การควบคุมที่เวลาจริง

MATLAB REALTIME WORKSHOP

นายพชร วรพจน์

นายพันธุ์พิศิษฐ์ ภูแซมโชติ

นายเรืองสกุล ศิริเภตรา

นางสาววริสรา เบ้าทอง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2553

MATLAB REALTIME WORKSHOP

PACHARA WORRAPOJ

PHANPHISIT PHOOSAMCHOT

RUANGSAKUL SIRIPETRA

WARISARA BAOTHONG

THIS THESIS IS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN CONTROL ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2010

ปริญญานิพนธ์ปีการศึกษา 2553

สาขาวิชาวิศวกรรมการวัดและควบคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมที่เวลาจริง

MATLAB REALTIME WORKSHOP

ผู้จัดทำ

นายพชร	วรพจน์	50011025
นายพันธุ์พิศิษฐ์	ภูแซมโชติ	50011088
นายเรืองสกุล	ศิริเภตรา	50011324
นางสาววริสรา	เบ้าทอง	50011401

.....อาจารย์ที่ปรึกษา

(รศ.สุเธียร เกียรติสุนทร)

การควบคุมที่เวลาจริง

โดย

นายพชร	วรพจน์	50011025
นายพันธุ์พิศิษฐ์	ภูแซมโชติ	50011088
นายเรืองสกุล	ศิริเภตรา	50011324
นางสาววริสรา	เบ้าทอง	50011401

อาจารย์ที่ปรึกษา

รศ.สุเธียร เกียรติสุนทร

บทคัดย่อ

โครงงานนี้เรียบเรียงขึ้น เพื่อจุดประสงค์ที่จะเพิ่มประสิทธิภาพให้กับฟังก์ชันของซิมูลิงค์ในโปรแกรมแมทแลป เนื่องจากซิมูลิงค์เป็นฟังก์ชันที่สามารถวิเคราะห์ได้เพียงระบบจำลองเท่านั้น เราจึงนำมาพัฒนาให้สามารถควบคุมการทำงานที่เวลาจริงได้ โดยฟังก์ชันของซิมูลิงค์จะสามารถรับและส่งค่าตัวแปรต่างๆ ให้กับระบบจริงหรืออุปกรณ์ภายนอกผ่านทางพอร์ตอนุกรม ซึ่งพอร์ตอนุกรมทำหน้าที่รับและส่งข้อมูลระหว่างซิมูลิงค์กับระบบจริงหรืออุปกรณ์ภายนอก

MATLAB REALTIME WORKSHOP

By

Pachara Worrapoj

Phanphisit Phoosamchot

Ruangsakul Siripetra

Warisara Baothong

Advisor

Assoc.Prof. Suthian Kiatsunthon

ABSTRACT

The main purpose of this study is getting more efficiency of Simulink function in Matlab which is analyzing the simulators. The high efficiency simulator can analyze Real System using the Matlab to connect the Simulink and the Real System can transfer variable each other and connect interface devices by serial port in order to connect to computer. It is received the data which is conversion from analog to digital and digital to analog.

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์ฉบับนี้ สามารถสำเร็จลุล่วงไปได้ด้วยดี เพราะได้รับความช่วยเหลือเป็นอย่างดี จากอาจารย์สุเธียร เกียรติสุนทร กรุณาให้คำปรึกษาแนะนำที่ดีมาโดยตลอดตั้งแต่ต้นรวมทั้งให้ความช่วยเหลืออื่นๆที่เป็นประโยชน์ต่อโครงงาน ผู้จัดทำรู้สึกซาบซึ้งและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณพี่ๆ ทุกคนที่คอยให้ความช่วยเหลือ และให้คำแนะนำดีๆ มากมายโดยเฉพาะพี่สมพบ แซ่เจี่ย และพี่ธนศักดิ์ โสพส

ขอบคุณเพื่อนๆ และน้องๆ ทุกคน ที่ให้กำลังใจ กระตุ้นเตือน รวมทั้งคอยถามไถ่ความคืบหน้าของโครงงานอยู่เสมอ

สุดท้ายนี้ผู้จัดทำขอกราบขอบพระคุณบิดา มารดา และครอบครัว ที่คอยเป็นกำลังใจที่ดีตลอดมา รวมถึงการสนับสนุนในเรื่องของงบประมาณที่ขาดเหลือ ตลอดจนเป็นแรงบันดาลใจที่ดีที่สุดที่ทำให้โครงงานนี้สำเร็จสมบูรณ์ลงได้

ผู้จัดทำ

นายพชร วรพจน์

นายพันธุ์พิศิษฐ์ ภูแซมโชติ

นายเรืองสกุล ศิริเภตรา

นางสาววริสรา เบ้าทอง

สารบัญ

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

เป็นที่ทราบกันดีว่าในปัจจุบันโปรแกรม แมทแลป เป็นโปรแกรมที่สามารถพัฒนาวิเคราะห์และออกแบบระบบควบคุมได้อย่างมีประสิทธิภาพ และเป็นที่นิยมใช้กันอย่างแพร่หลาย โดยเฉพาะการใช้ซิมูลิงค์ในการออกแบบ วิเคราะห์ระบบเพราะทำได้ง่าย ด้วยการสร้างโมเดลขึ้นมาเป็นบล็อกโดยใช้การปฏิบัติการณ์คลิกและลากเมาส์ (Click and Drag Mouse Operation) แต่เป็นที่น่าเสียดายที่สามารถทำการวิเคราะห์ได้เพียงระบบจำลองเท่านั้น (Simulated System) เท่านั้น ซึ่งหมายความว่าผู้ใช้ต้องทำการจำลองระบบจริงก่อนทำการวิเคราะห์ ซึ่งอาจทำให้เกิดความคลาดเคลื่อนของตัวแปรต่างๆ โดยเฉพาะระบบที่มีความละเอียดอ่อนสูงหรือระบบที่ตัวแปรมีการเปลี่ยนแปลงค่าตลอดเวลา อาจไม่สามารถทำการวิเคราะห์ได้ แต่ถ้าเราสามารถเชื่อมโยงให้ซิมูลิงค์ ติดต่อกับระบบจริงได้โดยตรงที่เวลาใกล้เคียงความจริง ก็สามารถนำตัวแปรต่างๆ ที่เกิดขึ้นจริงในเวลานั้นเข้ามาทำการวิเคราะห์ได้ทันทีเป็นการเพิ่มประสิทธิภาพของซิมูลิงค์ในโปรแกรมแมทแลปให้สูงขึ้น

ในการออกแบบระบบควบคุมระบบหนึ่งนั้น ยกตัวอย่างเช่นระบบควบคุมมอเตอร์ (Motor Control) ทำได้ช้ามาก เพราะจะต้องเสียเวลาในการออกแบบตัวควบคุม โดยการจำลองระบบที่ต้องการควบคุมลงไปในซิมูลิงค์ก่อนจึงจะทำการออกแบบตัวควบคุมได้ และ เมื่อได้ตัวควบคุมที่เป็นที่พอใจแล้วก็ต้องนำมาออกแบบวงจรควบคุมก่อน จึงจะสามารถนำไปต่อทดลองกับระบบจริงได้ ซึ่งจะต้องทำกลับไปกลับมาทำให้เสียเวลาเป็นอย่างมาก ดังนั้นเราจึงควรนำเอาเอาท์พุตจากระบบจริงเข้ามาคำนวณ ผลในซิมูลิงค์โดยตรงจึงจะให้ตัวควบคุมที่ถูกต้องกว่า แต่ แมทแลปไม่สามารถติดต่อกับระบบภายนอกได้ด้วยตัวเอง เราจึงจำเป็นต้องสร้างซีเมคเอสฟังก์ชั่นบล็อกขึ้นมา เพื่อเป็นตัวช่วย ในการส่งผ่านตัวแปรต่างๆ เข้ามาที่ซิมูลิงค์ ของโปรแกรมแมทแลปได้ทันที

1.2 วัตถุประสงค์ของโครงงาน

1) ศึกษาการจำลองระบบทั้งแบบซิมูเลชันและแบบเรียลไทม์

2) ศึกษาการเขียนโปรแกรมในรูปแบบ ซีเมคเอสฟังก์ชัน

3) เพื่อให้เข้าใจขั้นตอนการติดต่อกับระบบภายนอกโดยใช้พอร์ตอนุกรม

1.3 ขอบเขตของโครงงาน

สร้างบล็อกไดอะแกรมบนหน้าต่างซิมูลิงค์ในแบบเรียลไทม์ และรับส่งค่าสัญญาณจากอุปกรณ์ภายนอกผ่านทางพอร์ตอนุกรมได้

1.4 วิธีการดำเนินงาน

1) ซอฟแวร์

- ศึกษาการใช้งานโปรแกรมMATLAB, Simulink, การเขียนC-MEX S-Funtion
- โปรแกรมการทำงานของARM7 LPC2148 ด้วย โปรแกรม Keil uVision

2) ฮาร์ดแวร์

- ศึกษาการเชื่อมต่อของวงจรของบอร์ด JX-2148 เพื่อนำมาใช้งาน

บทที่ 2
ทฤษฎีและหลักการ

2.1 โปรแกรมแมทแลป (MATLAB)

โปรแกรมที่มีประสิทธิภาพสูงในการคำนวณด้านเทคนิค เป็นการรวมกันของการคำนวณ การมองเห็นได้และการเขียนโปรแกรม ในรูปแบบที่ง่ายต่อการใช้ โดยที่ปัญหาและการแก้ปัญหาถูกแสดงในรูปแบบทางคณิตศาสตร์ ลักษณะการใช้ทั่วไปของแมทแลป ได้แก่

- คณิตศาสตร์ และการคำนวณ
- การพัฒนาทางด้านอัลกอลิทึม
- การจำลองระบบ และการสร้างโมเดล
- การวิเคราะห์ข้อมูล การสำรวจ และการแสดงระบบโดยใช้ภาพ
- กราฟฟิคทางด้านวิทยาศาสตร์และทางด้านวิศวกรรม
- การพัฒนาและการสร้างการติดต่อกับผู้ใช้ในแบบกราฟฟิค

แมทแลปเป็นระบบปฏิสัมพันธ์ (interactive system) ที่ส่วนประกอบของข้อมูลพื้นฐานเป็นแบบอาร์เรย์ซึ่งไม่มีมิติ ทำให้สามารถแก้ปัญหาการคำนวณทางด้านเทคนิค โดยเฉพาะการคำนวณโดยใช้สูตรในทางเมตริกซ์และเวคเตอร์ ซึ่งต้องใช้การเขียนโปรแกรมในรูปแบบภาษา สเกลาร์นอนอินเทอร์เรคทีฟ (scalar noninteractive) เช่น ภาษาซีและ ฟอร์แทรน

แมทแลป จะมีชุดของคำสั่งเฉพาะที่ใช้ในการแก้ปัญหาเรียกว่า ทูลบล็อก (toolboxes) ซึ่งมีความสำคัญอย่างมากต่อผู้ใช้ ซึ่งผู้ใช้จะได้รับเทคนิคจากการเรียนรู้และพัฒนาทูลบล็อกจะรวบรวมเอ็มไฟล์ ซึ่งเป็นตัวช่วยให้แมทแลปสามารถแก้ปัญหาที่หลากหลายได้ เช่น การประมวลสัญญาณ (signal processing), การควบคุมระบบ (control system), ฟัซซี่ ลอจิก (fuzzy logic), เวฟเลท (wave lets), การจำลองระบบ (simulation) ฯลฯ

2.2 ซิมูลิงค์ (Simulink)

ซิมูลิงค์เป็นซอฟต์แวร์ ที่ใช้สำหรับการสร้างโมเดล, การจำลอง และการวิเคราะห์ระบบแบบไดนามิค ซึ่งสามารถใช้ได้กับ ระบบที่เป็นเชิงเส้นและไม่เชิงเส้นโดยสร้างโมเดลในรูปแบบระบบเวลาต่อเนื่อง, ระบบเวลาไม่ต่อเนื่อง และระบบไฮบริด โดยที่ระบบเป็นแบบมัลติเรต (multirate) ได้

2.3 เรียลไทม์เวอร์คชอพ (Realtime Workshop)

มีหน้าที่สร้างโปรแกรมภาษาซี โดยอัตโนมัติให้กับซิมูลิงค์ มันจะสร้างรหัสภาษาซีขึ้นมาโดยตรงจากบล็อกไดอะแกรมของซิมูลิงค์ และจะสร้างโปรแกรมอัตโนมัติที่สามารถทำงานในเวลาจริงในสิ่งแวดล้อมที่ต่างๆ เช่น การทำงานกับไมโครคอนโทรลเลอร์และตัวประมวลสัญญาณดิจิตอล ซึ่งการทำจะต้องใช้ความรู้ในด้านต่างๆดังนี้คือ

2.4 เรียลไทม์วินโดวส์ทาร์เกต (Real Time Window Target)

คือซอฟต์แวร์ที่อนุญาตให้ภาษาซีที่ถูกสร้างโดยเรียลไทม์เวอร์คชอพ ทำงานได้ในเวลาจริงบนคอมพิวเตอร์ โดยที่ในคอมพิวเตอร์จะต้องมีโปรแกรมแมทแลป, ซิมูลิงค์ และ เรียลไทม์เวอร์คชอพ การประยุกต์ใช้ทั่วๆไปของเรียลไทม์วินโดวส์ทาร์เกต ได้แก่ การควบคุมในเวลาจริง, การประมวลผลสัญญาณและ การจำลองแบบฮาร์ดแวร์อินเดอะลูป (Hardware in The Loop) รหัสภาษาซีที่สร้างขึ้น จะทำงานได้เร็วบนวินโดวส์95,98 และเอ็นที(NT) ขณะที่ทำงานกับโมเดลในหมวดเวลาจริง เรียลไทม์วินโดวส์ทาร์เกต จะจับข้อมูลในช่วงเวลาที่กำหนดจากช่องสัญญาณหนึ่งหรือมากกว่านั้น และใช้ข้อมูลที่จับได้นั้นเป็นอินพุตให้กับโมเดลในซิมูลิงค์ที่เขียนขึ้นเมื่อทำการประมวลข้อมูลเรียบร้อยแล้ว ข้อมูลจะถูกส่งข้อมูลไปยังภายนอกผ่านทางช่องทางสัญญาณเอาต์พุตของอินพุตเอาต์พุตพอร์ต เรียลไทม์วินโดวส์ทาร์เกต ใช้ความสามารถที่มีอยู่ในเรียลไทม์ของเคอร์เนลพิเศษ สำหรับสนับสนุนแต่ละปฏิบัติการณ์ของวินโดวส์ และสำหรับความเร็วของการคอมไพล์ (compile) ซีโปรแกรมมันจะบอกให้บล็อกคำสั่ง ซึ่งมีจำนวนมากกว่า 60 บล็อคคำสั่งที่ใช้ในการติดต่อกับฮาร์ดแวร์ สัญญาณบางที่ถูกจับและถูกบันทึกข้อมูลลงในสโคป โดยใช้การติดต่อระบบภายนอกของซิมูลิงค์ เพื่อที่จะทำให้คุณสามารถสังเกตพฤติกรรมของระบบ เรียลไทม์ ได้

เรียลไทม์วินโดวส์ทาร์เกต ประกอบด้วยชุดของแหล่งกำเนิด, เลขฐานสองสำหรับไดรเวอร์ไลบรารี่ (driver libraries) ของอุปกรณ์อินพุตเอาต์พุต, ตารางที่ใช้ในการสร้างไฟล์ และเมคไฟล์อินเตอร์เฟส (Mex-files interface) หลังจากที่ได้สร้างบล็อกไดอะแกรมโมเดลในซิมูลิงค์และใช้เรียลไทม์เวอร์คชอพ สร้างเรียลไทม์โมเดล โดยการคลิกที่ปุ่มบิลด์ (build) ที่หน้าเรียลไทม์เวอร์ชอพของไดอะลอกบล็อก ซิมูลิงค์พารามิเตอร์ หน่วยของเมคไฟล์ อินเตอร์เฟส จะถูกใช้เพื่ออนุญาตให้โหมดการทำงานภายนอกของซิมูลิงค์ ทำการส่งค่าตัวแปรใหม่ให้เรียลไทม์โมดูล และทำการนำสัญญาณเรียลไทม์โมดูลกลับมา เรายังสามารถแสดงสัญญาณนี้ออกทางสโคปของซิมูลิงค์ได้

2.5 การเริ่มต้นการปฏิบัติการณ์เรียลไทม์

เมื่อการสร้างเรียลไทม์โมเดลได้สำเร็จแล้วเราสามารถทดสอบได้โดยการทดสอบการทำงานที่เวลาจริงและดูผลที่เกิดขึ้นจากการปฏิบัติการณ์ เริ่มต้นการปฏิบัติการณ์ rtvdp โมเดล

