล่าวให้มีค่าเพียงพอสำหรับการนำไปประมวลผลต่อในส่วนของ ภารกิจนำร่องด้วยความเฉื่อย (Inertia navigation system task) และส่งข้อมูลที่ปรับปรุงไปยัง ภารกิจควบคุมพื้นผิวควบคุมการบิน (Control surface controls task) เพื่อให้ข้อมูลดังกล่าวส่งกลับไปยัง ชุดควบคุมเซอร์โว เพื่อที่จะปรับท่าทางการบินให้ถูกต้อง

เครื่องจะถูกบินด้วยนักบินอัตโนมัติที่อยู่ในส่วนของ ภารกิจควบคุมการบิน ทำให้เครื่องสามารถบินได้แม้จะไม่มีนักบินควบคุม ในการทำการบินปกติ เครื่องบินจะทำการรับคำสั่งจากศูนย์ควบคุมและสั่งการระยะไกล ผ่านภารกิจสื่อสารระยะไกล (Long range communication task) บ้างเป็นครั้งคราว เช่น เพื่อเปลี่ยนเส้นทาง เป็นต้น

2.7.2 การทำงานเมื่อเครื่องยูเอวีทำการบินทำการบินขึ้นและลง

ปกติแล้วอากาศยานไร้คนขับขนาดเล็กจะไม่มีระบบอัตโนมัติสำหรับนำเครื่องขึ้นและลง จำเป็นที่จะต้องให้นักบินภายนอกทำการนำเครื่องขึ้นและลงให้ โดยใช้คันบังคับและอุปกรณ์สื่อสารสำหรับการสื่อสารระยะใกล้ โดยเสาอากาศที่ใช้จะเป็นเสาอากาศแบบรอบทิศทาง ที่สามารถสื่อสารภายในระยะใกล้ๆได้ดี โดยชุดสื่อสารจะติดต่อกับ ภารกิจสื่อสารระยะใกล้ (Short range communication task) ซึ่งภารกิจดังกล่าวนี้เป็นภารกิจเวลาจริงแบบแข็ง แตกต่างกับภารกิจสื่อสารระยะไกลที่เป็นภารกิจเวลาจริงแบบอ่อน

บทที่ 3

กระบวนการสร้างแบบจำลองรูปนัย

ในบทนี้จะครอบคลุมเนื้อหาเกี่ยวกับกระบวนการสร้างแบบจำลองเชิงรูปนัยสำหรับระบบยูเอวี โดยเริ่มจากการวิเคราะห์เพื่อหาข้อกำหนดความต้องการ (Requirements specification) ของระบบยูเอวีและแจกแจงเป็นรายการข้อกำหนดความต้องการที่ครอบคลุมลักษณะโครงสร้างและพฤติกรรมหลักของระบบยูเอวี จากนั้นผู้วิจัยจะนำรายการข้อกำหนดความต้องการที่ได้มานำลงรายละเอียดและนำเสนอเป็นแผนภาพกิจกรรมและแผนภาพสถานะ (State diagram) เพื่อใช้เป็นประโยชน์ในการแปลงเป็นภาษาโปรเมลาและใช้ในการทวนสอบ อนึ่ง รายการข้อกำหนดความต้องการที่ได้มานั้นยังสามารถนำมาวิเคราะห์เพื่อสกัดหาประโยคแบบอย่างคุณลักษณะพึงประสงค์ที่ใช้ในการทวนสอบได้เช่นกัน โดยประโยคแบบอย่างคุณลักษณะนี้จะถูกนำไปแปลให้เป็นภาษาแอลทีแอลต่อไป

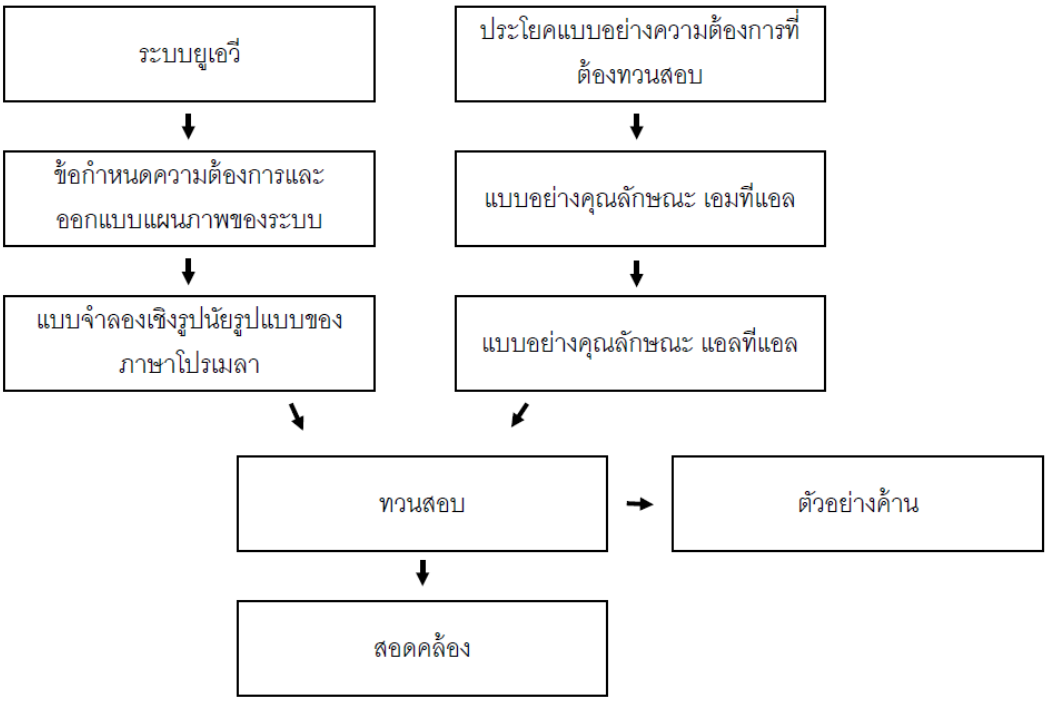
3.1 กระบวนการสร้างแบบจำลองรูปนัย

กระบวนการสร้างแบบจำลองเชิงรูปนัยของระบบยูเอวีในวิทยานิพนธ์นี้แสดงตามรูปที่ 3.1 ผู้วิจัยจะเริ่มนำระบบยูเอวีที่มีอยู่ในกรณีศึกษาซึ่งประกอบด้วยแผนภาพองค์ประกอบของระบบยูเอวีตามรูปที่ 2.1 และเอกสารคู่มือระบบมาเพื่อสกัดหาข้อกำหนดความต้องการหลักของระบบยูเอวีและเขียนได้เป็นรายการให้ชัดเจนตามวิธีการวิเคราะห์ระบบงานทั่วไป จากก็จะนำข้อกำหนดนั้นในข้อกำหนดความต้องการมาวิเคราะห์และออกแบบเพื่อให้ได้แผนภาพกิจกรรมและแผนภาพสถานะเพื่อใช้ในการแปลงเป็นแบบจำลองเชิงรูปนัยที่เขียนด้วยภาษาโปรเมลา เราพบว่าข้อกำหนดความต้องการที่ได้ของระบบยูเอวีจะเป็นไปตามคุณลักษณะหลักของระบบเวลาจริง (Real-time system) ในขณะเดียวกันข้อกำหนดความต้องการที่ได้มาสามารถนำมาสกัดหาประโยคแบบอย่างคุณลักษณะที่จะนำไปใช้ในการทวนสอบระบบยูเอวีได้ด้วยเช่นกัน ประโยคที่สกัดได้เหล่านี้จะถูกแปลให้เป็นภาษาแอลทีแอลหรือภาษาเอ็มทีแอล เพื่อใช้ในการทวนสอบต่อไปเช่นกัน

ผู้วิจัยทำการแปลงแผนภาพกิจกรรมและแผนภาพสถานะให้อยู่ในภาษาโปรเมลา โดยใช้เทคนิคการสร้างแม่แบบพร๊อกไทป์ (Proctype template) เพื่อช่วยในการเขียนภาษาโปรเมลาได้ง่ายและสะดวกขึ้น โดยแผนภาพของระบบดังกล่าวจะถูกนำมาแปลงเป็นภาษาโปรเมลาได้ครบถ้วนในขณะเดียวกัน ผู้วิจัยก็ทำการแปลประโยคแบบอย่างคุณลักษณะที่ต้องทวนสอบมาเป็นภาษาเอ็มทีแอลจากนั้นค่อยแปลไปเป็นแอลทีแอล โดยใช้แม่แบบแอลทีแอล (LTL Template)

การทวนสอบทำได้ด้วยเครื่องมือตรวจสอบแบบจำลองสปินซึ่งรับแบบจำลองเชิงรูปนัยที่เขียนด้วยภาษาโปรเมลาและแอลทีแอลที่ใช้ในการทวนสอบมาประมวลผล ซึ่งมีผลลัพธ์ได้สองกรณีคือ กรณี

ที่แบบจำลองเชิงรูปนัยทำงานสอดคล้องกับแอลทีแอล หรือกล่าวได้ว่าแบบจำลองนี้ทำงานได้ถูกต้องตามคุณลักษณะที่ต้องการทวนสอบ ในทางตรงกันข้ามกรณีที่แบบจำลองเชิงรูปนัยทำงานไม่สอดคล้องจะปรากฏตัวอย่างค้านเป็นผลลัพธ์ออกมาเพื่อผู้วิจัยจะได้นำไปแก้ไขแบบจำลองให้ถูกต้องและทำการทวนสอบซ้ำอีกรอบหนึ่ง



รูปที่ 3.1 แผนภาพแสดงกระบวนการสร้างแบบจำลองเชิงรูปนัย

3.2 ข้อกำหนดความต้องการของระบบยูเอวี

จากการวิเคราะห์การทำงานของระบบยูเอวีแล้วพบว่า ระบบยูเอวีเป็นชุดของภารกิจที่ทำงานภายใต้ระบบเวลาจริงที่ทำงานบนระบบสมองกลฝังตัว ดังนั้นผู้วิจัยจึงทำการรวบรวมความต้องการของระบบให้เป็นรายการคุณลักษณะที่จำเป็นดังนี้

1) ระบบยูเอวีเป็นระบบที่ทำงานต่อเนื่องไม่มีวันสิ้นสุด

ระบบสัญญาณนาฬิกาในระบบเวลาจริงเป็นระบบที่ทำงานต่อเนื่องไม่มีวันหยุด (Infinite execution) ส่งผลให้ระบบยูเอวีเป็นระบบที่ทำงานต่อเนื่องไม่มีวันสิ้นสุดเช่นเดียวกัน คุณลักษณะนี้จึงเป็นคุณลักษณะสำคัญที่จะทำให้ระบบนาฬิกามีความใกล้เคียงระบบสัญญาณนาฬิกาในระบบเวลาจริง

2) ระบบยูเอวีจะต้องสามารถทำงานแบบออโตมาตาเวลาได้

ระบบนาฬิกาที่ทำงานอยู่บนระบบเวลาจริงจะต้องมีการนับเวลาและมีค่าปริมาณเวลากำกับไว้ด้วยเพื่อให้ภารกิจต่างๆ สามารถที่จะอ้างอิงปริมาณเวลาที่ผ่านไป และจะต้องมีการรีเซตให้กลับเป็นเริ่มต้นเมื่อถึงค่าที่กำหนดไว้

3) ระบบยูเอวีจะต้องมีจำนวนสถานะจำกัด

ระบบยูเอวีต้องมีจำนวนสถานะจำกัด เป็นคุณลักษณะที่สำคัญสำหรับการทวนสอบด้วยวิธีการตรวจสอบแบบจำลอง

4) ระบบยูเอวีสามารถที่จะจองทรัพยากรและนับเวลาในการทำงานเมื่อถือครองภารกิจ

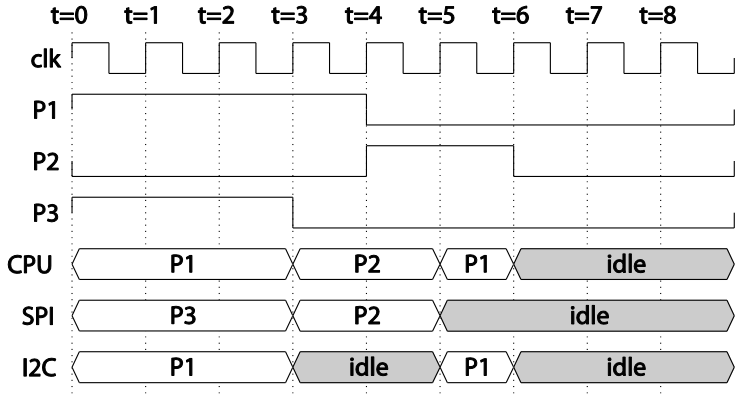
ระบบยูเอวีจะต้องรองรับการทำงานของภารกิจที่จะเข้าครอบครองทรัพยากรที่กำหนดไว้และทำการคืนทรัพยากรให้กับระบบเมื่อได้ครอบครองตามเงื่อนไขเวลาแล้ว

5) ระบบยูเอวีมีการใช้ระดับความสำคัญในการกำหนดสิทธิ์ในการครอบครองทรัพยากร

ระบบยูเอวีจะต้องสามารถเลือกภารกิจที่เหมาะสมได้หากภารกิจสองอย่างขึ้นไปที่ต้องการจะทำงานเข้าถึงทรัพยากรเดียวกัน การเลือกว่าภารกิจได้จะได้สิทธิในการเข้าถึงทรัพยากรนั้นๆจะถูกกำหนดไว้โดยลำดับความสำคัญของแต่ละภารกิจ

6) ระบบยูเอวีสามารถขัดจังหวะการทำงานระหว่างภารกิจตามระดับความสำคัญ

ระบบยูเอวีจะต้องมีคุณลักษณะเฉพาะคือความสามารถในการขัดจังหวะระหว่างการทำงานของภารกิจโดยการทำพรีเอมทีฟ (Pre-emptive) ถ้า P1 และ P2 เป็นภารกิจที่ระบบยูเอวีมีและทำงานพร้อมๆกัน โดยที่ P2 มีระดับความสำคัญมากกว่า P1 รูปที่ 3.2 แสดงการขัดจังหวะโดยภารกิจ P2 ที่ขัดจังหวะการทำงานของภารกิจ P1 และเมื่อ P2 ทำงานจนเสร็จแล้วจึงปล่อยให้กับ P1 ดำเนินการต่อ



รูปที่ 3.2 แผนภาพเวลาของกิจกรรม

7) ระบบยูเอวีจะต้องอนุญาตให้ภารกิจทำงานขนานกันได้หากไม่มีการแย่งทรัพยากรกัน

ระบบที่สร้างต้องมีคุณลักษณะเฉพาะในระบบสมองกลฝังตัวคือการทำงานขนานเช่น หากภารกิจสองภารกิจทำงานพร้อมกันโดยไม่มีการแย่งทรัพยากรกัน เป็นต้น

8) ระบบยูเอวีจะอนุญาตให้ภารกิจมีโอกาสครอบครองทรัพยากรได้ หากภารกิจมีระดับความสำคัญเท่ากัน

การเกิดกรณีที่มีภารกิจที่มีระดับความสำคัญเท่ากัน ทำงานในช่วงเวลาเดียวกันและต้องการทรัพยากรเหมือนกัน สิทธิในการครอบครองทรัพยากรของภารกิจจะมีค่า คือไม่สามารถคาดการณ์ได้ (Undeterministic) หมายความว่าภารกิจทั้งคู่สามารถที่จะครอบครองทรัพยากรได้ แล้วแต่โอกาสว่าภารกิจไหนสามารถเข้าถึงได้ก่อน

9) ระบบยูเอวีสามารถสร้างภารกิจใหม่เมื่อภารกิจที่กำหนดเสร็จสิ้นได้

ภารกิจใหม่สามารถถูกสร้างขึ้นมาหลังจากที่ภารกิจที่กำหนดถูกสร้างขึ้นแล้วและสามารถเปลี่ยนแปลงเส้นทางการทำงานได้

10) ระบบยูเอวีสามารถจัดกำหนดรายการเวลาแบบระดับความสำคัญคงที่

ระบบที่สร้างสามารถทำการเปรียบเทียบว่าภารกิจใดมีระดับความสำคัญ (Priority) มากที่สุดและเลือกภารกิจนั้นให้มีสิทธิในการครอบครองทรัพยากร

11) ระบบยูเอวีสามารถจัดกำหนดรายการเวลาแบบเส้นตายเป็นหลัก

ระบบที่สร้างสามารถการเปรียบเทียบว่าระยะเวลากระทำการ (Duration) ของภารกิจใดสั้นที่สุด และเลือกภารกิจนั้นให้มีสิทธิ์ในการครอบครองทรัพยากร

12) ระบบยูเอวีสามารถจัดกำหนดรายการเวลาแบบเส้นตายเร็วสุด

ทำการเปรียบเทียบว่าเส้นตาย (Deadline) ของภารกิจใดสั้นที่สุด และเลือกภารกิจนั้นให้มีสิทธิในการครอบครองทรัพยากร

13) ระบบยูเอวีสามารถทวนสอบโดยใช้ตัวดำเนินการ □ และกำหนดช่วงเวลาได้

ระบบที่สร้างสามารถที่จะใช้ตัวดำเนินการ □ (Infinitely) ในการสำหรับทวนสอบความถูกต้องของเงื่อนไขเวลาที่กำหนดไว้ได้แบบจำลองได้

14) ระบบยูเอวีสามารถทวนสอบโดยใช้ตัวดำเนินการ ◊ และกำหนดช่วงเวลาได้

ระบบที่สร้างสามารถที่จะใช้ตัวดำเนินการ ◊ (Eventually) ในการสำหรับทวนสอบความถูกต้องของเงื่อนไขเวลาที่กำหนดไว้ได้แบบจำลองได้

15) ระบบยูเอวีมีการกำหนดหน่วยเวลา (Time unit)

ในการสร้างแบบจำลองในระบบเวลาจริงนั้น มีความจำเป็นที่จะต้องนำเอาฐานเวลาจริง (Real-Value Clock) เข้าไปเป็นส่วนประกอบเพื่อใช้สำหรับในการวัดหน่วยเวลา การเทียบหน่วยเวลาในระบบเวลาจริงจะใช้หน่วยเรียกว่าติ๊ก (Tick) เทียบแทนหน่วยเวลาที่น้อยที่สุดที่ระบบสามารถทำได้

16) ระบบยูเอวีมีการกำหนดจำนวนติ๊กต่อรอบ (TotalTickPerLoop)

ในระบบเวลาจริงจะมีการกำหนดรอบการทำงาน ไว้สำหรับให้หน่วยประมวลใช้ในการกำหนดจังหวะการทำงาน เช่น รอบการทำงานเท่ากับ 50 มิลลิวินาที เป็นต้น และจากค่าที่กำหนดไว้ที่ *หน่วยเวลา* จึงสามารถกำหนดได้ว่าในหนึ่งรอบการทำงานมีการทำงานทั้งหมด 50 ติ๊ก

17) ระบบยูเอวีมีการกำหนดแบบจำลองทรัพยากร (Resource model)

ในการทำงานในแต่ละภารกิจในระบบเวลาจริงนั้นต้องการทรัพยากรเพื่อการทำงาน จากระบบตัวอย่างจึงกำหนดทรัพยากรสามอย่าง แตกต่างกันในแต่ละภารกิจ อย่างแรกคือหน่วยประมวลผลและบัสข้อมูลอีกสองประเภทที่ทุกภารกิจต้องแบ่งกันใช้ร่วมกัน

3.3 การออกแบบแบบจำลอง

เมื่อผู้วิจัยทำการวิเคราะห์ความต้องการของระบบแล้วจึงแบ่งกลุ่มความต้องการออกเป็น 5 กลุ่มตามลักษณะเฉพาะที่มีความคล้ายกันและได้มีการออกแบบให้มี 5 แบบจำลองย่อย เพื่อทำหน้าที่ต่างๆ ที่แจกแจงในแต่ละกลุ่มลักษณะเฉพาะเหล่านี้ดังนี้

1) แบบจำลองย่อยสัญญาณนาฬิกา (Ticking Submodel)

- ระบบยูเอวีเป็นระบบที่ทำงานต่อเนื่องไม่มีวันสิ้นสุด
- ระบบยูเอวีจะต้องสามารถทำงานแบบออโตมาตาเวลาได้
- ระบบยูเอวีจะต้องมีจำนวนสถานะจำกัด

2) แบบจำลองย่อยการจัดกำหนดรายการเวลา (Scheduler Submodel)

- ระบบยูเอวีสามารถจัดกำหนดรายการเวลาแบบระดับความสำคัญคงที่
- ระบบยูเอวีสามารถจัดกำหนดรายการเวลาแบบเส้นตายเป็นหลัก
- ระบบยูเอวีสามารถจัดกำหนดรายการเวลาแบบเส้นตายเร็วสุด

3) แบบจำลองย่อยของภารกิจ (Task submodel)

- ระบบยูเอวีสามารถที่จะจองทรัพยากรและนับเวลาในการทำงานเมื่อถือครองภารกิจ
- ระบบยูเอวีมีการใช้ระดับความสำคัญในการกำหนดสิทธิในการครอบครองทรัพยากร
- ระบบยูเอวีสามารถขัดจังหวะการทำงานระหว่างภารกิจตามระดับความสำคัญ

- ระบบจะอนุญาตให้ภารกิจทำงานขนานกันได้หากไม่มีการแย่งทรัพยากรกัน
- ระบบจะอนุญาตให้ภารกิจมีโอกาสครอบครองทรัพยากรได้ หากภารกิจมีระดับความสำคัญเท่ากัน

4) แบบจำลองย่อยภารกิจไม่อิสระ (Dependent task submodel)

- ระบบยูเอวีสามารถสร้างภารกิจใหม่เมื่อภารกิจที่กำหนดเสร็จสิ้นได้

5) แบบจำลองย่อยเครื่องประมวลผลเอมทีแอล (MTL Engine Submodel)

- ระบบยูเอวีสามารถทวนสอบโดยใช้ตัวดำเนินการ □ และกำหนดช่วงเวลาได้
- ระบบยูเอวีสามารถทวนสอบโดยใช้ตัวดำเนินการ ◊ และกำหนดช่วงเวลาได้

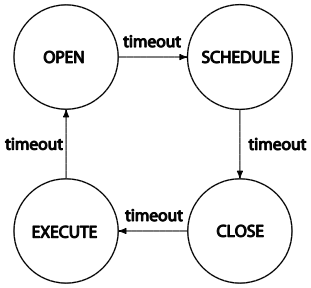
6) แบบจำลองพารามิเตอร์ของระบบ

- ระบบยูเอวีมีการกำหนดหน่วยเวลา (Time unit)
- ระบบยูเอวีมีการกำหนดจำนวนติ๊กต่อรอบ (TotalTickPerLoop)
- ระบบยูเอวีมีการกำหนดแบบจำลองทรัพยากร (Resource model)

3.4 แบบจำลองย่อยสัญญาณนาฬิกา (Ticking Submodel)

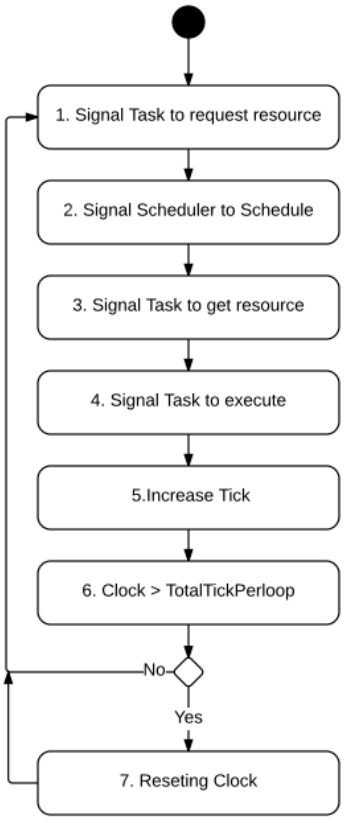
การออกแบบแบบจำลองย่อยสำหรับให้จังหวะเวลากับระบบ และกำหนดค่านาฬิการะบบเพื่อกำหนดว่าภารกิจในระบบเวลาจริงควรจะเริ่มทำงานเมื่อใด แบบจำลองย่อยดังกล่าวนี้จะเขียนให้อยู่ในลักษณะของ ออโตมาตาเวลา (Timed automata) โดยจะมีส่วนประกอบเป็นส่วนที่นับเวลาที่ระบบทำงานเป็นหน่วยติ๊ก

จากการวิเคราะการทำงานของภารกิจบนระบบยูเอวีพบว่า จำเป็นที่ต้องแบ่งการทำงานของภารกิจออกเป็น 4 สถานะ เพื่อให้สะดวกกับการทำงานของระบบจัดกำหนดรายการให้เกิดการประสานการทำงาน (Syncronize) ระหว่างภารกิจที่ทำงานพร้อมๆ กัน โดยแบ่งออกเป็นสถานการณ์ทำงานได้สี่สถานะดังรูปที่ 3.3 แต่ละสถานะจะรอภารกิจทำงานจนเสร็จก่อนถึงเปลี่ยนเป็นสถานะถัดไป



รูปที่ 3.3 สถานะของ GlobalClockState

- สถานะ Open เป็นสถานะที่แสดงสัญญาณให้แต่ละภารกิจ ตรวจสอบว่าสามารถเปลี่ยนสถานะเป็น แอกทีฟ (Active) หรือได้หรือไม่
- สถานะ Scheduling เป็นสถานะที่มีไว้เพื่อให้แบบจำลองย่อยการจัดกำหนดรายการเวลา ได้ทำตามคำสั่ง
- สถานะ Close เป็นสถานะที่ทำหน้าที่ในการเลือกเอาภารกิจที่ได้ถูกจัดเรียงโดยสถานะการจัดกำหนดรายการเวลาให้มีสิทธิ์ในการเข้าถึงทรัพยากร
- สถานะ Execute เป็นสถานะที่กำหนดให้ภารกิจที่ได้ครอบครองทรัพยากร สามารถที่จะเพิ่มเวลากระทำการได้ 1 หน่วยเวลา



รูปที่ 3.4 แผนภาพกิจกรรมของแบบจำลองย่อยสัญญาณนาฬิกา

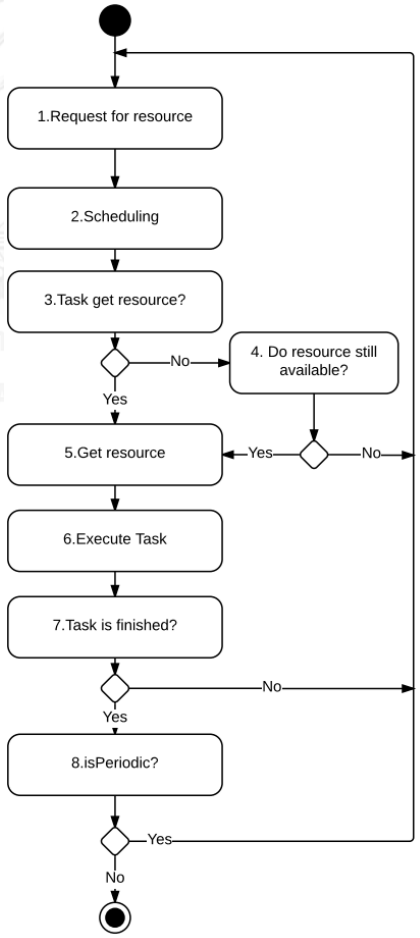
จากแผนภาพกิจกรรมของแบบจำลองย่อยสัญญาณนาฬิกาที่แสดงในรูปที่ 3.4 สามารถแสดงรายละเอียดของแต่ละกิจกรรมได้ดังนี้

1) แบบจำลองจะส่งสัญญาณให้ทุกๆ ภารกิจร้องขอทรัพยากร
2) แบบจำลองส่งสัญญาณให้ตัวจัดกำหนดรายการเวลาทำการเรียงภารกิจตามโทโพโลยี

3) แบบจำลองส่งสัญญาณให้แบบจำลองจัดกำหนดรายการเวลาเลือกภารกิจให้กับทรัพยากร
4) แบบจำลองส่งสัญญาณให้ภารกิจที่ได้ทรัพยากรทำงาน
5) แบบจำลองเพิ่มเวลา 1 หน่วยเวลา
6) ตรวจค่าหน่วยเวลาของแบบจำลอง
7) ถ้าเวลามากกว่าค่าเวลาต่อรอบจะทำการรีเซตเป็นค่าเริ่มต้นใหม่ หากไม่กลับไปทำข้อ 1)

3.5 แบบจำลองย่อยภารกิจ (Task submodel)

เป็นแบบจำลองย่อยที่อธิบายการทำงานของแต่ละภารกิจที่สามารถทำงานพร้อมๆ กัน ในระบบเวลาจริง และเมื่อมีภารกิจที่ทำงานพร้อมๆ กันก็จะอนุญาตให้ทำงานพร้อมกันในลักษณะจวบกัน เมื่อถึงช่วงเวลาที่กำหนดไว้ภารกิจแต่ละภารกิจจะพยายามเข้าถึงทรัพยากรเพื่อที่จะทำตามเงื่อนของตนเอง ภารกิจที่เป็นคาบเวลาต้องครอบครองทรัพยากรให้ได้ตามเวลาที่กำหนดไว้ล่วงหน้าเพื่อที่จะทำให้ภารกิจสำเร็จ แต่ภารกิจแบบที่ไม่เป็นคาบเวลาสามารถที่จะทำเมื่อโอกาสอำนวย และสามารถที่จะเลื่อนเวลาออกไปได้หากมีงานอื่นๆ ครองทรัพยากรที่ต้องการอยู่