โดยไปเลือกคำสั่งเอ็กซ์เทอร์นอล (external)ในเมนูของซิมูเลชัน และเลือกคำสั่งคอนเนคท์ทูทาร์เกต (Connect to target) และจะเห็นว่าสามารถที่จะเลือกคำสั่งสตาร์ทเรียลไทม์โค้ด (Start realtime code) จากนั้นจะสังเกตเส้นกราฟปรากฏขึ้นที่สโคปของซิมูลิงค์ค่อยๆเคลื่อนที่สอดคล้องกับเวลาจริง หลังจากนั้นทำการทดลองนี้อีกครั้ง โดยการทดสอบการทำงานของโมเดลนี้ขึ้นตรงกับซิมูลิงค์โดยไม่ใช้เรียลไทม์วินโดวส์ทาร์เกต จะเห็นว่าการทำงานจะไม่เป็นไปตามเวลาจริง

2.6 เอสฟังก์ชัน (S-Functions) คือ

เอสฟังก์ชันเป็นการบรรยายภาษาคอมพิวเตอร์ของระบบไดนามิก ซึ่งสามารถถูกเขียนโดยใช้แมทแลปหรือภาษาเอสฟังก์ชันของซี (C language S-function) จะถูกคอมไพล์ขณะที่เมคไฟล์กำลังใช้เมคยูทิลิตี้ (Mex utility) ถูกบรรยายอยู่ใน The Application Program Interface Guid ในขณะที่เมคไฟล์อื่นๆจะติดต่ออยู่ตลอดเวลาในแมทแลป เมื่อจำเป็นเอสฟังก์ชัน จะใช้ซินแทกซ์พิเศษเพื่อที่คุณสามารถติดต่อกับสมการการแก้ปัญหาของซิมูลิงค์ (Simulink's Equation solvers) การติดต่อแบบนี้จะคล้ายคลึงกันมากกับการติดต่อระหว่างผู้แก้ปัญหา และการสร้างซิมูลิงค์บล็อกรูปแบบของเอสฟังก์ชัน จะเป็นรูปแบบง่ายมาก และสามารถใช้กับระบบต่อเนื่อง,ไม่ต่อเนื่องและไฮบริดเป็นสาเหตุให้ ซิมูลิงค์โมเดล ทั้งหมดมีความใกล้เคียงกับเอสฟังก์ชัน เอสฟังก์ชันจะถูกใช้เสมือนเป็นโมเดลหนึ่งในซิมูลิงค์โดยสามารถเรียกขึ้นมาได้จากเอสฟังก์ชันบล็อกในไลบรารี่ย่อย (sublibraly) การใช้เอสฟังก์ชั่นบล็อกของไดอะลอกบอกซ์ เพื่อที่จะระบุชื่อของช่องว่างของเอสฟังก์ชันที่เว้นไว้ ซึ่งอธิบายตามรูปที่ 2.1

ในตัวอย่างนี้โมเดล จะประกอบด้วย 2 เอสฟังก์ชันบล็อก ซึ่งทั้งคู่จะอ้างอิงในแหล่งกำเนิดไฟล์เดียวกัน (mysfun, สามารถเป็นได้ทั้งซีเมคไฟล์ และเอมไฟล์) ถ้าทั้งซีเมคไฟล์ และเอมไฟล์ปรากฏด้วยชื่อเดียวกันซีเมคไฟล์ จะเป็นส่วนที่ถูกเรียกใช้ คุณสามารถใช้แมสกิง (masking) ของซิมูลิงค์ อำนวยความสะดวกเพื่อที่จะสร้างไดอะลอกบอกซ์ และ ไอคอน (Icons) สำหรับ เอสฟังก์ชันบล็อกของคุณ และแมสไดอะลอกบอกซ์ สามารถที่จะระบุการเพิ่มด้วยพารามิเตอร์สำหรับเอสฟังก์ชันการบรรยายของการเพิ่ม พารามิเตอร์และแมสกิงดูใน Using Simulink

2.6.1 เมื่อไหร่ที่จะใช้เอสฟังก์ชัน

เป็นการง่ายที่สุดของการใช้ เอสฟังก์ชัน สร้างซิมูลิงค์บล็อก คุณสามารถใช้เอสฟังก์ชันสำหรับการใช้ที่หลากหลาย รวมถึง

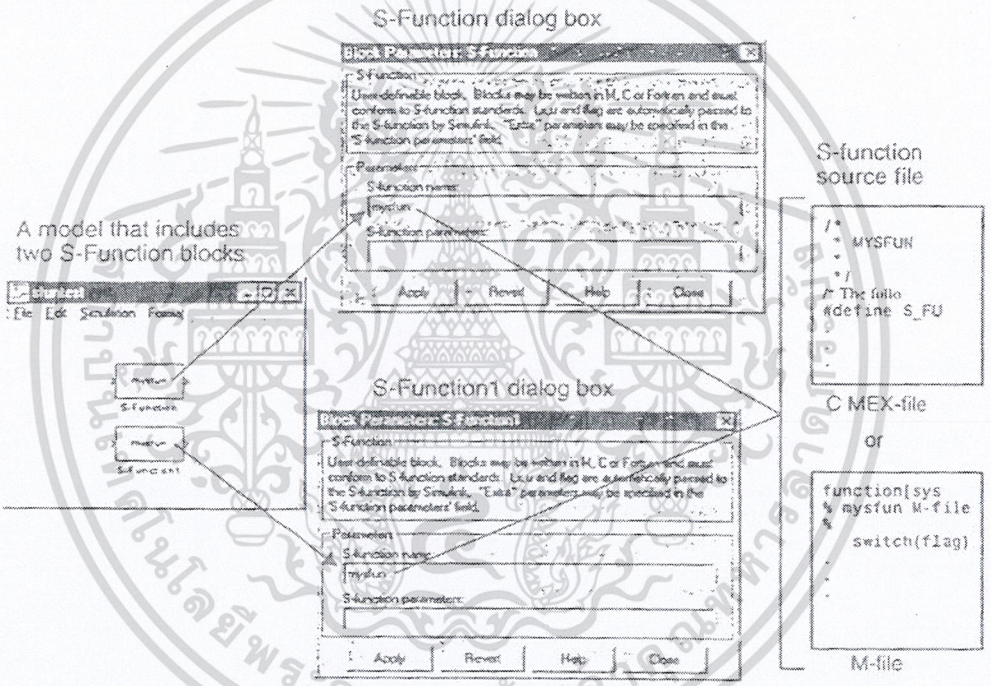
- การเพิ่มบล็อกที่มีจุดประสงค์ใหม่ไปยังซิมูลิงค์
- การร่วมมือกับซีโปรแกรมในการซิมูเลชัน

- อธิบายระบบที่เป็นรูปแบบสมการทางคณิตศาสตร์
- ใช้ กราฟฟิคเอนิเมชัน (Graphical animations)

ข้อได้เปรียบของการใช้เอสฟังก์ชันที่ว่า คุณสามารถสร้างบล็อกจุดประสงค์ทั่วไปและคุณสามารถใช้หลายๆครั้ง ใน 1 โมเดล ในการเปลี่ยนแปลงค่าต่างๆในส่วนของแต่ละบล็อก

2.6.2 ใช้เอสฟังก์ชันได้อย่างไร

แต่ละบล็อกในซิมูลิงค์ โมเดลจะเป็นไปตามลักษณะทั่วๆไปของเวคเตอร์อินพุต u เวคเตอร์เอาต์พุต y และเวคเตอร์ของสถานะต่างๆ



รูปที่ 2.1 แสดงความสัมพันธ์ระหว่าง เอสฟังก์ชันบล็อก, ไดอะลอกบอกซ์, และซอสไฟล์ที่ให้คำจำกัดความของพฤติกรรมของบล็อก

2.7 ซีเมคเอสฟังก์ชัน (C-MEX S-Function)

ในเอ็มไฟล์ เอสฟังก์ชัน ,ซิมูลิงค์ประกอบด้วยเวคเตอร์สถานะใน 2 ส่วนคือ สถานต่อเนื่องและไม่ต่อเนื่อง สถานะต่อเนื่องจะจับจองในส่วนแรกของเวคเตอร์ และไม่ต่อเนื่องจะจับจองส่วนที่ 2 ส่วนบล็อกที่ไม่มีสถานะ ก็คือ x ที่เป็นเวคเตอร์ว่างเปล่า สถานการณ์จำลองและ เอสฟังก์ชันรูทีน (Simulation stages & S-Function Routines) ซิมูลิงค์จะทำการเรียกซึ่งระหว่างการเจาะจงสถานะ

ของซิมูเลชันในแต่ละบล็อกในโมเดลไปยังมันโดยตรงเพื่อจะปฏิบัติหน้าที่ เช่น คำนวณเอาต์พุตของมัน อัพเดทค่าของระบบไม่ต่อเนื่อง หรือคำนวณอนุพันธ์ (Deriavatives) ส่วนการเรียกเพิ่มเติมคือคำสั่งการเริ่มต้นและจบของการ ซิมูเลชันเพื่อต้องปฏิบัติหน้าที่เริ่มต้นและสิ้นสุด

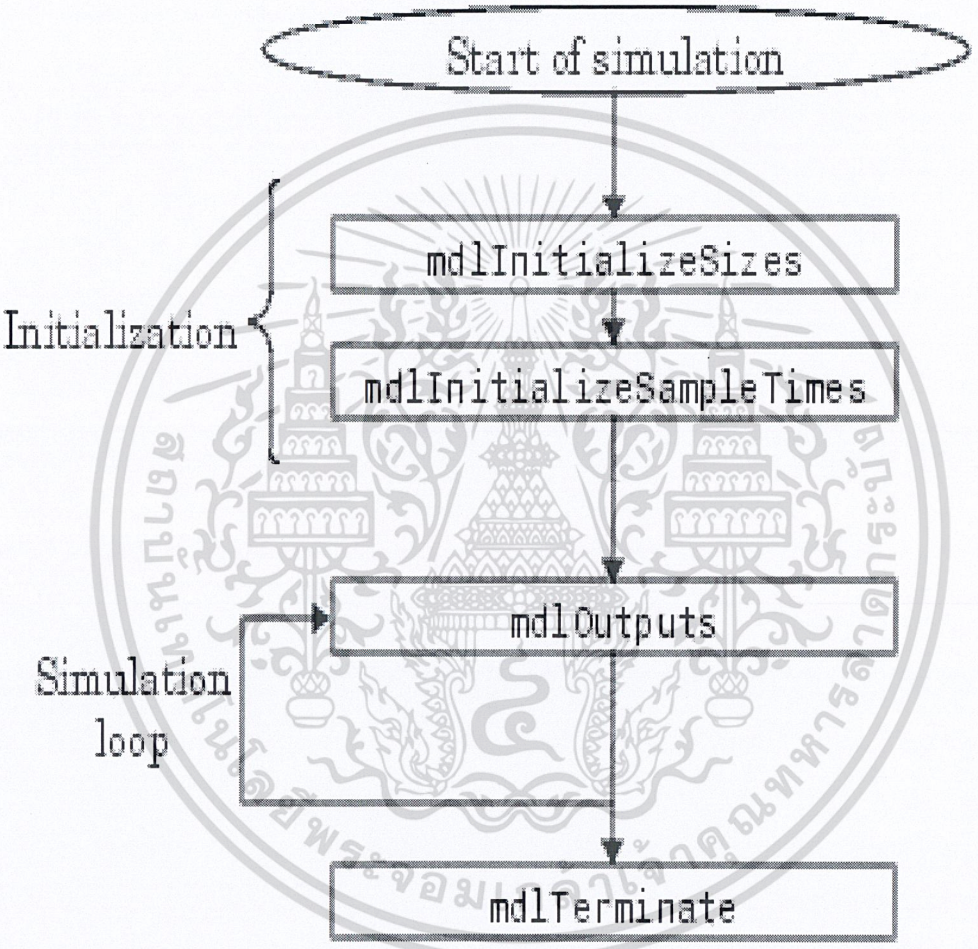
รูปที่ 2.2 เป็นการแสดงว่าซิมูลิงค์ จะทำการซิมูเลชันอย่างไร อย่างแรกซิมูลิงค์จะทำการเริ่มต้นโมเดล (การเริ่มต้นแต่ละบล็อกรวมถึงเอสฟังก์ชัน) และซิมูเลชันลูปที่ซึ่งแต่ละเส้นทางจะผ่านตามลูปซึ่งจะอ้างถึงซิมูเลชันสเตประหว่างแต่ละซิมูเลชันสเตป, ซิมูลิงค์จะปฏิบัติการณ์เอสฟังก์ชันบล็อกของคุณอย่างต่อเนื่อง จนกระทั่งการซิมูเลชันเรียบร้อยซิมูลิงค์จะทำการเรียกซ้ำไปยังเอสฟังก์ชันในโมเดลของคุณในระหว่างการเรียกนี้ซิมูลิงค์เรียกเอสฟังก์ชั่นรูทีน, ปฏิบัติตามหน้าที่ที่ถูกต้องการแต่ละสถานะหน้าที่รวมถึง

- Initialization ก่อเริ่มลูปแรกของการซิมูเลชัน ซิมูลิงค์จะทำการเริ่มต้นเอสฟังก์ชันระหว่างสถานะของซิมูลิงค์
- Initializes the Simustruct (a simulation structure) ซึ่งประกอบด้วยข้อมูลเกี่ยวกับเอสฟังก์ชัน
- เซตตัวเลขและขนาดของ อินพุตและเอาต์พุตพอร์ต
- เซตของบล็อกการสุ่มค่า
- จัดพื้นที่การจัดเก็บและขนาดของอาร์เรย์
- ทำการคำนวณของค่า next sample hit
- ทำการคำนวณเอาต์พุตในช่วงของเวลาหลักหลังจากการเรียกนี้สมบูรณ์เอาต์พุตพอร์ตทั้งของบล็อกจะมีค่าของช่วงเวลาในปัจจุบัน
- Update สถานะไม่ต่อเนื่องในช่วงเวลาหลักในการเรียกนี้จะปฏิบัติการณ์ 1 ครั้งช่วงเวลาการทำงาน
- Integration จะใช้กับโมเดลที่ต่อเนื่องและ/หรือ nonsampled zero crossings ถ้าเอสฟังก์ชันของคุณเป็นแบบต่อเนื่อง ซิมูลิงค์จะเรียกส่วนของเอาต์พุตและอนุพันธ์ของเอสฟังก์ชันของคุณที่ไมเนอร์ไทม์สเตป ซึ่งซิมูลิงค์สามารถคำนวณสถานะของเอสฟังก์ชัน

ซีเมคไฟล์จะให้ข้อมูลเกี่ยวกับโมเดลไปยังซิมูลิงค์ระหว่างซิมูเลชัน ในขณะที่ดำเนินการซิมูเลชันซิมูลิงค์ ODE SOLVER และเมคไฟล์จะติดต่อกันเพื่อจะทำงานตามหน้าที่ที่เจาะจงไว้ หน้าที่นี้ได้แก่การนิยามค่าเริ่มต้นและบล็อกคาแรกเตอร์ริสติคส์ และคำนวณอนุพันธ์, สถานะไม่ต่อเนื่อง, เอาต์พุตซีเมคไฟล์ของเอสฟังก์ชันจะมีโครงสร้างและการปฏิบัติการณ์เหมือนกับเอมไฟล์ แต่จะต่างกันในเรื่องของการปฏิบัติการณ์

2.7.1 การเขียนซีเมคเอสฟังก์ชั่นพื้นฐาน

ในส่วนจะกล่าวถึงการเขียนซีเมคเอสฟังก์ชันพื้นฐาน ซึ่งจะเป็นส่วนที่ต้องการในเอสฟังก์ชันรูทีน แต่อย่างไรก็ตามผู้ใช้สามารถทำให้โปรแกรมซับซ้อนอย่างไรก็ได้ ในที่นี้จะขอกล่าวถึงโปรแกรม timestwo ซึ่งโปรแกรม timestwo ของเอสฟังก์ชัน จะมีส่วนที่เอสฟังก์ชันต้องการตามรูปที่ 2-2นี้



รูปที่ 2.2 แสดงส่วนที่ใช้อยู่ในเอสฟังก์ชัน

เพื่อที่จะใช้เอสฟังก์ชันในซิมูลิงค์ จะต้องสร้าง source file เพื่อที่จะเรียกใช้ timestwo.c โดยพิมพ์ mex timestwo.c ในที่นี้จะขอกล่าวถึงรูทีนเอสฟังก์ชันระหว่างการซิมูเลชันของ timestwo

- mdlInitializeSizes ซิมูลิงค์จะเรียกรูทีนนี้ ขณะที่อิดิตโมเดล เพื่อที่จะกำหนดจำนวนของอินพุต และเอาต์พุตพอร์ต ซิมูลิงค์จะเรียกมันขณะที่เริ่มต้นการซิมูเลชันเพื่อร้องขอขนาดของพอร์ตและวัตถุประสงค์อื่น

- mdlInitializeSampletimes ซิมูลิงค์จะเรียกรูทีนนี้ เพื่อกำหนดขนาดของการสุ่มของเอสฟังก์ชัน