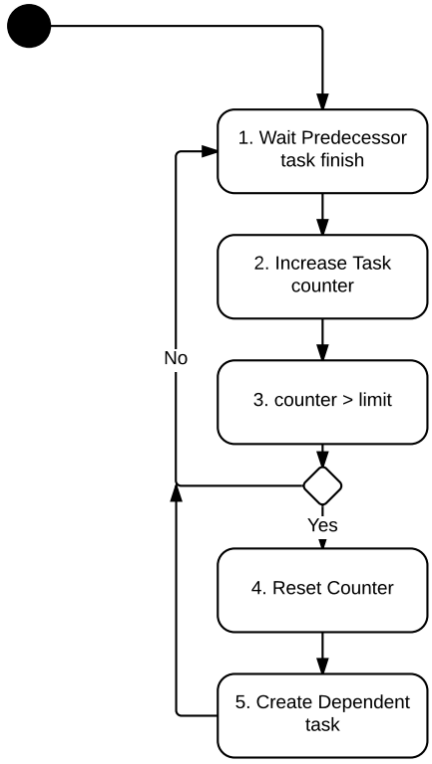


รูปที่ 3.5 แผนภาพกิจกรรมของแบบจำลองย่อยภารกิจ

1) ภารกิจร้องขอทรัพยากร
2) ภารกิจทำการจัดลำดับรายการเวลาตามโทโพโลยี
3) ตรวจว่าภารกิจได้รับทรัพยากรหรือไม่
4) ถ้าไม่ได้รับทรัพยากรตรวจดูว่ายังมีทรัพยากรอื่นๆ เหลือหรือไม่ หากไม่มีให้ไปที่เริ่มต้นใหม่
5) หากได้รับทรัพยากรไปยังกิจกรรมถัดไป
6) ภารกิจทำงาน 1 หน่วยเวลา
7) ตรวจว่าภารกิจทำงานสำเร็จแล้วหรือไม่ หากยังไม่เสร็จให้เริ่มต้นใหม่
8) ตรวจว่าเป็นภารกิจคาบเวลาหรือไม่หากเป็นให้เริ่มต้นใหม่ หากไม่เป็นให้สิ้นสุดการทำงาน

3.6 แบบจำลองย่อยภารกิจไม่อิสระ (Dependent task submodel)

เป็นแบบจำลองสำหรับสร้างภารกิจไม่อิสระ ภารกิจใดๆ สามารถที่จะสร้างภารกิจอื่นๆ สร้างขึ้นมาใหม่ได้อย่างต่อเนื่อง ยกตัวอย่างการทำงาน เช่น เมื่อวัดสัญญาณจากเซนเซอร์ ก็จำเป็นที่จะต้องรวบรวมไว้จำนวนหนึ่งก่อนที่จะทำการแปลงหรือจัดการข้อมูลดังกล่าวและทำการจดบันทึกลงในหน่วยความจำสถิต เป็นต้น แผนภาพกิจกรรมการทำงานของแบบจำลองย่อยภารกิจไม่อิสระ ในรูปที่ 3.6



รูปที่ 3.6 แผนภาพกิจกรรมของแบบจำลองย่อยภารกิจไม่อิสระ

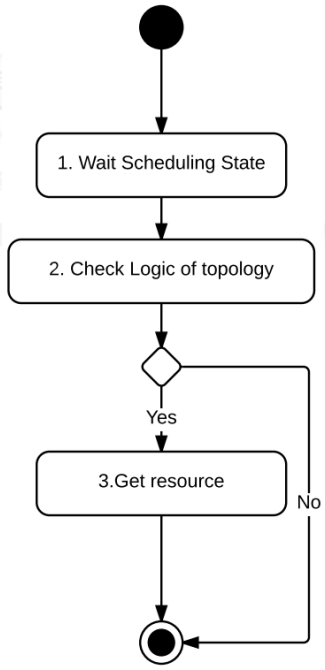
จากแผนภาพกิจกรรมของแบบจำลองย่อยภารกิจไม่อิสระที่แสดงในรูปที่ 3.6 สามารถแสดงรายละเอียดของแต่ละกิจกรรมได้ดังนี้

1) เมื่อเริ่มต้น แต่ละเส้นทางของภารกิจไม่อิสระเริ่มทำงานจะรอ ให้ภารกิจเริ่มต้นทำงานเสร็จแล้วเพื่อที่จะเริ่มการทำงานในเส้นทางดังกล่าว
2) เพิ่มจำนวนนับสำหรับเส้นทางภารกิจไม่อิสระ
3) ตรวจว่าจำนวนนับมีค่ามากกว่าที่กำหนดไว้หรือไม่ หากไม่ให้กลับไปรอที่ 1)
4) ทำการรีเซตตัวนับ
5) สร้างภารกิจไม่อิสระเมื่อเงื่อนไขครบ

3.7 แบบจำลองย่อยการจัดกำหนดรายการเวลา (Scheduler Submodel)

เป็นแบบจำลองภาษาโปรเมลาสำหรับจำลองการจัดกำหนดรายการเวลาบนระบบเวลาจริง เป็นชุดแบบจำลองที่ทำหน้าที่ตัดสินในกรณีที่มีภารกิจมากกว่าหนึ่งภารกิจ ต้องการทรัพยากรเดียวกันในช่วงเวลาเดียวกัน การตัดสินจะเลือกตามโทโพโลยีการจัดกำหนดรายการเวลาที่ผู้ใช้เลือกไว้

แผนภาพกิจกรรมการทำงานของแบบจำลองย่อยสัญญาณนาฬิกา แบบจำลองย่อยเริ่มต้นทำงานเมื่อตัวแปรภารกิจอยู่ในสถานะจัดเรียง



รูปที่ 3.7 แผนภาพกิจกรรมของแบบจำลองย่อยจัดกำหนดรายการเวลา

จากแผนภาพกิจกรรมของแบบจำลองย่อยการจัดกำหนดรายการเวลาแสดงในรูปที่ 3.7 สามารถแสดงรายละเอียดของแต่ละกิจกรรมได้ดังนี้

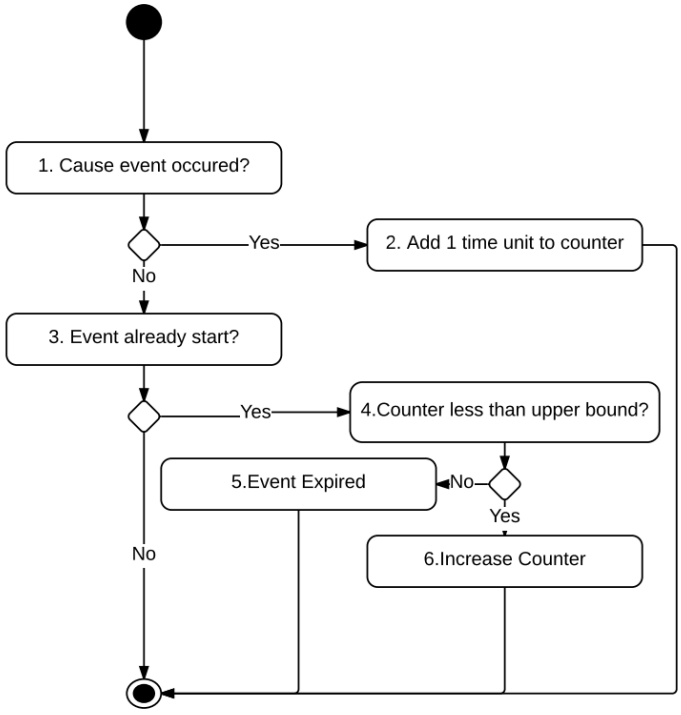
1) รอจนกว่าทุกภารกิจจะอยู่ในสถานะจัดเรียงเพื่อให้ทุกภารกิจทำงานพร้อมกัน

2) ตรวจตรรกะตามโทโพโลยี ให้ค่าเป็นจริงหากได้รับเลือกตามโทโพโลยี

3) หากเป็นจริงจะจัดสรรทรัพยากรให้กับภารกิจที่ได้รับเลือก

3.8 แบบจำลองย่อยเครื่องประมวลผลเอมทีแอล (MTL engine submodel)

เป็นแบบจำลองสำหรับช่วยให้การแปลภาษาเอมทีแอล (Metric Temporal Logic : MTL) ซึ่งเป็นภาษาที่ได้รับความนิยมในการอธิบายระบบเวลาจริงเพราะภาษาเอมทีแอลสามารถอธิบายคุณลักษณะเฉพาะของระบบเวลาจริงได้ซับซ้อนกว่าแอลทีแอล เพื่อเขียนให้อยู่ในรูปของสูตรแบบอย่างคุณลักษณะ (Specification Pattern) สำหรับทวนสอบความถูกต้องของแบบจำลองได้ ผู้วิจัยจำเป็นที่จะต้องสร้างวิธีการแปลสูตรแบบเอมทีแอล ให้เป็นสูตรแบบแอลทีแอล เพื่อที่จะสามารถอธิบายการทำงานในระบบเวลาจริงได้อย่างครบถ้วน และสามารถที่จะตรวจสอบด้วยเครื่องมือตรวจสอบแบบจำลองสปินได้



รูปที่ 3.8 แผนภาพกิจกรรมของแบบจำลองย่อยเครื่องประมวลผลเอมทีแอล

จากแผนภาพกิจกรรมของแบบจำลองย่อยเครื่องประมวลผลเอมทีแอลที่แสดงในรูปที่ 3.8 สามารถแสดงรายละเอียดของแต่ละกิจกรรมได้ดังนี้

1) ทุกๆ รอบของนาฬิกาทำการตรวจว่ามีเหตุการณ์ที่เป็น เหตุการณ์เริ่มต้นหรือไม่
2) หากมีให้เพิ่มเวลา 1 หน่วย และกำหนดให้เหตุการณ์เริ่ม
3) หากไม่มีเหตุการณ์ ตรวจว่าเหตุการณ์เคยเริ่มแล้วหรือไม่
4) หากเหตุการณ์เคยเริ่มแล้ว ตรวจว่าเวลาเกินขอบเขตบนหรือไม่
5) หากเกินให้ตั้งว่าเหตุการณ์เกินกำหนดแล้ว
6) หากไม่เกินให้เพิ่มเวลา 1 หน่วย

3.9 การสร้างแบบอย่างคุณลักษณะเอมทีแอล

ผู้วิจัยใช้ เอมทีแอล สำหรับสร้างแบบอย่างคุณลักษณะ โดยเอมทีแอลมีข้อดีกว่าแอลทีแอลคือสามารถกำหนดเงื่อนไขด้านเวลา (Timing Constraint) สำหรับตรวจสอบว่าภารกิจดังกล่าวได้ทำงานตรงกับห้วงเวลาที่กำหนดไว้ตามเงื่อนไขหรือไม่ โดยมีส่วนประกอบคือ เหตุการณ์ลั่นไก (TriggerEvent) และเหตุการณ์ปฏิกิริยา (Reaction Event) คั่นด้วยเครื่องหมายอิมพลิเคชั่น (Implication) แบบอย่างคุณลักษณะเอมทีแอลเขียนได้ดังนี้

$$\Box(\text{TriggerEvent} \rightarrow \Box_{[t1,t2]}\text{ReactionEvent})$$

จากข้อกำหนดเงื่อนไขด้านเวลาของระบบเวลาจริงนั้นสามารถแบ่งออกได้เป็นสองแบบคือ

1) เงื่อนไขสำหรับเหตุการณ์โดยใช้ตัวดำเนินการ □

เป็นเงื่อนไขสำหรับการเกิดเหตุการณ์ที่มีเวลาแน่นอน เช่น ภารกิจที่เป็นระบบเวลาจริงแบบแข็ง มีข้อบังคับให้ภารกิจนั้นต้องทำงานในห้วงเวลาที่กำหนดเฉพาะ เช่นเมื่อภารกิจเซนเซอร์ 1 เริ่มทำงาน ภารกิจนำร่องด้วยความเฉื่อยต้องทำงานในติ๊กที่ 5 ถึง 7 เสมอ เป็นต้น หากภารกิจนำร่องด้วยความเฉื่อยไม่สามารถทำงานในเฟสที่กำหนดไว้ได้ถือว่าภารกิจและระบบทำงานผิดพลาด จะใช้สูตรเอมทีแอลเขียนได้ดังนี้

$$\Box(\text{TriggerEvent} \rightarrow \Box_{[5,7]}\text{ReactionEvent})$$

- กำหนดตัวแปรเหตุการณ์ TriggerEvent เป็นจริงเมื่อภารกิจเซนเซอร์ 1 เริ่มทำงาน
- กำหนดตัวแปรเหตุการณ์ ReactionEvent เป็นจริงเมื่อภารกิจนำร่องด้วยความเฉื่อยเริ่มทำงาน

2) เงื่อนไขสำหรับเหตุการณ์โดยใช้ตัวดำเนินการ ◊

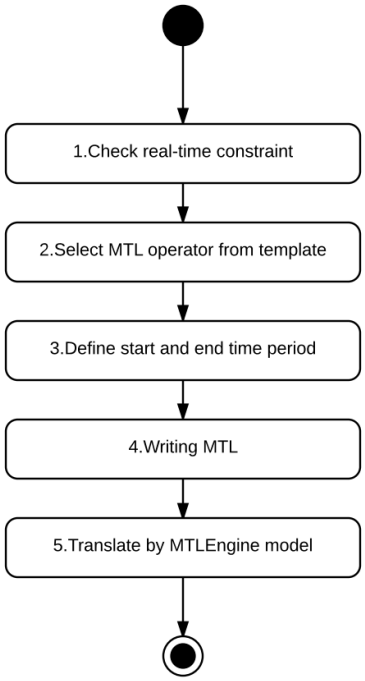
สำหรับข้อกำหนดสำหรับการเกิดเหตุการณ์ที่ต้องมีการตอบสนองภายในเงื่อนไขเวลา เช่น เมื่อภารกิจเซนเซอร์ 1 ต้องทำงานแล้ว จะต้องมีการทำงานของภารกิจนำร่องด้วยความเฉื่อยอย่างน้อย 1

ครั้งภายใน 5 ถึง 7 หน่วยเวลา เป็นต้นหากภารกิจดังกล่าวไม่สามารถทำงานในเฟสที่กำหนดไว้ได้ถือว่าภารกิจและระบบทำงานผิดพลาด จะใช้สูตรเอมทีแอลเขียนได้ดังนี้

$$\Box(\text{TriggerEvent} \rightarrow \Diamond_{[5,7]}\text{ReactionEvent})$$

- กำหนดตัวแปรเหตุการณ์ TriggerEvent เป็นจริงเมื่อภารกิจเซนเซอร์ 1 เริ่มทำงาน
- กำหนดตัวแปรเหตุการณ์ ReactionEvent เป็นจริงเมื่อภารกิจนำร่องด้วยความเฉื่อยเริ่มทำงาน

การแปลงประโยคแบบอย่างที่ต้องการทวนสอบ ให้เป็นรูปแบบแบบอย่างคุณลักษณะเอมทีแอล นั้นสามารถที่จะทำได้ดังรูปที่ 3.9



รูปที่ 3.9 แผนภาพกิจกรรมกระบวนการสร้างแบบอย่างคุณลักษณะเอมทีแอล

จากแผนภาพกิจกรรมของสามารถแสดงรายละเอียดของแต่ละกิจกรรมได้ดังนี้

1) ทำการตรวจสอบเงื่อนไขด้านเวลาของของระบบยูเอวี ดังนี้
   - เหตุการณ์ที่เกิดขึ้นประกอบด้วยเหตุการณ์ลั่นไก และ เหตุการณ์ปฏิกิริยา ตัวแปรใช้สอดคล้องกันทั้งแบบจำลอง
   - จำเป็นที่จะต้องมีตัวดำเนินการอิมพลิเคชั่น (Implication) ที่มีสัญลักษณ์แทนด้วย → ในสูตรเอมทีแอลส่วนของตัวดำเนินการสามารถใช้ได้เพียง ◊ และ □ เท่านั้น

- ส่วนของหน่วยเวลาที่กำหนดไว้กับตัวดำเนินการ โดยกำหนดให้เป็นตัวเลขจำนวนเต็มบวกเท่านั้น สามารถกำหนดหน่วยเวลาเริ่มต้นและสิ้นสุดได้

2) เลือกชนิดของตัวดำเนินตามเงื่อนไขของเงื่อนไขด้านเวลาว่าเป็นตัวดำเนินการแบบ □ หรือ ◊ ตามชนิดของเงื่อนไขด้านเวลา
3) กำหนดตัวแปร และเวลาเริ่มต้นของช่วงเวลา และ สิ้นสุดช่วงเวลา
4) สร้างสูตรเอมทีแอล
5) แปลเอมทีแอลเป็นแอลทีแอลแม่แบบแอลทีแอล โดยการเปลี่ยนตัวดำเนินการเอมทีแอลเป็นแอลทีแอลจากแม่แบบสำหรับแปล 5 แบบดังนี้ โดยกำหนด *p* เป็นเหตุการณ์ปฏิกิริยา และ Counter เป็นตัวนับที่ใช้ร่วมกับแบบจำลองเครื่องประมวลผลเอมทีแอล

- $\Diamond_{(t1,t2)}p$ แปลเป็น $\Diamond(p \wedge Counter>t1 \wedge Counter<t2)$
- $\Diamond_{[t1,t2]}p$ แปลเป็น $\Diamond(p \wedge Counter>=t1 \wedge Counter<=t2)$
- $\Box_{(t1,t2)}p$ แปลเป็น $\Box(p \wedge Counter>t1 \wedge Counter<t2)$
- $\Box_{[t1,t2]}p$ แปลเป็น $\Box(p \wedge Counter>=t1 \wedge Counter<=t2)$

จากสูตร

$$\Box(TriggerEvent \rightarrow \Diamond_{[t1,t2]} ReactionEvent)$$

เมื่อทำการแปลแล้วจะได้ดังนี้จะได้

$$\Box(TriggerEvent \rightarrow \Diamond(ReactionEvent \wedge Counter>=t1 \wedge Counter<=t2))$$

และทำการกำหนดตัวแปร Counter ที่ร่วมกับแบบจำลองย่อยเครื่องประมวลผลเอมทีแอล เพื่อให้ทำการนับเวลาเอมทีแอลโดยอัตโนมัติ

ตัวอย่างการแปลเงื่อนไขด้านเวลาของระบบเวลาจริงให้เป็นเอมทีแอล

กำหนดให้มีภารกิจสองภารกิจ คือ ภารกิจเซนเซอร์ 1 ตัวย่อคือ Sen1 เป็นภารกิจที่ทำงานตามเวลา และ ภารกิจนำร่องด้วยความเฉื่อยตัวย่อคือ INS ทั้งสองภารกิจมีลำดับเวลาที่ทำต่อเนื่องกันอย่างแม่นยำ โดยมีเงื่อนไขด้านเวลาคือ เมื่อภารกิจเซนเซอร์ 1 เริ่มทำงานแล้ว ภารกิจ นำร่องด้วยความเฉื่อยจะต้องทำงานในอีก 28 หน่วยเวลาให้หลังเท่านั้น เมื่อไม่มีภารกิจอื่นๆ ทำงานอยู่เลย นอกจากภารกิจ 2 ภารกิจข้างต้นภารกิจทั้งสองสามารถทำตามต่อกันได้อย่างแม่นยำ แต่ผู้ทวนสอบ

แบบจำลองต้องการทวนสอบว่าหากการทำงานจริงร่วมกับภารกิจอื่นๆ ที่ทำงานแทรกกันและอาจจะเกิดการขัดขวางกันได้ ภารกิจทั้งสองยังจะสามารถทำงานได้ตามเงื่อนไขเวลาที่กำหนดหรือไม่

1) **ทำการตรวจสอบเงื่อนของระบบเวลาจริง** ตรวจสอบพบว่า “เมื่อภารกิจเซนเซอร์ 1 เริ่มทำงานแล้ว ภารกิจนำร่องด้วยความเฉื่อยจะต้องทำงานในอีก 28 หน่วยเวลาให้หลังเท่านั้น” นั้นมีเหตุการณ์เริ่มต้นและสิ้นสุด และหน่วยเวลากำหนดไว้ถูกต้อง
2) **เลือกชนิดของตัวดำเนินการตามแม่แบบเอมทีแอล** จากเงื่อนไขด้านเวลาจากที่กำหนดแล้ว ทำการเลือกตัวดำเนินการคือ □ เนื่องจากเหตุการณ์มีการกำหนดให้เกิดเหตุการณ์เริ่มต้นตามช่วงเวลาที่กำหนดไว้
3) **กำหนดตัวแปร และเวลาเริ่มต้นของช่วงเวลา และ สิ้นสุดช่วงเวลา** กำหนดชื่อตัวแปร Sen1Event แทน เมื่อภารกิจเซนเซอร์ 1 เริ่มทำงาน และ ตัวแปร INSEvent แทน ภารกิจนำร่องด้วยความเฉื่อยทำงาน จากเงื่อนไขด้านเวลาเวลาเริ่มต้นและสิ้นสุดคือ 28 หน่วยเวลา
4) **เขียนเงื่อนไขเป็นเอมทีแอล** จากเงื่อนไขด้านเวลาสามารถเขียนเอมทีแอลได้ดังนี้

$$\Box(\text{Sen1Event} \rightarrow \Box_{[25]}\text{INSEvent})$$

5) **แปลเอมทีแอลเป็นแอลทีแอลโดยใช้แบบจำลองเครื่องประมวลผลเอมทีแอล** โดยการนำเอาสูตรเอมทีแอล ดังกล่าวเปลี่ยนตัวดำเนินการตามแม่แบบแอลทีแอล โดย เลือกแม่แบบ $\Box_{[t1,t2]}p$ ซึ่งเป็นแม่แบบสำหรับตัวดำเนินการแบบ □ สำหรับใช้ช่วงปิด t_1 ถึง t_2

$$\Box(\text{Sen1Event} \rightarrow \Box(\text{INSEvent} \wedge \text{Counter}>=28 \wedge \text{Counter}<=28))$$

และทำการกำหนดตัวแปร Counter สำหรับใช้ในการนับให้กับแบบจำลองเครื่องประมวลผลเอมทีแอลเพื่อใช้ในการนับโดยอัตโนมัติ

บทที่ 4

แบบจำลองเชิงรูปนัยระบบยูเอวี

จากกระบวนการสร้างบทที่แบบจำลองเชิงรูปนัยระบบยูเอวีที่กล่าวไว้ในบทที่ 3 ผู้วิจัยได้ลงมือสร้างแบบจำลองด้วยภาษาโปรเมลา โดยนำแผนภาพกิจกรรมและแผนภาพสถานะของแต่ละระบบมาขยายผลด้วยภาษาโปรเมลา แบบจำลองเชิงรูปนัยของระบบยูเอวีประกอบด้วย 5 แบบจำลองย่อย ดังนี้ แบบจำลองย่อยสัญญาณนาฬิกา แบบจำลองย่อยภารกิจ แบบจำลองย่อยเครื่องประมวลผลเอมทีแอล แบบจำลองย่อยภารกิจไม่อิสระ แบบจำลองย่อยการจัดกำหนดรายการเวลา

4.1 แบบจำลองย่อยสัญญาณนาฬิกา (Ticking submodel)

จากการวิเคราะห์แบบจำลองที่เขียนไว้ในหัวข้อที่ 3.4 ผู้วิจัยทำการ สร้างแบบจำลองย่อยดังกล่าวนี้ในลักษณะของออโตมาตาเวลา โดยที่แบบจำลองย่อยสัญญาณนาฬิกาจะประกอบด้วยส่วนประกอบสำคัญดังนี้

1) ตัวแปร GlobalClock

เป็นตัวแปรครอบคลุมชนิดจำนวนเต็มบวกเรียกว่า *GlobalClock* ทุกพร็อกไทป์สามารถอ่านค่าได้สำหรับใช้ในการอ้างอิงเวลาให้กับทุกพร็อกไทป์ในระบบ

2) ตัวแปร GlobalClockState

เป็นตัวแปรครอบคลุมเรียกว่า *GlobalClockState* ทุกพร็อกไทป์สามารถอ่านค่าได้ มีสี่สถานะ คือ *Open*, *Schedule*, *Close* และ *Execute* ซึ่งเป็นตัวกำหนดสถานะของแต่ละภารกิจว่าเหมาะสมในการทำงานใดได้บ้างในขณะนั้นเพื่อให้เกิดการประสานการทำงาน (Syncronize)

3) พร๊อกไทป์ Ticking

เป็นพร๊อกไทป์ที่ทำงานอิสระสำหรับนับสัญญาณนาฬิกา ทำงานวนลูปปรับสถานะเริ่มจากสถานะของ *GlobalClockState* เท่ากับ สถานะ Open จนถึง สถานะ Execute และกลับมาเริ่มการทำงานใหม่ แต่ละสถานะถูกคั่นด้วยคำสั่ง *timeout* เมื่อครบรอบการทำงาน 1 รอบจะเพิ่มค่าของ *GlobalClock* เท่ากับ 1 หน่วยและเมื่อครบ ตามค่าของ *TotalTickPerLoop* จะรีเซตค่า *GlobalClock* กลับเป็นค่าเริ่มต้น

4.1.1 การทำงานของแบบจำลองย่อยสัญญาณนาฬิกา

ผู้วิจัยทำการสร้างแบบจำลองย่อยสัญญาณนาฬิกาด้วยภาษาโปรเมลาเพื่อทำงานให้ได้ตามแผนภาพกิจกรรมที่กำหนดในแผนภาพกิจกรรมรูปที่ 3.4 ในบทที่ 3

```
1  #define TotalTickPerLoop 50
2  #define Priority_Highest 99
3  int GlobalClock = 1;
4  byte GlobalClockState = C_Open;
5  proctype Ticking(){
6  do ::atomic{((GlobalClockState == C_Open) && timeout)->
7      GlobalClockState=C_Schedule; }
8     ::atomic{((GlobalClockState == C_Schedule)&& timeout)->
9      GlobalClockState = C_Close; }
10    ::atomic{((GlobalClockState == C_Close)&& timeout)->
11     GlobalClockState = C_Execute; }
12    ::atomic{((GlobalClockState == C_Execute) && timeout)->
13     MTLEngine()-> Selected = none;
14     HighestGetResource = false;
15     printf ("== TICK :%d CPU[%e] SPI[%e] I2C[%e] \n",
16     GlobalClock-1,Resource[0].owner,Resource[1].owner,Resource[2].owner);
17     GlobalClock++;
18     if :: (GlobalClock > (TotalTickPerLoop)) -> GlobalClock = 1;
19        :: else;  fi;
20   Resource[0].owner = _IDLE_; Resource[1].owner = _IDLE_;
21   Resource[2].owner = _IDLE_;
22   GlobalClockState = C_Open;}
23 od;
```

รูปที่ 4.1 แบบจำลองย่อยสัญญาณนาฬิกาภาษาโปรเมลา

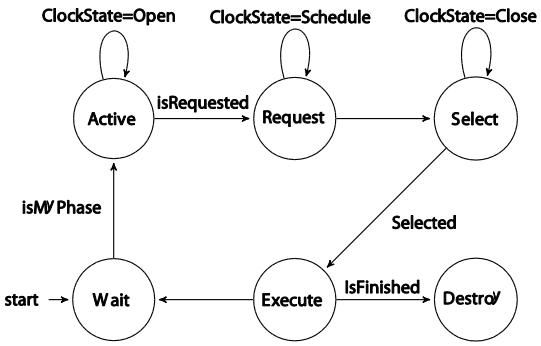
รูปที่ 4.1 แสดงแบบจำลองย่อยสัญญาณนาฬิกาภาษาโปรเมลาโดยมีคำอธิบายการทำงานดังนี้

1) เริ่มต้นแบบจำลองย่อยสัญญาณนาฬิกาจะรอ จนสถานะของตัวแปร *GlobalClockState* จะอยู่ในสถานะ Open แสดงในบรรทัดที่ 6 และ สถานะของ *timeout* เป็นจริง ทำให้สถานะของ *GlobalClockState* จะไม่เลื่อนไปยังสถานะถัดไปจนกว่าทุกๆพร็อกไทป์ที่อยู่บนแบบจำลองได้ทำงานจนสิ้นสุด คำสั่ง timeout เป็นคำสั่งเฉพาะของสปินจะรอจนกว่าทุกๆ พร็อกไทป์ทำงานจนไม่สามารถทำงานต่อได้แล้ว ความจำเป็นที่ต้องใช้คำสั่งนี้เพื่อเป็นการป้องกันไม่ให้ *GlobalClockState* เลื่อนสถานะก่อนที่ภารกิจจะทำตามขั้นตอนเสร็จสิ้น
2) ในบรรทัดที่ 7 ดำเนินการปรับสถานะ *GlobalClockState* ให้ค่าของตัวแปร *GlobalClockState* มีเท่ากับ Schedule
3) แบบจำลองจะรอจนกว่าสถานะ *GlobalClockState* จะอยู่ใน สถานะการจัดกำหนดรายการเวลา และ สถานะของ timeout เป็นจริง
4) ดำเนินการปรับสถานะ *GlobalClockState* ให้เป็นสถานะถัดไปคือ ค่าของตัวแปร *GlobalClockState* มีเท่ากับ Close

5) ในบรรทัดที่ 9 แบบจำลองจะรอจนกว่าสถานะ *GlobalClockState* จะอยู่ใน สถานะปิด และสถานะของ timeout เป็นจริง เพื่อรอให้แบบจำลองทำหน้าที่ในการเลือกเอาภารกิจที่มีระดับความสำคัญสูงสุดตามโทโพโลยีการจัดกำหนดรายการเวลา เมื่อไม่มีพร๊อกไทป์ใดๆ ทำงานแล้วจะไปยังสถานะถัด
6) ดำเนินการปรับสถานะ *GlobalClockState* ให้เป็นสถานะถัดไปคือ สถานะกระทำการ
7) แบบจำลองจะรอให้สถานะ *GlobalClockState* เป็นสถานะกระทำการ และสถานะของ timeout เป็นจริง เพื่อที่จะให้ภารกิจที่ได้รับเลือก จะเข้าครอบครองทรัพยากร เมื่อไม่มีพร๊อกไทป์ใดๆ ทำงานแล้วสถานะของ *timeout* จะเป็นจริงและไปสถานะถัดไป
8) ดำเนินการปรับสถานะ *GlobalClockState* ให้เป็นสถานะถัดไปคือ ค่าของตัวแปร *GlobalClockState* มีเท่ากับ Open
9) เพิ่มค่าของตัวแปร *GlobalClock* เพื่อทำการนับเวลา 1 ติ๊กแสดงในบรรทัดที่ 17
10) ตรวจสอบว่าค่า *GlobalClock* มีค่ามากกว่า *TotalTickPerLoop* หรือไม่ หากไม่ กลับไปทำข้อ 1) ใหม่ หากใช่ ทำข้อ 11)
11) ทำการรีเซต *GlobalClock*

4.2 แบบจำลองย่อยภารกิจ (Task submodel)