- mdlOutputs เป็นการคำนวณของค่าเอาต์พุต mdlOutputs จะรับค่าอินพุตและจะคูณมันด้วย 2 สำหรับโปรแกรม timestwo รูทีนนี้จะถูกเรียกขณะที่มีการซิมูเลชันลูปแต่ละเวลาซึ่งจะต้องมีการอัพเดทใน timestwo

- mdlterminate จะแสดงในส่วนท้ายสุดของโปรแกรมแต่ใน timestwo จะไม่มีการแสดงของหน้าที่นี้เนื่องจากรูทีนนี้ว่างเปล่า

สำหรับข้อมูลเพิ่มเติมของ timestwo จะต้องการ

การนิยามอันได้แก่ ชื่อของเอสฟังก์ชัน (timestwo) และเอสฟังก์ชันจะต้องอยู่ในการจัดรูปแบบ level2 หลังจากนิยามในสองหัวข้อนี้แล้ว จะยกตัวอย่าง simstruct.h ซึ่งจะเรียกว่าส่วนเฮดไฟล์ (head file) ซึ่งในการเข้าถึง SimStruct และ API ของ Matlab :

```
#define S_FUNCTION_NAME  timestwo
#define S_FUNCTION_LEVEL   2
#include "simstruct.h"
```

- mdlInitializeSizes แสดงได้โดยตัวอย่างโปรแกรม timestwo

Zero parameters หมายถึง เอสฟังก์ชันพารามิเตอร์ ของไดอะลอกบอกซ์จะต้องว่างถ้ามันมีตัวแปร ซิมูลิงค์อยู่ จะแสดงผลมาว่าเกิดการไม่เข้ากัน (mismatch) หนึ่งอินพุตและหนึ่งเอาต์พุต ความกว้างของพอร์ตอินพุตและเอาต์พุตสามารถเปลี่ยนแปลงขนาดได้สิ่งนี้บอก ด้วยว่าซิมูลิงค์ใช้คุณแต่ละอุปกรณ์แต่ละสัญญาณอินพุทกับเอสฟังก์ชัน โดยคูณกับ 2 และใส่ผลลัพธ์ลงในสัญญาณเอาต์พุต การที่ขนาดของมันเปลี่ยนแปลงได้ จะทำให้เอสฟังก์ชันของกรณีนี้มีความกว้างของอินพุตและเอาต์พุตเท่ากันหนึ่งการสุ่มเวลาควรจะต้องระบุค่าของSample timeในmdlInitializeSampletime

รหัสที่ถูกยกเว้นการระบุโค้ดนี้จะ ทำให้การจัดการของเอสฟังก์ชันคุณเร็วขึ้นจะต้องสนใจในเรื่องของการระบุใน option ด้วย โดยทั่วไปแล้วค่าเอสฟังก์ชันของคุณไม่สามารถติดต่อกับ Matlabได้ คุณควรจะระบุใน option นี้

- mdlIntializeSampletime จะถูกยับยั้งจากบล็อกผลักดันหมายถึงเอสฟังก์ชันจะทำงานได้เมื่อได้รับอินพุตจากบล็อกที่ติดต่อกับอินพุตพอร์ตของบล็อกเอสฟังก์ชัน

- mdloutput การคำนวณตัวเลข mdloutput ได้จากสัญญาณซิมูลิงค์คูณสัญญาณอินพุตด้วยสอง

การเข้าถึงสัญญาณอินพุต

ใช้ InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(s,0); ซึ่ง Uptrs เป็นเวคเตอร์ตัวหนึ่งของพอยเตอร์จะต้องถูกใช้ในรูป uPtrs(i)

การเข้าถึงสัญญาณเอาต์พุต

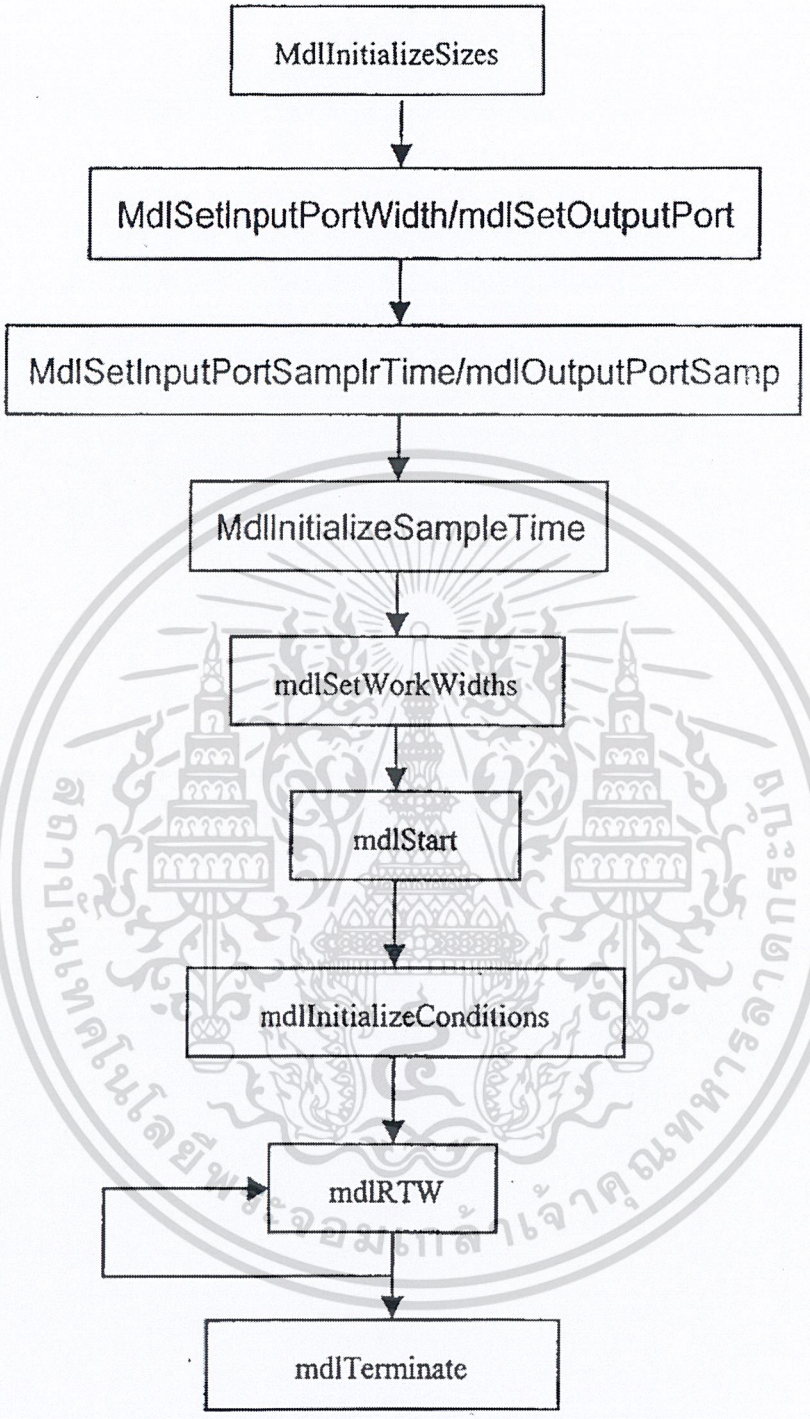
ใช้ real_T*Y=ssGetOutputPortRealSignalPtrs(s,0); ลูปของเอสฟังก์ชันจะกว้างกว่าความกว้างของสัญญาณที่กำลังผ่านเข้าไปในบล็อกเพื่อที่จะเข้าถึงบล็อกสัญญาณคุณสามารถตรวจสอบพอร์ตความกว้างพอร์ตอินพุตและพอร์ตเอาต์พุตได้ ซึ่งความกว้างของเอาต์พุตสามารถหาค่าได้

- mdlterminate ซึ่งเป็นเอสฟังก์ชันรูทีน อย่างไรก็ตาม timestwo ฟังก์ชันไม่จำเป็นต้องแสดงดังนั้นรูทีนนี้จึงว่างสุดท้ายของเอสฟังชันจะบอกรหัสที่ติดต่อกับตัวอย่างเพื่อเข้าสู่ซิมูลิงค์และเรียลไทม์เวิร์คชอพ

```
#ifdef MATLAB_MEX_FILE
#include “simulink.c”
#else
#include “cg_sfun.h”
#endif
```

2.7.2 โครงสร้างของการแบบเปลี่ยนได้สำหรับโหมดภายนอก

เมื่อรันโปรแกรมซิมูลิงค์ในโหมดภายนอก ลำดับการเรียนของเอสฟังก์ชันรูทีนจะเปลี่ยนแปลง และรูปข้างล่างนี้แสดงลำดับที่ถูกต้องของโหมดภายนอก



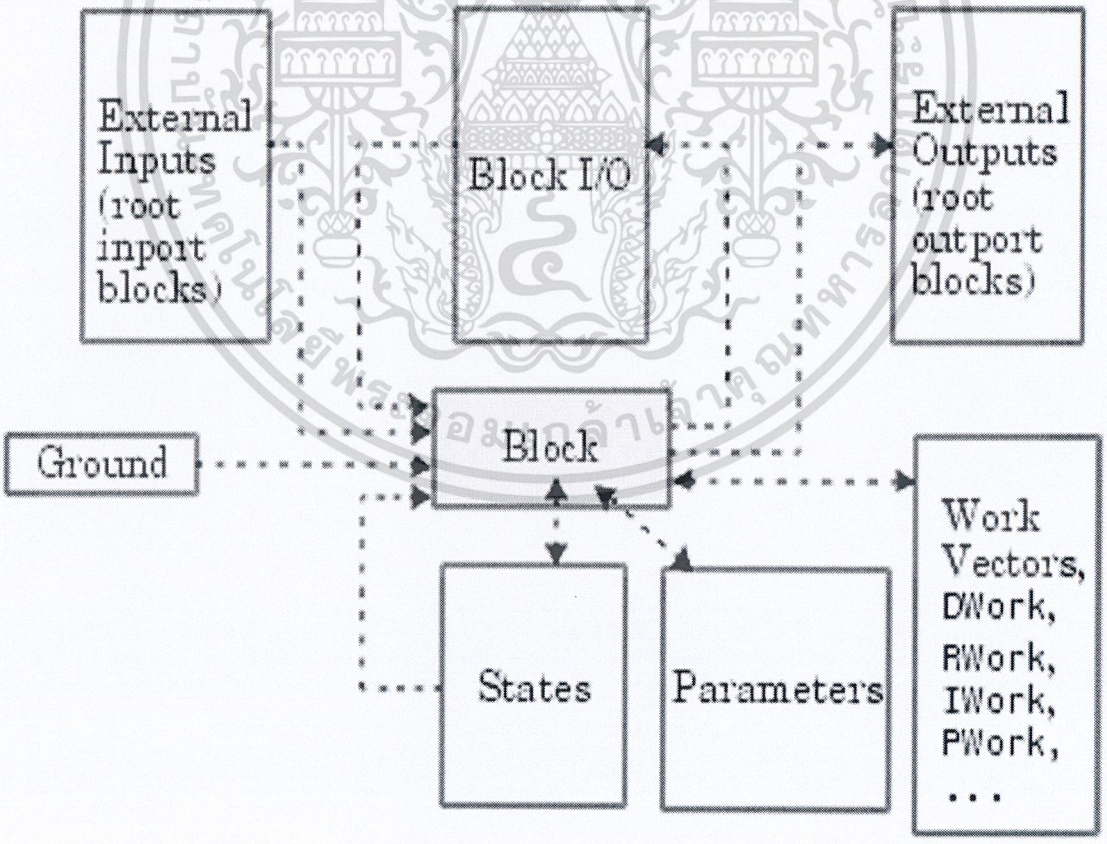
รูปที่ 2.3 ลำดับการเรียกแบบเอสฟังก์ชันเมื่อมีการรันซิมูลิงค์ในโหมดภายนอก

ซิมูลิงค์ที่เรียกว่า mdlRTW เมื่อเข้าสู่โหมดภายนอก การเข้าสู่โหมดภายนอกแต่ละครั้ง หรือเมื่อคุณเลือกอัพเดตเมื่อคุณเลือกอัพเดต ไดอะแกรม ภายใต้ Edit เมนูของรูปแบบของคุณ จะทำให้พารามิเตอร์เปลี่ยนไป

2.7.3 การแสดงข้อมูลของเอสฟังก์ชัน

บล็อกของเอสฟังก์ชันมีสัญญาณอินพุตและสัญญาณเอาต์พุต, พารามิเตอร์, สถานะภายในบวกกับพื้นที่ทำงานทั่วไปอื่นๆ โดยทั่วไป บล็อกอินพุตและเอาต์พุตจะถูกเขียนไปยัง และถูกอ่านจาก I/O เวคเตอร์บล็อก อินพุตต่างๆสามารถมาจาก

- อินพุตภายนอกโดยผ่านทางบล็อกของพอร์ตภายใน
- กราวนด์ ถ้าสัญญาณอินพุตไม่ได้ถูกเชื่อมต่อ หรือถูกต่อลงกราวนด์บล็อกเอาต์พุตสามารถไปถึงเอาต์พุตภายนอกได้ โดยผ่านทางบล็อกของพอร์ตภายนอก สิ่งที่เพิ่มเติมคือ สัญญาณอินพุต และสัญญาณเอาต์พุตเอสฟังก์ชันสามารถมีสถานะที่ต่อเนื่องและสถานะที่ไม่ต่อเนื่อง
- พื้นที่ทำงานอื่นๆ เช่น ค่าจริง, เลขจำนวนเต็ม, หรือเวคเตอร์ตัวชี้บล็อกของเอสฟังก์ชันสามารถถูกทำเป็นพารามิเตอร์โดยการผ่านพารามิเตอร์ไปยังบล็อกเหล่านั้น และการใช้ไดอะลอกบอกซ์ของบล็อกของเอสฟังก์ชันรูปที่ 2.4 แสดงแมปปิงทั่วไประหว่างข้อมูลแบบต่างๆกัน

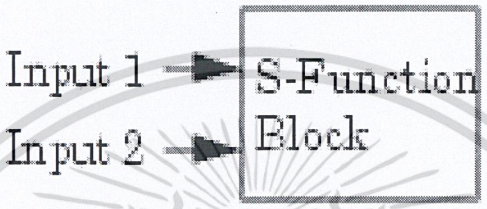


รูปที่ 2.4 แสดงความสัมพันธ์ระหว่างเอสฟังก์ชันและข้อมูลของเอสฟังก์ชัน

ความยาวสัญญาณต่างๆและเวคเตอร์จะมีลักษณะ mdlInitializeSizes รูทีน และความยาวของมันสามารถเข้าไปในรูทีนเอสฟังก์ชันจะถูกเรียกระหว่างลูปซิมูเลชันระหว่างซิมูเลชันลูป การเข้าถึงสัญญาณอินพุตจะถูกแสดงดังต่อไปนี้

= ssGetInputPortRealSignalPtrs InputRealPtrsType uPtrs (s,portindex);

สิ่งนี้คืออาร์เรย์ของพอยเตอร์ซึ่งพอร์ทอินเดคเริ่มที่ 0 จะมีหนึ่งอันสำหรับพอร์ทอินพุต เท่านั้นการเข้าถึงอุปกรณ์ของสัญญาณนี้คุณจะต้องใช้ uPtrs (element) สามารถแสดงให้เห็น



รูปที่ 2.5 แสดงถึงการเข้าถึงสัญญาณ

2.7.4 การเข้าสัญญาณอินพุตของพอร์ต

ในส่วนนี้กล่าวถึงจะทำอย่างไรจะเข้าถึงสัญญาณอินพุตทั้งหมดของพอร์ต และเขียนมันลงในเอาต์พุตของพอร์ตอยู่ในรูปแบบฟอร์มของเวคเตอร์ดังนั้นวิธีที่ถูกต้องในการเข้าถึงอุปกรณ์อินพุตและเขียนมันสู่สัญญาณเอาต์พุต (สมมติว่าพอร์ตอินพุตและพอร์ตเอาต์พุตมีความกว้างเท่ากัน) จะต้องใช้โค้ดเหล่านี้คือ

```
int_T element;
int_T portwidth = ssGetInputPortWidth (s,input PortIndex);
ssGetInputPortRealSignalPtrs InputRealPtrsType uPtrs (s,portindex);
real_T*Y=ssGetOutputPortRealSignalPtrs(s,outputPortIdx)
```

ความผิดพลาดโดยทั่วไป จะพยายามและเข้าถึงสัญญาณอินพุตทางพอยเตอร์ ตัวอย่างเช่น

```
real_T*U = *uPtrs
```

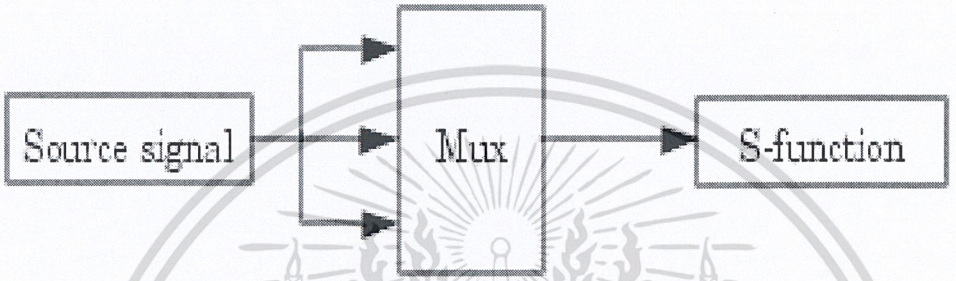
ซึ่งแบบนี้จะผิดหลังจากเริ่มต้นของ uptrs และการแทนเข้าไปในส่วนภายในของลูป

```
y++ = * u++
```

ซึ่งแบบนี้ก็จะผิดการแปลงรหัสสำหรับเมคไฟล์ อาจจะชนกับซิมูลิงค์ได้ อันเนื่องมาจากการเข้าไปยังหน่วยความจำที่สูญหายได้ ซึ่งจะขึ้นอยู่กับคุณว่าได้สร้างรูปแบบของคุณอย่างไร การเข้าถึงสัญญาณอินพุท เกิดการชนกันอย่างผิดปกติจะเกิดเมื่อสัญญาณที่เข้าเอสฟังก์ชันของคุณไม่

ติดกัน สัญญาณข้อมูลที่ไม่ติดกันนี้จะเกิดเมื่อสัญญาณผ่านเข้าสู่บล็อกที่ติดต่อกันแบบเห็นได้จริง เช่น บล็อกของซีเลคเตอร์หรือมัลติเพล็กเซอร์

เพื่อพิสูจน์ว่าคุณกำลังเข้าสู่สัญญาณอินพุตที่ถูกต้องสัญญาณจะลอกเลียนแบบไปยังพอร์ตอินพุตของแต่ละอันของเอสฟังก์ชันของคุณเมื่อขั้นตอนนี้ถูกกระทำเสร็จโดยการสร้าง mux บล็อกกับจำนวนพอร์ตอินพุต = ความกว้างของสัญญาณที่ต้องการ ที่กำลังเข้าสู่เอสฟังก์ชันของคุณ แหล่งขับเคลื่อนควรจะติดกับอินพุตพอร์ตแต่ละอันแสดงได้ดังภาพ



รูปที่ 2.6 แสดงถึงการเข้าถึงสัญญาณอินพุต

2.7.5 การตรวจสอบและการประมวลผลตัวแปรเอสฟังก์ชัน

คุณสามารถให้พารามิเตอร์ยังเอสฟังก์ชัน ซึ่งสามารถถูกเปลี่ยนโดยใช้การติดต่อกับเอสฟังก์ชันพารามิเตอร์ของไดอะลอกบอกซ์ ถ้าคุณนิยามพารามิเตอร์ ข้างล่างเป็นขั้นตอนเมื่อคุณจะสร้างเอสฟังก์ชัน

- กำหนดคำสั่ง ซึ่งพารามิเตอร์จะระบุในไดอะลอกบอกซ์

- ใน mdLinitializeSizes ฟังก์ชันใช้ ssSetNumSFcnParams สำหรับบอกซิมูลิงค์ถึงจำนวนตัวแปรจำนวนตัวแปรที่ถูกผ่านเข้าไปในเอสฟังก์ชัน ระบุเอสฟังก์ชันเป็นเหมือนอาร์กิวเมนต์ตัวแรก และจำนวนของตัวแปรที่คุณจำกัดความให้เป็นอาร์กิวเมนต์ตัวที่สอง

- การเข้าถึงอาร์กิวเมนต์อินพุตเหล่านี้ในเอสฟังก์ชันที่กำลังใช้ ssGetSFcnParam macro ระบุเอสฟังก์ชันเป็นอาร์กิวเมนต์แรก และตำแหน่งของตัวแปรในรายการที่ถูกเข้าไปยังไดอะลอกบอกซ์ (จะเป็นตำแหน่งแรก) เปรียบเสมือนอาร์กิวเมนต์ที่สอง เมื่อคุณทำการซิมูเลชันระบุชื่อตัวแปรหรือค่าในตัวแปรเอสฟังก์ชันของไดอะลอกบอกซ์ลำดับของชื่อ และค่าตัวแปรจะเหมือนกันเพื่อที่จะเรียงลำดับตามที่คุณกำหนดไว้ในขั้นตอนที่ 1 ถ้าคุณกำหนดชื่อคุณไม่จำเป็นต้องใช้ชื่อเดียวกันในเมคไฟล์

2.7.6 การใช้เอสฟังก์ชันกับเรียลไทม์เวิร์คชอพ

โดยทั่วไปคุณสามารถใช้เอสฟังก์ชั่นในเรียลไทม์เวิร์คชอพ อย่างไรก็ตามในบางกรณีต้องการการปรับปรุงเหล่านี้ เพื่อเอสฟังก์ชัน

- การกำหนดชื่อของรูปแบบพิเศษถูกใช้ในการสร้างเอสฟังก์ชันของคุณ
- การกำหนด RTW data สำหรับเอสฟังก์ชัน
- การเพิ่ม mdlRTW ฟังก์ชัน

2.7.7 ชื่อโมดูลสำหรับ RTW Build's

ถ้าเอสฟังก์ชันของคุณถูกสร้างด้วยโมดูลจำนวนมาก คุณจำเป็นต้องแบ่งการสร้างชื่อกระบวนการของโมดูลที่เพิ่มเข้าไป คุณจะสามารถทำขั้นตอนนี้ได้โดยคุณต้องผ่านเรียลไทม์เวิร์คชอพเทมเพลตเมคไฟล์ หรือเพื่อให้สะดวกมากยิ่งขึ้นให้ใช้ set_param MATLAB command สำหรับตัวอย่างถ้าเอสฟังก์ชันของคุณใช้หลายโมดูลต้องทำดังต่อไปนี้

mex sfun_main.c Sfun_modulel.c Sfun_module2.c

ต่อมาระบุชื่อของโมดูล โดยไม่ต้องใช้คำสั่ง

set_param=(sfun_block,-,'SfunctionModule','sfun_modulel sfun_module2')

ตัวแปรสามารถเปลี่ยนแปลงได้

module=('sfun_modulel sfun_module2

set_param(sfun_block,-'SfunctionModule','modules')

2.7.8 เอสฟังก์ชัน RTWdata สำหรับการสร้างรหัสด้วย RTW

มีคุณสมบัติของบล็อกอย่างหนึ่งถูกเรียกว่า RTW data ซึ่งสามารถถูกใช้โดยทีแอลซี เมื่อทำเอสฟังก์ชัน RTW data จะเป็นโครงสร้างหนึ่งของตัวอักษร ซึ่งคุณสามารถติดต่อได้กับบล็อกมันจะถูกบันทึกและแทนที่ในโหมด model.rtw เมื่อสร้างรหัสขึ้นสำหรับตัวอย่างคำสั่งกลุ่มนี้ของ Matlab คือ

mydata.field1 = 'information for field1';

mydata.field2 = 'information for field2';

set_param(gcb,'RTWdata',mydata)

Get_param(gcb'RTWdata'

ได้ผลลัพธ์ดังนี้

Ans = field': 'information for field1'

Field2:"iformation for field2'

ใน model.rtw สำหรับ บล็อกเอสฟังก์ชันจะมีข้อมูลเหล่านี้คือ

```
Block {
        Type    "S-Function"
        RTWdata{
                field1   " information for field1"
                field2   " information for field2"
```

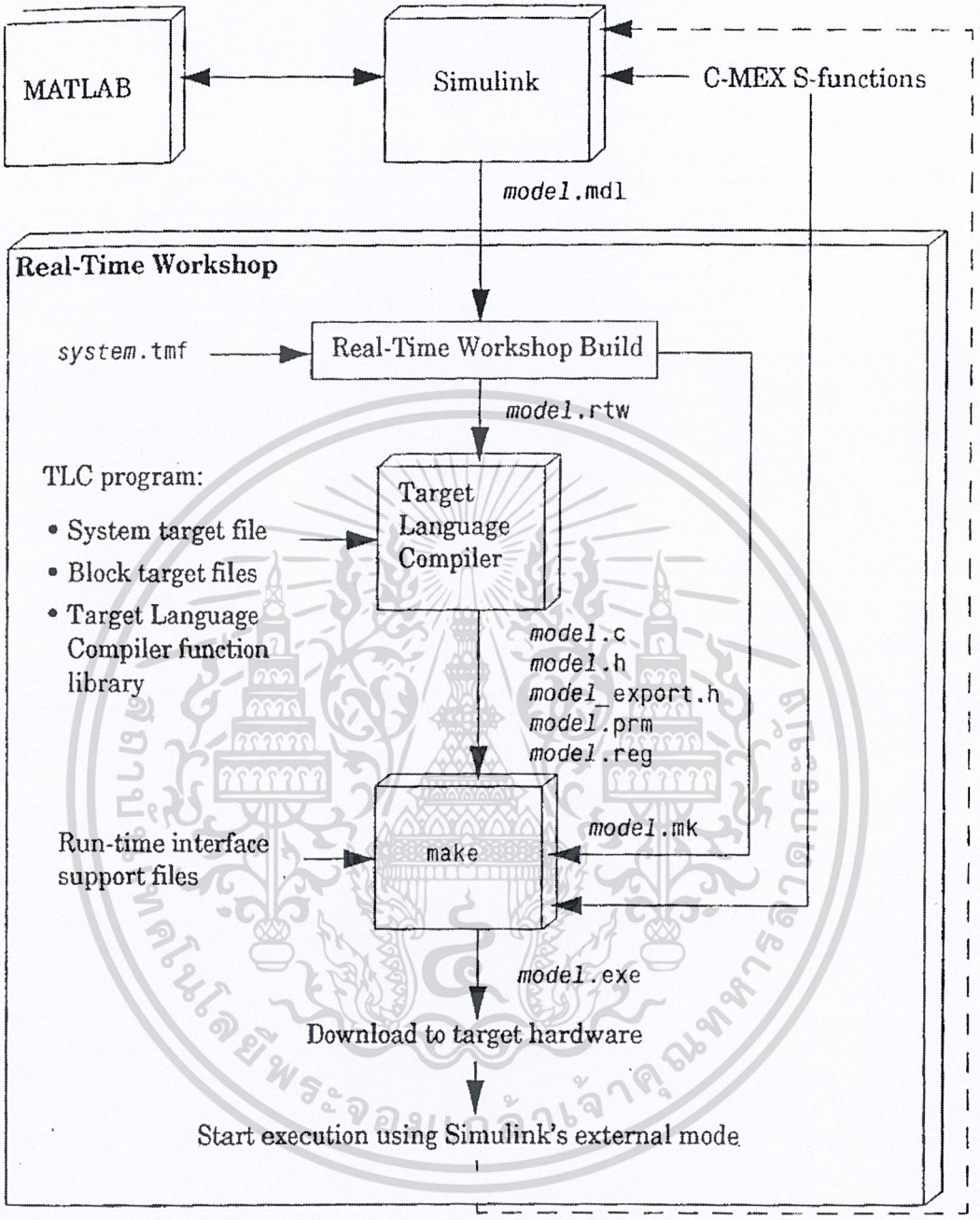
mdlRTW

mdIRTW รูทีนจะช่วยย่อยเอสฟังก์ชันของคุณในรหัสที่สร้างขึ้น การย่อในรหัสของคุณในเรียลไทม์เวอร์คชอพ โดยทั่วไปเมื่อเสร็จขั้นตอนนี้สำหรับการแสดงผล คุณอาจจะถูกต้องการอินลายฟังก์ชันของคุณอีกครั้งหนึ่ง ถ้าคุณมี mdlProcessParameters รูทีนหรือ ถ้าเอสฟังก์ชันคุณมีซิมูเลชันโหมด,และเรียลไทม์โหมด เช่น ฮาร์ดแวร์ I/0 เอสฟังก์ชัน ซึ่งประมวลอุปกรณ์อินพุตเอาต์พุตในซิมูลิงค์และติดต่อด้วยอุปกรณ์อินพุตเอาต์พุตในเรียลไทม์

2.8 วิธีการใช้เรียลไทม์เวิร์คชอพ

เรียลไทม์เวอร์คชอพ (Realtime Workshop) ใช้กับแมทแลป และซิมูลิงค์ ผลิตโค้ดโดยตรงจากซิมูลิงค์โมเดลและสร้างโปรแกรมโดยอัตโนมัติ ซึ่งสามารถทำงานในหลายรูปแบบได้แก่ระบบเวลาจริงและการจำลองแบบสแตนอโลน (stand alone simulation) โดยสามารถทำงานได้ในที่ความเร็วสูงบนเครื่องจักรหลัก และกับคอมพิวเตอร์ภายนอก การประยุกต์ใช้งานของเรียลไทม์เวอร์คชอพ

- การควบคุมที่เวลาจริง (realtime control) เราสามารถออกแบบระบบควบคุม โดยแมทแลปและซิมูลิงค์ และสร้างโค้ดจากบล็อกไดอะแกรมโมเดล คอมไพล์และดาวน์โหลดโค้ดนี้เข้าสู่ฮาร์ดแวร์เป้าหมายได้โดยตรง
- การประมวลผลสัญญาณที่เวลาจริง เราสามารถประมวลผลสัญญาณที่สร้างโดยแมทแลปและซิมูลิงค์ การสร้างโค้ดจากบล็อกไดอะแกรมของคุณสามารถคอมไพล์ และดาวน์โหลดไปยังฮาร์ดแวร์เป้าหมายได้
- การจำลองแบบฮาร์ดแวร์ อินเดอะลูป (Hardware in the loop) คุณสามารถสร้างซิมูลิงค์โมเดล ซึ่งใช้ในระบบจริงๆหรือระบบไดนามิก และสัญญาณจริงเพื่อใช้ในการติดต่อเปลี่ยนแปลงที่เวลาจริง



รูปที่ 2.7 สถาปัตยกรรมของเรียลไทม์เวอ์คชอพ

2.8.1 โค้ดที่ถูกสร้างขึ้น

ถูกสร้างจากซิมูลิงค์โมเดลใดๆ ไม่ว่าเป็นระบบต่อเนื่อง, ไม่ต่อเนื่อง และไฮบริดก็ตาม มีลักษณะเป็นรหัสทางภาษาซี ซิมูลิงค์บล็อกทั้งหมดถูกแปลงเป็นโปรแกรมโดยอัตโนมัติ โดยเราต้องเขียนบล็อกนี้เป็นซีเมคเอสฟังก์ชันเป็นแบบที่เราต้องการให้ทำงานกับเรียลไทม์เวอร์คชอพ

เรียลไทม์เวอร์คชอพ ประกอบด้วยชุดของไฟล์เป้าหมายซึ่งถูกคอมไพล์โดยทาร์เกตแลงเกวจคอมไพเลอร์ (ทีแอลซี) และกลายเป็น ANSI C CODE โดยก่อนหน้านี้ไฟล์เป้าหมายเป็นแอสกี้ที่อธิบายว่า เราจะแปลงโมเดลของซิมูลิงค์เป็นโค้ดได้อย่างไร เราสามารถรวมซีเมคเอสฟังก์ชันร่วมกับโค้ดที่ถูกสร้างขึ้นไว้ในโปรแกรมทำงาน และเรายังสามารถเขียนทาร์เกตไฟล์ให้กับซีเมคเอสฟังก์ชันเพื่อที่จะอินลาย (inline) เอสฟังก์ชันนั้น

2.8.2 ทีแอลซี (Target language compiler)

ในการสร้างโค้ดจำเป็นต้องมีทีแอลซี เพื่อใช้เป็นตัวแปลงคำอธิบายโมเดลที่สร้างโดยเรียลไทม์เวอร์คชอพให้กลายเป็นโค้ดที่มีเป้าหมายเฉพาะ และคำอธิบายโมเดลนั้นจะถูกบันทึกในไฟล์แอสกี้โดยมีชื่อว่า model.rtw คอมไพเลอร์จะอ่านไฟล์ model.rtw ขึ้นมา และปฏิบัติการณ์โปรแกรมทีแอลซี ซึ่งมีชุดของทาร์เกตไฟล์ (.tlc) ลงไป โปรแกรมทีแอลซีบอกให้รู้ว่าจะแปลงไฟล์ model.rtw ให้เป็นโค้ดที่ถูกสร้าง (generated code)

โปรแกรมทีแอลซีประกอบด้วย

- ไฟล์หลัก ซึ่งเรียกว่าซิสเตมทาร์เกตไฟล์ (system target files)
- ชุดของบล็อกทาร์เกตไฟล์ซึ่งบ่งชี้ว่าจะแปลงแต่ละบล็อกในโมเดลให้กลายเป็นโค้ดที่เฉพาะเจาะจงได้อย่างไร
- ทีแอลซีฟังก์ชันไลบรารี่ เป็นชุดของไลบรารี่ที่ทีแอลซีใช้ในการแปลงไฟล์ model.rtw

2.8.3 เมคยูทิลิตี้ (make utility)