ภารกิจจะมีสถานะภารกิจ ทำงานเป็นทีละสถานะและทำงานประสานกัน ผู้วิจัยได้แบ่งสถานะของภารกิจไว้ทั้งหมด 6 สถานะ แสดงแผนภาพสถานะ ตามรูปที่ 4.2 ผู้วิจัยกำหนดให้ 1 ภารกิจเท่ากับ 1 พร๊อกไทป์ทำงาน พร๊อกไทป์จะถูกสร้างและถูกทำลายลงเมื่อภารกิจสิ้นสุดลง แต่หากเป็นภารกิจที่เป็นแบบคาบเวลาจะมีการทำงานซ้ำต่อไปโดยไม่มีวันสิ้นสุด แบบจำลองจะไม่สามารถเรียกภารกิจที่เกิดขึ้นแล้วชนิดเดียวกันซ้ำได้จนกว่าภารกิจนั้นจะถูกทำลายไป



รูปที่ 4.2 สถานะของแบบจำลองย่อยภารกิจ

ส่วนประกอบที่สำคัญของแบบจำลองย่อยภารกิจมีดังนี้

1) พารามิเตอร์ภารกิจ (Task parameters)

พารามิเตอร์ที่สำคัญของภารกิจจะบรรจุอยู่ใน typedef ที่เรียกว่า *Taskdef* ใช้สำหรับการบันทึกค่ารายละเอียดของภารกิจต่างๆ สำหรับการสร้างภารกิจใหม่ ประกอบด้วย ทรัพยากรที่ต้องใช้ ระดับความสำคัญ คาบเวลา เฟสที่เริ่มทำงาน ระยะเวลากระทำการ เป็นต้น

2) มาโคร isMyPhase

เป็นมาโครที่ทำหน้าที่ในการตรวจว่าภารกิจมีคาบเวลาและเฟส ตรงกับ *GlobalClock* หรือไม่ ให้ค่าเป็นจริงหากคาบเวลาตรงกัน มาโครนี้ใช้อำนวยความสะดวกในการตรวจว่าภารกิจสามารถทำงานที่เวลาปัจจุบันได้หรือไม่

3) ตัวแปร Selected

เป็นตัวแปรครอบคลุมที่จะใช้ในการเก็บชื่อภารกิจ เพื่อระบุสิทธิ์ในการเข้าถึงทรัพยากร ภารกิจที่มีสิทธิ์มากที่สุดตามโทโพโลยีการจัดเรียงถูกระบุไว้ในตัวแปรนี้ ภารกิจอื่นๆ จะตรวจดูว่าภารกิจของตนเองถูกระบุไว้ในตัวแปรนี้หรือไม่ หากมีจะสามารถเข้าใช้ทรัพยากรได้

4) พร๊อกไทป์ CreateTask

ภารกิจต่างๆ จะถูกสร้างโดยพร๊อกไทป์ชื่อ *CreateTask* มีหน้าที่สำหรับการสร้างภารกิจใหม่ๆ เข้าไปในระบบแต่ละภารกิจมี 6 สถานะ คือ Wait, Active, Request, Selected, Execute และ Destroy

4.2.1 การทำงานของแบบจำลองย่อยภารกิจ

ผู้วิจัยทำการสร้างแบบจำลองย่อยภารกิจด้วยภาษาโปรเมลาเพื่อทำงานให้ได้ตามแผนภาพกิจกรรมที่กำหนดในแผนภาพกิจกรรมรูปที่ 3.5 ในบทที่ 3 โดยแบ่งส่วนของแบบจำลองออกเป็นส่วนๆ 6 ส่วน แต่ละส่วนถูกแบ่งออกตามเงื่อนไขของคำสั่ง do ผู้วิจัยจึงทำการอธิบายแยกส่วนตามแต่ละสถานะของภารกิจ ในส่วนแรกของแบบจำลองแสดงในรูปที่ 4.3

รูปที่ 4.3 ถึง 4.9 แสดงแบบจำลองย่อยสัญญาณนาฬิกาภาษาโปรเมลา โดยมีคำอธิบายการทำงานดังนี้

1) เมื่อแบบจำลองถูกเรียกใช้ คำสั่งในบรรทัดที่ 2-5 จะทำหน้าที่ในการตั้งค่าตัวแปรสำหรับภารกิจใหม่ เช่น ตัวแปรสำหรับนับรอบการทำงานเรียกว่า *cycleCount* ตัวแปรสำหรับนับเวลากระทำการเรียกว่า *durationCount* ตัวแปรสำหรับแสดงสถานการณ์ทำงานของภารกิจ เป็นต้น นอกจากนั้นมีส่วนที่ป้องกันไม่ให้มีการสร้างภารกิจเดิมซ้ำในบรรทัดที่ 5-9 ภารกิจที่เคยถูกสร้างไปแล้วจะไม่สามารถสร้างใหม่ได้อีกเพื่อป้องกันการซ้ำซ้อนของภารกิจ และป้องกันการเกิดการระเบิดของปริภูมิสถานะ โดยใช้เงื่อนไขในการป้องกันในบรรทัดที่ 9

เมื่อเงื่อนไขเป็นคือมีการสร้างภารกิจซ้ำ จะกระโดดไปที่ *exitWOclose* หรือเป็นการออกจากภารกิจนั้นโดยไม่เป็นการนับว่าทำภารกิจสำเร็จ หากภารกิจดังกล่าวยังไม่มีอยู่ในระบบก็จะสร้างใหม่ เมื่อภารกิจได้ถูกสร้างขึ้นแล้ว จะเริ่มต้นที่ *waitNextCycle* แสดงในรูปที่ 4.3 บรรทัดที่ 10 ทำหน้าที่ในการรอการทำงานในรอบถัดไป ในกรณีที่ภารกิจถูกสร้างระหว่างทำงาน เพื่อที่จะป้องกันการแทรกสลับระหว่างการทำงานของภารกิจอื่นๆ พร๊อกไทป์นี้จะมีลูป do เป็นลูปทำงานหลัก ภารกิจทั้งหมดจะถูกดำเนินการภายใต้ลูปนี้ยกเว้นภารกิจสำเร็จถึงจะสิ้นสุดและปิดพร๊อกไทป์

```
1  proctype CreateTask(taskdef myTask){
2  int cycleCount = 0;
3  int durationCount = 0;
4  bool isTaskActive = false;
5  mtype RES[MaxResource];        /* Resouce available from Global Clock */
6  if ::Tlist[myTask.my_type].isCreate == true
7      -> goto exitWOclose;
8     ::else->Tlist[myTask.my_type].isCreate = true;
9   fi;
10 waitNextCycle:
11    do   :: atomic{((GlobalClockState==C_Open)&&(isMyPhase)
12             &&(!isTaskActive))
13         -> isTaskActive= true;  }
```

รูปที่ 4.3 ส่วนเริ่มต้น และ ส่วนของสถานะ Wait ของแบบจำลองย่อยภารกิจ

2) สถานะของภารกิจที่มีค่า Wait เงื่อนไขแรกที่พร๊อกไทป์นี้ถูกการ์ดไว้คือ รอให้ *GlobalClockState* มีสถานะเป็น *Open* หรือเป็นสถานะที่เปิดให้ร้องขอทรัพยากรภารกิจอื่นๆ หรือ พร๊อกไทป์อื่นๆ ก็จะรอจังหวะสัญญาณนาฬิกานี้เช่นเดียวกันเพื่อที่จะทำงานกันเป็นแบบสอดประสาน กันและป้องกันการแทรกสลับระหว่างภารกิจ เงื่อนไขที่สองที่กำหนดไว้คือตรวจว่าภารกิจมีสถานะเป็น *Active* หรือไม่ หากภารกิจได้เริ่มทำงานแล้ว จะเริ่มทำงานเลยโดยไม่ต้องรอเฟสที่กำหนดไว้แล้วอีก เงื่อนไขที่สุดท้ายในที่ใช้ในการการ์ดคือมาโคร *isMyPhase* ที่ได้แสดงดังนี้

```
#define isMyPhase (((GlobalClock-(myTask.phase-1))%myTask.period==0)
                        ||(myTask.phase==0))
```

โดยมาโครนี้ถูกสร้างมาเพื่อที่จะอำนวนความสะดวกในการเรียกใช้แบบจำลอง โดยมาโครนี้จะทำการตรวจว่าเวลา *GlobalClock* มีค่าเท่ากับที่กำหนดไว้ของเฟสของภารกิจหรือไม่ หรือ มีค่าเท่ากับคาบเวลา ของภารกิจหรือไม่ โดยตรวจกับตัวแปร *myTask.phase* ที่ได้ถูกกำหนดไว้เมื่อสร้างภารกิจ นอกจากนั้น หากกำหนดให้ภารกิจทำงานเลยโดยที่ไม่ต้องรอเฟสที่กำหนดไว้ ให้กำหนดเฟสเท่ากับ -1

3) สถานะของภารกิจที่มีค่า Active เป็นที่สองคือ ภารกิจได้ถูกกระตุ้นและทำงานแล้วโดยเงื่อนไขคือผ่านเฟสที่กำหนดไว้แล้ว แต่อาจจะยังไม่ได้เริ่มทำงาน หรือเริ่มทำงานแล้วแต่ยังไม่ครบเงื่อนไขเวลาก็ได้ ภารกิจจะตรวจว่าสถานะของตนเองคือ Active หรือไม่โดยตรวจที่เงื่อนไข *isTaskActive* และตรวจว่าภารกิจนั้นๆ เคยมีการร้องขอทรัพยากรแล้วหรือไม่ แสดงในบรรทัดที่ 3 เพื่อป้องกันไม่ให้เกิดการร้องขอต่อเนื่อง หากยังไม่เคยร้องขอก็จะทำการร้องขอผ่านตัวแปรที่แสดงในบรรทัดที่ 4 นอกจากนั้น ยังมีการปรับปรุงค่าต่างๆ ของภารกิจ เพื่อที่ใช้ในการคำนวณการจัดกำหนดรายการเวลาในสถานะถัดไป ในบรรทัดที่ 5-8 ของรูปที่ 4.4

```
14      :: atomic{((GlobalClockState == C_Open)
15        &&(isTaskActive)
16        &&(!Tlist[myTask.my_type].is_Requested))
17        -> Tlist[myTask.my_type].is_Requested = true;
18          Tlist[myTask.my_type].myDeadline = myTask.period-
19         ((GlobalClock-(myTask.phase-1))%myTask.period);
20          Tlist[myTask.my_type].myDuration = myTask.duration;
21          Tlist[myTask.my_type].myResponse = myTask.period - myTask.duration;}
```

รูปที่ 4.4 ส่วนของสถานะแอกทีฟ ของแบบจำลองย่อยภารกิจ

4) สถานะของภารกิจที่มีค่า Request ในสถานะนี้ ภารกิจจะเข้าเงื่อนไขได้ต่อเมื่อสถานะนาฬิกาอยู่ในสถานะ จัดกำหนดรายการเวลา เมื่อภารกิจเข้ามาอยู่ในสถานะนี้แล้ว จะทำตามเงื่อนไขของแบบจำลองย่อยการจัดกำหนดรายการเวลาที่กำหนดไว้ล่วงหน้าแล้ว เมื่อภารกิจได้ทำตามคำสั่งที่กำหนดไว้เสร็จแล้วจะกำหนดค่าตัวแปร *is_Schedule* แสดงบนบรรทัดที่ 6 รูปที่ 4.5 เป็นจริงและทำให้ไม่เข้าสู่สถานะนี้อีก หากมีการดำเนินการจัดกำหนดรายการเวลา จะมีการกำหนดค่าตัวแปร *Selected* ให้เป็นภารกิจที่มีคุณสมบัติตามโทโพโลยีของการจัดกำหนดรายการเวลาแต่หากไม่มีการเลือกการจัดการเวลา ค่าตัวแปร *Selected* จะมีค่าเป็น none

```
22     :: atomic{((GlobalClockState == C_Schedule)
23       &&(isTaskActive)
24       &&(!Tlist[myTask.my_type].is_Scheduled))->
25            //Scheduler_DM(myTask.my_type);
26            //Scheduler_EDF(myTask.my_type);
27            Tlist[myTask.my_type].is_Scheduled = true;}
```

รูปที่ 4.5 สถานะร้องขอของแบบจำลองย่อยภารกิจ

5) สถานะของภารกิจที่มีค่า Select ในสถานะนี้ ภารกิจจะเข้าเงื่อนไขเมื่อสถานะนาฬิกาอยู่ในสถานะปิด โดยจะเป็นการตรวจดูว่าภารกิจนั้นๆมีเงื่อนไขคือ ทำงานอยู่หรือไม่จากตัวแปร isTaskActive มีระดับความสำคัญด้อยกว่าภารกิจอื่นๆ อยู่หรือไม่จากตัวแปร isLessPriority บนบรรทัดที่ 29 รูปที่ 4.6 มีภารกิจที่ได้รับเลือกไปแล้วหรือไม่ *Selected* และภารกิจนั้นได้มีการร้องของเอาทรัพยากรหรือไม่ เมื่อตรวจสถานะทั้งหมดแล้ว หากเงื่อนไขเป็นจริงหมายความว่าภารกิจนั้นๆ ได้รับการเลือกให้ถือครองทรัพยากร และกำหนดตัวแปร *Selected* ให้เป็นภารกิจนั้นๆ แต่หากเงื่อนไขไม่เป็นจริง เช่น มีภารกิจอื่นๆ ที่มีระดับความสำคัญสูงกว่าจะทำให้เงื่อนไขไม่เป็นจริง เป็นต้น ตัวแปร *Selected* สามารถที่จะถูกกำหนดค่าจาก แบบจำลองย่อยการจัดกำหนดรายการเวลาในสถานะร้องขอได้ หากไม่มีการเลือกใช้มาโครจัดกำหนดรายการเวลา ระบบจะเลือกภารกิจตามระดับความสำคัญ แต่หากมีการกำหนดโทโพโลยีการจัดกำหนดรายการเวลาในสถานะก่อนหน้าจะทำให้สถานะนี้ไม่ทำงาน

```
28      ::   atomic{((GlobalClockState == C_Close)
29        &&(isTaskActive)
30        &&(!isLessPriority(myTask.my_type))
31        &&(Selected == none)
32        &&(Tlist[myTask.my_type].is_Requested))->
33             Selected = myTask.my_type; }
```

รูปที่ 4.6 สถานะเลือกของแบบจำลองย่อยภารกิจ

6) สถานะของภารกิจที่มีค่า Excute แบบได้รับทรัพยากร เมื่อเงื่อนไขคือสถานะนาฬิกาคือกระทำการ ภารกิจจะตรวจดูว่าภารกิจของตนเองได้รับเลือก

```
34      ::  atomic{((GlobalClockState == C_Execute)/* For the highest priority */
35        &&(isTaskActive) &&(Tlist[myTask.my_type].is_Requested)
36        &&(Selected == myTask.my_type))->
37             Tlist[myTask.my_type].is_Requested = false;
38             Tlist[myTask.my_type].is_Scheduled = false;
39               if ::(Resource[0].owner==_IDLE_ && myTask.isreqRES0)
40                    ->Resource[0].owner=myTask.my_type;
41                  :: else; fi;
42               if ::(Resource[1].owner==_IDLE_ && myTask.isreqRES1)
43                    ->Resource[1].owner=myTask.my_type;
44                  :: else; fi;
45               if ::(Resource[2].owner==_IDLE_ && myTask.isreqRES2)
46                    ->Resource[2].owner=myTask.my_type;
47                  :: else; fi;
48               durationCount++;
49          if   :: (durationCount >= myTask.duration)->
50                    durationCount=0->goto TaskIsFinished;
51               :: else->skip; fi;
52          HighestGetResource = true; GlobalClockState == C_Open;       }
```

รูปที่ 4.7 สถานะของภารกิจที่มีค่า Execute กระทำการแบบได้รับทรัพยากร

ให้ครองทรัพยากรโดยตรวจดูที่ตัวแปร *Selected* แสดงบนบรรทัดที่ 35 รูปที่ 4.7 หากภารกิจนั้นได้รับสิทธิในการถือครองทรัพยากรจะทำการใส่ชื่อของตนเองลงในทรัพยากรที่ตัวเองต้องการ และทำการเพิ่มเวลาดำเนินการของตนเอง เสมือนว่าตนเองได้ทำงานสำเร็จไปแล้ว 1 รอบการทำงาน และ หากจำนวนหน่วยเวลาที่ได้ทำงานแล้วเท่ากับค่าระยะเวลาทำงาน แล้วก็จะถือว่างานสำเร็จแล้ว จะสั่งให้กระโดดไปที่ TaskIsFinished เพื่อสิ้นสุดการทำงานของ ภารกิจนั้นๆ และจะกำหนดให้ ตัวแปร *HighestGetResource* เป็นจริง เพื่อที่จะบอกกับภารกิจอื่นๆ ว่า ภารกิจที่มีระดับความสำคัญสูงสุดได้เข้าครองทรัพยากร

7) สถานะของภารกิจที่มีค่า Execute แบบไม่ได้รับทรัพยากร สถานะภารกิจนี้คล้ายกับสถานะก่อนหน้านี้ แต่แตกต่างตรงที่มีการรอจนกว่าภารกิจที่มีสิทธิในการครองทรัพยากรเข้าไปครอบครองก่อน แล้วภารกิจอื่นๆ จึงดำเนินการเข้าครองกันภายหลัง โดยจะทำการตรวจจาก ตัวแปร *HighestGetResouce* แสดงบนบรรทัดที่ 55 รูปที่ 4.8 หากตัวแปรนี้มีค่าเป็นจริงจึงอนุญาตให้ภารกิจอื่นๆ เข้าครอบครองทรัพยากร

```
53      ::  atomic{((GlobalClockState == C_Execute)   /* For lower priority task */
54        &&(isTaskActive)
55        &&(Tlist[myTask.my_type].is_Requested)
56        &&(HighestGetResource)
57        &&(CheckMyResource))->
58          Tlist[myTask.my_type].is_Requested = false;
59          Tlist[myTask.my_type].is_Scheduled = false;
60          if      ::(Resource[0].owner==_IDLE_ && myTask.isreqRES0)
61                   ->Resource[0].owner=myTask.my_type;
62                  :: else;
63               fi;
64          if      ::(Resource[1].owner==_IDLE_ && myTask.isreqRES1)
65                   ->Resource[1].owner=myTask.my_type;
66                  :: else;
67               fi;
68          if      ::(Resource[2].owner==_IDLE_ && myTask.isreqRES2)
69                    ->Resource[2].owner=myTask.my_type;
70                  :: else;
71               fi;
72               durationCount++;
73          if
74            :: (durationCount >= myTask.duration)->durationCount=0
75                     ->goto TaskIsFinished;
76            :: else->skip;
77          fi;
78          GlobalClockState == C_Open;
79        }
80  od;
```

รูปที่ 4.8 สถานะของภารกิจที่มีค่า Execute แบบไม่ได้รับทรัพยากร

8) สถานะของภารกิจที่มีค่า Destroy เป็นสถานะที่ทำหน้าที่เมื่อภารกิจได้ทำตามระยะเวลากระทำการ (Duration) ของแต่ละภารกิจแล้ว และทำการปิดพร๊อกไทป์ของตนเองเพื่อลดจำนวนหน่วยความจำที่ต้องใช้ ก่อนที่จะปิดตัวเอง ภารกิจจะทำการรีเซตค่าพารามิเตอร์ต่างๆ ของภารกิจเพื่อเตรียมตัวสำหรับการเรียกใช้ในครั้งถัดไป และทำการปรับตัวแปร *isCreate* แสดงบนบรรทัดที่ 92 รูปที่ 4.9 ให้เป็นเท็จเพื่อที่จะสามารถสร้างภารกิจใหม่ได้

```
81 TaskIsFinished:    atomic{isTaskFinished[myTask.my_type] = true;
82                           durationCount = 0;
83                           isTaskActive = false;
84                            if
85                             ::   (myTask.is_periodic)->cycleCount=0;
86                                goto waitNextCycle;
87                             ::   ((cycleCount+1) < myTask.cycle)->
88                            cycleCount++;printf("cycle count %d\n",cycleCount);
89                            goto waitNextCycle;
90                             ::   else->cycleCount=0;
91                           fi;
92                        }
93 Tlist[myTask.my_type].isCreate = false;
94 exitWOclose:  skip;}
```

รูปที่ 4.9 สถานะของภารกิจที่มีค่า Destroy การแบบได้รับทรัพยากร

9) พารามิเตอร์ภารกิจ (Task parameters) พารามิเตอร์ที่สำคัญของภารกิจจะบรรจุอยู่ใน typedef ที่เรียกว่า *tasklist* และ *TaskList* ใช้สำหรับการบันทึกค่ารายละเอียดของภารกิจต่างๆ สำหรับการสร้างภารกิจใหม่ แสดงบนบรรทัดที่ 1-14 รูปที่ 4.10 และผู้ใช้สามารถที่จะเรียกใช้หรืออ้างอิงภารกิจที่กำลังทำงานอยู่ในระบบได้โดยง่ายโดยการเรียกผ่านตัวแปรอาร์เรย์ *Tlist* ที่แสดงในรูปที่ 4.10 บรรทัดที่ 15-24 อาร์เรย์นี้จะทำการบันทึกข้อมูลของภารกิจต่างๆ ที่ทำงานในระบบ ทำให้ภารกิจหรือแบบจำลองต่างๆ สามารถที่จะเรียกใช้ได้ สำหรับการตั้งชื่อภารกิจต่างๆ เพื่อสำหรับการเรียกใช้และอ้างอิง ผู้วิจัยจะสร้าง ชนิดของตัวแปรใหม่เป็นชื่อย่อของภารกิจต่างๆ ทำให้การเรียกชื่อภารกิจในระบบเพื่อการอ้างอิงสามารถเรียกใช้ผ่านอาร์เรย์ Tlist[ชื่อภารกิจ] แสดงที่รูป 4.10 บนบรรทัดที่ 25 เช่น เมื่อต้องการทราบว่า ภารกิจควบคุมการบิน มีระยะเวลาการทำงานเท่าใดก็สามารถเรียกใช้ได้ตามมาโครที่แสดงบนบรรทัดที่ 44 เป็นต้น

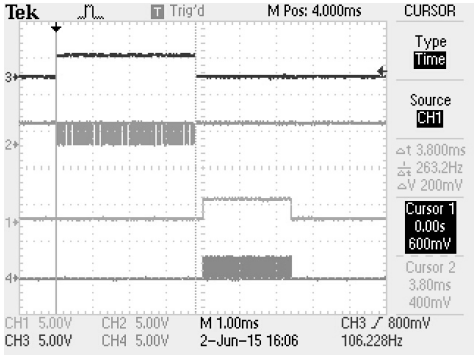
```
typedef taskdef{
mtype my_type;
bool isreqRES0;
bool isreqRES1;
bool isreqRES2;
byte my_priority;
bool is_Active;
bool is_periodic;
int phase;
  int duration;
  int phaseCycle;
  int cycle;
  int period;};
typedef TaskList {
  bool isCreate;
  bool is_Requested;
  bool is_Scheduled;
  byte myPriority;
  byte myDeadline;
  byte myDuration;
  byte myResponse; };
TaskList Tlist[[30;
 inline Create(Taskname,R1, R2, R3, isPer ,dura, pha, pri, cy, per, pCy){
taskdef taskTemplete;
atomic{taskTemplete.my_type = Taskname;
    taskTemplete.isreqRES0 = R1;
    taskTemplete.isreqRES1 = R2;
    taskTemplete.isreqRES2 = R3;
    taskTemplete.duration = dura;
    taskTemplete.is_periodic = isPer;
    taskTemplete.phase = pha+1;          /* to handle phase -1 */
    taskTemplete.cycle = cy;
    taskTemplete.period = per;
    taskTemplete.phaseCycle = pCy;
    taskTemplete.my_priority = pri;
    Tlist[Taskname].myPriority=pri;
} run CreateTask(taskTemplete);}
mtype= {FC,INS,CSC,SDA,DSM,LCom,SCom,SPS,Serv,
        Sen1,Sen2,Sen3,SSD,GPS,DR,ODR,none};
#define FC_Duration Deadline = Tlist[FC].myDeadline;
```

รูปที่ 4.10 พารามิเตอร์ของภารกิจ

4.2.2 การแปลงภารกิจของระบบยูเอวีให้เป็นแบบจำลองพารามิเตอร์ภารกิจ

การแปลงภารกิจจากกรณีตัวอย่างให้เป็นแบบจำลองเริ่มจาก การรวบรวมข้อมูลของภารกิจต่างๆ ให้อยู่ในรูปแบบตาราง ข้อมูลของภารกิจสามารถรวบรวมได้จากคุณลักษณะของแต่ละภารกิจ สำหรับจำนวนติ๊กที่ต้องใช้ในการทำงานในแต่ละภารกิจสามารถใช้วิธีการวัดสัญญาณ โดยการแทรกคำสั่งเครื่องมือ (Instrument Code) ลงในระบบสมองกลฝังตัวเพื่อให้แสดงห้วงเวลาที่ระบบทำงานตามที่แสดงในรูปที่ 4.11 ทดลองโดยการแทรกคำสั่งเครื่องมือสำหรับการวัดเวลาการทำงาน เมื่อ

ภารกิจนำร่องด้วยความเฉื่อยทำงาน สัญญาณที่ 3 แสดงในรูปที่ 4.11 จะมีสถานะเป็น High และสัญญาณที่สองแสดงถึงปริมาณการประมวลผลข้อมูลระหว่างทำงาน ทำให้ทราบระยะเวลาการทำงานของภารกิจดังกล่าวคือประมาณ 4 มิลลิวินาที



รูปที่ 4.11 การวัดเวลากระทำการของภารกิจโดยการใช้เครื่องออสซิลโลสโคป

ผู้วิจัยทำการรวบรวมข้อมูลที่มีความสำคัญในการแปลงภารกิจให้เป็นแบบจำลองดังที่แสดงในตารางที่ 4.1 เพื่อความสมบูรณ์ และ ถูกต้องในการแปลงภารกิจให้เป็นแบบจำลองย่อยภารกิจ

ตารางที่ 4.1 แม่แบบพารามิเตอร์ภารกิจสำหรับแปลงภารกิจให้เป็นแบบจำลองย่อยภารกิจ

ชื่อพารามิเตอร์	**ชนิดตัวแปร**	**คำอธิบาย**	**สูตรการแปลง**	**ค่าที่หามาได้**
my_type	Char	ชื่อภารกิจ	-	INS
isreqRES0	Boolean	ต้องใช้ CPU ในการทำงาน	-	True
isreqRES1	Boolean	ใช้บัส SPI ในการทำงาน	-	False
isreqRES2	Boolean	ใช้บัส I2C ในการทำงาน	-	False
Is_periodic	-	ภารกิจเป็นคาบเวลาหรือไม่	-	False
PckSize	Byte	ขนาดของข้อมูลต่อชุด	-	-
BitRate	Bit/s	ความเร็วในการส่งข้อมูล	-	-
WCET	Msec	ระยะเวลาในการประมวลผล	วัดสัญญาณ	4
TickPerMs	Tick/ms	หน่วยเวลา	-	1
Duration	Tick	จำนวนติ๊กที่ต้องใช้	Round ((8 x PckSize) / (1000 x TickPerMs))	4
SamplingRate	Hz	อัตราการวัดข้อมูล	-	-
Phase		เริ่มต้นการทำงานที่เฟส	-	-1
Cycle		TotalTickPerLoop	1000 / SamplingRate	50
Priority		ระดับความสำคัญ	-	4

เมื่อได้ข้อมูลตามตารางที่ 4.1 แล้ว โดยการนำค่าของตัวแปรในตารางที่ 4.1 มากรอกตามรายชื่อของพารามิเตอร์ตามตารางที่ 4.2 ดังนี้

ตารางที่ 4.2 แม่แบบพารามิเตอร์ภารกิจสำหรับแปลงภารกิจให้เป็นมาโคร

my_type	isreqRES0	isreqRES1	isreqRES1	periodic	Duration	Phase	Priority
INS	True	False	False	False	4	-1	4

นำค่าของพารามิเตอร์ภารกิจมารวบรวมไว้ในตารางที่ 4.2 และทำการสร้างมาโครตามแม่แบบ โดยการเติมค่าของตัวแปรแทนแต่ละตัวแปรบนมาโคร

```
#define CreateTask_INS Create(my_type,isreqRES0,isreqRES1,isreqRES2,
Is_periodic, Duration, Phase, Priority, Cycle,50,1)
```

เมื่อทำการกรอกแล้วจะได้มาโครสำหรับการสร้างภารกิจดังนี้

```
#define CreateTask_ INS Create(INS,true,false,false,false,4,-1,4,50,50,1)
```

4.2.3 การแปลงแผ่นข้อมูลของเซนเซอร์เป็นภารกิจ

สำหรับภารกิจที่ต้องมีการติดต่อกับเซนเซอร์ภายนอก เช่น เซนเซอร์จะใช้การแปลงจากคุณลักษณะจำเพาะ ที่ระบุไว้ นำมากรอกใส่แม่แบบพารามิเตอร์ภารกิจในตารางที่ 4.3

ตารางที่ 4.3 แม่แบบพารามิเตอร์ภารกิจสำหรับแปลงแผ่นข้อมูลเซนเซอร์เป็นภารกิจ

ชื่อพารามิเตอร์	ชนิดตัวแปร	คำอธิบาย	สูตรการแปลง	ค่าที่หามาได้
my_type	char	ชื่อภารกิจ	-	Sen1
isreqRES0	Boolean	ต้องใช้ CPU ในการทำงาน	-	True
isreqRES1	Boolean	ใช้บัส SPI ในการทำงาน	-	True
isreqRES2	Boolean	ใช้บัส I2C ในการทำงาน	-	False
Is_periodic	-	ภารกิจเป็นคาบเวลาหรือไม่	-	True
PckSize	Byte	ขนาดของข้อมูลต่อชุด	-	200
WCET	Msec	ความเร็วในการส่งข้อมูล	Round ((8 x PckSize) / (1000 x TickPerMs))	2
BitRate	Bit/s	ระยะเวลาในการประมวลผล	-	1Mbit
TickPerMs	Tick/ms	หน่วยเวลา	-	1
Duration	Tick	จำนวนติ๊กที่ต้องใช้	WCET / TickPerMs	2
SamplingRate	Hz	อัตราการวัดข้อมูล	-	20Hz
Phase	-	เริ่มต้นการทำงานที่เฟส	-	1
Cycle	Tick/Cycle	TotalTickPerLoop	1000 / SamplingRate	50
Priority	-	ระดับความสำคัญ	-	10