เป็นเครื่องมือในการคอมไพล์และเชื่อมโค้ดที่ถูกสร้าง เพื่อสร้างการปฏิบัติ งานเราจะสามารถกำหนดรูปร่างของเมคได้โดยไปทำการเปลี่ยนแปลงที่ซิสเตมเทมเพลตเมคไฟล์ (system template makefile)

2.8.4 เอสฟังก์ชัน

เอสฟังก์ชันสามารถทำให้เราสร้างโค้ดได้เองลงในซิมูลิงค์ เราสามารถใส่เอสฟังก์ชันโค้ดโดยตรงลงในโค้ดที่ถูกสร้างไว้ เรียกวิธีนี้ว่าอินลายนิ่ง ทีแอลซี เป็นหัวใจสำคัญขึ้นการสร้างขึ้นมาใช้ในตัวของมันเอง โดยวิธีการสร้างนี้เกี่ยวข้องกับอินลายนิงเอสฟังก์ชัน

2.8.5 ขั้นตอนการสร้างในเรียลไทม์เวอร์คชอพ

สร้างไฟล์ model.rtw ขึ้นมาซึ่งเป็นตัวแทนของซิมูลิงค์บล็อกในไฟล์นี้จะมีข้อมูลต่างเช่น ค่าของตัวแปร ความกว้างของเวคเตอร์เวลาในการสุ่มค่าและลำดับการปฏิบัติการณ์ของบล็อก

โมเดลนั้น หลังจากที่ได้ไฟล์นี้แล้วต้องร้องขอให้ทีแอลซีมาแปลงไฟล์นี้ให้เป็นโค้ดทีเฉพาะเจาะจงโดยทีแอลซีจะเริ่มอ่านไฟล์ model.rtw หลังจากนั้นจะคอมไพล์และปฏิบัติการณ์กับคำสั่งในทาร์เกตไฟล์ ผลลัพธ์ของทีแอลซีจะได้ซอสโค้ด (source code) ที่เป็นฟังก์ชันของซิมูลิงค์บล็อก

ขั้นตอนไปคือการสร้างไฟล์ system ขึ้นจากเทมเพลทเมคไฟล์ (system.tmf)ไฟล์model.mk ถูกสร้างโดยการบันทึกหัวข้อของ system.tmf และขยายออก หลังจากได้ model.mk คำสั่งเมคจะสร้างการปฏิบัติการณ์ และยังสามารถดาวน์โหลดการปฏิบัติการณ์ลงในฮาร์ดแวร์เป้าหมาย และหลังจากดาวน์โหลดแล้วถ้าเราใช้โหมดภายนอก (external mode) คุณยังสามารถติดต่อนำสัญญาณกลับมาที่ซิมูลิงค์เพื่อปรับแต่งค่าต่างๆในขณะที่ทำงานได้

2.8.6 ความคิดพื้นฐานที่ควรรู้ในเรียลไทม์เวอร์คชอพ

เรียลไทม์ทั่วไป (generic realtime)

- สิ่งแวดล้อมสำหรับการจำลองโมเดลทีเป็นขั้นตอนตายตัว ในโหมดเดี่ยว หรือโหมดหลายหน้าที่ (single mode or multitashking mode)
- ในความที่สามารถแสดงค่าของโค้ดได้
- เป็นจุดเริ่มต้นของการทำเป้าหมาย

เป้าหมาย

ในการใช้เรียลไทม์เวอร์คชอพเราต้องตัดสินใจว่าจะนำโค้ดที่จะสร้างไปวางไว้ในสิ่งแวดล้อมใด ในสิ่งแวดล้อมนั้นจะเรียกว่าทาร์เกตโฮส (host) ที่ซึ่งแมทแลป ซิมูลิงค์และเรียลไทม์เวอร์คชอพทำงานอยู่ ในการใช้เครื่องมือ build ที่อยู่บนโฮสเราจะได้โค้ด และการปฏิบัติการณ์ที่ทำงานบนระบบของเป้าหมาย

เราจำเป็นต้องนิยามว่า เป้าหมายที่ใช้เป็นชนิดใด ตารางข้างล่างแสดงตัวอย่างของเป้าหมาย, ไฟล์เป้าหมายของระบบ และเทมเพลตเมคไฟล์ของแต่ละสิ่งแวดล้อม

ตารางที่ 2.1 ตารางแสดงทาร์เกตไฟล์ที่ใช้ และเทมเพลตเมคไฟล์ที่ใช้

Target	System Target File	Template Makefile
Generic real-time	grt.tlc	grt_default.tmf
Embedded-C	ert.tlc	ert_default.tmf
DOS(4GW) real-time	drt.tlc	drt_watc.tmf
Tornado(VxWorks) real- times	tornado.tlc	tornado.tmf
Rapid Simulation	rsim.tlc	rsim_default.tmf

ไฟล์ทีแอลซี (target language compiler file)

เป็นไฟล์ที่ใช้สำหรับคอมไพล์และปฏิบัติการ ซึ่งจะอธิบายถึงการสร้างโค้ดให้กับเป้าหมายของเรา เรียลไทม์เวอร์คชอพใช้ไฟล์ทีแอลซีแปลงซิมูลิงค์โมเดลให้เป็นโค้ดซิสเตมทาร์เกตไฟล์เป็นทางเข้าของโปรแกรมทีแอลซี เพื่อสร้างการปฏิบัติการณ์ส่วนบล็อกทาร์เกตไฟล์จะบอกว่าโค้ดมองหาบล็อกซิมูลิงค์ได้อย่างไร

กระบวนการสร้าง

กระบวนการนี้ถูกควบคุมด้วย make_rtw ซึ่งจะถูกขอร้องเมื่อคุณกดปุ่มบิลด์ในเรียลไทม์เวอร์คชอพ โดยอันแรก make_rtw จะแปลงบล็อกไดอะแกรมเพื่อสร้างไฟล์ model.rtw และอันดับต่อไป make_rtw จะบอกให้ทีแอลซีมาสร้างโค้ด เราจะต้องกำหนดซิสเตมทาร์เกตไฟล์ที่อยู่บนหน้าเรียลไทม์เวอร์คชอพสำหรับทีแอลซีด้วย ต่อจากนั้น make_rtwจะสร้างเมคไฟล์ model.mkขึ้นมาจากเทมเพลตเมคไฟล์(template make file) ในหน้าต่างเรียลไทม์เวอร์คชอพ และสุดท้ายถ้าโฮสที่เราใช้งานอยู่ตรงกันกับโฮสในเทมเพลตเมคไฟล์ เมคจะถูกเรียกขึ้นเพื่อสร้างโปรแกรมจากโค้ด

เทมเพลตเมคไฟล์

เรียลไทม์เวอร์คชอพใช้เทมเพลตเมคไฟล์ทำการสร้างปฏิบัติการณ์จากโค้ด เพื่อให้สะดวกเทมเพลตเมคไฟล์จะมีนามสกุลเป็น.tmf และมีชื่อตามเป้าหมาย ตัวอย่างเช่น grt_unix.tmf เป็นเทมเพลตเมคไฟล์สำหรับ unix ไฟล์ที่ถูกสร้างจากเทมเพลตเมคไฟล์จะเลียนแบบแต่ละบรรทัดจากเทมเพลตเมคไฟล์ และขยายโทเค่น (token) เข้าไปในเมคไฟล์ ชื่อของไฟล์ที่ถูกสร้างขึ้นคือ model.mk และไฟล์ model.mk จะผ่านไปยังเครื่องมือเมคเพื่อสร้างการปฏิบัติการจากชุดของไฟล์เหล่านั้น เครื่องมือจะแสดงวันที่ของการเกี่ยวเนื่องกันระหว่างเป้าหมายกับซีไฟล์ และจะสร้างไฟล์เป้าหมายใหม่ถ้าจำเป็น

รูปร่างลักษณะของเทมเพลตเมคไฟล์

คุณสามารถกำหนดกระบวนการสร้างโดยการเปลี่ยนแปลงที่เทมเพลตเมคไฟล์ อาจทำได้โดยการก๊อปปี้กระบวนการนั้น ไดเรคทอรี่ของตัวที่ทำงานอยู่และแก้ไขใหม่ หรืออีกทางหนึ่งคุณสามารถกำหนดการปฏิบัติการณ์ของเทมเพลตเมคไฟล์ โดยไปกำหนดที่ตัวเลือกของการสร้างให้กับการสร้างคำสั่ง make_rtw ดังตัวอย่าง

Make_rtw OPT_OPTS=-g

การระบุพารามิเตอร์ของโมเดล

คุณจะสามารถเปลี่ยนแปลงพารามิเตอร์ของโมเดลที่ควบคุมเรื่องการจำลองระบบเช่นเวลาเริ่มและเวลาหยุดโดยไปเลือกที่ไดอะลอกบอกซ์ซิมูเลชันพารามิเตอร์ (Simulation parameter) พารามิเตอร์เหล่านี้ถูกใช้โดยตรงกับการสร้างโค้ดและโปรแกรม ดังนั้นก่อนที่จะสร้างโค้ดและโปรแกรมจะต้องแน่ใจว่าพารามิเตอร์ได้ถูกเซตอย่างถูกต้องในไดอะลอกบอกซ์ซิมูเลชัน

2.9 แมสกิง (masking)

เป็นการซิมูลิงค์หรือเชื่อมต่อกันของไดอะลอกบอกซ์ และไอคอน (หน่วยย่อย) หรือเป็นระบบย่อย ลักษณะของแมสกิง

- ในโมเดลของคุณสามารถแทนด้วยไดอะลอกบอกซ์หลายตัวในระบบย่อยเพียงตัวเดียว สามารถเปิดทีละบล็อกและได้ค่าพารามิเตอร์บนพารามิเตอร์แมสไดอะลอกบอกซ์ และในระบบย่อยของแมสกิง
- มีคำอธิบายและช่วยแนะนำผู้ใช้ในบล็อกคำบรรยาย (block description) คำบรรยายของพารามิเตอร์ (parameter filed labels) และตัวช่วย (help text)
- ให้กำหนดคำสั่งและประมวลผลในค่าพารามิเตอร์
- สร้างบล็อกย่อย
- ทำการปรับปรุงระบบย่อยโดยใช้การซ่อนไว้หลังการอินเตอร์เฟส

การแซมเปิ้ลแมสของระบบย่อย

ระบบย่อยอย่างง่ายๆ เป็นสมการเส้นตรง Y= mx+b โดยแสดงการเปิดบล็อกระบบย่อยในวินโดวส์ซึ่งสมการ mx+b ประกอบด้วย บล็อกเกนคือ ความชัน (m) ค่าคงที่ที่จุดตัดแกนคือ b เราสามารถดับเบิ้ลคลิก ในการเปิดระบบย่อยซึ่งภายในพารามิเตอร์จะประกอบด้วย 2 ค่าที่ตั้งไว้ และเราสามารถสร้างค่าพารามิเตอร์เองได้เรียกค่าเหล่านี้ว่า แมสพารามิเตอร์

2.9.1 การสร้างแมสสำหรับระบบย่อย

- กำหนดค่าแมสไดอะลอกบอกซ์พารามิเตอร์ เช่นตัวอย่างของความชันและค่าคงที่
- กำหนดชื่อตัวแปร เพื่อสำหรับเก็บค่าของแต่ละพารามิเตอร์
- กรอกข้อมูลของบล็อกประกอบด้วย การอธิบายและตัวช่วยบล็อก
- ทำการวาดภาพด้วยคำสั่งและสร้างบล็อกไอคอน
- เติมคำสั่งและตัวแปรต่างๆในดรออิงคอมมานต์

2.9.2 การสร้างแมสไดอะลอกบอกซ์พรอมต์ (Mask Dialog Box Prompts)

เราสามารถเลือกระบบย่อยบล็อก และเลือกการสร้างแมสจากเมนู อิดิต (Edit) ส่วนบนของไดอะลอกบลอกซ์จะมีหน้าที่อินนิเชียลไลเซชัน ในหน้านี้จะประกอบด้วย

- ค่าพรอมต์คือข้อความเลเบลที่ใช้แทนค่าพารามิเตอร์
- ชนิดของการควบคุม (control type) คือรูปแบบของการควบคุมค่าพารามิเตอร์โดยการใส่ค่าหรือเลือกค่า
- ตัวแปรคือชื่อของตัวแปรที่จะเก็บค่าพารามิเตอร์นั้น

โดยปกติแล้วเราสามารถอ้างอิงค่าพารามิเตอร์หลายๆตัวในพรอมต์ เช่น ความชันก็กำหนดสโลปพารามิเตอร์และค่าคงที่กำหนดเป็นจุดตัด ทั้งค่าความชันและค่าคงที่ล้วนอยู่ในหมวดอิดิต ซึ่งผู้ใช้สามารถเปลี่ยนแปลงค่าและสามารถเก็บค่าตัวแปรไว้ในแมสเวอร์คสเปซ (maskworkspace) จะเก็บเฉพาะตัวแปรเท่านั้น เช่นความชันมีตัวแปรเป็น m ซึ่งจะมีค่าความชันมาจากแมสเวอร์คสเปซและจะแสดงในส่วนของอิดิตเตอร์ (editor) หลังจากที่คุณสร้างค่าพารามิเตอร์และปิดแล้วสามารถดับเบิ้ลคลิกเพื่อใส่ค่าตัวเลข เช่น 3 ลงในความชัน 2 ลงในค่าคงที่

- การสร้างบล็อกการบรรยายและตัวช่วย สองส่วนที่จะอยู่ในส่วนของหน้าเอกสาร
- การสร้างบล็อกไอคอนการสร้างบล็อกย่อย mx+b ซึ่งมีการซิมูลิงค์ภายในระบบย่อยซึ่งค่าไอคอนนี้เราสามารถพล็อตเส้นตรงที่มีความชัน และในหน้าไอคอนประกอบด้วย

1. ดรออิงคอมมานต์ (Drawing commands)

- the drawing command เป็นการพล็อตเส้นจากจุด (0,0) ถึง (0,m)
- สามารถเข้าถึงตัวแปร โดยการใส่ค่าต่างๆลงในความชัน ซึ่งไอคอนจะมีการพล็อตให้โดยอัตโนมัติ

2. ไอคอนพรอพเพอตี้ ประกอบด้วยแบบปกติกับการจัดการที่กำหนดจุดโดยจาก (0,0) ไปยัง(1,1)

สรุปในการสร้างแมสไดอะลอกบอกซ์พรอมพต์ ต้องประกอบด้วย

- กำหนดค่าไดอะลอกบอกซ์พรอมต์ (ค่าพารามิเตอร์ต่างๆ) และตัวอักษร
- กำหนดการบรรยายแมสบล็อก (คำอธิบายฟังก์ชั่นการทำงาน) และตัวช่วยแนะนำการใช้งาน
- กำหนดคำสั่งและสร้างแมสบล็อกไอคอน (หน่วยย่อย)

2.9.3 แมสอิดิตเตอร์ (mask editor)

รูปภายนอกบล็อกการเลือกบล็อกและสร้างแมสจากเมนูอิดิต ซึ่งในส่วนของเดอะแมสอิดิตเตอร์ประกอบด้วย 3 เพจต่างๆกันดังนี้

1. อินนิเชียลไลเซชันเพจ คุณต้องกำหนดค่าพารามิเตอร์พรอมต์ ชื่อตัวแปรของค่าพารามิเตอร์และคำสั่ง

2. ไอคอนเพจ สามารถกำหนดบล็อกไอคอน

3. หน้าเอกสาร โดยกำหนดชนิดของแมสกำหนดคำอธิบายบล็อกและส่วนช่วยการใช้บล็อกและมี 5 ปุ่ม ด้านล่างของ แมสอิดิตเตอร์

- **ปุ่มแอพพลายด์** สร้างและเปลี่ยนแปลงข้อมูลในทุกหน้าในการเปิดอิดิตเตอร์
- **ปุ่มรีเวอร์ต** การแทนค่าให้กลับมาสู่ค่ามาตรฐาน
- **ปุ่มอันแมส** ไม่กระทำการแมสและปิดแมสอิดิตเตอร์ ข้อมูลที่ได้รับ การสร้างหรือการเซตไว้จะไปกระทำการเมื่อได้มีการปิดและไม่สามารถค้นหาอีกได้
- **ปุ่มช่วยเหลือ** แสดงเนื้อหาในบทนี้
- **ปุ่มปิด (close)** เมื่อทำการเซตแมสทุกหน้าแล้วก็ต้องปิดแมสอิดิตเตอร์ด้วย

ในการมองระบบภายใต้แมส ปราศจาก อันแมสกิง (unmasking) การเลือกบล็อกและเลือก Look under Masking จากเมนูอิดิต ถ้าบล็อกอยู่ในระบบย่อย กำลังที่จะไม่มีผลเกิดขึ้น

2.9.4 การเริ่มต้นหน้า (The Initialization page)

ผู้ใช้สามารถกำหนดค่าพารามิเตอร์ในระบบแมสและสามารถสร้างแมสให้ติดต่อกับผู้ใช้ โดยค่าพารามิเตอร์ใน หน้าเริ่มต้นจากตัวอย่าง mx+b ในการเริ่มต้นหน้าประกอบด้วย

1. พรอมต์ และกลุ่มตัวแปร

การกำหนดข้อมูลพรอมต์จะช่วยให้ผู้ใช้เต็มหรือเลือกค่าพารามิเตอร์และจะแสดงรายการพรอมต์ เมื่อตัวแปรถูกกำหนดค่าแล้ว เลือกรูปแบบการควบคุมและเก็บค่าตัวในตัวแปร ถ้ามีการกำหนดชนิดเป็นการประมาณค่า (Evaluate) ค่าสตริงเกนจะถูกตีค่าโดยแมทแลปและผลจะถูกกำหนดค่าตัวแปร ถ้าเป็นชนิดลิเทอรัลสตริง (literal string) จะไม่ถูกตีค่าโดยแมทแลป แต่จะให้สตริงที่มีข้อความว่า “gain” ถ้าคุณต้องการกำหนดสตริงเพื่อตีค่าในการเลือกชนิดลิเทอรอล โดยใช้คำสั่ง eval matlab ใน อินนิเชียลไลซ์คอมมานต์ (initialization command)

2. การสร้างฟิวด์พรอมพต์ (field prompt)

ในการสร้างพรอมต์แรกในรายการ โดยการเอนเทอร์พรอมต์ (enter prompt) ลงในพรอมต์ฟิวด์(prompt field) ตัวแปรจะมีค่าพารามิเตอร์ในฟิวด์ตัวแปรและเลือกรูปแบบการควบคุมและกำหนดชนิด

3.การแทรกพรอมต์

- เลือกพรอมต์ และต้องการแทรกพรอมต์ใหม่โดยการคลิกปุ่ม Addของปุ่มทางซ้ายมือ
- ใส่ค่าพรอมต์ลงใน พรอมต์ฟิวด์ใส่ตัวแปรโดยคงค่าพารามิเตอร์ในฟิวด์ตัวแปร

4. การแก้ไข พรอมต์

- เลือกพรอมต์ในรายการพรอมต์, ชื่อตัวแปร, รูปแบบควบคุม และชนิดใช้ในฟิวด์ตามรายการ

5. การลบพรอมต์

- เลือกพรอมต์ที่ต้องการลบ
- คลิกปุ่มดิลิต(delete) ทางด้านซ้ายรายการ

6. การย้ายพรอมต์

- เลือกพรอมต์ที่ต้องการย้าย
- ย้ายค่าขึ้นไป 1 ตำแหน่งคลิกอัพ หรือย้ายค่าลงมา 1ตำแหน่งคลิกปุ่มดาวน์โหลด Default ค่าของแมสบล็อกพารามิเตอร์ (Masked Block parameters)

การบักทึกบล็อกไว้ในส่วนของบล็อกค่าไลบรารี่ตามลำดับดังนี้

- ลากบล็อกไปไว้ใน บล็อกไลบรารี่
- เปิดบล็อกแล้วเติมค่าที่ต้องการและปิดไดอะลอกบอกซ์
- บันทึกบล็อกไลบรารี่

คำสั่ง Initialization

คำสั่งของตัวแปรจะถูกกำหนดในแมสเวอร์คสเปซ ตัวแปรนี้จะถูกดำเนินตามคำสั่ง Initialization กำหนดแมสโดยเขียนบล็อกในระบบย่อยเขียนคำสั่งในการวาดบล็อกไอคอนซิมูลิงค์จะดำเนินการเมื่อ

- โมเดลถูกโหลด
- การซิมูเลชันจะเริ่มหรือบล็อกไดอะแกรมจะถูกสร้างขึ้นไอคอนของบล็อกจะถูกวาดหรือพล็อตตามคำสั่ง initialization
- คำสั่งจะผิดพลาดในแมทแลปพิจารณาที่ functions, ตัวกระทำ, ตัวแปร และจะไม่สามารถเข้าถึงเวอร์คสเปซ เมื่อมีเซมิโคลอนบนคำสั่งวินโดวส์

แมสเวอร์คสเปซ

ซิมูลิงค์จะสร้างพื้นที่ถูกเรียกว่าเวอร์คสเปซ เมื่อแมสค์มีคำสั่ง initialization หรือมีพรอมต์และตัวแปร แมสบล็อกไม่สามารถเข้าถึงฐานเวอร์คสเปซหรือเวอร์คสเปซอื่นๆ ในเวอร์คสเปซประกอบด้วยตัวแปรในพารามิเตอร์และตัวแปรจากคำสั่ง initialization ตัวแปรเหล่านี้ในเวอร์คสเปซจะถูกเข้าถึงโดยแมสบล็อก ถ้าบล็อกเป็นระบบย่อยหนึ่งเราสามารถเข้าถึงทุกบล็อกในระบบย่อย แมสเวอร์คสเปซที่เป็นอะนาลอกจะมีการใช้เอมไฟล์ฟังก์ชันคิดจำนวนการพูดลงในไดอะลอกบอกซ์ของอันเดอร์ลายอิงค์บล็อก และคำสั่ง initialization บนอิดิตเตอร์ของฟังก์ชัน เอมไฟล์การใช้อะนาลอกจะอยู่ในรูปฟังก์ชัน

2.10 แผงวงจร JX-2148

2.10.1 คุณสมบัติเด่นของบอร์ด JX-2148

ใช้ไมโครคอนไทรลเลอร์ 32 บิตในตระกูล ARM7TDMI-S เบอร์ LPC2148 ของNXP ซึ่งมีหน่วยความจำ โปรเกรมแบบแฟลชความจำ 512 กิโลไบต์ มีหน่วยความจำสำหรับประมวลผลขนาด 128 บิต เพื่อช่วยให้สามารถประมวลผลข้อมูลขนาด 32 บิตได้ด้วยความเร็วสูง ทำงานกับความถี่สัญญาณนาฬิกาได้ถึง 6OMHz มีหน่วยความจำสแตติกแรม 40 กิโลไบต์

- มีคอนเน็กเตอร์ JTAG สำหรับรองรับการดีบักจากเครื่องมือดีบักเกอร์ภายนอก
- มี USB คอนเน็กเตอร์แบบ B พร้อม LED แสดงการทำงานเชื่อมต่อตรงกับขาพอร์ต USB ของ LPC2148
- มีวงจรเชื่อมต่อพอร์ตอนุกรม RS-232 จำนวน 2 ช่อง โดยUART-0 ใช้สื่อสารข้อมูลแล้วดาวน์โหลดโปรแกรมแบบ ISP ส่วน UART-1 สามารถเลือกสื่อสารข้อมูลกับพอร์ตอนุกรมหรือโมดูลบลูทูธ ESD -200 (ต้องจัดหาเพิ่มเติม)
- มีซ็อกเก็ตสาหรับติดตอ MMC หรือ SD card
- มีคอนเนกเตอร์ PS2 สำหรับรองรับการเชื่อมต่อกับคีย์บอร์ดหรือเมาส์
- สวิตช์กดติดปล่อยดับ 2 ชุด พร้อมต่อตัวต้านทานพูลอัป
- LED แสดงผลการทำงาน 2 ดวงพร้อมตัวต้านทานจำกัดกระแส
- มีตัวต้านทานปรับค่า ได้ต่อกับขาพอร์ตอินพุตอะนาลอกเพื่อทดสอบการทำงานของโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล
- มีบัซเซอร์สำหรับขับเสียง
- มีแผงต่อวงจรหรือเบรดบอร์ดขนาดเล็ก จำนวน 170 จุดต่อ
- มีคริสตอล 32.768kHZ และแบตเตอรี่เพื่อรองรับระบบเวลานาฬิกาจริงภายในไมโครคอนโทรลเลอร์
- ใช้ไฟเลี้ยงได้ย่านกว้างตั้งแต่ +6V ถึง +12V แนะนำให้ใช้ในย่าน +6V ถึง +9V กระแส 100mA โดยบนบอร์ดมีวงจรควบคุมไฟเลี้ยงคงที่ที่ +3.3V สำหรับเลี้ยงไมโครคอนโทรลเลอร์
- มีวงจรจัดขั้วแรงดันไฟเลี้ยง จึงทำให้สามารถใช้งาน กับอะแดปเตอร์ไฟตรงที่มีการจัดขั้วแบบใดก็ได้

2.10.2 ความต้องการของระบบ

ในการใช้งานบอร์ด JX 2148 ต้องเตรียมเครื่องมือและอุปกรณ์ดังนี้

1. คอมพิวเตอร์ PC ที่ติดตั้งระบบปฏิบัติการวินโดวส์ XP service pack 2 และควรมีพอร์ตอนุกรมว่างอย่างน้อย 1 พอร์ต ถ้ามีถึง 2 พอร์ตจะช่วยให้การทดลองเรียนรู้สะดวกขึ้นอย่างมาก พร้อมกับติดตั้งซอฟต์แวร์ mVision3 หรือ Keil ARM tool kit ของ Keil (www.keil.com) จะเป็นรุ่นทดลองใช้งาน (evaluation) หรือรุ่นสมบูรณ์

2. ตัวแปลงสัญญาณพอร์ต USB เป็นพอร์ตอนุกรม RS-232 ในกรณีที่คอมพิวเตอร์ใช้งานไม่มีพอร์ต อนุกรม (แนะนำ UCON 232S ของ inex

3. อะแดปเตอร์ +6Vdc 100mA ขึ้นไปสูงสุดไม่เกิน +12Vdc

4. สายต่อพอร์ตอนุกรมแบบที่หัวท้ายเป็นคอนเน็กเตอร์ DB-9 ตัวผู้และตัวเมีย มีการต่อสายสัญญาณครบทั้ง 9 เส้น แบบขาต่อขา

5. โมดูลบลูทูธ ESD-200 ในกรณีที่ต้องการทดสอบการทำงานกับระบบสื่อสารไร้สายแบบบลูทูธ

6. USB บลูทูธดองเกิล ในกรณีที่ต้องการทำกับระบบสื่อสารไร้สายแบบบลูทูธร่วมกับคอมพิวเตอร์

7. คีย์บอร์ดแบบ PS/2 ในกรณีที่ต้องการทดสอบการทำงานระหว่าง LPC2148 กับคีย์บอร์ด

8. โมดูลกราฟิก LCD รุ่น GLCD5110 สำหรับทดลองใช้งานกับโมดูลกราฟิก LCD

9. สาย USB แบบ AB มีความยาวไม่เกิน 3 เมตร

10. ULINKUSB -JTAG adaptor อุปกรณ์เสริมสำหรับการทำดีบักผ่านทางคอนเน็กเตอร์ JTAG บนบอร์ด JX-2148

2.10.3 การทำงานของบอร์ด JX-2148

การทำงานโดยรวมของบอร์ด JX-2148 สามารถแบ่งการทำงานออกเป็น 3 ส่วนหลักคือ

(1) ส่วนของ ไมโครคอนโทรลเลอร์ LPC2148

(2) ภาคจ่ายไฟ

(3) วงจรอินพุตเอาต์พุต

ในส่วนของไมโครคอนโทรลเลอร์ประกอบด้วยไมโครคอนโทรลเลอร์ LPC2148 และวงจรกำเนิดสัญญาณนาฬิกา ซึ่งมีด้วยกัน 2 ชุดคือ 12 MHz สำหรับการทำงานหลัก และ 32.768KHz สำหรับวงจรฐานเวลานาฬิกาจริง ภาคจ่ายไฟ บอร์ด JX-2148 สามารถรับแรงดันไฟเลี้ยงจากภายนอกตั้งแต่ +6V จนถึง 16V โดยบนบอร์ด ได้บรรจุวงจรควบคุมไฟเลี้ยงคงที่ที่ +3.3V สำหรับ

เลี้ยงวงจรโดยส่วนใหญ่รวมทั้ง ไมโครคอนโทรลเลอร์ LPC2148ด้วย และ +5V สำหรับวงจรเชื่อมต่ออุปกรณ์ในระบบบัส แบบ PS2

วงจรอินพุตเอาต์พุตของบอร์ด JX-2148 ประกอบด้วย LED สวิตช์กดติดปล่อยดับ ตัวต้านทานปรับค่าได้สำหรับทดสอบโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล วงจรเชื่อมต่อพอร์ตอนุกรม RS-232 จำนวน 2 ช่อง คอนเน็กเตอร์เชื่อมต่ออุปกรณ์ผ่านแจ็ค PS/2, SD การ์ดซ็อกเก็ตสำหรับติดต่อกับ SD การ์ด, คอนเน็กเตอร์เชื่อมต่อพอร์ต USB คอนเน็กเตอร์ JTAG สำหรับติดต่อกับ JTAG debugger อาทิ ULINK ของ Keil และคอนเน็กเตอร์สำหรับติดต่อกับโมดูลบลูทูธ ESD-200 นอกจากนั้นยังมีพอร์ตว่างสำหรับเชื่อมต่อกับอุปกรณ์อื่นๆ ได้และมีแผงต่อวงจรหรือเบรดบอร์ดขนาดเล็กจำนวนจุดต่อ 170 จุดสำหรับสนับสนุนการต่อวงจรทดลองด้วยLPC2148 มีโมดูล SPI และ SSP อย่างละ 1 ชุด โดยโมดูล SSP สามารถกำหนดให้ทำงานในโหมด SPI ได้ บนบอร์ด JX-2148 ได้จัดสรรให้ขาพอร์ตของโมดูล SSP ทำงานในโหมด SPI เพื่อเชื่อมต่อกับซ็อกเก็ตการ์ดหน่วยความจำแบบ MMC หรือ SD ส่วนโมดูล SPI อีกชุดหนึ่งกำหนดเป็น SPI0 สำรองไว้เป็นจุดต่อใช้งานอิสระ เช่น เดียวกับโมดูลเชื่อมต่ออุปกรณ์ในระบบบัส 2 สาย (TWI: Two Wire interface) ซึ่งสามารถใช้เชื่อมต่อกับอุปกรณ์ในระบบบัส I^2C ได้

ส่วนการทดสอบโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล บนบอร์ดได้จัดเตรียมตัวต้านทานปรับค่าได้ต่อเข้ากับขาพอร์ต P0.30 สามารถทดสอบได้ทันที

ทางด้านการทดสอบพอร์ตอินพุตเอาต์พุตนั้น ได้จัดสรร LED 2 ดวงพร้อมตัวต้านทานจำกัดกระแสต่อเข้ากับขาพอร์ต P0.21 และ P0.22 ส่วนอุปกรณ์อินพุตนั้นได้ต่อสวิตช์กดติดปล่อยดับพร้อมตัวต้านทานพูลอัปเข้าที่ขาพอร์ต P0.28 และ P0.29 การขับสัญญาณเสียงเลือกใช้บัซเซอร์โดยต่อเข้ากับขาพอร์ต P0.12 และ P0.13วงจรกำเนิดสัญญาณนาฬิกามี 2 ส่วนคือ กำเนิดสัญญาณนาฬิกาหลักใช้คริสตอล PS/2 ที่นำมาต่อร่วมด้วย ไม่ว่าจะเป็นคีย์บอร์ดหรือเมาส์ โดยมีไอซีเบอร์ LM2931-5.0 ควบคุมแรงดันคงที่ 5V

การเชื่อมต่อพอร์ต USB ใช้ขา USBD+ และ USBD- ต่อผ่านตัวต้านทาน 27 Ω เพื่อป้องกันกระแสไหลเกิน ส่วนแรงดัน +5V ที่มาจากพอร์ต USB นั้นต่อเข้ากับขา 0.23/VBUS ส่วนการควบคุมการเชื่อมต่อกับพอร์ต USB นั้นจะใช้ขาพอร์ต P0.31 โดยเมื่อต้องการเชื่อมต่อกับพอร์ต USB ต้องทำให้ขา P0.31 นี้เป็นลอจิก “0” ดังนั้นจึงสามารถควบคุมการต่อและยกเลิกการติดต่อพอร์ต USB ผ่านทางซอฟต์แวร์ได้

จั๊มเปอร์ REF ใช้เลือกแรงดันอ้างอิงของโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล LPC2148 ปกติต่อกับ +3.3V แต่ถ้าหากต้องการใช้แรงดันอ้างอิงจากภายนอกทำได้โดยปลดจั๊มเปอร์ที่ตำแหน่ง REF ออก แล้วต่อแรงดันอ้างอิงภายนอกเข้าที่ขากลางของคอนเน็กเตอร์ที่ใช้รับจั๊มเปอร์ REF