เมื่อได้ข้อมูลตามตารางที่ 4.3 แล้วทำการแปลงโดยเรียงลำดับค่าตามกำหนดในตารางที่ 4.2 และแปลงเป็นมาโครจะได้มาโครสำหรับการสร้างภารกิจดังนี้

```
#define CreateTask_Wear Create(Sen1,true,true,false,true,2,1,10,50,50,1)
```

4.3 แบบจำลองย่อยเครื่องประมวลผลเอมทีแอล (MTL engine submodel)

สูตรแบบเอมทีแอลเป็นสูตรที่ใช้สำหรับการบรรยายการทำงานของระบบเวลาจริง สามารถใช้เป็นแบบอย่างคุณลักษณะ สำหรับการทวนสอบความถูกต้องของแบบจำลองได้ การนำมาใช้ในการตรวจสอบแบบจำลองจำเป็นที่จะต้องทำการแปลเอมทีแอลให้เป็นสูตรแบบแอลทีแอลก่อน เพื่อที่จะสามารถใช้ความสามารถในการทวนสอบแบบจำลองของสปินได้ ส่วนประกอบที่สำคัญของเครื่องประมวลผลเอมทีแอลมีดังนี้

1) มาโคร MTLEngine

ถูกเรียกใช้ทุกครั้งเมื่อพร๊อกไทป์ Ticking เข้าสู่สถานะ GlobalClockState เท่ากับ Execute ทำหน้าที่ในการอัพเดทและส่งต่อสถานะของเหตุการณ์ต่างๆ ให้กับมาโคร MTLCounter

2) มาโคร MTLCounter

ถูกเรียกใช้ในมาโคร MTLEngine ทำหน้าที่ในการนับเวลาตัวแปรของเอมทีแอล และตรวจว่าตัวแปรแต่ละตัวมีสถานะเป็น Expired หรือไม่

3) ตัวแปรชนิด MTLEvent

ทำหน้าที่ในการบรรจุพารามิเตอร์ของเอมทีแอลแต่ละตัว มีการจัดเก็บเหตุการณ์ที่เป็นเหตุการณ์ริเริ่ม เหตุการณ์ที่เป็นการตอบสนอง ขอบเขตเวลา เป็นต้น

4) แม่แบบการแปลเอมทีแอลเป็นแอลทีแอล

เป็นชุดมาโครเพื่อที่ใช้แปลสูตรเอมทีแอลให้เป็นสูตรแอลทีแอลและใช้ร่วมกันในแบบจำลองก่อน การใช้งานสูตรเอมทีแอลทุกๆครั้งจะมีการถ่ายทอดค่าต่างๆ ไปยังตัวแปรที่อยู่ในสูตรเพื่อที่ใช้การคำนวณเปรียบเทียบได้โดยง่าย โดยมีพารามิเตอร์ที่ต้องเพิ่มดังรูปที่ 4.12 เพื่อให้รองรับการใช้สูตรเอมทีแอลจะทำการเพิ่มมาโคร ดังกล่าวนี้ลงในส่วนของ แบบจำลองย่อยสัญญาณนาฬิกา โดยการเพิ่มมาโคร *MTLEngine* แสดงในรูปที่ 4.12 บรรทัดที่ 15 ดังกล่าวให้เพิ่มลงบนตำแหน่งดังกล่าวเนื่องจากในขณะที่กำลังดำเนินการคำสั่ง timeout นั้นจะรอจนกว่ากระบวนการอื่นๆ ในระบบดำเนินการเสร็จสิ้นแล้วตัวแปรที่ใช้ถ่ายทอดไปยังแบบจำลองย่อยเครื่องประมวลผลเอมทีแอลมีดังนี้

1) ตัวแปร Name ชื่อ กำหนดชื่อเพื่อที่จะใช้ในการตรวจสอบและอ้างอิง
2) ตัวแปร Event เป็นเหตุการณ์ลั่นไกให้ ตัวดำเนินการทำงาน
3) ตัวแปร Reaction เป็นเหตุการณ์ที่เป็นเหตุการณ์ตอบสนอง

4) ตัวแปร PTime1 ช่วงเปิด หรือ ช่วงปิด สำหรับเวลาเริ่มต้น

5) ตัวแปร PTime2 ช่วงเปิด หรือ ช่วงปิด สำหรับเวลาสิ้นสุด

```
1  MTLevent Event0;
2  inline MTLEngine(){
3     Selected_MTL = Selected; // do not change
4     // Set up for Event 0
5     Event0.name = Event_p
6     Event0.Event = pEvent;
7     Event0.Reaction = qEvent;
8     Event0.pTime1 = 0;
9     Event0.pTime2 = 10;
10    MTLCounter(Event0);
11 }
```

รูปที่ 4.12 แบบจำลองย่อยเครื่องประมวลผลเอมทีแอล

5) แบบจำลองย่อยเอมทีแอลเคาเตอร์ (MTL Counter submodel)

แบบจำลองเอมทีแอลเคาเตอร์ แสดงในรูปที่ 4.13 เป็นแบบจำลองที่เขียนด้วยภาษาโปรเมลา ใช้ในการนับเวลาให้กับสูตรเอมทีแอลแต่ละตัว โดยที่แบบจำลองนี้ถูกเขียนในลักษณะของมาโคร ทำหน้าที่ในการนับปริมาณเวลาโดยอัตโนมัติเมื่อเวลาดำเนินไป มาโครนี้จะทำงานอัตโนมัติทุกครั้งที่มีการนับเวลา 1 ติ๊ก

```
1  inline MTLCounter(CauseEvent){
2  if
3   ::atomic{(CauseEvent.Event) -> printf("TriggerEvent Occured\n");
4  if
5  !: CauseEvent.EventFlag->CauseEvent.EventCounter = 0;
6     CauseEvent.EventFlag = true; // set P
7  :: else->skip;
8  fi; }
9   ::atomic{(!CauseEvent.Event&&CauseEvent.EventFlag)-> if
10 ))::CauseEvent.EventCounter<=CauseEvent.pTime (2
11   &&(CauseEvent.EventCounter!=expired))-> CauseEvent.EventCounter++;
12 if
13  :: CauseEvent.Reaction->printf("%e Reaction at count = %d \n"
14     ,CauseEvent.name,CauseEvent.EventCounter);
15 ! :: CauseEvent.Reaction->printf("%e Count = %d \n"
16     ,CauseEvent.name,CauseEvent.EventCounter);
17 fi;
18 ::else->printf("%e Counter = expired > %d \n"
19   ,CauseEvent.name,CauseEvent.EventCounter);
20   CauseEvent.EventCounter=expired;
21   CauseEvent.EventFlag = false;
22   fi; }
23 :: else->skip;
24  fi;}
```

รูปที่ 4.13 แบบจำลองย่อยเครื่องประมวลผลเอมทีแอล

จากการสร้างสูตรแบบอย่างคุณลักษณะเอมทีแอลแล้วสามารถแปลเป็นสูตรแอลทีแอลโดยใช้ตามแม่แบบแอลทีแอลที่แสดงไว้ที่รูป 4.14 และใช้งานร่วมกับแบบจำลองย่อยเครื่องประมวลผลเอมทีแอล

$\Diamond_{(t1,t2)}p$

```
#define MTL_Fpp(effect) <>(effect.Reaction&&(effect.EventCounter>
  effect.pTime1)&&(effect.EventCounter<effect.pTime2))
```

$\Diamond_{[t1,t2]}p$

```
#define MTL_Fss(effect) <>(effect.Reaction&&(effect.EventCounter>=
  effect.pTime1)&&(effect.EventCounter<=effect.pTime2))
```

$\Box_{(t1,t2)}p$

```
#define MTL_Gpp(effect) [](!((effect.EventCounter>effect.pTime1)&&
  (effect.EventCounter<effect.pTime2))||((effect.Reaction
&&(effect.EventCounter>effect.pTime1))&&(effect.EventCounter<effect.pTime2)))
```

$\Box_{[t1,t2]}p$

```
#define MTL_Gss(effect) [](!((effect.EventCounter>=effect.pTime1)&&
(effect.EventCounter<=effect.pTime2))||((effect.Reaction&&
(effect.EventCounter>=effect.pTime1))&&(effect.EventCounter<=effect.pTime2)))
```

รูปที่ 4.14 แม่แบบการแปลเอมทีแอลเป็นแอลทีแอล

4.3.1 ตัวอย่างการแปลเอมทีแอลโดยใช้แม่แบบการแปลเอมทีแอลเป็นแอลทีแอล

จากตัวอย่างการแปลเงื่อนไขด้านเวลาของระบบเวลาจริงให้เป็นเอมทีแอลในบทที่ 3 หัวข้อ 3.9 จะได้แบบอย่างคุณลักษณะเอมทีแอลดังนี้

$\Box(\text{Sen1Event} \rightarrow \Box_{[25]}\text{INSEvent})$

นำเอาตัวแปรดังกล่าวมาเติมลงในแบบจำลอง MTLEngine จากรูป 4.2 โดยมีตัวแปรดังนี้

1) ตัวแปร Name ตั้งชื่อเหตุการณ์นี้คือ Sen1toINS
2) ตัวแปร Event เป็นเหตุการณ์ลั่นไกให้คือ Sen1Event
3) ตัวแปร Reaction เป็นเหตุการณ์ที่เป็นเหตุการณ์ตอบสนองคือ INSEvent
4) ตัวแปร PTime1 ช่วงเปิด หรือ ช่วงปิด สำหรับเวลาเริ่มต้นเท่ากับ 28
5) ตัวแปร PTime2 ช่วงเปิด หรือ ช่วงปิด สำหรับเวลาสิ้นสุดเท่ากับ 28

นำค่าของตัวแปรดังกล่าวมาใส่ลงในแม่แบบการแปลเอมทีแอลเป็นแอลทีแอล

```
1   MTLevent Event0;
2   inline MTLEngine(){
3     Selected_MTL = Selected; // do not change
4     // Set up for Event 0
5     Event0.name = Sen1toINS
6     Event0.Event = Sen1Event;
7     Event0.Reaction = INSEvent;
8     Event0.pTime1 = 28;
9     Event0.pTime2 = 28;
10    MTLCounter(Event0);}
```

รูปที่ 4.15 ตัวอย่างการแปลเอมทีแอลโดยใช้แม่แบบการแปลเอมทีแอลเป็นแอลทีแอล

เลือกแม่แบบการแปลเอมทีแอลเป็นแอลทีแอลจากรูปที่ 4.14 จากสูตรเอมทีแอลที่ให้ประกอบด้วยตัวดำเนินการเอมทีแอลแบบ $\Box_{[t1,t2]}$ เป็นแบบช่วงปิด จึงทำการแทนค่าลงในแม่แบบได้ดังรูปที่ 4.16 โดยทำการเปลี่ยนเหตุการณ์ลั่นไกจาก Sen1Event ให้เป็นตัวแปร Event0.Event และเปลี่ยนตัวดำเนินการเอมทีแอลให้เป็นมาโครและให้มาโครเรียกใช้ตัวแปร Event0 จากรูปที่ 4.15

```
ltl ltl001 {[](Event0.Event -> MTL_Gss(Event0))}
```

รูปที่ 4.16 สูตรแอลทีแอลที่ได้จากการแปลโดยใช้แม่แบบการแปลเอมทีแอลเป็นแอลทีแอล

นำเอามาโคร MTLEngine ที่แสดงในรูป 4.15 และ สูตรแอลทีแอลในรูปที่ 4.16 ใส่แบบจำลองก็จะสามารถทวนสอบแบบจำลองด้วยเอมทีแอลที่กำหนดได้

4.4 แบบจำลองย่อยภารกิจไม่อิสระ (Dependent task submodel)

เป็นแบบจำลองภาษาโปรเมลาสำหรับสร้างภารกิจไม่อิสระ ผู้วิจัยทำการสร้างพร็อกไทป์แยกออกมาจากพร็อกไทป์อื่นๆ เพื่อที่ใช้ในการจัดการภารกิจไม่อิสระโดยเฉพาะ ภารกิจใดๆ สามารถที่จะสร้างภารกิจอื่นๆ สร้างขึ้นมาใหม่ได้อย่างต่อเนื่อง ยกตัวอย่างการทำงาน เช่น เมื่อวัดสัญญาณจากเซนเซอร์ ก็จำเป็นที่จะต้องรวบรวมไว้จำนวนหนึ่งก่อนที่จะทำการแปลงหรือจัดการข้อมูลดังกล่าวและทำการจดบันทึกลงในหน่วยความจำสถิต เป็นต้น ส่วนประกอบที่สำคัญของแบบจำลองย่อยภารกิจไม่อิสระมีดังนี้

1) ตัวแปรอาร์เรย์ isTaskFinished

เป็นตัวแปรทำหน้าที่ในการชี้ว่าแต่ละภารกิจได้เสร็จสมบูรณ์แล้วหรือไม่ยกตัวอย่าง เช่น เมื่อภารกิจที่ 1 ทำงานเสร็จแล้วตัวแปร *isTaskFinished* จะมีสถานะเป็นจริง เป็นต้น

2) พร็อกไทป์ DependentTask

ผู้วิจัยได้สร้างพร็อกไทป์ขึ้นมาชื่อ *DependentTask* ทำหน้าที่คือใช้ในการควบคุมภารกิจไม่อิสระของระบบ แล้วทำหน้าที่ในการตรวจสอบตัวแปร *isTaskFinish* ในแต่ละภารกิจว่าพร้อมเสร็จสมบูรณ์แล้วหรือไม่ แบบจำลองย่อยภารกิจไม่อิสระ แสดงในรูปที่ 4.17 ผู้วิจัยทำการแยกเอา

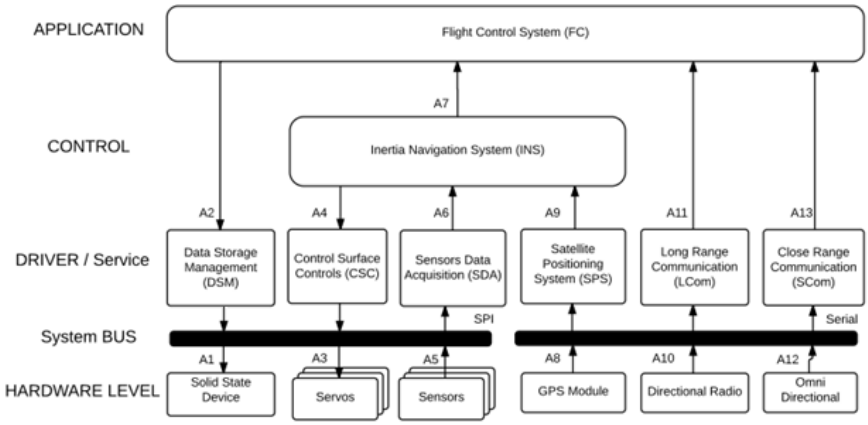
แบบจำลองย่อยภารกิจไม่อิสระ (*DependentTask)* ออกมาเป็นพร๊อกไทป์อิสระ ทำให้ผู้ที่นำแบบจำลองไปใช้สามารถที่จะปรับเปลี่ยนการทำงานของระบบได้โดยง่าย จากแบบจำลองดังกล่าว พร๊อกไทป์นี้เป็นพร๊อกไทป์ที่จะทำงานอัตโนมัติ แสดงในบรรทัดที่ 1 รูปที่ 4.17 เมื่อทำงานพร๊อกไทป์นี้จะรอจนกว่าเงื่อนไขที่อยู่ในลูป do จะทำงาน โดยแต่ละหัวข้อจะทำหน้าที่แทนแต่ละเส้นทางของภารกิจไม่อิสระ ในรูปที่ 4.17 บรรทัดที่ 3 ถึง 5 แสดงแม่แบบของภารกิจไม่อิสระ 1 เส้นทาง ในประกอบด้วยภารกิจตัวนำหน้า (Predecessor Task) และ ภารกิจไม่อิสระ (Dependent Task) เมื่อภารกิจริเริ่มสำเร็จจะมีการเปลี่ยนสถานะในอาร์เรย์ตัวแปรที่เรียกว่า *isTaskFinished* ทำให้เงื่อนของเส้นทางนั้นๆ เป็นจริง และทำการรีเซตค่า *isTaskFinished* ในบรรทัดที่ 4 เพื่อที่จะรอรับภารกิจต่อไป และทำการเรียกมาโครเพื่อสร้างภารกิจใหม่ในบรรทัดที่ 5

```
1  active proctype DependentTask () {
2   do
3     :: ::atomic{isTaskFinished[Predecessor Task] ->  // Dependent Task A1
4        isTaskFinished[Predecessor Task]=false;
5        CreateTask_Dependent Task;
6     :: <<Dependent Task A2>>
7     :: <<Dependent Task A3>>
8  Od;}
```

รูปที่ 4.17 แม่แบบแบบจำลองย่อยภารกิจไม่อิสระ

4.4.1 การแปลงภารกิจไม่อิสระเป็นรูปนัย (Formalize dependent task)

จากกรณีตัวอย่างทำการแปลงภารกิจโดยการนำเอาความสัมพันธ์ระหว่างภารกิจต่างๆ มาทำการตั้งชื่อแต่ละความสัมพันธ์เพื่อความครบถ้วนในการเรียกใช้ แสดงในรูปที่ 4.18



รูปที่ 4.18 แผนภาพความสัมพันธ์ของภารกิจในระบบ

จากทิศทางของลูกศรจะเริ่มจากภารกิจที่เป็นภารกิจตัวนำหน้า ชี้ไปยังภารกิจที่ไม่อิสระ รายชื่อของความสัมพันธ์ของภารกิจต่างๆ จะเป็นดังตาราง ในช่องแรกจะเป็นชื่อเส้นทาง ตามด้วยภารกิจที่เป็นผู้ภารกิจตัวนำหน้า และ ภารกิจไม่อิสระ และจำนวนครั้งก่อนเริ่มภารกิจแสดงในตาราง 4.4

ตารางที่ 4.4 รายชื่อความสัมพันธ์ระหว่างภารกิจของระบบยูเอวี

เส้นทาง	Predecessor Task	Dependent Task	Count
A1	DSM	SSD	1
A2	FC	DSM	1
A3	CSC	Servo	1
A4	INS	CSC	1
A5	Sensors	SDA	1
A6	SDA	INS	1
A7	INS	FC	1
A8	GPS	SPS	5
A9	SPS	INS	1
A10	DR	Lcom	10
A11	Lcom	INS	1
A12	OD	Scom	10
A13	Scom	FC	1

เมื่อได้รายชื่อและความสัมพันธ์ระหว่างภารกิจแล้ว ต่อไปนี้ผู้วิจัยขอแสดงตัวอย่างการแปลงเพียงเฉพาะเส้นทาง A6 ทำการแปลงโดยใช้แม่แบบการแปลง จากข้อมูลแสดงความสัมพันธ์ของภารกิจเส้นทาง A6 ระหว่างภารกิจ รับสัญญาณเซนเซอร์ และภารกิจนำร่องด้วยความเฉื่อย

แม่แบบการแปลงกำหนดให้ใส่ชื่อภารกิจที่เป็นภารกิจตัวนำหน้าลงในแม่แบบตามรูปที่ 4.15 จะสามารถใส่ข้อมูลได้ดังรูปที่ 4.19

```
::atomic{isTaskFinished[SDA] ->
isTaskFinished[SDA]=false;
CreateTask_INS;     }          //A6
```

รูปที่ 4.19 ผลการแทนค่าเส้นทาง A6 ลงในแม่แบบแบบจำลองย่อยภารกิจไม่อิสระ

สำหรับภารกิจที่มีการนับจำนวนครั้งที่ภารกิจสำเร็จก่อนที่จะเริ่มภารกิจไม่อิสระก็สามารถทำได้โดยการเติมชื่อภารกิจลงในแม่แบบเช่นเดียวกัน ต่อไปนี้ผู้วิจัยขอแสดงตัวอย่างการแปลงเพียงเฉพาะเส้นทาง A8 จากตารางที่ 4.4ส่วนที่มีการเติมเพิ่มคือการสร้างตัวแปรสำหรับนับจำนวนภารกิจ และใส่ชื่อตัวแปรนั้นลงในแม่แบบของภารกิจไม่อิสระแสดงในรูปที่ 4.20

```
::atomic{isTaskFinished[Predecessor Task] -> PredecessorTaskCount++;
isTaskFinished[Predecessor Task]=false;
if   :: PredecessorTaskCount >Count->
        PredecessorTaskCount = 0;
        CreateTask_Dependent Task;
     ::else->skip;
fi;     }
```

รูปที่ 4.20 แม่แบบของภารกิจไม่อิสระที่มีตัวแปรควบคุม

เมื่อทำการเติมข้อมูลภารกิจแล้วจะได้ข้อมูลดังรูปที่ 4.21 ให้ทำจนครบจำนวนของเส้นทางที่เป็นไปได้ทั้งหมดและใส่ไว้ในแบบจำลองย่อยภารกิจไม่อิสระ รายการภารกิจไม่อิสระอื่นๆ ผู้วิจัยทำการรวบรวมไว้ที่ภาคผนวก

```
1     ::atomic{isTaskFinished[GPS] -> GPSdataCount++;
2       isTaskFinished[GPS]=false;
3       If   ::GPSdataCount>5->
4              GPSdataCount = 0;
5              CreateTask_SPS;              //A8
6            ::else->skip;
7       fi;     }
```

รูปที่ 4.21 เส้นทางของภารกิจไม่อิสระที่มีตัวแปรควบคุมที่เติมตัวแปรแล้ว

4.5 แบบจำลองย่อยการจัดกำหนดรายการเวลา (Scheduler submodel)

เป็นแบบจำลองภาษาโปรเมลาสำหรับจำลองการจัดกำหนดรายการเวลาบนระบบเวลาจริง เป็นชุดแบบจำลองที่ทำหน้าที่ตัดสินในกรณีที่มีภารกิจมากกว่าหนึ่งภารกิจ ต้องการทรัพยากรเดียวกันในช่วงเวลาเดียวกัน การตัดสินจะเลือกตามโทโพโลยีการจัดกำหนดรายการเวลาที่ผู้ใช้เลือกไว้ ส่วนประกอบของแบบจำลองย่อยการจัดกำหนดรายการเวลา

1) ตัวแปร Selected

ตัวแปร *Selected* เป็นตัวแปรที่สุดท้ายที่ระบบใช้ในการเลือกภารกิจใดๆ เข้ามาทำงานด้วย โดยเมื่อการจัดกำหนดรายการเวลาสำเร็จแล้ว แบบจำลองย่อยการจัดกำหนดรายการเวลาจะกำหนดค่าภารกิจให้กับตัวแปร *Selected* เพื่อที่ในขั้นตอนในสถานะภารกิจมีค่าเท่ากับ Execute ของแบบจำลองย่อยภารกิจ

2) ส่วนการกำหนดโทโพโลยีการจัดเวลา

เป็นส่วนที่แทรกอยู่ในแบบจำลองย่อยภารกิจ อยู่ในส่วนของสถานะของภารกิจที่มีค่า Request แสดงในรูป 4.22 ผู้นำแบบจำลองไปใช้สามารถเลือกโทโพโลยีกำจัดกำหนดรายการเวลาโดยการเลือกเอามาโครของการจัดกำหนดรายการเวลาที่ต้องการ ตามรูปที่ 4.22 หากไม่เลือกโทโพโลยีใดเลยจะใช้การจัดเวลาตามลำดับความสำคัญมากสุด

```
1       :: atomic{((GlobalClockState == C_Schedule)
2         &&(isTaskActive)
3         &&(!Tlist[myTask.my_type].is_Scheduled))->
4              //Scheduler_DM(myTask.my_type);
5              //Scheduler_EDF(myTask.my_type);
6              Tlist[myTask.my_type].is_Scheduled = true;
7         }
```

รูปที่ 4.22 การปรับโทโพโลยีการจัดกำหนดรายการเวลา

เป็นมาโครที่ใช้ในการตรวจสิทธิในการเข้าครอบครองทรัพยากร เมื่อเรียกใช้มาโครจะทำการเรียกมาโครเปรียบเทียบ เช่น จากในรูป 4.23 เป็นการเรียกใช้การจัดกำหนดรายการเวลาตามเวลาดำเนินการเป็นหลัก เป็นต้น มาโครนี้จะเรียกใช้มาโครอีกตัวหนึ่ง แสดงบรรทัดที่ 4 ในรูปที่ 4.23 ที่จะแสดงค่าความจริงเป็นจริงหากระยะเวลาดำเนินการของภารกิจมีค่าน้อยที่สุด เรียกว่า ตัวแปร iMoreDuration และ แสดงค่าเป็นเท็จหากมีภารกิจอื่นที่มีระยะเวลาดำเนินการน้อยกว่าตนเอง

```
1  inline Scheduler_DM(TaskType){
   2  atomic{
   3    if
   4    ::(!iMoreDuration(TaskType))-> Selected = myTask.my_type;
   5    ::else->skip;
   6    fi;}}
```

รูปที่ 4.23 มาโครโทโพโลยีระยะเวลาดำเนินการน้อยสุดทำก่อน

3) ส่วนของตรรกะโทโพโลยี

ตรรกะของโทโพโลยี (Topology Logic) เป็นชุดคำสั่งมาโครทำหน้าที่ในการเปรียบเทียบระหว่างภารกิจเพื่อที่จะหาว่าภารกิจใดมีค่ามากหรือน้อยที่สุด เช่น ใช้ในการเปรียบเทียบระดับความสำคัญ เป็นต้น ให้ค่าจริงหากระดับความสำคัญของภารกิจที่เรียกใช้มีค่ามากที่สุด และให้ค่าเป็นเท็จหากมีภารกิจอื่นมีระดับความสำคัญมากกว่า เช่น มาโคร iMoreDeadline isLessPriority iMoreDuration เป็นต้น ในแบบจำลองนี้ผู้วิจัยได้กำหนดตัวเลขของลำดับความสำคัญให้เป็นจำนวนเต็มบวก ตั้งแต่ 1 ถึง 255 แต่ละภารกิจจะเปรียบเทียบอันดับความสำคัญของตนเองเทียบกับภารกิจอื่นๆ ตามรูป ที่ 4.24

```
1  #define isLessPriority(t) \
2  ((( Tlist[(t)].myPriority<Tlist[0].myPriority)&&(Tlist[0].is_Requested))\
3  ||((Tlist[(t)].myPriority<Tlist[1].myPriority)&&(Tlist[1].is_Requested))\
4  ||((Tlist[(t)].myPriority<Tlist[2].myPriority)&&(Tlist[2].is_Requested))\
```

รูปที่ 4.24 ส่วนของตรรกะโทโพโลยีความสำคัญมากสุดทำก่อน

ชุดมาโครนี้จะตรวจเฉพาะงานที่มีสถานะแอกทีฟและได้ลงทะเบียน ในการเข้าถึงทรัพยากรแล้วเท่านั้น โดยการใช้ชุดมาโคร isLessPriority โดยผลของตรรกะทีเกิดจากมาโครดังกล่าวเป็นจริง หมายถึงว่าภารกิจนั้นๆ มีความสำคัญด้อยกว่าภารกิจอื่นๆ อย่างน้อยหนึ่งภารกิจ แต่ผลของมาโครดังกล่าวเป็นเท็จแสดงว่า ไม่มีภารกิจใดเลยที่มีความสำคัญสูงกว่าภารกิจ จากรูปที่ 4.25 แสดงมาโครเปรียบเทียบสำหรับโทโพโลยีการจัดกำหนดรายการเวลาที่เรียงตามระยะเวลาดำเนินการ

```
1 #define iMoreDuration(t)
2 ((( Tlist[(t)].myDuration>Tlist[0].myDuration) &&(Tlist[0].is_Requested))\
3 ||((Tlist[(t)].myDuration>Tlist[1].myDuration) &&(Tlist[1].is_Requested))\
4 ||((Tlist[(t)].myDuration>Tlist[2].myDuration) &&(Tlist[2].is_Requested))\
```

รูปที่ 4.25 ส่วนของตรรกะโทโพโลยีระยะเวลาดำเนินการน้อยสุดทำก่อน

บทที่ 5

การทวนสอบแบบจำลอง

5.1 การทวนสอบแบบจำลองย่อยสัญญาณนาฬิกา

การทวนสอบแบบจำลองย่อยสัญญาณนาฬิกา (Ticking Submodel) โดยการทวนสอบเทียบกับคุณสมบัติของระบบนับเวลาในระบบเวลาจริง

1) แบบจำลองย่อยสัญญาณนาฬิกาสามารถในทำงานต่อเนื่องไม่มีวันสิ้นสุด

ผลลัพธ์ที่คาดว่าจะได้ ตรวจพบรอบการทำงานไม่ก้าวหน้า (Non-progress) ซึ่งเป็นวงรอบการทำงานที่ไม่มีวันจบ เมื่อแสดงตัวอย่างค้านพบรอบการทำงานที่ไม่ก้าวหน้าวนกลับมาที่เดิมซ้ำ

กรรมวิธีการทวนสอบ ทำการทวนสอบแยกส่วนเฉพาะส่วนของแบบจำลองย่อยสัญญาณนาฬิกา ปิดการทำงานของมาโคร *MTLEngine()* ในพร็อกไทป์ Ticking

```
1   ::atomic{((GlobalClockState == C_Execute) && timeout)->
2     //MTLEngine();  //Disabled
3     Selected = none;
```

รูปที่ 5.1 การปิดมาโคร MTLEngine เพื่อการทวนสอบ

ใช้สปินทำการทวนสอบแบบจำลอง โดยเลือกวิธีการคือ ตรวจหา non-progress cycle พบว่าตรวจพบรอบการทำงานที่ไม่มีวันจบภายในแบบจำลอง (จากรูปที่ 5.2 บรรทัดที่ 5)

```
1  pan: :1non-progress cycle (at depth (14
2  pan: wrote ThesisMini.01pml.trail
3  ...
4  ...
```

รูปที่ 5.2 ผลของการทวนสอบด้วยโปรแกรมสปิน

เมื่อทำเรียกดูตัวอย่างค้าน (Counter Example) ของผลการทดลองก่อนหน้าจะได้ผลดังรูปที่ 5.3 (จากรูปจะตัดเอาติ๊กที่ 4 – 47 ออก) จากบรรทัดที่ 2 ตัวอย่างค้านแสดงการพบรอบการทำงานซ้ำเริ่มต้นจากติ๊กที่ 1 จนถึง 49 และทำการรีเซต

```
1  == TICK :1 CPU[0] SPI[0] I2C[0]
2  <<<<<START OF CYCLE>>>>>
3  == TICK :2 CPU[_IDLE_] SPI[_IDLE_] I2C[_IDLE_]
4  ...
5  == TICK :50 CPU[_IDLE_] SPI[_IDLE_] I2C[_IDLE_]
```