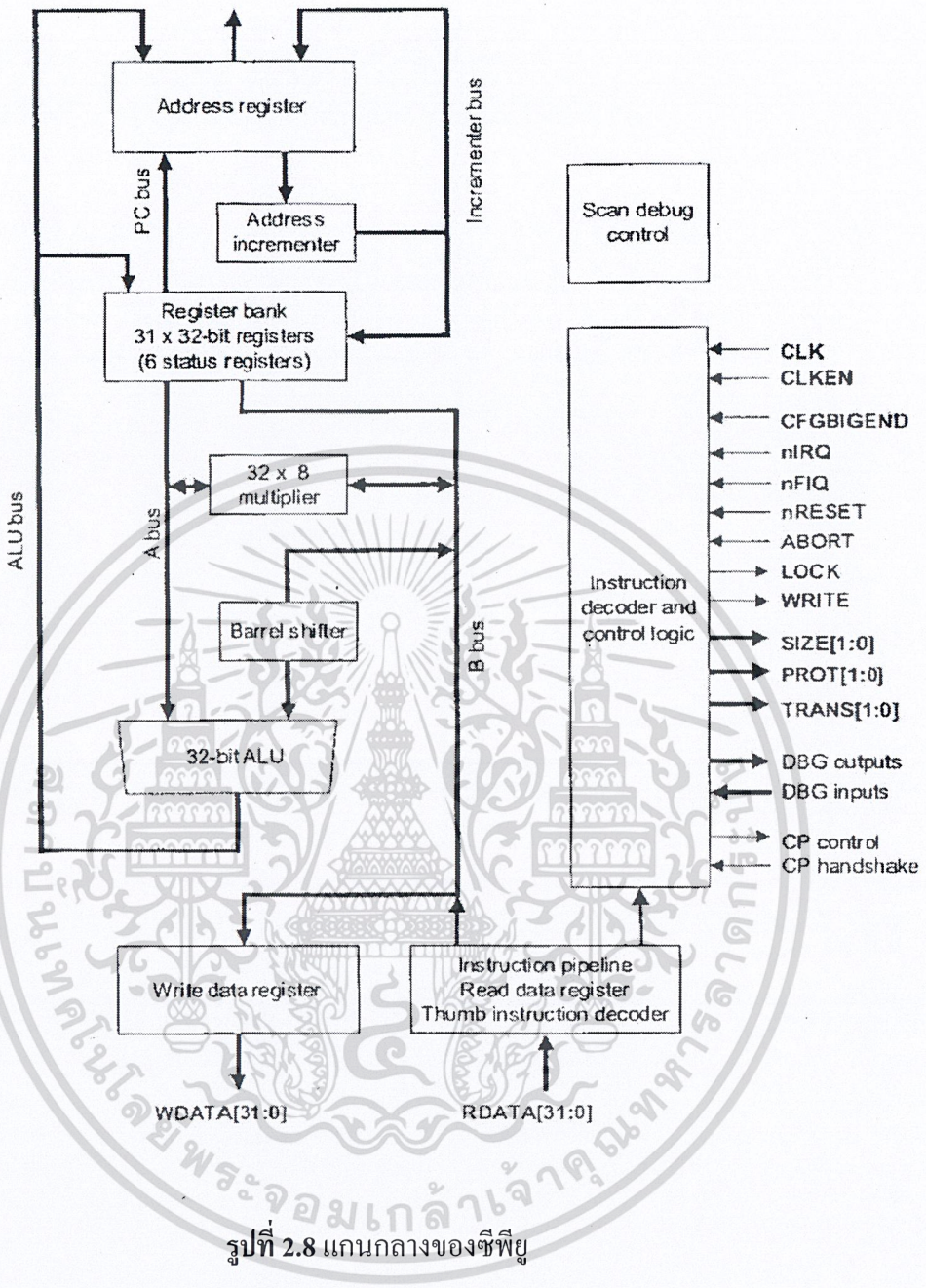
จั๊มเปอร์ DEBUG EN. ใช้เอ็นเอเบิลการดีบักผ่านทางพอร์ต JTAG ปกติต่อไว้ทางขวาในตำแหน่ง Enable เนื่องจากบอร์ด JX-2148 ได้สงวนขาพอร์ตในส่วนนี้เอาไว้เพื่อการดีบักโดยเฉพาะ

2.11 Microcontroller ARM7

2.11.1 สถาปัตยกรรมของซีพียู ARM7

สถาปัตยกรรมของ ARM7 เป็นซีพียูแบบ RISC (Reduce lnstrucnon SetComputer) ขนาด 32 บิต ภายในมีบัสขนาด 32 บิตตัวเดียวที่ใช้สำหรับรับส่งข้อมูลและคำสั่ง ชุดคำสั่งจะมีขนาด32บิต ลงที่ในขณะที่ข้อมูลสามารถเลือกได้ว่าจะมีขนาด 8, 6 หรือ 32 บิต โดยแสดงแกนหลัก (core) ของซีพียู ARM7 ได้ดังรูปที่ 2.8

- โครงสร้างของ ARM7 จะเป็นแบบที่เรียบง่าย มีชุดคำสั่งไม่มากนัก ประหยัดพื้นที่สารกึ่งตัวนำที่ใช้สร้าง ประหยัดพลังงาน
- สถาปัตยกรรมของ ARM7 จะเป็นนเบบ load-and-store ในการประมวลผลข้อมูลใดๆ ต้องกระทำผ่านทางรีจิสเตอร์ เริ่มต้นด้วยการโหลดค่าจากหน่วยความจำเก็บในรีจิสเตอร์นำค่ามาประมวลผล เสร็จแล้วจะเขียนค่าเก็บในหน่วยความจำดังเดิม
- รีจิสเตอร์ของ ARM7 ที่ใช้งานได้สำหรับผู้ใช้มีทั้งหมด 16 ตัวคือ R0-R15 โดยทุกตัวมีขนาด 32 บิตโดย R0-R12 เป็นรีจิสเตอร์ทั่วไปที่ไม่ได้กำหนดหน้าที่การทำงานพิเศษ ส่วน R12 ทำหน้าที่เป็น stack pointer (SP) R14 ทำหน้าที่เป็น link register (LR) และ R15 ทำหน้าที่เป็น Program Counter (PC)



รูปที่ 2.8 แกนกลางของซีพียู

2.11.2 คุณสมบัติหลักของ ARM โปรเซสเซอร์

ในปัจจุบันมีผู้ผลิตไมโครคอนโทรลเลอร์และไมโครโปรเซสเซอร์ที่ใช้ ARM โปรเซสเซอร์หลายราย อาทิ Atmel, Philips, Samsung, ST ซึ่งแต่ละผู้ผลิตได้บรรจุความสามารถพิเศษต่างๆ แตกต่างกันไป แต่ยังคงต้องมีคุณสมบัติหลักของ ARM โปรเซสเซอร์ไว้ นั่นคือ

- มีกลุ่มของรีจิสเตอร์จำนวนมาก เพื่อรองรับการทำงานที่หลากหลาย
- ใช้สถาปัตยกรรมแบบ Load-store
- คำสั่งทั้งหมดต้องใช้รูปแบบ 3 แอดเดรส และมีขนาด 32 บิต

2.11.3 สถาปัตยกรรมแบบ Load-Store

เป็นสถาปัตยกรรมที่สนับสนุนการทำงานของคำสั่งในลักษณะที่เน้นกระบวนการเป็นหลัก โดยเมื่อมีการกระทำคำสั่ง ข้อมูลของคำสั่งจะถูกคัดลอกจากหน่วยความจำไปยังรีจิสเตอร์ กระบวนการนี้เรียกว่า การอ่านคำสั่ง หรือ Load instructions และกระบวนคัดลอกข้อมูลจากรีจิสเตอร์กลับไปยังหน่วยความจำ จะถูกเรียกว่า การเก็บคำสั่งหรือ store instructions โดยปกติสำหรับ CISC (Complexed Instrument Set Computer) หรือไมโครคอนโทรลเลอร์ทั่วไปจะยอมให้ค่าจากหน่วยจำสามารถรวมเข้ากับค่าในรีจิสเตอร์ แต่สำหรับ ARM โปรเซสเซอร์จะไม่รองรับการทำงานระหว่างหน่วยความจำ (memory-to-memory operations) คำสั่งการทำงานของ ARM โปรเซสเซอร์จึงมีลักษณะการทำงานได้สามกรณีดังนี้

- คำสั่งประมวลผลข้อมูล (Data processing instruction) การทำงานลักษณะนี้จะมีการใช้และเปลี่ยนแปลงเฉพาะค่าของรีจิสเตอร์ ยกตัวอย่าง คำสั่งบวกค่าระหว่างรีจิสเตอร์ 2 ตัวเข้าด้วยกัน แล้วนำค่าผลลัพธ์เก็บลงในรีจิสเตอร์
- คำสั่งถ่ายทอดข้อมูล (Data processing instructions) เป็นการคัดลอกข้อมูลจากหน่วยความจำลงในรีจิสเตอร์ (Load) หรือคัดลอกค่าจากรีจิสเตอร์ลงในหน่วยความจำ (store) ในรูปแบบการทำงานแบบนี้ยังสามารถใช้ในการกำหนดค่าของระบบ และเปลี่ยนค่าของหน่วยความจำด้วยค่าของรีจิสเตอร์
- คำสั่งควบคุมการดำเนินไปของโปรแกรม (Control flow instructions) โดยปกติการเอ็กซิคิวต์คำสั่งจะกระทำเรียงตามลำดับแอดเดรสที่กำหนดในหน่วยความจำ คำสั่งควบคุมการดำเนินไปของโปรแกรมจะทำให้การเอ็กซิคิวต์สามารถเปลี่ยนตำแหน่งแอดเดรสได้ หรือบันทึกค่าแอดเดรสที่ต้องการกลับมาหลังจากกระโดดไปทำงานที่อื่นเพื่อให้สามารถทำงานตามลำดับของโปรแกรมหลักได้ โดยใช้กลุ่มคำสั่งกระโดดและเชื่อมโยง (branch and link instructions) รวมไปถึงการตรวจสอบรหัสโปรแกรมของระบบ (supervisor call) ได้

2.11.4 คุณสมบัติชุดคำสั่งของ ARM โปรเซสเซอร์

คำสั่งของ ARM โปรเซสเซอร์ทั้งหมดมีขนาด 32 บิต ยกเว้นกลุ่มคำสั่งที่เกี่ยวข้องกับการทำงานไมโครคอนโทรลเลอร์ Thumb ซึ่งมีขนาด 16 บิต มีคุณสมบัติที่ควรทราบดังนี้

- รองรับกับสถาปัตยกรรมแบบ load-store
- ใช้รูปแบบประมวลผลข้อมูล 3 แอดเดรส ประกอบด้วย 2 แอดเดรสของรีจิสเตอร์ที่กระทำคำสั่ง และแอดเดรสของรีจิสเตอร์ที่ใช้เก็บผลลัพธ์
- สามารถทำงานร่วมกับรีจิสเตอร์ได้หลายตัว

- สามารถกระทำคำสั่งเลื่อนข้อมูลและประมวลผลคณิตศาสตร์และลอจิกได้หนึ่งคำสั่ง และใช้เวลาเพียง 1 ไซเกิลสัญญาณนาฬิกา
- เป็นคำสั่งแบบเปิดที่สามารถเพิ่มเติมรีจิสเตอร์ที่เกี่ยวข้องและรูปแบบของข้อมูลได้

2.11.5 ไปป์ไลน์

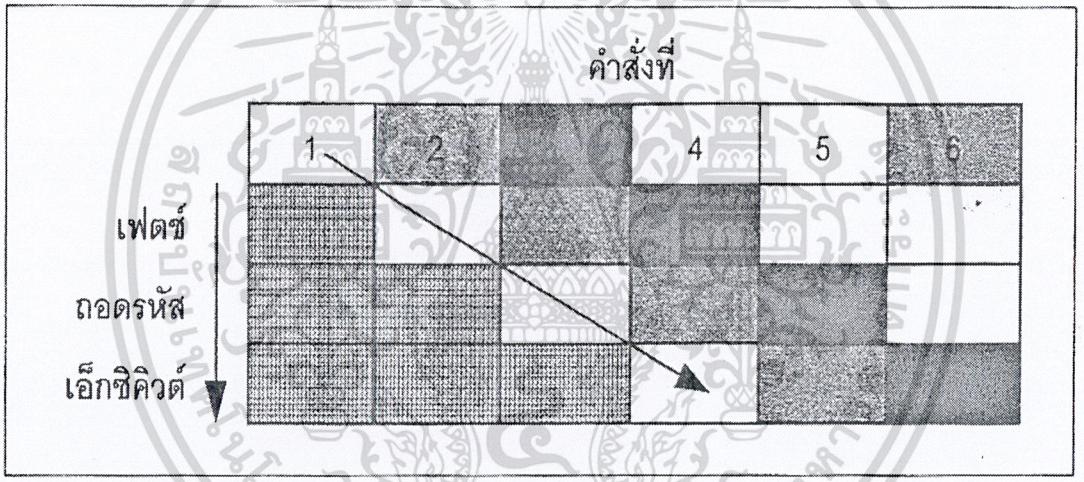
หัวใจสำคัญในการทำงานของ ARM7 คือ กระบวนการไปป์ไลน์ (instruction pipeline) ซึ่งมีด้วยกัน 3 ระดับ ในรูปที่ 2-1 แสดงกระบวนการทำงานของไปป์ไลน์ใน ARM7 อย่างง่าย จะเห็นได้อย่างชัดเจนว่า ในขณะที่กำลังเอ็กซิคิวต์คำสั่งที่ 1 ซีพียูก็กำลังถอดรหัสคำสั่งที่ 2 พร้อมกันนั้นยังทำการเฟตซ์คำสั่งที่ 3 ด้วย

ดังนั้นจึงทำให้ ARM7 สามารถทำงานได้อย่างรวดเร็ว และยิ่งสามารถรับสัญญาณนาฬิกาที่มีความถี่สูงได้ (กรณีLPC2148 ทำงานได้ถึง60MHz) ก็ยิ่งทำให้ ARM7 สามารถรองรับการประมวลผลข้อมูลขนาดใหญ่ได้ด้วยความเร็วสูง

2.11.6 คุณสมบัติทางเทคนิคที่สำคัญ

- หน่วยความจำสแตติกแรม 40 กิโลไบต์แบ่งเป็น 32 กิโลไบต์สำหรับใช้งานทั่วไป และ 8 กิโลไบต์สำหรับการติดต่อแบบเข้าถึงโดยตรง (DMA: Direct Memory Access) เมื่อติดพอร์ต USB
- หน่วยความจำแบบแฟลชความสูง 512 กิโลไบต์ ลบ-เขียนใหม่ได้ 100,000 รอบ รักษาข้อมูลได้นาน20ปี สามารถเร่งความสามารถในการประมวลผล โดยใช้บัฟเฟอร์พิเศษขนาด 128บิต เพื่อรองรับการทำงานที่ความถี่สัญญาณนาฬิกาสูง 60 MHz
- สามารถโปรแกรมในวงจรหรือ In-System Programming (ISP) รวมทั้งการโปรแกรมในขณะทำงานหรือ In-Application Programming (IAP) ผ่านซอฟต์แวร์บูตโหลดเดอร์ที่บรรจุอยู่ภายในชิป สามารถลบหน่วยความจำทั้งหมดภายในเวลา 400 มิลลิวินาที และใช้เวลาในการโปรแกรม 256 ไบต์ภายในเวลา 1 มิลลิวินาที
- รองรับการดีบักและติดตามผลการทำงานในแบบเรียลไทม์ผ่านทางขาพอร์ตฟังก์ชันพิเศษและ JTAG Debugger
- มีโมดูลเชื่อมต่อพอร์ต USB 2.0 แบบ Full-speed พร้อมทั้งหน่วยความจำแรมภายในชิป 8 กิโลไบต์สามารถติดต่อได้ในแบบ DMA เพื่อรองรับการติดต่อผ่านพอร์ต USB
- มีโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล ความละเอียด 10บิต จำนวน 2 ชุดคือ โมดูล ADC0 มีอินพุต 6 ช่องและมีโมดูล ADC1 มีอินพุตอะนาลอก 14ช่อง ใช้เวลาในการแปลงสัญญาณเพียง 2.44 มิลลิวินาทีต่อช่อง

- มีโมดูลแปลงสัญญาณดิจิตอลเป็นอะนาลอก ความละเอียด 10 บิต 1 ช่อง
- มีไทเมอร์ 32บิต 2ชุดที่สามารถรองรับการทำงานเป็นเคาน์เตอร์, ตรวจจับสัญญาณหรือแคปเจอร์และเปรียบเทียบสัญญาณได้ 4 ช่องต่อชุด, รองรับการสร้างสัญญาณPWM อีก 6 ช่อง และมีวอตช์ด็อกไทเมอร์
- มีระบบฐานเวลานาฬิกาจริงหรือเรียลไทม์คล็อก (RTC) ใช้แบตเตอรี่แบบลิเธียมขนาด3V สำหรับการสำรองข้อมูลเวลา รวมทั้งแยกคริสตอลกำลังงานต่ำ 32 KHz ออกมาเพื่อกำหนดจังหวะการทำงานเฉพาะสำหรับเรียลไทม์คล็อก
- มีโมดูลสื่อสารข้อมูลผ่านพอร์ตอนุกรมหรือ UART2ชุด โดย UART0ยังใช้ในการดาวน์โหลดโปรแกรมแบบ ISP โดยใช้แบบบูตโหลดเดอร์ที่ติดตั้งมาภายในตัวไมโครคอนโทรลเลอร์แล้ว