รูปที่ 5.3 ตัวอย่างค้านจากผลการทวนสอบ

สรุปผล แบบจำลองสามารถจำลองสัญญาณนาฬิกาได้อย่างถูกต้อง โดยมีรอบการทำงานที่ไม่สิ้นสุดและมีการวนกลับมาที่เดิม

2) แบบจำลองทำงานแบบออโตมาตาเวลา

ผลลัพธ์ที่คาดว่าจะได้ เมื่อทำการจำลองการทำงานของระบบ พบว่าเลขแสดงลำดับติ๊ก มีการรีเซตเป็น 1 ใหม่ทุก 50 ครั้ง

กรรมวิธีการทวนสอบ ทำการจำลองการทำงานของ แสดงผลการทำงานได้รูปที่ 5.4 จากผลการทำงานมีการตัดเอา ติ๊กที่ 3 – 48 ออก การจำลองการทำงานแสดงการวนซ้ำไปเรื่อยๆ ไม่สิ้นสุด เริ่มจากติ๊กที่ 1 ถึง ติ๊กที่ 50 และทำการรีเซต

```
1  == TICK :1 CPU[_IDLE_] SPI[_IDLE_] I2C[_IDLE_]
2  == TICK :2 CPU[_IDLE_] SPI[_IDLE_] I2C[_IDLE_]
3  ...
4  == TICK :49 CPU[_IDLE_] SPI[_IDLE_] I2C[_IDLE_]
5  == TICK :50 CPU[_IDLE_] SPI[_IDLE_] I2C[_IDLE_]
6  ------------------------------------------------------------
7  == TICK :1 CPU[_IDLE_] SPI[_IDLE_] I2C[_IDLE_]
8  == TICK :2 CPU[_IDLE_] SPI[_IDLE_] I2C[_IDLE_]
```

รูปที่ 5.4 ผลจากการจำลองการทำงานแบบจำลอง

สรุปผล แบบจำลองทำงานแบบ Timed-Automata ได้ถูกต้อง

3) แบบจำลองมีจำนวนสถานะจำกัด

ผลลัพธ์ที่คาดว่าจะได้ ทำการทวนสอบแบบจำลอง โดยเลือกให้เป็นการทวนสอบ แบบ Safety เพื่อที่จะตรวจหาขนาดของแบบจำลอง

กรรมวิธีการทวนสอบ ทำการจำลองการทำงานของแบบจำลอง โดยใช้แบบจำลองย่อยสัญญาณนาฬิกา โดยการทดสอบแบบ Safety เพื่อหาจำนวนสถานะของแบบจำลอง ผลของการทวนสอบได้ดังรูป

```
1  (Spin Version 6.4.3 -- 16 December 2014)
2   + Partial Order Reduction
3  Full statespace search for:
4   never claim          - (none specified)
5   assertion violations +
6   cycle checks         - (disabled by -DSAFETY)
7   invalid end states +
8  State-vector 332 byte, depth reached 305, ••• errors: 0 •••
```

รูปที่ 5.5 ผลจากการทวนสอบแบบ Safety

สรุปผล พบว่าแบบแบบจำลองย่อยสัญญาณนาฬิกา หากทำงานเพียงแบบจำลองเดียว จะมีสถานะจำกัดเท่ากับ 205 สถานะ และมีจำนวนสถานะจำกัด

5.2 การทวนสอบแบบจำลองย่อยภารกิจและกรณีตัวอย่าง

1) แบบจำลองสามารถที่จะจองทรัพยากรได้

แบบจำลองสามารถที่จะจองทรัพยากร และนับเวลาในการทำงานเมื่อถือครองภารกิจ ภารกิจชุดเซนเซอร์จำนวนสามภารกิจมีการทำงานดังตารางที่ 5.1

ตารางที่ 5.1 รายชื่อของภารกิจสำหรับการทวนสอบและพารามิเตอร์ที่เกี่ยวข้อง

Name	CPU	SPI	Serial	Periodic	Duration	Phase	Priority	Period
Sen1	False	True	False	True	2	1	10	50
Sen2	False	True	False	True	1	3	10	50
Sen3	False	True	False	True	2	4	10	50

ผลลัพธ์ที่คาดว่าจะได้ ภารกิจทั้งสาม สามารถครอบครองทรัพยากรภายในห้วงเวลาที่กำหนด และทำการทวนสอบแบบจำลองเทียบกับ แบบอย่างคุณลักษณะที่บรรยายด้วยภาษาแอลทีแอล

กรรมวิธีการทวนสอบ

ทำการแปลงภารกิจทั้งสามเป็นคำสั่งมาโครสำหรับการสร้างภารกิจและสั่งให้ภารกิจทั้งสามทำงานเมื่อเริ่มการทำงานของแบบจำลองดังรูปที่ 5.6

```
1  #define CreateTask_Sen1Create(Sen1,false,true,false,true, 2,1,10,1,50,1)
2  #define CreateTask_Sen2Create(Sen2,false,true,false,true,1,3,10,1,50,1)
3  #define CreateTask_Sen3Create(Sen3,false,true,false,true,1,5,10,1,50,1)
```

รูปที่ 5.2 มาโครภารกิจที่ต้องการทวนสอบ

ทำการเพิ่มชุดคำสั่งดังกล่าวลงในแบบจำลองระบบและเพิ่มชุดคำสั่ง init ตามรูปที่ 5.3

```
1  init { atomic{   CreateTask_Sen1; CreateTask_Sen2; CreateTask_Sen3;   }
2  run Ticking();}
```

รูปที่ 5.3 ชุดคำสั่งเริ่มต้นของกรณีทวนสอบ

ทำการจำลองการทำงาน ได้ผลออกมาดังรูปที่ 5.4 (ในรูปไม่แสดงผลของ “timeout”) เมื่อสั่งให้ทำการจำลองการทำงานจะพบว่า ภารกิจทั้งสามสามารถที่จะจองทรัพยากรในห้วงเวลาที่กำหนดไว้ได้

```
1  == TICK :1 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
2  == TICK :2 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
3  == TICK :3 CPU[_IDLE_] SPI[Sen2] I2C[_IDLE_]
4  == TICK :4 CPU[_IDLE_] SPI[Sen3] I2C[_IDLE_]
```

รูปที่ 5.4 ผลการจำลองการทำงานของกรณีทวนสอบ

ในการทวนสอบจะใช้สูตรแอลทีแอลมาช่วยในการทวนสอบด้วยเพื่อความถูกต้องและครอบคลุมสถานะทั้งหมดในปริภูมิสถานะในการทวนสอบ โดยจะกำหนดเวลาเพียงชั่วขณะเดียวที่ใช้ในการอ้างอิงเพื่อตรวจสอบสถานะคือเวลาที่สถานะนาฬิกาคือ *Execute* และกำลังจะเปลี่ยนเป็นสถานะต่อไป ผู้วิจัยใช้ชุดคำสั่ง *timeout* มาเป็นตรรกะสำหรับช่วงเวลาดังกล่าว

จากรูปที่ 5.5 ในบรรทัดที่ 1 คือมาโครที่เรียกว่า *ExecutionTime* เพื่อความสะดวก เมื่อเวลาที่มาโคร *ExecutionTime* มีค่าความจริงเป็นจริงคือช่วงเวลาที่ใช้ในการนับการทำงาน 1 ครั้ง

ในบรรทัดที่ 2 คือมาโคร Sen1TC (Sensor1 Timing Constraint) เป็นมาโครที่แสดงตำแหน่งเวลาที่ภารกิจ Sen1 ต้องครองทรัพยากร SPI หาก *GlobalClock* มีค่าเท่ากับ 1 และ 2 จะทำให้ Sen1TC มีค่าความจริงเป็นจริง

ในบรรทัดที่ 3 คือมาโคร Sen1TC (Sensor1 Resources Constraint) เป็นชนิดของทรัพยากรที่ ภารกิจ Sen1 ต้องถือครอง

ในบรรทัดที่ 8 สูตรแอลทีแอลให้ความหมายว่า เวลาใดก็ตามถ้าถึงเวลาที่ GlobalClock เท่ากับ 1 และ 2 แล้วผู้ครองทรัพยากร คือภารกิจ Sen1 และ เวลาใดก็ตามถ้าถึงเวลาที่ GlobalClock เท่ากับ 3 แล้วผู้ครองทรัพยากร คือภารกิจ Sen2 และ เวลาใดก็ตามถ้าถึงเวลาที่ GlobalClock เท่ากับ 4 แล้วผู้ครองทรัพยากร คือภารกิจ Sen3

```
1  #define ExecuteTime (timeout && GlobalClockState == C_Execute)
2  #define Sen1TC (ExecuteTime && (GlobalClock >=1)&&(GlobalClock <=2))
3  #define Sen1RC (Resource[1].owner==Sen1)
4  #define Sen2TC (ExecuteTime)&&(GlobalClock ==3)
5  #define Sen2RC (Resource[1].owner==Sen2)
6  #define Sen3TC (ExecuteTime)&&(GlobalClock ==4)
7  #define Sen3RC (Resource[1].owner==Sen3)
8  ltl LTLSens { [](Sen1TC->Sen1RC)&&[](Sen2TC->Sen2RC)&&[](Sen3TC->Sen3RC) }
```

รูปที่ 5.5 สูตรแอลทีแอลสำหรับทวนสอบ

เมื่อทำการทวนสอบแล้วพบว่าแบบจำลองสอดคล้องกับสูตรแอลทีแอลที่ให้

สรุปผล แบบจำลองสามารถที่จะจองทรัพยากร นับเวลาการทำงานได้อย่างถูกต้อง

2) แบบจำลองสามารถจัดลำดับความสำคัญและเลือกสิทธิในการครองทรัพยากร

กำหนดให้มีภารกิจสองภารกิจแสดงในตารางที่ 5.2

ตารางที่ 5.2 รายชื่อของภารกิจสำหรับการทวนสอบและพารามิเตอร์ที่เกี่ยวข้อง

Name	CPU	SPI	Serial	Periodic	Duration	Phase	Priority	Period
FC	True	False	False	False	2	-1	4	50
DSM	True	False	False	False	2	-1	3	50

ผลลัพธ์ที่คาดว่าจะได้ ภารกิจที่มีระดับความสำคัญสูงกว่าจะครอบครองทรัพยากรให้ห้วงเวลาที่กำหนดได้เสมอ และทำการทวนสอบเทียบกับ แบบอย่างคุณลักษณะที่บรรยายด้วยแอลทีแอล

กรรมวิธีการทวนสอบ แปลงภารกิจให้เป็นมาโครและแทรกคำสั่งสร้างภารกิจเมื่อตอนที่เริ่มทำงานจะได้ดังรูปที่ 5.6

```
1  #define CreateTask_FC Create(FC,true,false,false,false,2,-1,4,1,50,1)
2  #define CreateTask_DSM Create(DSM,true,true,false,false,2,-1,3,1,50,1)
```

รูปที่ 5.6 มาโครภารกิจที่ต้องการทวนสอบ

ทำการเพิ่มชุดคำสั่งดังกล่าวลงในแบบจำลองระบบและเพิ่มชุดคำสั่ง init ตามรูปที่ 5.7

```
1  init{atomic{CreateTask_FC;   CreateTask_DSM;}
2  run Ticking();}
```

รูปที่ 5.7 ชุดคำสั่งเริ่มต้นของกรณีทวนสอบ

เมื่อทำการจำลองการทำงานได้ผลดังรูปที่ 5.8 ภารกิจควบคุมการบินมีระดับความสำคัญสูงกว่าจะสามารถที่จะถือครองทรัพยากรได้เสมอ แสดงว่าการทำงานของการจัดระดับความสำคัญถูกต้อง

```
1   ==TICK :1 CPU[FC] SPI[_IDLE_] I2C[_IDLE_]
2   ==TICK :2 CPU[FC] SPI[_IDLE_] I2C[_IDLE_]
3   ==TICK :3 CPU[DSM] SPI[DSM] I2C[_IDLE_]
```

รูปที่ 5.8 ผลการจำลองการทำงานของกรณีทวนสอบ

และทำการทวนสอบแบบอย่างคุณลักษณะที่บรรยายด้วยภาษาแอลทีแอล

```
1  #define ExecuteTime (timeout && GlobalClockState == C_Execute)
2  #define isCreated(task) Tlist[task].isCreate
3  #define FC1TC (isCreated(FC)&&(ExecuteTime \
4          && (GlobalClock >=1)&&(GlobalClock <=2)))
5  #define FC1RC (Resource[0].owner==FC)
6  ltl LTLFC {[](FC1TC->FC1RC)}
```

รูปที่ 5.9 สูตรแอลทีแอลสำหรับทวนสอบ

จากรูปที่ 5.3 จากบรรทัดที่ 2 เป็นมาโครที่เรียกว่า *isCreated[ชื่อภารกิจ]* สำหรับตรวจว่าภารกิจนั้นๆ ถูกสร้างขึ้นมาแล้วหรือไม่แสดงค่าความจริงเป็นจริง สูตรแอลทีแอลให้ความหมายคือ เวลาใดก็ตามหากภารกิจ FC ถูกสร้างขึ้น และ ถึงเวลาที่ GlobalClock เท่ากับ 1 และ 2 แล้ว ผู้ครองทรัพยากรคือภารกิจ FC เมื่อทำการตรวจแบบจำลองแล้วพบว่าแบบจำลองสอดคล้องกับสูตรแอลทีแอลที่ให้

สรุปผล แบบจำลองสามารถจัดลำดับความสำคัญและเลือกสิทธิในการครองทรัพยากรได้ถูกต้อง

3) แบบจำลองสามารถขัดจังหวะการทำงานระหว่างภารกิจ

แบบจำลองสามารถขัดจังหวะการทำงานระหว่างภารกิจตามระดับความสำคัญได้ กำหนดให้มีภารกิจมีรายละเอียดของทั้งสองภารกิจแสดงในตารางที่ 5.3

ตารางที่ 5.3 รายชื่อของภารกิจสำหรับการทวนสอบและพารามิเตอร์ที่เกี่ยวข้อง

Name	CPU	SPI	Serial	Periodic	Duration	Phase	Priority	Cycle
Sen1	True	False	False	True	2	2	10	50
DSM	True	False	False	False	2	-1	3	50

ผลลัพธ์ที่คาดว่าจะได้ ภารกิจเซนเซอร์ 1 สามารถขัดจังหวะการทำงานของภารกิจการจัดเก็บข้อมูลระหว่างการทำงานได้ และทำการทวนสอบแบบจำลองเทียบกับ แบบอย่างคุณลักษณะที่บรรยายด้วยภาษาแอลทีแอล

กรรมวิธีการทวนสอบ แปลงภารกิจให้เป็นมาโครดังรูปที่ 5.10

```
1   #define CreateTask_Sen1 Create(Sen1,false,true,false,true,2,2,10,1,50,1)
2   #define CreateTask_DSM Create(DSM,true,true,false,false,2,-1,3,1,50,1)
```

รูปที่ 5.10 มาโครภารกิจที่ต้องการทวนสอบ

ทำการเพิ่มชุดคำสั่งดังกล่าวลงในแบบจำลองระบบและเพิ่มชุดคำสั่ง init ตามรูปที่ 5.11

```
1   atomic{   CreateTask_Sen1; CreateTask_DSM;}
2   run Ticking();}
```

รูปที่ 5.11 ชุดคำสั่งเริ่มต้นของกรณีทวนสอบ

พบว่าทั้งสองภารกิจมีส่วนที่ใช้ทรัพยากรร่วมกันคือเมื่อเวลาเท่ากับ 2 ต้องใช้ ทรัพยากร[1] (SPI) ร่วมกัน เมื่อทำการจำลองการทำงานได้ผลดังรูปที่ 5.12 พบว่าเมื่อเวลาเท่ากับ 1 ภารกิจการจัดเก็บข้อมูลสามารถครอบครองทรัพยากรได้ ได้แต่เมื่อ ภารกิจเซนเซอร์ 1 เริ่มทำงานเมื่อเวลา 2 ภารกิจการจัดเก็บข้อมูลถูกขัดขวางการทำงานโดยภารกิจเซนเซอร์ 1 จนกระทั่ง ภารกิจภารกิจเซนเซอร์ 1 ทำงานเสร็จจึงสามารถทำงานต่อได้

```
1    ==TICK :1 CPU[DSM] SPI[DSM] I2C[_IDLE_]
2    ==TICK :2 CPU[_IDLE_] SPI[Sen [1I2C[_IDLE_]
3    ==TICK :3 CPU[_IDLE_] SPI[Sen [1I2C[_IDLE_]
4    ==TICK : 4CPU[DSM] SPI[DSM] I2C[_IDLE_]
```

รูปที่ 5.12 ผลการจำลองการทำงานของกรณีทวนสอบ

ทำการทวนสอบโดยใช้สูตรแอลทีแอลดังรูปที่ 5.13

```
1  #define ExecuteTime (timeout && GlobalClockState == C_Execute)
2  #define isCreated(task) Tlist[task].isCreate
3  #define Sen11TC (isCreated(Sen1)&&(ExecuteTime \
4          && (GlobalClock >=2)&&(GlobalClock <=3)))
5  #define Sen1RC (Resource[1].owner==Sen1)
6  ltl LTLSen1 {[]( Sen11TC -> Sen1RC)}
```

รูปที่ 5.13 สูตรแอลทีแอลสำหรับทวนสอบ

จากบรรทัดที่ 1 เป็นมาโครสำหรับการนับการทำงาน 1 ครั้ง

จากบรรทัดที่ 2 เป็นมาโครจะแสดงค่าความจริงเป็นจริงหากภารกิจดังกล่าวถูกสร้างมาแล้ว

จากบรรทัดที่ 3 และ 4 สร้างมาโครสำหรับการตรวจห้วงเวลาการทำงานหาก *GlobalClock* มีค่าเท่ากับ 2 และ 3 จะให้ค่าความจริงเป็นจริง

จากบรรทัดที่ 5 สร้างมาโครสำหรับตรวจเงื่อนไขด้านทรัพยากร หากทรัพยากร[1] (คือ SPI) ถูกครอบครองโดย ภารกิจเซนเซอร์ 1 จะให้ค่าความจริงเป็นจริง

สูตรแอลทีแอลให้ความหมายคือคือ เวลาใดก็ตาม หากภารกิจเซนเซอร์ 1 ถูกสร้างขึ้น และ ถึงเวลาที่ GlobalClock เท่ากับ 2 และ 3 แล้ว ผู้ครองทรัพยากรคือภารกิจเซนเซอร์ 1

สรุปผล แบบจำลองสามารถขัดจังหวะการทำงานระหว่างภารกิจตามระดับความสำคัญได้อย่างถูกต้อง

4) แบบจำลองสามารถทำงานขนานกันได้หากไม่มีการแย่งทรัพยากรกัน

จากกรณีตัวอย่าง กำหนดให้มีภารกิจสองภารกิจ คือ ภารกิจเซนเซอร์ 1 (Sen1) มีระดับความสำคัญเท่ากับ 10 เป็นภารกิจที่ทำงานตามเวลา และ ภารกิจควบคุมการบิน (FC) มีระดับความสำคัญเท่ากับ 4 รายละเอียดของทั้งสองภารกิจแสดงในตารางที่ 5.4

ตารางที่ 5.4 รายชื่อของภารกิจสำหรับการทวนสอบและพารามิเตอร์ที่เกี่ยวข้อง

Name	CPU	SPI	Serial	Periodic	Duration	Phase	Priority	Cycle
Sen1	True	False	False	True	2	1	10	50
FC	True	False	False	False	2	-1	4	50

ผลลัพธ์ที่คาดว่าจะได้ ภารกิจเซนเซอร์1 และ ภารกิจควบคุมการบินสามารถทำงานให้ห้วงเวลาเดียวกันได้ และทำการทวนสอบแบบจำลองเทียบกับ แบบอย่างคุณลักษณะที่บรรยายด้วยภาษาแอลทีแอล

กรรมวิธีการทวนสอบ แปลงภารกิจให้เป็นมาโครและแทรกคำสั่งสร้างภารกิจเมื่อตอนที่เริ่มทำงานจะได้ดังรูปที่ 5.14

```
#define CreateTask_Sen1 Create(Sen1,false,true,false,true,2,2,10,1,50,1)
#define CreateTask_FC Create(FC,true,false,false,false,2,-1,4,1,50,1)
```

รูปที่ 5.14 มาโครภารกิจที่ต้องการทวนสอบ

ทำการเพิ่มชุดคำสั่งดังกล่าวลงในแบบจำลองระบบและเพิ่มชุดคำสั่ง init ตามรูปที่ 5.15

```
init{
 atomic{
   Resource[.[0owner = _IDLE_;
   Resource[.[1owner = _IDLE_;
   Resource[.[2owner = _IDLE_;
   CreateTask_Sen1;
   CreateTask_FC;
 }
 run Ticking();
}
```

รูปที่ 5.15 ชุดคำสั่งเริ่มต้นของกรณีทวนสอบ

พบว่าทั้งสองภารกิจไม่มีส่วนที่ใช้ทรัพยากรร่วมกัน เมื่อทำการจำลองการทำงานได้ผลดังรูปที่ 5.16

```
 ==TICK :1 CPU[FC] SPI[Sen [1I2C[_IDLE_]
 ==TICK :2 CPU[FC] SPI[Sen [1I2C[_IDLE_]
 ==TICK :3 CPU[_IDLE_] SPI[_IDLE_] I2C[_IDLE_]
```

รูปที่ 5.16 ผลการจำลองการทำงานของกรณีทวนสอบ

พบว่าเมื่อเวลาเท่ากับ 1 และ 2 ภารกิจทั้งสองสามารถทำงานพร้อมกันได้ และทำการทวนสอบโดยใช้สูตรแอลทีแอลดังรูปที่ 5.17

```
#define ExecuteTime (timeout && GlobalClockState == C_Execute)
#define isCreated(task) Tlist[task].isCreate
#define FC1TC (isCreated(FC)&&(ExecuteTime \
        && (GlobalClock >=1)&&(GlobalClock <=2)))
#define FC1RC (Resource[0].owner==FC)
#define Sen1TC (isCreated(Sen1)&&(ExecuteTime \
        && (GlobalClock >=1)&&(GlobalClock <=2)))
#define Sen1RC (Resource[1].owner==Sen1)
ltl LTLFC {[](FC1TC->FC1RC) && [](Sen1TC->Sen1RC)}
```

รูปที่ 5.17 สูตรแอลทีแอลสำหรับทวนสอบ

จากบรรทัดที่ 1 เป็นมาโครสำหรับการนับการทำงาน 1 ครั้ง

จากบรรทัดที่ 2 เป็นมาโครจะแสดงค่าความจริงเป็นจริงหากภารกิจดังกล่าวถูกสร้างมาแล้ว

จากบรรทัดที่ 3 และ 4 สร้างมาโครสำหรับการตรวจห้วงเวลาการทำงานหาก *GlobalClock* มีค่าเท่ากับ 1 และ 2 จะให้ค่าความจริงเป็นจริง

จากบรรทัดที่ 5 สร้างมาโครสำหรับตรวจเงื่อนไขด้านทรัพยากร หากทรัพยากร[0] (คือ CPU) ถูกครอบครองโดย ภารกิจควบคุมการบินจะให้ค่าความจริงเป็นจริง

จากบรรทัดที่ 6 และ 7 สร้างมาโครสำหรับการตรวจห้วงเวลาการทำงานหาก *GlobalClock* มีค่าเท่ากับ 1 และ 2 จะให้ค่าความจริงเป็นจริง

จากบรรทัดที่ 8 สร้างมาโครสำหรับตรวจเงื่อนไขด้านทรัพยากร หากทรัพยากร[1] (คือ SPI) ถูกครอบครองโดย ภารกิจเซนเซอร์ 1 จะให้ค่าความจริงเป็นจริง

สูตรแอลทีแอลให้ความหมายคือ เวลาใดก็ตาม ถ้าภารกิจควบคุมการบินถูกสร้างขึ้นและถึงเวลาที่ GlobalClock เท่ากับ 1 และ 2 แล้ว ผู้ครองทรัพยากร[0] คือภารกิจควบคุมการบิน และ เวลาใดก็ตามถ้าภารกิจเซนเซอร์ 1 ถูกสร้างขึ้น และ ถึงเวลาที่ GlobalClock เท่ากับ 1 และ 2 แล้ว ผู้ครองทรัพยากร[1] คือภารกิจเซนเซอร์ 1

สรุปผล แบบจำลองสามารถทำงานขนานกันได้หากไม่มีการแย่งทรัพยากรกันได้อย่างถูกต้อง

5) แบบจำลองสามารถมีสิทธิครอบครองทรัพยากรได้หากมีระดับความสำคัญเท่ากัน

จากกรณีตัวอย่าง กำหนดให้มีภารกิจสองภารกิจ คือ ภารกิจเซนเซอร์ 1 (Sen1) มีระดับความสำคัญเท่ากับ 10 เป็นภารกิจที่ทำงานตามเวลา และ ภารกิจเซนเซอร์ 2 (Sen2) มีระดับความสำคัญเท่ากับ 10 รายละเอียดของทั้งสองภารกิจแสดงในตารางที่ 5.5

ตารางที่ 5.5 รายชื่อของภารกิจสำหรับการทวนสอบและพารามิเตอร์ที่เกี่ยวข้อง

Name	CPU	SPI	Serial	Periodic	Duration	Phase	Priority	Cycle
Sen1	False	True	False	True	2	1	10	50
Sen2	False	True	False	True	2	1	10	50

ผลลัพธ์ที่คาดว่าจะได้ ทั้งสองภารกิจมีโอกาสที่จะครอบครองทรัพยากรได้ และทำการทวนสอบแบบจำลองเทียบกับ แบบอย่างคุณลักษณะที่บรรยายด้วยภาษาแอลทีแอล

กรรมวิธีการทวนสอบ แปลงภารกิจให้เป็นมาโครและแทรกคำสั่งสร้างภารกิจเมื่อตอนที่เริ่มทำงานจะได้ดังรูปที่ 5.17

```
1   #define CreateTask_Sen1 Create(Sen1,false,true,false,true,2,2,10,1,50,1)
2   #define CreateTask_Sen2 Create(Sen2,false,true,false,true,2,2,10,1,50,1)
```

รูปที่ 5.17 มาโครภารกิจที่ต้องการทวนสอบ

ทำการเพิ่มชุดคำสั่งดังกล่าวลงในแบบจำลองระบบและเพิ่มชุดคำสั่ง init ตามรูปที่ 5.18

```
1  init{
2   atomic{
3     Resource[.[0owner = _IDLE_;
4     Resource[.[1owner = _IDLE_;
5     Resource[.[2owner = _IDLE_;
6     CreateTask_Sen1;
7     CreateTask_Sen2;
8   }
9   run Ticking();}
```

รูปที่ 5.18 ชุดคำสั่งเริ่มต้นของกรณีทวนสอบ

พบว่าทั้งสองภารกิจใช้ห้วงเวลาเดียวกัน ทรัพยากรเดียวกันและระดับความสำคัญเท่ากันทำให้ผลที่ออกมาจะ ไม่สามารถคาดการณ์ได้ (Undeteministic) เมื่อทำการจำลองสามครั้ง ผลการจำลองได้ผลดังรูปที่ 5.19

ครั้งที่ 1

```
1    ==TICK :1 CPU[_IDLE_] SPI[Sen [2I2C[_IDLE_]
2    ==TICK :2 CPU[_IDLE_] SPI[Sen [2I2C[_IDLE_]
3    ==TICK :3 CPU[_IDLE_] SPI[Sen [1I2C[_IDLE_]
4    ==TICK :4 CPU[_IDLE_] SPI[Sen [1I2C[_IDLE_]
```

ครั้งที่ 2

```
1    ==TICK :1 CPU[_IDLE_] SPI[Sen [1I2C[_IDLE_]
2    ==TICK :2 CPU[_IDLE_] SPI[Sen [1I2C[_IDLE_]
3    ==TICK :3 CPU[_IDLE_] SPI[Sen [2I2C[_IDLE_]
4    ==TICK :4 CPU[_IDLE_] SPI[Sen [2I2C[_IDLE_]
```

ครั้งที่ 3

```
1    ==TICK :1 CPU[_IDLE_] SPI[Sen [1I2C[_IDLE_]
2    ==TICK :2 CPU[_IDLE_] SPI[Sen [2I2C[_IDLE_]
3    ==TICK :3 CPU[_IDLE_] SPI[Sen [1I2C[_IDLE_]
4    ==TICK : 4CPU[_IDLE_] SPI[Sen [2I2C[_IDLE_]
```

รูปที่ 5.19 ผลการจำลองการทำงานของกรณีทวนสอบ

พบว่าเมื่อเวลาเท่ากับ 1 และ 2 ภารกิจทั้งสองมีโอกาสที่จะทำงานได้ และทำการทวนสอบโดยใช้สูตรแอลทีแอลดังรูปที่ 5.20

```
1  #define ExecuteTime (timeout && GlobalClockState == C_Execute)
2  #define isCreated(task) Tlist[task].isCreate
3  #define Sen1TC (isCreated(Sen1)&&(ExecuteTime \
4          && (GlobalClock >=1)&&(GlobalClock <=2)))
5  #define Sen1RC (Resource[1].owner==Sen1)
6  #define Sen2TC (isCreated(Sen2)&&(ExecuteTime \
7          && (GlobalClock >=1)&&(GlobalClock <=2)))
8  #define Sen2RC (Resource[1].owner==Sen2)
9  ltl LTLFC {<>(Sen1TC->Sen1RC) && <>(Sen2TC->Sen2RC)}
```

รูปที่ 5.20 สูตรแอลทีแอลสำหรับทวนสอบ

จากบรรทัดที่ 1 เป็นมาโครสำหรับการนับการทำงาน 1 ครั้ง

จากบรรทัดที่ 2 เป็นมาโครจะแสดงค่าความจริงเป็นจริงหากภารกิจดังกล่าวถูกสร้างมาแล้ว

จากบรรทัดที่ 3 และ 4 สร้างมาโครสำหรับการตรวจห้วงเวลาการทำงานหาก *GlobalClock* มีค่าเท่ากับ 1 และ 2 จะให้ค่าความจริงเป็นจริง

จากบรรทัดที่ 5 สร้างมาโครสำหรับตรวจเงื่อนไขด้านทรัพยากร หากทรัพยากร[1] (คือ SPI) ถูกครอบครองโดย ภารกิจควบคุมการบินจะให้ค่าความจริงเป็นจริง

จากบรรทัดที่ 6 และ 7 สร้างมาโครสำหรับการตรวจช่วงเวลาการทำงานหาก *GlobalClock* มีค่าเท่ากับ 1 และ 2 จะให้ค่าความจริงเป็นจริง

จากบรรทัดที่ 8 สร้างมาโครสำหรับตรวจเงื่อนไขด้านทรัพยากร หากทรัพยากร[1] (คือ SPI) ถูกครอบครองโดย ภารกิจเซนเซอร์ 1 จะให้ค่าความจริงเป็นจริง

สูตรแอลทีแอลให้ใช้ตัวดำเนินการ ◊ (Eventually) ความหมายคือ จะมีสักครั้งหนึ่งที่ ถ้าภารกิจเซนเซอร์ 1 ถูกสร้างขึ้นและถึงเวลาที่ GlobalClock เท่ากับ 1 และ 2 แล้ว ผู้ครองทรัพยากร [1] คือภารกิจเซนเซอร์ 1 และ จะมีสักครั้งหนึ่งที่ถ้าภารกิจเซนเซอร์ 2 ถูกสร้างขึ้น และ ถึงเวลาที่ GlobalClock เท่ากับ 1 และ 2 แล้ว ผู้ครองทรัพยากร[1] คือภารกิจเซนเซอร์ 2

สรุปผล แบบจำลองสามารถมีสิทธิครอบครองทรัพยากรได้หากมีระดับความสำคัญเท่ากัน

5.3 การทวนสอบแบบจำลองย่อยการจัดกำหนดรายการเวลา

1) แบบจำลองสามารถจัดกำหนดรายการเวลาแบบระดับความสำคัญคงที่

ตารางที่ 5.6 รายชื่อของภารกิจสำหรับการทวนสอบและพารามิเตอร์ที่เกี่ยวข้อง

Name	CPU	SPI	Serial	Periodic	Duration	Phase	Priority	Period
Sen1	False	True	False	True	2	1	10	50
Sen2	False	True	False	True	2	1	9	50
Sen3	False	True	False	True	2	1	8	50

ผลลัพธ์ที่คาดว่าจะได้ เมื่อจำลองการทำงาน ภารกิจจะต้องเรียงลำดับตาม ระดับความสำคัญจากมากไปน้อย

กรรมวิธีการทวนสอบ ปรับวิธีการกำหนดรายการเวลาในแบบจำลองย่อยภารกิจให้เป็นดังรูป

```
1  :: atomic{((GlobalClockState == C_Schedule)
2          &&(isTaskActive)
3          &&(!Tlist[myTask.my_type].is_Scheduled))->
4               Tlist[myTask.my_type].is_Scheduled = true;}
```

รูปที่ 5.21 กำหนดให้การจัดรายการเวลาไม่ใช้โทโพโลยีใดเลย

เมื่อไม่กำหนดโทโพโลยีการกำหนดรายการเวลา จะถือว่าเป็นแบบจัดตามลำดับความสำคัญโดยอัตโนมัติ ทำการแปลงภารกิจทั้งสามเป็นคำสั่งมาโครสำหรับการสร้างภารกิจและสั่งให้ภารกิจทั้งสามทำงานเมื่อเริ่มการทำงานของแบบจำลองดังรูป ทำการเพิ่มชุดคำสั่งดังกล่าวลงในแบบจำลองระบบในรูปที่ 5.22

```
1  #defineCreateTask_Sen1Create(Sen1,false,true,false,true,2,1,10,1,50,1)
2  #defineCreateTask_Sen2Create(Sen2,false,true,false,true,2,1,9,1,50,1)
3  #defineCreateTask_Sen3Create(Sen3,false,true,false,true,2,1,8,1,50,1)
4  ...
5  init {
6   atomic{
7     Resource[0].owner = _IDLE_;
8     Resource[1].owner = _IDLE_;
9     Resource[2].owner = _IDLE_;
10    CreateTask_Sen1;
11    CreateTask_Sen2;
12    CreateTask_Sen3; }
13  run Ticking();}
```

รูปที่ 5.22 ชุดคำสั่งเริ่มต้นของกรณีทวนสอบ

ทำการจำลองการทำงาน ได้ผลออกมาดังรูปที่ 5.23 เมื่อสั่งให้ทำการจำลองการทำงานจะพบว่า ภารกิจทั้งสามสามารถที่จะครอบครองทรัพยากรในช่วงเวลาที่กำหนดไว้ได้โดยลำดับของการครอบครองภารกิจเรียงตามลำดับความสำคัญของภารกิจ

```
1  == TICK :1 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
2  == TICK :2 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
3  == TICK :3 CPU[_IDLE_] SPI[Sen2] I2C[_IDLE_]
4  == TICK :4 CPU[_IDLE_] SPI[Sen2] I2C[_IDLE_]
```

รูปที่ 5.23 ผลการจำลองการทำงานของกรณีทวนสอบ

สรุปผล แบบจำลองสามารถจัดกำหนดรายการเวลาแบบระดับความสำคัญคงที่ได้ถูกต้อง

2) แบบจำลองสามารถจัดกำหนดรายการเวลาแบบเส้นตายเป็นหลัก

เพื่อที่จะทวนสอบการทำงานของแบบจำลองย่อยการจัดกำหนดรายการเวลาว่าสามารถทำงานได้ถูกต้องหรือไม่ ภารกิจชุดเซนเซอร์มีการทำงานดังตารางที่ 5.7

ตารางที่ 5.7 รายชื่อของภารกิจสำหรับการทวนสอบและพารามิเตอร์ที่เกี่ยวข้อง

Name	CPU	SPI	Serial	Periodic	Duration	Phase	Priority	Cycle
Sen1	False	True	False	True	6	1	10	1
Sen2	False	True	False	True	4	1	9	1
Sen3	False	True	False	True	2	1	8	1

ผลลัพธ์ที่คาดว่าจะได้ เมื่อจำลองการทำงาน ภารกิจจะต้องเรียงลำดับตาม เวลาในการประมวลผล (Duration) จากน้อยไปมาก

กรรมวิธีการทวนสอบ ปรับวิธีการกำหนดรายการเวลาในแบบจำลองย่อยภารกิจให้เป็นดังรูปที่ 5.21

```
1       :: atomic{((GlobalClockState == C_Schedule)
2          &&(isTaskActive)
3          &&(!Tlist[myTask.my_type].is_Scheduled))->
4               Scheduler_DM(myTask.my_type);
5               Tlist[myTask.my_type].is_Scheduled = true;
6          }
```

รูปที่ 5.21 กำหนดให้การจัดรายการเวลาไม่ใช้โทโพโลยีเรียงลำดับตามเวลาประมวลผล

ทำการแปลงภารกิจทั้งสามเป็นคำสั่งมาโครสำหรับการสร้างภารกิจและสั่งให้ภารกิจทั้งสามทำงานเมื่อเริ่มการทำงานของแบบจำลองดังรูป ทำการเพิ่มชุดคำสั่งดังกล่าวลงในแบบจำลองแสดงในรูปที่ 5.22

```
1  #defineCreateTask_Sen1Create(Sen1,false,true,false,true,6,1,10,1,50,1)
2  #defineCreateTask_Sen2Create(Sen2,false,true,false,true,4,1,9,1,50,1)
3  #defineCreateTask_Sen3Create(Sen3,false,true,false,true,2,1,8,1,50,1)
4  init {
5   atomic{
6     Resource[0].owner = _IDLE_;
7     Resource[1].owner = _IDLE_;
8     Resource[2].owner = _IDLE_;
9     CreateTask_Sen1;
10    CreateTask_Sen2;
11    CreateTask_Sen3; }
12  run Ticking();}
```

รูปที่ 5.22 ชุดคำสั่งเริ่มต้นของกรณีทวนสอบ

ทำการจำลองการทำงาน ได้ผลออกมาดังรูปที่ 5.23 เมื่อสั่งให้ทำการจำลองการทำงานจะพบว่า ภารกิจทั้งสามสามารถที่จะครอบครองทรัพยากรในช่วงเวลาที่กำหนดไว้ได้โดยลำดับของการครอบครองภารกิจเรียงตามเวลาในการประมวลผลภารกิจ

```
1   == TICK :1 CPU[_IDLE_] SPI[Sen3] I2C[_IDLE_]
2   == TICK :2 CPU[_IDLE_] SPI[Sen3] I2C[_IDLE_]
3   == TICK :3 CPU[_IDLE_] SPI[Sen2] I2C[_IDLE_]
4   == TICK :4 CPU[_IDLE_] SPI[Sen2] I2C[_IDLE_]
5   == TICK :5 CPU[_IDLE_] SPI[Sen2] I2C[_IDLE_]
6   == TICK :6 CPU[_IDLE_] SPI[Sen2] I2C[_IDLE_]
7   == TICK :7 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
8   == TICK :8 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
9   == TICK :9 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
10  == TICK :10 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
11  == TICK :11 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
12  == TICK :12 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
```

รูปที่ 5.23 ผลการจำลองการทำงานของกรณีทวนสอบ

สรุปผล แบบจำลองสามารถจัดกำหนดรายการเวลาแบบเส้นตายเป็นหลักได้ถูกต้อง

3) แบบจำลองสามารถจัดกำหนดรายการเวลาแบบเส้นตายเร็วสุด

เพื่อที่จะทวนสอบการทำงานของแบบจำลองย่อยการจัดกำหนดรายการเวลาว่าสามารถทำงานได้ถูกต้องหรือไม่ ภารกิจชุดเซนเซอร์มีการทำงานดังตารางที่ 5.8

ตารางที่ 5.8 รายชื่อของภารกิจสำหรับการทวนสอบและพารามิเตอร์ที่เกี่ยวข้อง

Name	CPU	SPI	Serial	Periodic	Duration	Phase	Priority	Period
Sen1	False	True	False	True	2	1	10	50
Sen2	False	True	False	True	2	1	9	30
Sen3	False	True	False	True	2	1	8	10

ผลลัพธ์ที่คาดว่าจะได้ เมื่อจำลองการทำงานภารกิจจะเรียงลำดับตามเส้นตายจากน้อยไปมาก

กรรมวิธีการทวนสอบ ปรับวิธีการกำหนดรายการเวลาในแบบจำลองย่อยภารกิจให้เป็นดังรูป

```
1       :: atomic{((GlobalClockState == C_Schedule)
2          &&(isTaskActive)
3          &&(!Tlist[myTask.my_type].is_Scheduled))->
4               Scheduler_DM(myTask.my_type);
5               Tlist[myTask.my_type].is_Scheduled = true;
6          }
```

รูปที่ 5.24 กำหนดให้การจัดรายการเวลาแบบโทโพโลยีเส้นตายใกล้สุดทำงานก่อน

ทำการแปลงภารกิจทั้งสามเป็นคำสั่งมาโครสำหรับการสร้างภารกิจและสั่งให้ภารกิจทั้งสามทำงานเมื่อเริ่มการทำงานของแบบจำลองดังรูป เพิ่มชุดคำสั่งดังกล่าวลงในแบบจำลองระบบแสดงในรูปที่ 5.25

```
7  #defineCreateTask_Sen1Create(Sen1,false,true,false,true,2,1,10,1,50,1)
8  #defineCreateTask_Sen2Create(Sen2,false,true,false,true,2,1,9,1,30,1)
9  #defineCreateTask_Sen3Create(Sen3,false,true,false,true,2,1,8,1,10,1)
10 ...
11 init {
12  atomic{
13    Resource[0].owner = _IDLE_;
14    Resource[1].owner = _IDLE_;
15    Resource[2].owner = _IDLE_;
16    CreateTask_Sen1;
17    CreateTask_Sen2;
18    CreateTask_Sen3;
19  }
20  run Ticking();}
```

รูปที่ 5.25 ชุดคำสั่งเริ่มต้นของกรณีทวนสอบ

ทำการจำลองการทำงาน ได้ผลออกมาดังรูปที่ 5.26 เมื่อสั่งให้ทำการจำลองการทำงานจะพบว่า ภารกิจทั้งสามสามารถที่จะครอบครองทรัพยากรในช่วงเวลาที่กำหนดไว้ได้โดยลำดับของการครอบครองภารกิจเรียงตามเวลาเส้นตายของภารกิจ (คาบเวลามากเส้นตายไกล คาบเวลาน้อยเส้นตายใกล้)

```
1  == TICK :1 CPU[_IDLE_] SPI[Sen3] I2C[_IDLE_]
2  == TICK :2 CPU[_IDLE_] SPI[Sen3] I2C[_IDLE_]
3  == TICK :3 CPU[_IDLE_] SPI[Sen2] I2C[_IDLE_]
4  == TICK :4 CPU[_IDLE_] SPI[Sen2] I2C[_IDLE_]
5  == TICK :7 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
6  == TICK :8 CPU[_IDLE_] SPI[Sen1] I2C[_IDLE_]
```

รูปที่ 5.26 ผลการจำลองการทำงานของกรณีทวนสอบ

5.4 การทวนสอบแบบจำลองย่อยภารกิจไม่อิสระและกรณีตัวอย่าง

1) แบบจำลองย่อยภารกิจไม่อิสระสามารถสร้างภารกิจใหม่ได้

จากกรณีตัวอย่าง กำหนดให้มีภารกิจสองภารกิจ คือ ภารกิจรับสัญญาณเซนเซอร์ (SDA) มีระดับความสำคัญเท่ากับ 4 และ ภารกิจนำร่องด้วยความเฉื่อย (INS) มีระดับความสำคัญเท่ากับ 6 รายละเอียดของทั้งสองภารกิจแสดงในตารางที่ 5.9

ตารางที่ 5.9 รายชื่อของภารกิจสำหรับการทวนสอบและพารามิเตอร์ที่เกี่ยวข้อง

Name	CPU	SPI	Serial	Periodic	Duration	Phase	Priority	Cycle
SDA	True	False	False	True	2	-1	4	50
INS	True	False	False	False	5	-1	6	50

ผลลัพธ์ที่คาดว่าจะได้ เมื่อภารกิจรับสัญญาณเซนเซอร์ ทำงานเสร็จและภารกิจนำร่องด้วยความเฉื่อยต้องทำงานต่อทันที และทำการทวนสอบแบบจำลองเทียบกับแบบอย่างคุณลักษณะที่บรรยายด้วยภาษาแอลทีแอล

กรรมวิธีการทวนสอบ แปลงภารกิจเป็นมาโครและแทรกคำสั่งตอน init จะได้ดังรูปที่ 5.27

```
1  #define CreateTask_SDA Create(SDA,true,false,false,false,2,-1,4,1,50,1)
2  #define CreateTask_INS Create(INS,true,false,true,false,5,-1,6,1,50,1)
```

รูปที่ 5.2 มาโครภารกิจที่ต้องการทวนสอบ

ทำการเพิ่มชุดคำสั่งดังกล่าวลงในแบบจำลองระบบและเพิ่มชุดคำสั่ง init ตามรูปที่ 5.28

```
1  init{  atomic{
2     Resource[.[0owner = _IDLE_;
3     Resource[.[1owner = _IDLE_;
4     Resource[.[2owner = _IDLE_;
5     CreateTask_SDA;
6   }
7   run Ticking();
8  }
```

รูปที่ 5.28 ชุดคำสั่งเริ่มต้นของกรณีทวนสอบ

ภารกิจนำร่องด้วยความเฉื่อยเป็นภารกิจที่ขึ้นอยู่กับภารกิจรับสัญญาณเซนเซอร์ เพื่อที่จะนำเอาสัญญาณที่ได้มาประมวลผลดังนั้น ภารกิจนำร่องด้วยความเฉื่อยจะทำงานต่อเมื่อภารกิจรับสัญญาณเซนเซอร์เสร็จสิ้น จึงนำไปเขียนเป็นแบบจำลองย่อยภารกิจไม่อิสระได้ดังรูปที่ 5.29

```
1  active proctype DependentControl()
2  {  do
3       ::atomic{isTaskFinished[SDA] ->
4        isTaskFinished[SDA]=false;
5        CreateTask_INS;  //A6
6     }od; }
```

รูปที่ 5.29 แบบจำลองย่อยภารกิจไม่อิสระที่ต้องการทวนสอบ

เมื่อใดก็ตามที่ภารกิจรับสัญญาณเซนเซอร์ทำงานเสร็จจะเรียกให้ภารกิจนำร่องด้วยความเฉื่อยทำงานต่อทุกครั้ง

```
1   #define ExecuteTime (timeout && GlobalClockState == C_Execute)
2   #define SDATC ((ExecuteTime && (GlobalClock >=1)&&(GlobalClock <=2)))
3   #define SDARC (Resource[0].owner==SDA)
4   #define INSTC ((ExecuteTime && (GlobalClock >=3)&&(GlobalClock <=7)))
5   #define INSRC (Resource[0].owner==INS)
6   ltl LTLFC {[](SDATC->SDARC) && [](INSTC->INSRC)}
```

รูปที่ 5.30 สูตรแอลทีแอลสำหรับทวนสอบ

จากบรรทัดที่ 1 เป็นมาโครสำหรับการนับการทำงาน 1 ครั้ง

จากบรรทัดที่ 3 สร้างมาโครสำหรับการตรวจช่วงเวลาการทำงานหาก *GlobalClock* มีค่าเท่ากับ 1 และ 2 จะให้ค่าความจริงเป็นจริง

จากบรรทัดที่ 4 สร้างมาโครสำหรับตรวจเงื่อนไขด้านทรัพยากร หากทรัพยากร[0] (คือ CPU) ถูกครอบครองโดย ภารกิจอ่านสัญญาณเซนเซอร์จะให้ค่าความจริงเป็นจริง

จากบรรทัดที่ 5 สร้างมาโครสำหรับการตรวจช่วงเวลาการทำงานหาก *GlobalClock* มีค่าเท่ากับ 3 และ 7 จะให้ค่าความจริงเป็นจริง

จากบรรทัดที่ 6 สร้างมาโครสำหรับตรวจเงื่อนไขด้านทรัพยากร หากทรัพยากร[0] (คือ CPU) ถูกครอบครองโดย ภารกิจนำร่องด้วยความเฉื่อย จะให้ค่าความจริงเป็นจริง

สูตรแอลทีแอลใช้ [] (Infinitely) ความหมายคือ ทุกครั้งที่เมื่อถึงเวลาที่ GlobalClock เท่ากับ 1 และ 2 แล้ว ผู้ครองทรัพยากร[0] คือภารกิจรับสัญญาณเซนเซอร์ และ ทุกครั้งที่เมื่อถึงเวลาที่ GlobalClock เท่ากับ 3 ถึง 7 แล้ว ผู้ครองทรัพยากร[0] คือนำร่องด้วยความเฉื่อย

เมื่อทำการตรวจแบบจำลองแล้วพบว่าแบบจำลองสอดคล้องกับสูตรแอลทีแอลที่ให้

สรุปผล แบบจำลองย่อยภารกิจไม่อิสระ สามารถสร้างภารกิจใหม่เมื่อภารกิจที่กำหนดเสร็จสิ้น

5.5 การทวนสอบแบบจำลองย่อยเครื่องประมวลผลเอมทีแอลและกรณีตัวอย่าง

1) สามารถใช้ตัวดำเนินการเอมทีแอล □ (Infinitely) ได้

จากกรณีตัวอย่าง กำหนดให้มีภารกิจสองภารกิจ คือ ภารกิจเซนเซอร์ 1 (Sen1) มีระดับความสำคัญเท่ากับ 10 เป็นภารกิจที่ทำงานตามเวลา และ ภารกิจรับสัญญาณเซนเซอร์ (SDA) มีระดับความสำคัญเท่ากับ 4 และภารกิจนำร่องด้วยความเฉื่อย (INS) ความสำคัญเท่ากับ 6 รายละเอียดของภารกิจแสดงในตารางที่ 5.10

ตารางที่ 5.10 รายชื่อของภารกิจสำหรับการทวนสอบและพารามิเตอร์ที่เกี่ยวข้อง

Name	CPU	SPI	Serial	Periodic	Duration	Phase	Priority	Cycle
Sen1	False	True	False	True	2	1	10	50
SDA	True	False	False	False	2	-1	6	50
INS	True	False	Flase	False	5	-1	4	50

ผลลัพธ์ที่คาดว่าจะได้ เมื่อกำหนดให้ PTime1 และ PTime2 เท่ากับ 4 จะสอดคล้องกับแบบจำลอง หาก PTime1 และ PTime2 มีค่าเป็นอย่างอื่นจะได้ผลไม่สอดคล้อง

กรรมวิธีการทวนสอบ แปลงภารกิจให้เป็นมาโคร และแทรกคำสั่งสร้างภารกิจเมื่อตอนเริ่มทำงานจะได้ดังรูปที่ 5.31

```
#define CreateTask_Sen1 Create(Sen1,false,true,false,true,2,1,10,1,50,1)
#define CreateTask_SDA Create( SDA,true,false,false,false,2,-1,4,1,50,1)
#define CreateTask_INS Create(INS,true,false,true,false,5,-1,6,1,50,1)

init{
 atomic{
   Resource[.[0owner = _IDLE_;
   Resource[.[1owner = _IDLE_;
   Resource[.[2owner = _IDLE_;
   CreateTask_Sen1;
 }
 run Ticking();}
```

รูปที่ 5.31 ชุดมาโครและคำสั่งสร้างภารกิจสำหรับการทวนสอบ

ภารกิจทั้งสามภารกิจเป็นภารกิจที่มีความเกี่ยวเนื่องกันโดยที่ภารกิจรับสัญญาณเซนเซอร์จะรอจนกว่าจะรับสัญญาณจากเซนเซอร์สำเร็จ ถึงจะเริ่มการทำงาน และ ภารกิจนำร่องด้วยความเฉื่อยจะเริ่มทำงานหลังจากที่ภารกิจรับสัญญาณเซนเซอร์ทำงานเสร็จ โดยทั้งสองภารกิจถูกเชื่อมผ่านแบบจำลองควบคุมภารกิจไม่อิสระดังรูปที่ 5.32


```
active proctype DependentControl(){
 do
   ::atomic{isTaskFinished[Sen-[1>
     isTaskFinished[Sen=[1false;
     CreateTask_SDA;
   }

   ::atomic{isTaskFinished[SDA] ->
     isTaskFinished[SDA]=false;
     CreateTask_INS;
   }

   ::atomic{isTaskFinished[INS] ->
     isTaskFinished[INS]=false;
   }
 od
}
```

รูปที่ 5.32 ชุดภารกิจไม่อิสระสำหรับการทวนสอบ

จากบรรทัดที่ 1 เมื่อภารกิจเซนเซอร์ 1 (Sen1) ทำงานเสร็จจะทำให้ตรรกะที่แสดงในบรรทัดที่ 3 เป็นจริงทำให้แบบจำลองทำการสร้างภารกิจใหม่คือภารกิจรับสัญญาณเซนเซอร์ (SDA) และเช่นเดียวกัน เมื่อภารกิจรับสัญญาณเซนเซอร์ทำงานเสร็จก็จะเรียกให้ภารกิจนำร่องด้วยความเฉื่อย (INS) ทำงานอย่างต่อเนื่องกัน เมื่อทำการจำลองการทำงานพบว่าจะผลการจำลองดังรูปที่ 5.33

```
1   ==TICK :1 CPU[_IDLE_] SPI[Sen [1I2C[_IDLE_]
2   ==TICK :2 CPU[_IDLE_] SPI[Sen [1I2C[_IDLE_]
3   ==TICK :3 CPU[SDA] SPI[_IDLE_] I2C[_IDLE_]
4   ==TICK :4 CPU[SDA] SPI[_IDLE_] I2C[_IDLE_]
5   ==TICK :5 CPU[INS] SPI[_IDLE_] I2C[INS]
6   ==TICK :6 CPU[INS] SPI[_IDLE_] I2C[INS]
7   ==TICK :7 CPU[INS] SPI[_IDLE_] I2C[INS]
8   ==TICK :8 CPU[INS] SPI[_IDLE_] I2C[INS]
9   ==TICK : 9CPU[INS] SPI[_IDLE_] I2C[INS]
```

รูปที่ 5.33 ผลการจำลองการทำงานของกรณีทวนสอบ

หากไม่มีภารกิจอื่นๆ แทรกระหว่างกลางการทำงานจะพบว่า ภารกิจทั้งสองสามารถทำตามต่อกันได้อย่างแม่นยำ และไม่ล่าช้า แต่หากการทำงานจริงนั้น จะมีภารกิจอื่นๆ ที่ทำงานแทรกกันและอาจจะเกิดการขัดขวางกันได้ ดังนั้นหากต้องการกำหนดให้ภารกิจดังกล่าวทำงาน ณ ตำแหน่งเวลาที่กำหนดสามารถที่จะเขียนเป็นสูตรเอมทีแอลได้ดังนี้

$$\square(\text{Sen1Event} \rightarrow \square_{[25]}\text{INSEvent})$$

แปลได้ว่า ไม่ว่าเวลาใดก็ตาม เมื่อเกิดเหตุการณ์เริ่มต้นภารกิจเซนเซอร์ 1 และจะต้องเกิดเหตุการณ์เริ่มต้นภารกิจนำร่องด้วยความเฉื่อย เมื่อเวลาเท่ากับ 5 หน่วยเวลา ผู้วิจัยจะใช้สูตรเอมทีแอลดังกล่าวนี้แปลงให้อยู่ในรูปของแบบจำลองโปรเมลา และ สูตรแอลทีแอลเพื่อที่จะใช้ในการตรวจสอบแบบจำลองว่ามีความสอดคล้องกับสูตรดังกล่าวหรือไม่ ผู้วิจัยใช้ตรรกกะชื่อ *isTaskStarted[ชื่อภารกิจ]* ตรรกะนี้จะเป็นจริงเมื่อขณะที่ภารกิจต่างๆทำงานเป็นช่วงเวลาแรก แสดงในรูปที่ 5.34

```
1   #define pEvent (isTaskStarted[Sen1])
2   #define qEvent (isTaskStarted[INS])
3   MTLevent Event0;
4   ltl LTL003 { [](Event0.EventFlag->(MTL_Gss(Event0))) }
5   inline MTLEngine(){
6     Selected_MTL = Selected; // do not change
7     Event0.name = Event_p
8     Event0.Event = pEvent;
9     Event0.Reaction = qEvent;
10    Event0.pTime1 = 4;
11    Event0.pTime2 = 4;
12    MTLCounter(Event0);}
```

รูปที่ 5.34 แบบจำลองย่อยเครื่องประมวลผลเอมทีแอลและสูตรแอลทีแอล

ในบรรทัดที่ 1 ทำการสร้างมาโคร *pEvent* สำหรับเป็นเหตุการณ์เริ่มต้น โดย *pEvent* จะเป็นจริงเมื่อภารกิจเซนเซอร์ 1 เริ่มทำงานในครั้งแรก

ในบรรทัดที่ 2 ทำการสร้างมาโคร *qEvent* สำหรับเป็นเหตุการณ์ตอบสนอง โดย *qEvent* จะเป็นจริงเมื่อภารกิจนำร่องความเฉื่อยเริ่มทำงาน

ในบรรทัดที่ 3 ทำการสร้างตัวแปร *Event0* เพื่อใช้ในการปรับค่าของตัวแปรเอมทีแอล

ในบรรทัดที่ 4 ทำการสร้างสูตรแอลทีแอล โดยแปลจากสูตรเอมทีแอล

ทำการทวนสอบแบบจำลองดังกล่าวแสดงในตารางที่ 5.11 โดยปรับค่า pTime1 และ pTime2 พบว่าหาก pTime เท่ากับ 4 จะทำให้แบบจำลองสอดคล้องกับแบบอย่างคุณลักษณะที่กำหนด

ตารางที่ 5.11 ผลการทดสอบเมื่อกำหนด PTime เป็นค่าต่างๆ

Execution	PTime1,2	Verification Result
SEN1	n/a	Fail
SEN1	1	Fail
SDA	2	Fail
SDA	3	Fail
INS	4	Satisfy
INS	5	Fail

สรุปผล แบบจำลองสามารถใช้ตัวดำเนินการ □ (Infinitely) และกำหนดช่วงเวลาได้อย่างถูกต้อง

2) สามารถใช้ตัวดำเนินการเอมทีแอล ◊ (Eventually) ได้

สำหรับกรณีที่มีภารกิจอื่นๆ ที่ทำงานแทรกกันดังนั้นหากต้องการกำหนดให้ภารกิจดังกล่าวสามารถล่าช้าได้แต่ไม่เกิดช่วงเวลาที่กำหนดทำงาน ณ ตำแหน่งเวลาที่กำหนดสามารถที่จะเขียนเป็นสูตรเอมทีแอลได้ดังนี้

$$[](\text{Sen1Event} \rightarrow \Diamond_{[2,6]} \text{INSEvent})$$

แปลได้ว่า ไม่ว่าเวลาใดก็ตาม เมื่อเกิดเหตุการณ์เริ่มต้นภารกิจเซนเซอร์ 1 และจะต้องเกิดเหตุการณ์เริ่มต้นภารกิจนำร่องด้วยความเฉื่อยหลังจากนั้น 0 หน่วยเวลาถึง 6 หน่วยเวลา อย่างน้อย 1 ครั้ง ผู้วิจัยจะใช้สูตรเอมทีแอลดังกล่าวนี้แปลงให้อยู่ในรูปของแบบจำลองโปรเมลา และ สูตรแอลทีแอลเพื่อที่จะใช้ในการตรวจสอบแบบจำลองว่ามีความสอดคล้องกับสูตรดังกล่าวหรือไม่ จากกรณีศึกษาก่อนหน้านี้ (5.5.3.1)

ผลลัพธ์ที่คาดว่าจะได้ เมื่อกำหนดให้ PTime1 เท่ากับ 0 และ PTime2 ตั้งแต่ 4 ขึ้นไปจะสอดคล้องกับแบบจำลอง PTime2 มีค่าน้อยกว่า 4 จะไม่สอดคล้องกับแบบจำลอง

กรรมวิธีการทวนสอบ ผู้วิจัยใช้ตรรกะชื่อ *isTaskStarted[ชื่อภารกิจ]* ตรรกะนี้จะเป็นจริงเมื่อขณะที่ภารกิจต่างๆทำงานเป็นช่วงเวลาแรก

```
1   #define pEvent (isTaskStarted[Sen([1
2   #define qEvent (isTaskStarted[INS])
3   MTLevent Event0;
4   ltl LTL003 { [](Event0.EventFlag->(MTL_Fss(Event0))) }
5   inline MTLEngine(){
6      Selected_MTL = Selected; // do not change
7      // Set up for Event 0
8      Event0.name = Event_p
9      Event0.Event = pEvent;
10     Event0.Reaction = qEvent;
11     Event0.pTime1 = 0;
12     Event0.pTime2 = 6;
13     MTLCounter(Event0);}
```

รูปที่ 5.35 แบบจำลองย่อยเครื่องประมวลผลเอมทีแอลและสูตรแอลทีแอล

ในบรรทัดที่ 1 ทำการสร้างมาโคร *pEvent* สำหรับเป็นเหตุการณ์เริ่มต้น โดย *pEvent* จะเป็นจริงเมื่อภารกิจเซนเซอร์ 1 เริ่มทำงานในครั้งแรก

ในบรรทัดที่ 2 ทำการสร้างมาโคร *qEvent* สำหรับเป็นเหตุการณ์ตอบสนอง โดย *qEvent* จะเป็นจริงเมื่อภารกิจนำร่องความเฉื่อยเริ่มทำงาน

ในบรรทัดที่ 3 ทำการสร้างตัวแปร *Event0* เพื่อใช้ในการปรับค่าของตัวแปรเอมทีแอล

ในบรรทัดที่ 4 ทำการสร้างสูตรแอลทีแอล โดยแปลจากสูตรเอมทีแอล โดยใช้ตัวดำเนินการ Eventually และ ทำการทวนสอบแบบจำลองดังกล่าวแสดงผลในตารางที่ 5.12 โดยปรับค่า pTime1 และ pTime2 พบว่าหาก pTime2 มีค่ามากกว่าหรือเท่ากับ 4 จะทำให้แบบจำลองสอดคล้องกับแบบอย่างคุณลักษณะที่กำหนด

ตารางที่ 5.11 ผลการทดสอบเมื่อกำหนด PTime เป็นค่าต่างๆ

Tick	Excution	PTime1	PTime2	Verification Result
2	SEN1	0	1	Fail
3	SDA	0	2	Fail
4	SDA	0	3	Fail
5	INS	0	4	Satisfy
6	INS	0	5	Satisfy
7	INS	0	6	Satisfy

สรุปผล แบบจำลองสามารถใช้ตัวดำเนินการ ◊ (Eventually)และกำหนดช่วงเวลาได้อย่างถูกต้อง

5.6 การทวนสอบระบบเมื่อเครื่องทำการบินปกติ

การทวนสอบการจัดกำหนดรายการเวลากรณีแรก เป็นการทวนสอบเมื่อ เครื่องบินทำการบินปกติ มีการทำงานของอุปกรณ์แบบคาบเวลาอย่างต่อเนื่องกันทุกอุปกรณ์ มีการประกันว่าอุปกรณ์แบบคาบเวลาทุกตัวสามารถทำงานโดยไม่พลาดเส้นตายแม้มีการบันทึกข้อมูลการบินอย่างต่อเนื่อง และมีการสื่อสารกับนักบินภายนอกบางครั้ง

ผลลัพธ์ที่คาดว่าจะได้

เมื่อทำการทวนสอบแบบจำลองด้วยข้อกำหนดที่ระบุไว้เทียบกับแบบอย่างคุณลักษณะ พบว่าแบบจำลองสอดคล้องกับแบบอย่างคุณลักษณะ

กรรมวิธีการทวนสอบ

ทำการนิยามความหมายของแต่ละตัวแปร NormalFlight คือ การบินปกติ จะมีภารกิจดังนีเ

$$\square\ (\text{Sen1Event} \rightarrow \Diamond_{[2,10]}\ \text{ServEvent})$$

RecordON คือ อุปกรณ์บันทึกข้อมูล หมายถึงภารกิจบันทึกข้อมูล

$$\square\ (\text{FCEvent} \rightarrow \Diamond_{[0,10]}\text{SSDEvent})$$

LCommON คือ มีการสื่อสารกับนักบินภายนอก หมายถึงมีการใช้ภารกิจการสื่อสารระยะไกล

$$\square\ (\text{LComEvent} \rightarrow \Diamond_{[0,10]}\ \text{FCEvent})$$

เงื่อนไขดังกล่าวสามารถเขียนเป็นแบบอย่างคุณลักษณะได้ดังนี้

$$\square\ (\text{NormalFlight} \wedge \text{RecordON} \wedge \text{LCommOn})$$

นำแบบจำลองย่อยภารกิจที่เสร็จสมบูรณ์ และ ภารกิจไม่อิสระที่เสร็จสมบูรณ์มาใช้ในแบบจำลอง กำหนดให้ชุดคำสั่งเริ่มต้น (Init) ดังรูปที่ 5.36

```
1  init { atomic {
2     Resource[0].owner = _IDLE_;
3     Resource[1].owner = _IDLE_;
4     Resource[2].owner = _IDLE_;
5     CreateTask_Sen1;
6     CreateTask_Sen2;
7     CreateTask_Sen3; }
8   run Ticking();}
9  #define Sen1Event  (isTaskFinished[Sen1])
10 #define ServEvent (isTaskStarted[Serv])
11 #define FCEvent  (isTaskStarted[FC])
12 #define SSDEvent (isTaskStarted[SSD])
13 #define LongCommEvent (isTaskFinished[DR])
14 #define SesnEvent isTaskStarted[Sen1] && isTaskStarted[Sen2] &&
    isTaskStarted[Sen3]
15 MTLevent Event0; MTLevent Event1; MTLevent Event2;
16 mtype {NormalFlight_Constraint,RecordON_Constraint, LongCommON_Constraint}
17 #define NormalFlight [](Event0.Event->MTL_Fss(Event0))
18 #define RecordON [](Event1.Event->MTL_Fss(Event1))
19 #define  LongCommON [](Event2.Event->MTL_Fss(Event2))
20 ltl ltl001 {NormalFlight&&RecordON&&LongCommON}
21 inline MTLEngine(){
22    Selected_MTL = Selected; // do not change
23    Event0.name = NormalFlight_Constraint;      // Set up for Event 0
24    Event0.Event = Sen1Event;
25    Event0.Reaction = ServEvent;
26    Event0.pTime1 = 5;
27    Event0.pTime2 = 20;
28    MTLCounter(Event0);
29    Event1.name = RecordON_Constraint;     // Set up for Event 1
30    Event1.Event = FCEvent;
31    Event1.Reaction = SSDEvent;
32    Event1.pTime1 = 0;
33    Event1.pTime2 =10;
34    MTLCounter(Event1);
35    Event2.name = LongCommON_Constraint;     // Set up for Event 2
36    Event2.Event =  LongCommEvent;
37    Event2.Reaction = FCEvent;
38    Event2.pTime1 = 0;
39    Event2.pTime2 = 10;
40    MTLCounter(Event2);
```

รูปที่ 5.33 แบบจำลองกรณีเครื่องทำการบินปกติ

เมื่อทำการทวนสอบพบว่าแบบจำลองสอดคล้องกับแบบอย่างคุณลักษณะที่ให้

สรุปผล แบบจำลองสามารถทำงานภายใต้เงื่อนไขที่กำหนดได้อย่างถูกต้อง

5.7 การทวนสอบเมื่อนำเครื่องขึ้นและลง

กรณีทวนสอบการจัดกำหนดรายการเวลากรณีที่สองเป็นการทวนสอบ เมื่อนำเครื่องขึ้นและลง มีการทำงานของอุปกรณ์แบบคาบเวลาอย่างต่อเนื่องทุกอุปกรณ์ มีการประกันว่าอุปกรณ์แบบคาบเวลาทุกตัวสามารถทำงานโดยไม่พลาดเส้นตาย มีการสื่อสารกับบินภายนอกอย่างต่อเนื่อง

NormalFlight คือ การบินปกติ จะมีภารกิจ

$$\Box \text{ (Sen1Event} \rightarrow \Diamond_{[2,10]} \text{ ServEvent)}$$

SCommON คือ มีการสื่อสารกับผ่านเสาอากาศรอบทิศทาง หมายถึงมีการใช้ภารกิจการสื่อสารระยะใกล้

$$\Box \text{ (SComEvent} \rightarrow \Box_{[10,10]} \text{ FCEvent)}$$

เงื่อนไขดังกล่าวสามารถเขียนเป็นแบบอย่างคุณลักษณะได้ดังนี้

$$\Box \text{ (NormalFlight} \wedge \text{ SCommOn)}$$

นำแบบจำลองย่อยภารกิจที่เสร็จสมบูรณ์ และ ภารกิจไม่อิสระที่เสร็จสมบูรณ์มาใช้ในแบบจำลอง

ทำการสร้างแบบจำลองย่อยการจัดกำหนดรายการเวลา ตามข้อ 5.6

ทำการทวนสอบ พบว่า แบบจำลองสอดคล้องกับแบบอย่างคุณลักษณะที่ให้

สรุปผล แบบจำลองสามารถทำงานภายใต้เงื่อนไขที่กำหนดได้อย่างถูกต้อง

บทที่ 6

สรุปผลงานวิจัยและข้อเสนอแนะ

6.1 สรุปผลงานวิจัย

การทวนสอบการจัดกำหนดรายการเวลาบนระบบยูเอวีขนาดเล็ก เป็นงานวิจัยที่นำเครื่องมือสปินมาประยุกต์ใช้ในการจำลองการทำงานของระบบเวลาจริงบนระบบสมองกลฝังตัว ที่ได้มาจากการแปลงเอาการทำงานของระบบเวลาจริงสำคัญ 5 ส่วน มาแปลงเป็นแบบจำลองย่อย 5 แบบจำลอง ประกอบด้วย แบบจำลองย่อยระบบสัญญาณนาฬิกา แบบจำลองย่อยภารกิจ แบบจำลองย่อยภารกิจไม่อิสระ แบบจำลองย่อยการจัดกำหนดรายการเวลา และ แบบจำลองย่อยเครื่องประมวลผลเอมทีแอล แบบจำลองทั้งหมดนี้เมื่อประกอบกันสามารถที่จะจำลองการทำงานของระบบเวลาจริงที่มีการแบ่งทรัพยากรตามโทโพโลยีการจัดกำหนดรายการเวลาอย่างมีประสิทธิภาพ

แบบจำลองนี้รองรับการทวนสอบด้วยแบบรูปคุณลักษณะที่เขียนด้วยภาษาเอมทีแอล ซึ่งเป็นภาษาหนึ่งที่สามารถใช้ในการอธิบายการทำงานระบบเวลาจริงได้อย่างลึกซึ้งที่สุด ความสามารถในการรองรับการทวนสอบด้วยภาษาเอมทีแอลนี้ ในปัจจุบันยังไม่มีเครื่องมืออัตโนมัติที่สามารถนำเอมทีแอลไปใช้ในการตรวจสอบระบบเวลาจริง ทำให้แบบจำลองนี้เป็นแบบจำลองแรกๆ ที่สามารถใช้เอมทีแอลในลักษณะของแบบรูปคุณลักษณะได้ การใช้แบบจำลองนี้สามารถลดเวลาในการทวนสอบระบบให้กับผู้ทดสอบ แบบจำลองสามารถใช้ในการทวนสอบระบบกรณีศึกษาได้อย่างมีประสิทธิภาพ

6.2 ข้อจำกัด

1) การทวนสอบด้วยแบบจำลองนี้หากนำไปใช้กับโครงการใหญ่ที่มีความซับซ้อนมาก หรือมีเส้นทางของภารกิจไม่อิสระมากกว่า 1 เส้นทางจะทำให้เกิดปริมาณของสถานะมหาศาลจะทำให้ต้องใช้หน่วยความจำขนาดใหญ่และเวลาในการทวนสอบเป็นเวลานาน
2) การเก็บค่าพารามิเตอร์ภารกิจหากเวลาประมวลผลมากสุดไม่เท่ากันทุกครั้งจะทำให้การคำนวนและสร้างแบบจำลองภารกิจเกิดความคลาดเคลื่อนและทำให้ผลการทวนสอบไม่มีประสิทธิภาพ

6.3 ข้อเสนอแนะ

ขั้นตอนวิธีการทวนสอบที่ได้นำเสนอในงานวิจัยนี้ เป็นการประยุกต์นำสปินมาช่วยในการจำลองการทำงานของระบบเวลาจริง แม้ว่าแบบจำลองจะสามารถทำงานได้อย่างที่ออกแบบไว้ แต่หากทำการดัดแปลงเครื่องมือสปินเพื่อสนับสนุนการทำงานที่เป็นระบบเวลาจริงโดยเฉพาะน่าจะสามารถเพิ่มประสิทธิภาพในการทวนสอบมากขึ้น

รายการอ้างอิง

1. Saeedloei, N. and G. Gupta, *Verifying complex continuous real-time systems with coinductive CLP(R)*, in *Proceedings of the 4th international conference on Language and Automata Theory and Applications*. 2010, Springer-Verlag: Trier, Germany. p. 536-548.
2. Dill, D.L., *Timing Assumptions and Verification of Finite-State Concurrent Systems*, in *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*. 1990, Springer-Verlag. p. 197-212.
3. Ostroff, J.S., *Temporal logic for real time systems*. 1989: John Wiley & Sons, Inc. 209.
4. Henzinger, T.A., Z. Manna, and A. Pnueli, *Temporal proof methodologies for real-time systems*. 1991, Stanford University.
5. Baier, C. and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. 2008: The MIT Press. 975.
6. Mihai Florian, E.G., Gerard Holzmann. *Logic Model Checking of Time-Periodic Real-Time System*. in *Aerospace 2012 Conference*. 2012.
7. Wilhelm, R.a.E., *The worst-case execution-time problem overview of methods and survey of tools.* ACM Trans. Embed. Comput. Syst., 2008. **7**(3): p. 36:1--36:53.
8. Burns, A. and A. Wellings, *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. 2009: Addison-Wesley Educational Publishers Inc. 624.
9. Stankovic, J.A., *Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems.* Computer, 1988. **21**(10): p. 10-19.
10. Amir, E., S. McCanne, and R. Katz, *An active service framework and its application to real-time multimedia transcoding*, in *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*. 1998, ACM: Vancouver, British Columbia, Canada. p. 178-189.

11. Buttazzo, G.C., *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. 2004: Springer-Verlag TELOS.
12. Alur, R., *Principles of Cyber-Physical Systems*. 2015: The MIT Press. 464.
13. Silberschatz, A., P.B. Galvin, and G. Gagne, *Operating System Concepts*. 2008: Wiley Publishing. 992.
14. Cheng, A.M.K., *Real-Time Systems: Scheduling, Analysis, and Verification*. 2002: John Wiley & Sons, Inc. 552.
15. Kopetz, H., *Real-Time Systems : Design Principles for Distributed Embedded Applications*. 2011, New York: Springer.
16. Pnueli, A., *The Temporal Logic of Programs*. 1997, Weizmann Science Press of Israel.
17. Alur, R. and T.A. Henzinger, *Real-time logics: complexity and expressiveness.* Inf. Comput., 1993. **104**(1): p. 35-77.
18. Bellini, P., R. Mattolini, and P. Nesi, *Temporal logics for real-time system specification.* ACM Comput. Surv., 2000. **32**(1): p. 12-42.
19. Holzmann, G.J., *Design and validation of computer protocols*. 1991: Prentice-Hall, Inc. 500.
20. Holzmann, G., *The SPIN Model Checker: Primer and Reference Manual*. 2011: Addison-Wesley Professional. 608.
21. Holzmann, G.J., *Design and validation of protocols: a tutorial.* Comput. Netw. ISDN Syst., 1993. **25**(9): p. 981-1017.
22. Tripakis, S. and C. Courcoubetis, *Extending Promela and Spin for Real Time*, in *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*. 1996, Springer-Verlag. p. 329-348.
23. Bosnacki, D. and D. Dams, *Integrating Real Time into Spin: A Prototype Implementation*, in *Proceedings of the FIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV XVIII)*. 1998, Kluwer, B.V. p. 423-438.

24. Bosnacki, D., *Partial Order Reduction in Presence of Rendez-vous Communications with Unless Constructs and Weak Fairness*, in *Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking*. 1999, Springer-Verlag. p. 40-56.

25. Jo\, et al., *Some Recent Results in Metric Temporal Logic*, in *Proceedings of the 6th international conference on Formal Modeling and Analysis of Timed Systems*. 2008, Springer-Verlag: Saint Malo, France. p. 1-13.

ภาคผนวก

ภาคผนวก ก

แบบจำลองย่อยภารกิจที่เสร็จสมบูรณ์ (Finished submodel)

เมื่อทำการแปลงภารกิจทั้งหมด แล้วจะได้เป็นมาโครสำหรับการสร้างแบบจำลองดังนี้

```
41 #define CreateTask_Sen1 Create( Sen1,false ,true,false,true,4 ,1 ,10,-1,50,1)
42 #define CreateTask_Sen2 Create( Sen2,false ,true,false,true,2 ,3 ,9 ,-1,50,1)
43 #define CreateTask_Sen3 Create( Sen3,false ,true,false,true,2 ,4 ,8 ,-1,50,1)
44 #define CreateTask_Serv Create( Serv ,false ,true,false,false ,3 ,-1,10,1 ,50,1)
45 #define CreateTask_GPS Create( GPS ,true,false ,false,true,4 ,1 ,4 ,1 ,50,1);
46 #define CreateTask_ODR Create( ODR ,true,false ,false,true,1 ,10,10,1 ,50,1);
47 #define CreateTask_INS Create( INS ,true,false ,true ,false ,5 ,-1,6 ,1 ,50,1)
48 #define CreateTask_FCCreate( FC ,true,false ,false,false ,5 ,-1,4 ,1 ,50,1)
49 #define CreateTask_CSC Create( CSC ,true,false ,false,false ,2 ,-1,8 ,1 ,50,1)
50 #define CreateTask_DSMCreate( DSM ,true,false ,false,false ,2 ,-1,3 ,1 ,50,1)
51 #define CreateTask_SDA Create( SDA ,true,false ,false,false ,2 ,-1,4 ,1 ,50,1)
52 #define CreateTask_SSD Create( SSD ,false ,false ,true ,false ,5 ,-1,4 ,1 ,50,1)
53 #define CreateTask_SPS Create( SPS ,true,false ,false,false ,1 ,-1,4 ,1 ,50,1);
54 #define CreateTask_SCom Create( SCom,true,false ,false,false ,2 ,-1,4 ,1 ,50,1)
55 #define CreateTask_LCom Create( LCom,true,true,false,false ,5 ,-1,5 ,1 ,50,1)
56 #define CreateTask_DRCreate( DR ,false,false ,true,true,7 ,1 ,8 ,1 ,50,1);
```

แบบจำลองย่อยภารกิจไม่อิสระที่เสร็จสมบูรณ์ (Finished dependent task submodel)

เมื่อทำการแปลงภารกิจไม่อิสระทั้งหมด แล้วจะได้เป็นมาโครสำหรับการสร้างแบบจำลองดังนี้

```
57 active proctype DependentTask() {
58   do ::atomic{isTaskFinished[Sen1] && isTaskFinished[Sen2] && isTaskFinished[Sen3]->
59      isTaskFinished[Sen1]=false;
60      isTaskFinished[Sen2]=false;
61      isTaskFinished[Sen3]=false;
62      CreateTask_SDA; }            //A5
63      ::atomic{isTaskFinished[SDA] ->
64      isTaskFinished[SDA]=false;
65      CreateTask_INS; }            //A6
66      ::atomic{isTaskFinished[INS] ->
67      isTaskFinished[INS]=false;
68      CreateTask_CSC;            //A4
69      CreateTask_FC; }             //A7
70      ::atomic{isTaskFinished[CSC] ->
71      isTaskFinished[CSC]=false;
72      CreateTask_Serv;           //A3    }
73      ::atomic{isTaskFinished[FC] -> FlightControlLogCount++;
74      isTaskFinished[FC]=false;
75      if     ::FlightControlLogCount>1->
76             FlightControlLogCount = 0;
77             CreateTask_DSM;          //A2
78          ::else->skip;
79      fi;}
80      ::atomic{isTaskFinished[DSM] ->
81      isTaskFinished[DSM]=false;
82      CreateTask_SSD; }              //A1
83      ::atomic{isTaskFinished[GPS] -> GPSdataCount++;
```

```
84      isTaskFinished[GPS]=false;
85      if   ::GPSdataCount>5->
86             GPSdataCount = 0;
87             CreateTask_SPS;            //A8
88        ::else->skip;
89      fi;}
90      ::atomic{isTaskFinished[SPS] ->
91      isTaskFinished[SPS]=false;
92      CreateTask_INS; }                   //A9
93      ::atomic{isTaskFinished[DR] -> LongRangeRadioDataCount++;
94      isTaskFinished[DR]=false;
95      if   ::LongRangeRadioDataCount>10->
96             LongRangeRadioDataCount = 0;
97             CreateTask_LCom;           //A10
98        ::else->skip;
99      fi;}
100     ::atomic{isTaskFinished[ODR] -> ShortRangeRadioDataCount++;
101     isTaskFinished[ODR]=false;
102     if   ::LongRangeRadioDataCount>10->
103            LongRangeRadioDataCount = 0;
104            CreateTask_SCom;           //A12
105       ::else->skip;
106     fi;}
107     ::atomic{isTaskFinished[SCom] ->
108     isTaskFinished[LCom]=false;
109     CreateTask_FC;}                 //A13
110     ::atomic{isTaskFinished[LCom] ->
111     isTaskFinished[LCom]=false;
112     CreateTask_FC;}                 //A11
113  od;  }
```

แบบจำลองระบบที่สำเร็จแล้ว (Finished system model)


```
114 //  #define debugGlobalClock
115 //  #define debugTaskExec
116 //  #define debugResource
117 //  #define debugTask
118 //  #define debugScheduler
119 mtype = {FC,INS,CSC,SDA,DSM,LCom,SCom,SPS,Serv,Sen1,Sen2,Sen3,SSD,GPS,DR,ODR,none};
120 mtype = {C_Open, C_Close, C_Execute,C_Schedule}
121 #define expired -1
122 /* GlobalClock Definition ------------------------------------------------- */
123  #define TotalTickPerLoop 50
124  #define Priority_Highest 99
125 /* GlobalClock Param and Init --------------------------------------------- */
126  int GlobalClock = 1;
127  byte GlobalClockState = C_Open;
128 /* ControlFlow Param and Init ---------------------------------------------*/
129  byte ControlFlowEventHandlerState;
130  byte activeTask;
131 /* Resource Definition ----------------------------------------------------*/
132  #define MaxResource 3
133  #define CheckMyResource ((!myTask.isreqRES0||(Resource[0].owner==_IDLE_))      &&
          (!myTask.isreqRES1||(Resource[1].owner==_IDLE_)) &&
          (!myTask.isreqRES2||(Resource[2].owner==_IDLE_)))
```

```
typedef res{
    byte owner;
};
typedef TaskList{
    bool isCreate;
    bool is_Requested;           /* Register for competition  */
    bool is_Scheduled;           /* Register for competition  */
    byte myPriority;
    byte myDeadline;
    byte myDuration;
    byte myResponse;
    byte myLax;
};
/* Resource Param and Init ------------------------------------------------------*/
 res Resource[MaxResource];
 bool ResPolling[MaxResource];
 bool PoolingResourceGuard;
 bool isTaskFinished[30];
 bool isTaskStarted[30];
 TaskList Tlist[30];
 /* Task Definition ------------------------------------------------------------*/
 typedef taskdef              /* Task definitions  */
 {
   mtype my_type;
   bool isreqRES0;            /* Required Resource */
   bool isreqRES1;            /* Required Resource */
   bool isreqRES2;            /* Required Resource */
   byte my_priority;          /* higher is better */
   bool is_Active;
   bool is_periodic;          /* Restart Every Main Cycle infinitely*/
   int phase;                /* run at which (minor) tick, 0 would be ASAP */
   int duration;             /* cycle that task have to do the job */
   int phaseCycle;           /* run at which (major) tick, 0 would be ASAP) */
   int cycle;                /* Restart Every Major Cycle for x Cycles  */
   int period;              /* Restart next Major cycle after x Cycles   */
 };
 #define isAnyActive(t)
 ((Tlist[0].isActive)||(Tlist[1].isActive)||(Tlist[2].isActive)||(Tlist[3].isActive)
||(Tlist[4].isActive)||(Tlist[5].isActive)||(Tlist[6].isActive)||(Tlist[7].isActive)
||(Tlist[8].isActive)||(Tlist[9].isActive)||(Tlist[10].isActive)
||(Tlist[11].isActive)||(Tlist[12].isActive)||(Tlist[13].isActive)
||(Tlist[14].isActive)||(Tlist[15].isActive)||(Tlist[16].isActive)
||(Tlist[17].isActive)||(Tlist[18].isActive)||(Tlist[19].isActive)
||(Tlist[20].isActive)||(Tlist[21].isActive)||(Tlist[22].isActive)
||(Tlist[23].isActive)||(Tlist[24].isActive)||(Tlist[25].isActive)
||(Tlist[26].isActive)||(Tlist[27].isActive))

 #define iMoreDeadline(t) ((( Tlist[(t)].myDeadline>Tlist[0].myDeadline)
&&(Tlist[0].is_Requested))||((Tlist[(t)].myDeadline>Tlist[1].myDeadline)
&&(Tlist[1].is_Requested))||((Tlist[(t)].myDeadline>Tlist[2].myDeadline)
&&(Tlist[2].is_Requested))||((Tlist[(t)].myDeadline>Tlist[3].myDeadline)
&&(Tlist[3].is_Requested))||((Tlist[(t)].myDeadline>Tlist[4].myDeadline)
&&(Tlist[4].is_Requested))||((Tlist[(t)].myDeadline>Tlist[5].myDeadline)
&&(Tlist[5].is_Requested))||((Tlist[(t)].myDeadline>Tlist[6].myDeadline)
&&(Tlist[6].is_Requested))myDeadline>Tlist[7].myDeadline)
```

```
189 &&(Tlist[7].is_Requested))||((Tlist[(t)].myDeadline>Tlist[8].myDeadline)
190 &&(Tlist[8].is_Requested))||((Tlist[(t)].myDeadline>Tlist[9].myDeadline)
191 &&(Tlist[9].is_Requested))||((Tlist[(t)].myDeadline>Tlist[10].myDeadline)
192 &&(Tlist[10].is_Requested))||((Tlist[(t)].myDeadline>Tlist[11].myDeadline)
193 &&(Tlist[11].is_Requested))||((Tlist[(t)].myDeadline>Tlist[12].myDeadline)
194 &&(Tlist[12].is_Requested))}||((Tlist[(t)].myDeadline>Tlist[13].myDeadline)
195 &&(Tlist[13].is_Requested))||((Tlist[(t)].myDeadline>Tlist[14].myDeadline)
196 &&(Tlist[14].is_Requested))||((Tlist[(t)].myDeadline>Tlist[15].myDeadline)
197 &&(Tlist[15].is_Requested))||((Tlist[(t)].myDeadline>Tlist[16].myDeadline)
198 &&(Tlist[16].is_Requested))||((Tlist[(t)].myDeadline>Tlist[17].myDeadline)
199 &&(Tlist[17].is_Requested))||((Tlist[(t)].myDeadline>Tlist[18].myDeadline)
200 &&(Tlist[18].is_Requested))||((Tlist[(t)].myDeadline>Tlist[19].myDeadline)
201 &&(Tlist[19].is_Requested))||((Tlist[(t)].myDeadline>Tlist[20].myDeadline)
202 &&(Tlist[20].is_Requested))||((Tlist[(t)].myDeadline>Tlist[21].myDeadline)
203 &&(Tlist[21].is_Requested))||((Tlist[(t)].myDeadline>Tlist[22].myDeadline)
204 &&(Tlist[22].is_Requested))||((Tlist[(t)].myDeadline>Tlist[23].myDeadline)
205 &&(Tlist[23].is_Requested))||((Tlist[(t)].myDeadline>Tlist[24].myDeadline)
206 &&(Tlist[24].is_Requested))||((Tlist[(t)].myDeadline>Tlist[25].myDeadline)
207 &&(Tlist[25].is_Requested))||((Tlist[(t)].myDeadline>Tlist[26].myDeadline)
208 &&(Tlist[26].is_Requested))||((Tlist[(t)].myDeadline>Tlist[27].myDeadline)
209 &&(Tlist[27].is_Requested)))
210
211  #define isLessPriority(t) ((( Tlist[(t)].myPriority<Tlist[0].myPriority)
212  &&(Tlist[0].is_Requested))||((Tlist[(t)].myPriority<Tlist[1].myPriority)
213  &&(Tlist[1].is_Requested))||((Tlist[(t)].myPriority<Tlist[2].myPriority)
214  &&(Tlist[2].is_Requested))||((Tlist[(t)].myPriority<Tlist[3].myPriority)
215  &&(Tlist[3].is_Requested))||((Tlist[(t)].myPriority<Tlist[4].myPriority)
216  &&(Tlist[4].is_Requested))||((Tlist[(t)].myPriority<Tlist[5].myPriority)
217  &&(Tlist[5].is_Requested))||((Tlist[(t)].myPriority<Tlist[6].myPriority)
218  &&(Tlist[6].is_Requested))||((Tlist[(t)].myPriority<Tlist[7].myPriority)
219  &&(Tlist[7].is_Requested))||((Tlist[(t)].myPriority<Tlist[8].myPriority)
220  &&(Tlist[8].is_Requested))||((Tlist[(t)].myPriority<Tlist[9].myPriority)
221  &&(Tlist[9].is_Requested))||((Tlist[(t)].myPriority<Tlist[10].myPriority)
222  &&(Tlist[10].is_Requested))||((Tlist[(t)].myPriority<Tlist[11].myPriority)
223  &&(Tlist[11].is_Requested))||((Tlist[(t)].myPriority<Tlist[12].myPriority)
224  &&(Tlist[12].is_Requested))||((Tlist[(t)].myPriority<Tlist[13].myPriority)
225  &&(Tlist[13].is_Requested))||((Tlist[(t)].myPriority<Tlist[14].myPriority)
226  &&(Tlist[14].is_Requested))||((Tlist[(t)].myPriority<Tlist[15].myPriority)
227  &&(Tlist[15].is_Requested))||((Tlist[(t)].myPriority<Tlist[16].myPriority)
228  &&(Tlist[16].is_Requested))||((Tlist[(t)].myPriority<Tlist[17].myPriority)
229  &&(Tlist[17].is_Requested))||((Tlist[(t)].myPriority<Tlist[18].myPriority)
230  &&(Tlist[18].is_Requested))||((Tlist[(t)].myPriority<Tlist[19].myPriority)
231  &&(Tlist[19].is_Requested))||((Tlist[(t)].myPriority<Tlist[20].myPriority)
232  &&(Tlist[20].is_Requested))||((Tlist[(t)].myPriority<Tlist[21].myPriority)
233  &&(Tlist[21].is_Requested))||((Tlist[(t)].myPriority<Tlist[22].myPriority)
234  &&(Tlist[22].is_Requested))||((Tlist[(t)].myPriority<Tlist[23].myPriority)
235  &&(Tlist[23].is_Requested))||((Tlist[(t)].myPriority<Tlist[24].myPriority)
236  &&(Tlist[24].is_Requested))||((Tlist[(t)].myPriority<Tlist[25].myPriority)
237  &&(Tlist[25].is_Requested))||((Tlist[(t)].myPriority<Tlist[26].myPriority)
238  &&(Tlist[26].is_Requested))||((Tlist[(t)].myPriority<Tlist[27].myPriority)
239  &&(Tlist[27].is_Requested)))
240
241  #define iMoreDuration(t) ((( Tlist[(t)].myDuration>Tlist[0].myDuration)
242  &&(Tlist[0].is_Requested))||((Tlist[(t)].myDuration>Tlist[1].myDuration)
243  &&(Tlist[1].is_Requested))||((Tlist[(t)].myDuration>Tlist[2].myDuration)
```

```
244 &&(Tlist[2].is_Requested))||((Tlist[(t)].myDuration>Tlist[3].myDuration)
245 &&(Tlist[3].is_Requested))||((Tlist[(t)].myDuration>Tlist[4].myDuration)
246 &&(Tlist[4].is_Requested))|||((Tlist[(t)].myDuration>Tlist[5].myDuration)
247 &&(Tlist[5].is_Requested))||((Tlist[(t)].myDuration>Tlist[6].myDuration)
248 &&(Tlist[6].is_Requested))||((Tlist[(t)].myDuration>Tlist[7].myDuration)
249 &&(Tlist[7].is_Requested))||((Tlist[(t)].myDuration>Tlist[8].myDuration)
250 &&(Tlist[8].is_Requested))||((Tlist[(t)].myDuration>Tlist[9].myDuration)
251 &&(Tlist[9].is_Requested))||((Tlist[(t)].myDuration>Tlist[10].myDuration)
252 &&(Tlist[10].is_Requested))||((Tlist[(t)].myDuration>Tlist[11].myDuration)
253 &&(Tlist[11].is_Requested))||((Tlist[(t)].myDuration>Tlist[12].myDuration)
254 &&(Tlist[12].is_Requested))||((Tlist[(t)].myDuration>Tlist[13].myDuration)
255 &&(Tlist[13].is_Requested))||((Tlist[(t)].myDuration>Tlist[14].myDuration)
256 &&(Tlist[14].is_Requested))||((Tlist[(t)].myDuration>Tlist[15].myDuration)
257 &&(Tlist[15].is_Requested))||((Tlist[(t)].myDuration>Tlist[16].myDuration)
258 &&(Tlist[16].is_Requested))||((Tlist[(t)].myDuration>Tlist[17].myDuration)
259 &&(Tlist[17].is_Requested))||((Tlist[(t)].myDuration>Tlist[18].myDuration)
260 &&(Tlist[18].is_Requested))||((Tlist[(t)].myDuration>Tlist[19].myDuration)
261 &&(Tlist[19].is_Requested))||((Tlist[(t)].myDuration>Tlist[20].myDuration)
262 &&(Tlist[20].is_Requested))||((Tlist[(t)].myDuration>Tlist[21].myDuration)
263 &&(Tlist[21].is_Requested))||((Tlist[(t)].myDuration>Tlist[22].myDuration)
264 &&(Tlist[22].is_Requested))||((Tlist[(t)].myDuration>Tlist[23].myDuration)
265 &&(Tlist[23].is_Requested))||((Tlist[(t)].myDuration>Tlist[24].myDuration)
266 &&(Tlist[24].is_Requested))||((Tlist[(t)].myDuration>Tlist[25].myDuration)
267 &&(Tlist[25].is_Requested))||((Tlist[(t)].myDuration>Tlist[26].myDuration)
268 &&(Tlist[26].is_Requested))||((Tlist[(t)].myDuration>Tlist[27].myDuration)
269 &&(Tlist[27].is_Requested)))
270
271 #define isLessPriority2(t) ((( Tlist[(t)].myPriority<Tlist[0].myPriority))
272           ||((Tlist[(t)].myPriority<Tlist[1].myPriority))    \
273           ||((Tlist[(t)].myPriority<Tlist[2].myPriority))    \
274           ||((Tlist[(t)].myPriority<Tlist[3].myPriority))    \
275           ||((Tlist[(t)].myPriority<Tlist[4].myPriority))    \
276           ||((Tlist[(t)].myPriority<Tlist[5].myPriority))    \
277           ||((Tlist[(t)].myPriority<Tlist[6].myPriority))    \
278           ||((Tlist[(t)].myPriority<Tlist[7].myPriority))    \
279           ||((Tlist[(t)].myPriority<Tlist[8].myPriority))    \
280           ||((Tlist[(t)].myPriority<Tlist[9].myPriority))    \
281           ||((Tlist[(t)].myPriority<Tlist[10].myPriority))   \
282           ||((Tlist[(t)].myPriority<Tlist[11].myPriority))   \
283           ||((Tlist[(t)].myPriority<Tlist[12].myPriority))   \
284           ||((Tlist[(t)].myPriority<Tlist[13].myPriority))   \
285           ||((Tlist[(t)].myPriority<Tlist[14].myPriority))   \
286           ||((Tlist[(t)].myPriority<Tlist[15].myPriority))   \
287           ||((Tlist[(t)].myPriority<Tlist[16].myPriority))   \
288           ||((Tlist[(t)].myPriority<Tlist[17].myPriority))   \
289           ||((Tlist[(t)].myPriority<Tlist[18].myPriority))   \
290           ||((Tlist[(t)].myPriority<Tlist[19].myPriority))   \
291           ||((Tlist[(t)].myPriority<Tlist[20].myPriority))   \
292           ||((Tlist[(t)].myPriority<Tlist[21].myPriority))   \
293           ||((Tlist[(t)].myPriority<Tlist[22].myPriority))   \
294           ||((Tlist[(t)].myPriority<Tlist[23].myPriority))   \
295           ||((Tlist[(t)].myPriority<Tlist[24].myPriority))   \
296           ||((Tlist[(t)].myPriority<Tlist[25].myPriority))   \
297           ||((Tlist[(t)].myPriority<Tlist[26].myPriority))   \
298           ||((Tlist[(t)].myPriority<Tlist[27].myPriority)))
```

```
299
300 #define iMoreLax(t)  \
301  ((( Tlist[(t)].myDeadline>Tlist[0].myDeadline)&&(Tlist[0].is_Requested))
302        ||((Tlist[(t)].myLax>Tlist[1].myLax)&&(Tlist[1].is_Requested))          \
303        ||((Tlist[(t)].myLax>Tlist[2].myLax)&&(Tlist[2].is_Requested))          \
304        ||((Tlist[(t)].myLax>Tlist[3].myLax)&&(Tlist[3].is_Requested))          \
305        ||((Tlist[(t)].myLax>Tlist[4].myLax)&&(Tlist[4].is_Requested))          \
306        ||((Tlist[(t)].myLax>Tlist[5].myLax)&&(Tlist[5].is_Requested))          \
307        ||((Tlist[(t)].myLax>Tlist[6].myLax)&&(Tlist[6].is_Requested))          \
308        ||((Tlist[(t)].myLax>Tlist[7].myLax)&&(Tlist[7].is_Requested))          \
309        ||((Tlist[(t)].myLax>Tlist[8].myLax)&&(Tlist[8].is_Requested))          \
310        ||((Tlist[(t)].myLax>Tlist[9].myLax)&&(Tlist[9].is_Requested))          \
311        ||((Tlist[(t)].myLax>Tlist[10].myLax)&&(Tlist[10].is_Requested)) \
312        ||((Tlist[(t)].myLax>Tlist[11].myLax)&&(Tlist[11].is_Requested)) \
313        ||((Tlist[(t)].myLax>Tlist[12].myLax)&&(Tlist[12].is_Requested)) \
314        ||((Tlist[(t)].myLax>Tlist[13].myLax)&&(Tlist[13].is_Requested)) \
315        ||((Tlist[(t)].myLax>Tlist[14].myLax)&&(Tlist[14].is_Requested)) \
316        ||((Tlist[(t)].myLax>Tlist[15].myLax)&&(Tlist[15].is_Requested)) \
317        ||((Tlist[(t)].myLax>Tlist[16].myLax)&&(Tlist[16].is_Requested)) \
318        ||((Tlist[(t)].myLax>Tlist[17].myLax)&&(Tlist[17].is_Requested)) \
319        ||((Tlist[(t)].myLax>Tlist[18].myLax)&&(Tlist[18].is_Requested)) \
320        ||((Tlist[(t)].myLax>Tlist[19].myLax)&&(Tlist[19].is_Requested)) \
321        ||((Tlist[(t)].myLax>Tlist[20].myLax)&&(Tlist[20].is_Requested)) \
322        ||((Tlist[(t)].myLax>Tlist[21].myLax)&&(Tlist[21].is_Requested)) \
323        ||((Tlist[(t)].myLax>Tlist[22].myLax)&&(Tlist[22].is_Requested)) \
324        ||((Tlist[(t)].myLax>Tlist[23].myLax)&&(Tlist[23].is_Requested)) \
325        ||((Tlist[(t)].myLax>Tlist[24].myLax)&&(Tlist[24].is_Requested)) \
326        ||((Tlist[(t)].myLax>Tlist[25].myLax)&&(Tlist[25].is_Requested)) \
327        ||((Tlist[(t)].myLax>Tlist[26].myLax)&&(Tlist[26].is_Requested)) \
328        ||((Tlist[(t)].myLax>Tlist[27].myLax)&&(Tlist[27].is_Requested)))
329
330 mtype = {ResPol,ResAck,_IDLE_,TASK_A,TASK_B,TASK_C,TASK_D,TASK_E};
331// p = parentheses, s = square
332 #define MTL_Fpp(effect) <>(effect.Reaction&&(effect.EventCounter>effect.pTime1)
333  &&(effect.EventCounter<effect.pTime2))
334 #define MTL_Fss(effect) <>(effect.Reaction&&(effect.EventCounter>=effect.pTime1)
335  &&(effect.EventCounter<=effect.pTime2))
336 #define MTL_Fps(effect) <>(effect.Reaction&&(effect.EventCounter>effect.pTime1)
337  &&(effect.EventCounter<=effect.pTime2))
338 #define MTL_Fsp(effect) <>(effect.Reaction&&(effect.EventCounter>=effect.pTime1)
339  &&(effect.EventCounter<effect.pTime2))
340
341 #define MTL_Gpp(effect)
342  [](!((effect.EventCounter>effect.pTime1)&&(effect.EventCounter<effect.pTime2))
343  ||((effect.Reaction&&(effect.EventCounter>effect.pTime1))
344  &&(effect.EventCounter<effect.pTime2)))
345 #define MTL_Gss(effect) [](!((effect.EventCounter>=effect.pTime1)
346  &&(effect.EventCounter<=effect.pTime2))
347  ||((effect.Reaction&&(effect.EventCounter>=effect.pTime1))
348  &&(effect.EventCounter<=effect.pTime2)))
349 #define MTL_Gps(effect) [](!((effect.EventCounter>effect.pTime1)
350  &&(effect.EventCounter<=effect.pTime2))
351  ||((effect.Reaction&&(effect.EventCounter>effect.pTime1))
352  &&(effect.EventCounter<=effect.pTime2)))
353 #define MTL_Gsp(effect) [](!((effect.EventCounter>=effect.pTime1)
```

```
354 &&(effect.EventCounter<effect.pTime2))
355 ||((effect.Reaction&&(effect.EventCounter>=effect.pTime1))
356 &&(effect.EventCounter<effect.pTime2)))
357
358/* Task Param and Init --------------------------------*/
359// Task Process  ------------------------------------------------------------------
360//#define isMyPhase ((myTask.phase-1 == GlobalClock)||(myTask.phase==0))
361//#define isMyPhaseCycle ((myTask.phaseCycle == GlobalMajorClock)
362  ||((GlobalMajorClock-myTask.phaseCycle)%myTask.period == 0))
363#define isMyPhase (((GlobalClock-(myTask.phase-1))%myTask.period==0)
364  ||(myTask.phase==0))
365
366byte Selected = none;
367byte Selected_MTL = none;
368inline Scheduler_EDF(TaskType){
369 atomic{     if
370           :: (!iMoreDeadline(TaskType))-> Selected = myTask.my_type;
371                                    #ifdef debugScheduler
372                                    printf("%e is less deadline\n",TaskType);
373                                    #endif
374                                    :: else->skip;
375         fi;  }
376}
377
378inline Scheduler_DM(TaskType){
379 atomic{   if
380           :: (!iMoreDuration(TaskType))-> Selected = myTask.my_type;
381                                      #ifdef debugScheduler
382                                    printf("%e is less duration\n",TaskType);
383                                      #endif
384                                     :: else->skip;
385         fi;  }
386}
387 bool HighestGetResource=false;
388 proctype CreateTask(taskdef myTask){
389  bool temp;
390  int cycleCount = 0;
391  int durationCount = 0;          /* Duration counter for task */
392  bool isTaskActive = false;      /* Guarding for this task if not Active yet*/
393  mtype RES[MaxResource];         /* Resouce available from Global Clock broadcast */
394  if
395    ::Tlist[myTask.my_type].isCreate == true; goto exitWOclose;
396    ::else->Tlist[myTask.my_type].isCreate = true;
397  fi;
398  waitNextCycle:             /* When finish every tick, task would wait here */
399    do
400
401    ::  atomic{((GlobalClockState==C_Open)
402      &&(isMyPhase)
403      &&(!isTaskActive))
404      -> isTaskActive=  true;
405        #ifdef debugTask
406          printf("%e is active\n",myTask.my_type);
407        #endif
408      }
```

```
409
410     ::  atomic{((GlobalClockState == C_Open)
411       &&(isTaskActive)
412       &&(!Tlist[myTask.my_type].is_Requested))
413       -> Tlist[myTask.my_type].is_Requested = true;
414         #ifdef debugTask
415           printf("%e Requested to List\n",myTask.my_type);
416         #endif
417         Tlist[myTask.my_type].myDeadline = myTask.period-
418             ((GlobalClock-(myTask.phase-1))%myTask.period);
419         Tlist[myTask.my_type].myDuration = myTask.duration;
420         Tlist[myTask.my_type].myResponse = myTask.period - myTask.duration;
421         }
422     ::  atomic{((GlobalClockState == C_Schedule)
423       &&(isTaskActive)
424       &&(!Tlist[myTask.my_type].is_Scheduled))->
425           //Scheduler_DM(myTask.my_type);
426           Scheduler_EDF(myTask.my_type);
427           Tlist[myTask.my_type].is_Scheduled = true;
428       }
429
430     ::  atomic{((GlobalClockState == C_Close)
431       &&(isTaskActive)
432       &&(!isLessPriority(myTask.my_type))
433       &&(Selected == none)
434       &&(Tlist[myTask.my_type].is_Requested))->
435         #ifdef debugTask
436           printf("%e is Selected \n",myTask.my_type);
437         #endif
438           Selected = myTask.my_type;
439       }
440
441     ::  atomic{((GlobalClockState == C_Execute)
442       &&(isTaskActive)
443       &&(Tlist[myTask.my_type].is_Requested)
444       &&(Selected == myTask.my_type))->
445           Tlist[myTask.my_type].is_Requested = false;
446           Tlist[myTask.my_type].is_Scheduled = false;
447
448             if::(Resource[0].owner==_IDLE_
449                  && myTask.isreqRES0)->Resource[0].owner=myTask.my_type;
450                :: else;
451             fi;
452             if::(Resource[1].owner==_IDLE_
453                   && myTask.isreqRES1)->Resource[1].owner=myTask.my_type;
454             :: else;
455             fi;
456             if::(Resource[2].owner==_IDLE_
457                 && myTask.isreqRES2)->Resource[2].owner=myTask.my_type;
458                :: else;
459             fi;
460             if  ::(durationCount == 0) -> isTaskStarted[myTask.my_type]=true;
461               ::else->isTaskStarted[myTask.my_type]=false;
462             fi;
463             durationCount++;
```

```
464         if:: (durationCount >= myTask.duration)->durationCount=0->
465                HighestGetResource = true;
466                GlobalClockState == C_Open;
467           goto TaskIsFinished;
468           :: else->skip;
469         fi;
470         HighestGetResource = true;
471         GlobalClockState == C_Open;
472       }
473     ::  atomic{((GlobalClockState == C_Execute)   /* For lower priority task */
474       &&(isTaskActive)
475       &&(Tlist[myTask.my_type].is_Requested)
476       &&(HighestGetResource)
477       &&(CheckMyResource))->
478         Tlist[myTask.my_type].is_Requested = false;
479         Tlist[myTask.my_type].is_Scheduled = false;
480              if::(Resource[0].owner==_IDLE_ && myTask.isreqRES0)
481                     ->Resource[0].owner=myTask.my_type;
482                :: else;
483              fi;
484              if::(Resource[1].owner==_IDLE_ && myTask.isreqRES1)
485                     ->Resource[1].owner=myTask.my_type;
486                :: else;
487              fi;
488              if::(Resource[2].owner==_IDLE_ && myTask.isreqRES2)
489                      ->Resource[2].owner=myTask.my_type;
490                :: else;
491              fi;
492              if::(durationCount == 0) -> isTaskStarted[myTask.my_type]=true;
493
494                ::else->isTaskStarted[myTask.my_type]=false;
495              fi;
496              durationCount++;
497         if   :: (durationCount >= myTask.duration)->durationCount=0
498              ->goto TaskIsFinished;
499            :: else->skip;
500         fi;
501         GlobalClockState == C_Open;
502       }
503     ::  atomic{((GlobalClockState == C_Execute)
504       &&(isTaskActive)
505       &&(Tlist[myTask.my_type].is_Requested)
506       &&(HighestGetResource)
507       &&(!CheckMyResource)
508       &&(Selected != myTask.my_type))->
509              #ifdef debugTask
510                printf("%e resetting\n",myTask.my_type);
511              #endif
512         Tlist[myTask.my_type].is_Scheduled =false;
513         Tlist[myTask.my_type].is_Requested= false;
514         GlobalClockState == C_Open;
515       }
516  od;
517 TaskIsFinished:      atomic{isTaskFinished[myTask.my_type] = true;
518                           durationCount = 0;
```

```
519                          isTaskActive = false;
520      if::   (myTask.is_periodic)->cycleCount=0;
521                            goto waitNextCycle; /* Run infinitely eventually */
522                            ::   ((cycleCount+1) < myTask.cycle)
523   -> cycleCount++;printf("cycle count %d\n",cycleCount);  goto waitNextCycle;
524                            ::   else->cycleCount=0;
525                          fi;
526                    }
527  Tlist[myTask.my_type].isCreate = false;
528  exitWOclose:skip;
529 }
530
531 bool pEventFlag = false;
532 mtype = { Event_p, Event_q, Event_r }
533 typedef MTLevent
534  {
535    bool   Event;
536    bool Reaction;
537    bool   EventFlag;
538    mtype  name;
539    int    EventCounter;
540    int  pTime1;
541    int  pTime2;
542  };
543
544 //#define MTL_F(effect) <>(effect.Reaction&&(effect.EventCounter>=effect.pTime1)
545 &&(effect.EventCounter<=effect.pTime2))
546 //#define MTL_G(effect) [](!((effect.EventCounter>=effect.pTime1)
547 &&(effect.EventCounter<=effect.pTime2))||((effect.Reaction
548 &&(effect.EventCounter>=effect.pTime1))&&(effect.EventCounter<=effect.pTime2)))
549
550 inline MTLCounter(CauseEvent)
551 {
552           if
553           :: atomic{(CauseEvent.Event) -> printf("TriggerEvent Occured\n");
554                                      if
555                                      ::!CauseEvent.EventFlag
556                                             ->CauseEvent.EventCounter = 0;
557                                          CauseEvent.EventFlag = true;
                                           ::else->skip;
558                                    fi;
559                        }
560           :: atomic{(!CauseEvent.Event&&CauseEvent.EventFlag)
561                -> if ::( (CauseEvent.EventCounter<=CauseEvent.pTime2)
562                && (CauseEvent.EventCounter!=expired))-> CauseEvent.EventCounter++;
563                If   :: CauseEvent.Reaction->printf("%e Reaction at count = %d \n"
564                        ,CauseEvent.name,CauseEvent.EventCounter);
565                     :: !CauseEvent.Reaction->printf("%e Count = %d \n"
566                        ,CauseEvent.name,CauseEvent.EventCounter);
567             fi;
568                  ::   else->printf("%e Counter = expired > %d \n"
569                         ,CauseEvent.name,CauseEvent.EventCounter);
570                          CauseEvent.EventCounter=expired;
571               CauseEvent.EventFlag = false;
572               fi;
```

```
573                         }
574           ::else->skip;
575         fi;
576 }
577 #define ExecuteTime (timeout && GlobalClockState == C_Execute)
578
579 #define Sen3TC (ExecuteTime)&&(GlobalClock ==4)
580 #define Sen3RC (Resource[1].owner==Sen3)
581
582
583 #define isCreated(task) Tlist[task].isCreate
584
585 #define FC1TC (isCreated(FC)&&(ExecuteTime && (GlobalClock >=1)&&(GlobalClock <=2)))
586 #define FC1RC (Resource[0].owner==FC)
587
588 #define Sen1TC (isCreated(Sen1)&&(ExecuteTime
589   && (GlobalClock >=1)&&(GlobalClock <=2)))
590 #define Sen1RC (Resource[1].owner==Sen1)
591
592 #define Sen2TC (isCreated(Sen2)&&(ExecuteTime
593  && (GlobalClock >=1)&&(GlobalClock <=2)))
594 #define Sen2RC (Resource[1].owner==Sen2)
595 #define Sen1Event  (isTaskFinished[Sen1])
596 #define ServEvent (isTaskStarted[Serv])
597 #define FCEvent  (isTaskStarted[FC])
598 #define SSDEvent (isTaskStarted[SSD])
599 #define LongCommEvent (isTaskFinished[DR])
600 #define SesnEvent isTaskStarted[Sen1] && isTaskStarted[Sen2] && isTaskStarted[Sen3]
601 MTLevent Event0;
602 MTLevent Event1;
603 MTLevent Event2;
604 mtype {NormalFlight_Constraint,RecordON_Constraint, LongCommON_Constraint}
605 #define NormalFlight [](Event0.Event->MTL_Fss(Event0))
606 #define RecordON [](Event1.Event->MTL_Fss(Event1))
607 #define  LongCommON [](Event2.Event->MTL_Fss(Event2))
608
609 ltl ltl001 {[](NormalFlight&&RecordON&&LongCommON)}
610 inline MTLEngine()
611 {
612     Selected_MTL = Selected; // do not change
613     // Set up for Event 0
614     Event0.name = NormalFlight_Constraint;
615     Event0.Event = Sen1Event;
616     Event0.Reaction = ServEvent;
617     Event0.pTime1 = 5;
618     Event0.pTime2 = 20;
619     MTLCounter(Event0);
620     // Set up for Event 1
621     Event1.name = RecordON_Constraint;
622     Event1.Event = FCEvent;
623     Event1.Reaction = SSDEvent;
624     Event1.pTime1 = 0;
625     Event1.pTime2 =10;
626     MTLCounter(Event1);
627
```

```
628    // Set up for Event 2
629    Event2.name = LongCommON_Constraint;
630    Event2.Event =  LongCommEvent;
631    Event2.Reaction = FCEvent;
632    Event2.pTime1 = 0;
633    Event2.pTime2 = 10;
634    MTLCounter(Event2);
635 }
636 // End Sandbox
637 // COpen Process ------------------------------------------------------------------------
638 proctype Ticking()
639 {
640
641  do
642
643    :: atomic {((GlobalClockState == C_Open) && timeout)
644       -> GlobalClockState=C_Schedule;
645             #ifdef debugTaskExec
646               printf("------------------Global Clock = %d %e\n"
647                  ,GlobalClock,GlobalClockState);
648               printf ("== TICK :%d CPU[%e] SPI[%e] I2C[%e] \n"
649              ,GlobalClock,Resource[0].owner,Resource[1].owner,Resource[2].owner);
650             #endif
651          }
652    :: atomic {((GlobalClockState == C_Schedule)     && timeout)
653      -> GlobalClockState = C_Close;
654             #ifdef debugTaskExec
655               printf("------------------Global Clock = %d %e\n"
656                  ,GlobalClock,GlobalClockState);
657               printf ("== TICK :%d CPU[%e] SPI[%e] I2C[%e] \n"
658              ,GlobalClock,Resource[0].owner,Resource[1].owner,Resource[2].owner)
659          #endif
660          }
661  :: atomic {((GlobalClockState == C_Close) && timeout)-> GlobalClockState= C_Execute;
662  #ifdef debugTaskExec
663  printf("------------------Global Clock = %d %e\n",GlobalClock,GlobalClockState);
664  printf ("== TICK :%d CPU[%e] SPI[%e] I2C[%e] \n"
665 ,GlobalClock,Resource[0].owner,Resource[1].owner,Resource[2].owner);
666  #endif
667  }
668    :: atomic {((GlobalClockState == C_Execute)    && timeout)
669          ->MTLEngine();
670             Selected = none;
671             HighestGetResource = false;
672             #ifdef debugTaskExec
673  printf("------------------Global Clock = %d %e\n",GlobalClock,GlobalClockState);
674             #endif
675             printf ("== TICK :%d CPU[%e] SPI[%e] I2C[%e] \n"
676       ,GlobalClock,Resource[0].owner,Resource[1].owner,Resource[2].owner);
677             GlobalClock++;
678             if
679             :: (GlobalClock > (TotalTickPerLoop)) -> GlobalClock = 1;
680             printf("-----------------------------------------------------------\n")
681             :: else;
682             fi;
```

```
683              GlobalClockState = C_Open;
684              Resource[0].owner = _IDLE_;
685              Resource[1].owner = _IDLE_;
686              Resource[2].owner = _IDLE_;
687              #ifdef debugTaskExec
688                printf("------------------Global Clock = %d %e\n"
689                   ,GlobalClock,GlobalClockState);
690                printf ("== TICK :%d CPU[%e] SPI[%e] I2C[%e] \n"
691                 ,GlobalClock,Resource[0].owner,Resource[1].owner,Resource[2].owner);
692              #endif
693                GlobalClockState = C_Open;
694              }
695  od;
696 }
697
698 // End  Process ---------------------------------------------------------------
699
700 inline Create(Taskname,R1, R2, R3, isPer ,dura, pha, pri, cy, per, pCy)
701 {
702  taskdef taskTemplete;
703  atomic{
704
705     taskTemplete.my_type = Taskname;
706     taskTemplete.isreqRES0 = R1;
707     taskTemplete.isreqRES1 = R2;
708     taskTemplete.isreqRES2 = R3;
709     taskTemplete.duration = dura;
710     taskTemplete.is_periodic = isPer;
711     taskTemplete.phase = pha+1;          /* to handle phase -1 */
712     taskTemplete.cycle = cy;
713     taskTemplete.period = per;
714     taskTemplete.phaseCycle = pCy;
715     taskTemplete.my_priority = pri;
716     Tlist[Taskname].myPriority=pri;
717  }
718  run CreateTask(taskTemplete);// priority pri;
719 }
720 int FlashCount = 0;
721 int HRDisplay = 0;
722 int FlightControlLogCount = 0;
723 int GPSdataCount = 0;
724 int LongRangeRadioDataCount = 0;
725 int ShortRangeRadioDataCount = 0;
726 init
727 { atomic {
728     Resource[0].owner = _IDLE_;
729     Resource[1].owner = _IDLE_;
730     Resource[2].owner = _IDLE_;
731     CreateTask_Sen1;
732     CreateTask_Sen2;
733     CreateTask_Sen3;
734     CreateTask_GPS;
735     CreateTask_DR;
736  }  run Ticking();
737 }
```

ประวัติผู้เขียนวิทยานิพนธ์

นายพันธ์เวสส์ สุขวนิช สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยมหิดลในปีการศึกษา 2547 และเมื่อปีการศึกษา 2555 เข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ประวัติการทำงานในอดีตมีความเกี่ยวข้องกับการออกแบบระบบเวลาจริงที่เกี่ยวข้องกับวิกฤติความปลอดภัยมาโดยตลอด เคยทำหน้าที่เป็นนักวิจัยแผนกควบคุมการยิงและอาวุธนำวิถี สถาบันเทคโนโลยีป้องกันประเทศ กระทรวงกลาโหม มีประสบการณ์การทำงานเกี่ยวข้องกับระบบควบคุมการบินอากาศยานไร้คนขับ และ ระบบอาวุธปล่อยนำวิถีพื้นสู่อากาศ ในปัจจุบันดำรงตำแหน่งหัวหน้าแผนกวิศวกรรม บริษัท เวลโลกราฟ จำกัด รับผิดชอบการออกแบบระบบเวลาจริงสำหรับเครื่องมือและอุปกรณ์ทางการแพทย์ ตลอดระยะเวลาการทำงานที่ผ่านมา ผลงานที่สร้างได้รับรางวัลทั้งระดับชาติ เช่น รางวัลชนะเลิศรางวัลนวัตกรรมแห่งชาติปี พ.ศ. 2558 และรางวัลรองชนะเลิศอันดับ 1 รางวัลนวัตกรรมแห่งชาติในปี พ.ศ. 2550 โดยสำนักงานนวัตกรรมแห่งชาติ กระทรวงวิทยาศาสตร์ และนานาชาติเช่น รางวัลนวัตกรรมด้านการออกแบบจากซีอีเอส (CES Innovation Award 2014) ประเทศสหรัฐอเมริกา เป็นต้น