รูปที่ 2.9 แสดงกระบวนการทำงานของไปป์ไลน์ใน ARM7 โปรเซสเซอร์

- มีโมดูลติดต่อกับอุปกรณ์ระบบบัส I^2C หรือระบบบัส 2 สาย จำนวน 2 ชุด มีความเร็วในการทำงาน 400 กิโลบิตต่อวินาที
- มีโมดูล SPI รองรับการติดต่ออุปกรณ์แบบอนุกรม
- กำหนดนัยสำคัญของเวกเตอร์อินเตอร์รัปต์ได้และมีอินพุตสัญญาณอินเตอร์รัปต์จากภายนอก21ขา
- กำหนดความถี่สัญญาณนาฬิกาด้วยกระบวนการ PLL โดยความถี่สูงสุดคือ60MHzจากคริสตอล 12 MHz และใช้ PLLx5 หากไม่ใช้ PLL สามารถใช้คริสตอลภายนอกได้ 1 MHz ถึง 25 MHz

- มีโหมดประหยัดพลังงาน ทั้งแบบไอเดิลและพาวเวอร์ดาวน์
- โปรเซสเซอร์ออกจากโหมดประหยัดพลังงานด้วยการอินเตอร์รัปต์จากภายนอก และจากบราวเอาต์ (BOD)
- ไฟเลี้ยงใช้ได้ในย่าน +3.0 ถึง +3.6V รองรับระดับสัญญาณ +5V ที่ขาพอร์ตได้

2.11.7 ไดอะแกรมการทำงานโดยรวมของ LPC2148

ในรูปที่ 2-10 แสดงไดอะแกรมการทำงานในภาพรวมของ LPC2148 หัวใจสำคัญคือ ARM โปรเซสเซอร์ ซึ่งใน LPC2148 ใช้ ARM7TDMI-S โดยมีการเชื่อมต่อโมดูลต่างๆ ภายในผ่านทางบัสซีพียูสมรรถนะสูงหรือ Advance High-performance Bus (AHB) ส่วนโมดูลอุปกรณ์หรือเพอริเฟอรัลจะส่งผ่านข้อมูลกับบัสเพอริเฟอรัลหรือ VLSI Peripheral Bus(VPB) ข้อมูลจากโมดูลทั้งหมดจะส่งมายังส่วนเชื่อมต่อบัส AHB และ VPB เพื่อถ่ายถอดไปยังซีพียูต่อไปด้วยการจัดสรรบัสแยกกันเช่นนี้ทำให้การทำงานโดยรวมเร็วขึ้น

นอกจากนั้น ARM7TDMI-S โปรเซสเซอร์ยังรองรับการทำงานในแบบไมโครคอนโทรลเลอร์ Thumb ด้วยดังนั้น ARM7TDMI-S โปรเซสเซอร์จึงมีชุดคำสั่ง 2 ชุดคือ ชุดคำสั่ง ARM มาจรฐาน 32 บิต และชุดคำสั่งThumb 16 บิต ในการทำงานด้วยคำสั่งของไมโครคอนโทรลเลอร์ Thumb จะทำให้ขนาดของโปรแกรมควบคุมเหลือเพียง 65% ของการใช้คำสั่ง ARM ปกติ ทั้งยังสามารถเพิ่มประสิทธิภาพในการทำงานขึ้นถึง 160% เมื่อทำงานกับหน่วยความจำในระบบ 16 บิต ส่วนในการทำงานด้วยคำสั่ง ARM นั้นจะมีข้อดีในด้านความเร็วในการทำงานเพราะสามารถเรียกใช้ทรัพยากรต่างๆ ได้อย่างเต็มที่ รวมถึงรองรับการทำงานประมวลผลสัญญาณดิจิตอลหรือ DSP algorithm แม้ว่าจะทำให้ขนาดของโปรแกรมใหญ่ แต่จะทำงานได้เร็วกว่าโหมด Thump ถึง 30%

2.11.8 หน่วยความจำภายใน LPC2148

LPC2148 มีหน่วยความจำ 2 แบบคือ หน่วยความจำแบบแฟลชความจุ 512 กิโลไบต์ ที่เป็นไปได้ทั้งหน่วยความจำโปรแกรมและข้อมูล โดยการโปรแกรมสามารถกระทำผ่านกระบวนการโปรแกรมในวงจร ผ่านโมดูล UART0 หรือ โมดูลสื่อสารพอร์ตอนุกรมชุด 0 โดยทำงานร่วมกับบูตโหลดเดอร์เฟิร์มแวร์ที่บรรจุอยู่ภายในชิป จะใช้เนื้อที่ของหน่วยความจำแฟลช 12 กิโลไบต์ ทำให้เหลือไว้ทำงาน 500 กิโลไบต์ หน่วยความจำอีกส่วนหนึ่งคือ

หน่วยความจำสแตติกความจุ 40 กิโลไบต์ แบ่งออกเป็น 32 กิโลไบต์ สำหรับเก็บโปรแกรมและข้อมูลทั่วไป และ 8 กิโลไบต์สำหรับรองรับ DMA เมื่อใช้งานโมดูล USB โดยหน่วยความจำส่วนนี้สามารถเก็บโปรแกรมและข้อมูลได้ด้วยเช่นกัน

2.11.9 ตัวควบคุมเวกเตอร์อินเตอร์รัปต์

ตัวควบคุมเวกเตอร์อินเตอร์รัปต์ (Vectored Interrupt Controller: VIC) ซึ่งทำหน้าที่รับสัญญาณร้องขออินเตอร์รัปต์ทั้งหมดที่เกิดขึ้น โดยแบ่งเป็นการร้องขออินเตอร์รัปต์แบบเร็ว (Fast Interrupt Request: FIQ), การร้องขอเวกเตอร์อินเตอร์รัปต์ (Vectored Interrupt Request: IRQ) และการร้องขออินเตอร์รัปต์แบบไม่มีเวกเตอร์ (Non-vectored IRQ) โดยจะมีการกำหนดระดับความสำคัญของแหล่งกำเนิดสัญญาณอินเตอร์รัปต์ต่างๆด้วยกระบวนการทางซอฟต์แวร์ และสามารถเปลี่ยนแปลงได้ตลอดเวลา

สัญญาณร้องขออินเตอร์รัปต์ที่มีนัยสำคัญสูงสุด

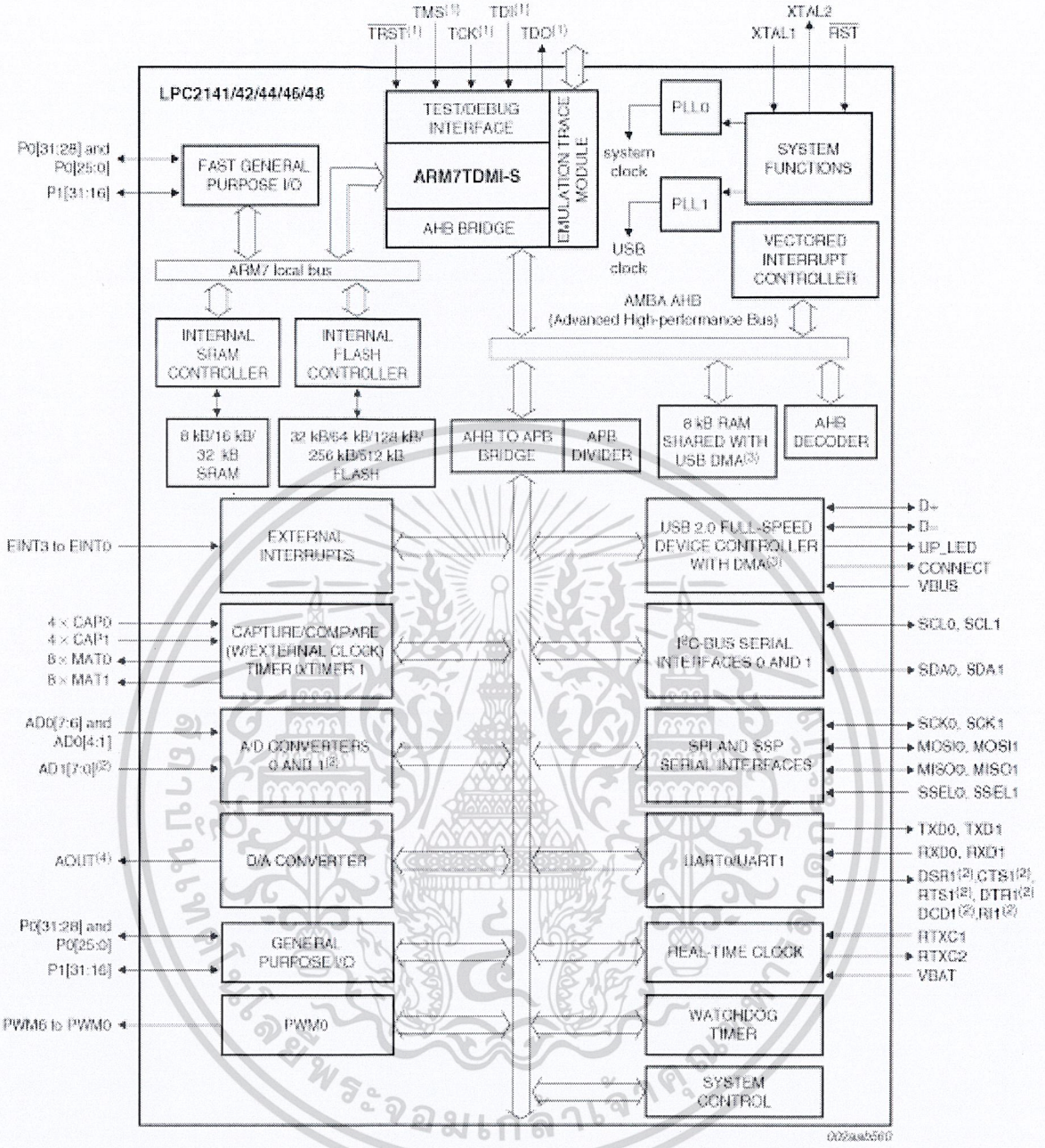
คือ FIQ แต่ ARM โปรเซสเซอร์ก็ยอมให้เกิดสัญญาณ FIQ ได้มากกว่า 1 แหล่งพร้อมกันได้ แต่การเข้าไปจัดการก็ยังมีระดับความสำคัญที่กำหนดได้และ ARM โปรเซสเซอร์ก็จะเลือกเข้าไปจัดการก่อนหลังอยู่ดี

สัญญาณร้องขอเวกเตอร์อินเตอร์รัปต์หรือ Vectored IRQ มีนัยสำคัญในลำดับถัดมา

มีแหล่งกำเนิดสัญญาณอินเตอร์รัปต์ในกลุ่มนี้ 16 แหล่งจึงมีเวกเตอร์อินเตอร์รัปต์ทั้งสิ้น 16 ตำแหน่ง ซึ่งเรียกว่าสล็อต (slot) โดยสล็อต 0จะมีนัยสำคัญสูงสุดและสล็อต 15 มีนัยสำคัญต่ำสุด

สัญญาณร้องขออินเตอร์รัปต์แบบไม่มีเวกเตอร์มีนัยสำคัญต่ำสุด

ในทางปฏิบัติ VIC 1 จะรวมสัญญาณร้องขออินเตอร์รัปต์ทั้งแบบมีและไม่มีเวกเตอร์เพื่อสัญญาณร้องขออินเตอร์รัปต์หรือ IRQส่งไปยัง ARM โปรเซสเซอร์ จากนั้นโปรแกรมบริการอินเตอร์รัปต์จะทำงานแล้วอ่านค่ารีจิสเตอร์จาก VIC แล้วกระโดดไปทำงานยังแอดเดรสหรือตำแหน่งที่กำหนด ถ้าหากยังไม่มีการระบุมาอย่างชัดเจน VIC จะเลือกแอดเดรสของสัญญาณอินเตอร์รัปต์ที่มีระดับความสำคัญสูงสุด แล้วกระโดดไปยังแอดเดรสนั้น ในกรณีที่มีการระบุแอดเดรสมาซึ่งอาจเป็นแอดเดรสของสัญญาณอินเตอร์รัปต์แบบไม่มีเวกเตอร์ก็ได้ ก็จะกระโดดไปทำงานยังแอดเดรสที่กำหนดนั้น และที่โปรแกรมย่อยบริการอินเตอร์รัปต์สามารถอ่านค่าของรีจิสเตอร์ใน VIC เพื่อตรวจสอบว่า สัญญาณอินเตอร์รัปต์ใดที่เกิดขึ้น ซึ่งก็คือการตรวจสอบบิตแฟลกของรีจิสเตอร์ควบคุมการอินเตอร์รัปต์นั่นเอง ด้านการตอบสนองอินเตอร์รัปต์จากภายนอก LPC2148 มีอินพุตรับสัญญาณอินเตอร์รัปต์ทั้งสิ้น 9 ช่องสามารถตรวจจับสัญญาณได้ทั้งแบบขอบขาและระดับลอจิก

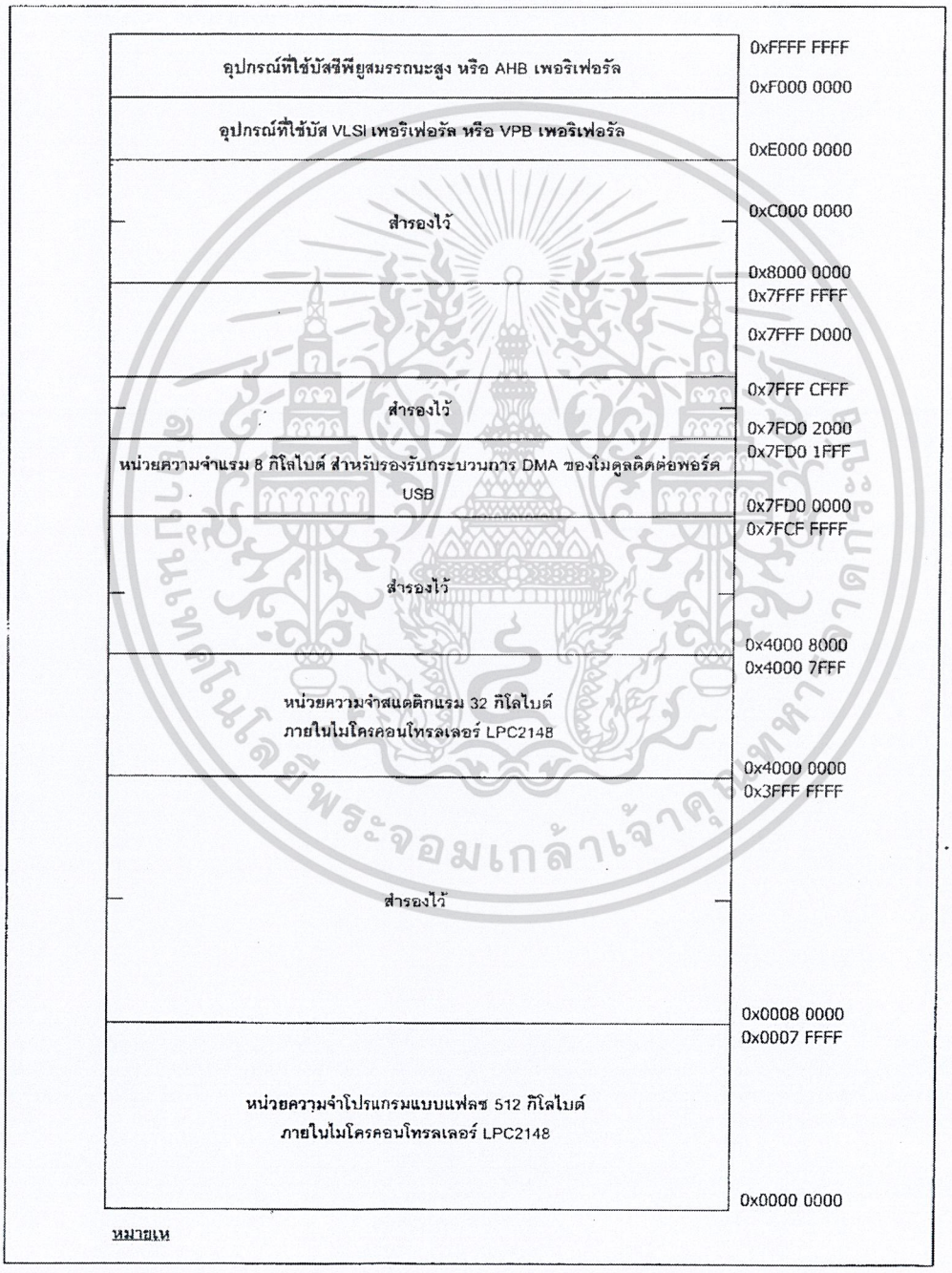


รูปที่ 2.10 ไดอะแกรมการทำงานของ LPC2148 ไมโครคอนโทรลเลอร์ 32 บิตตระกูล ARM7

2.11.10 พอร์ตอินพุตเอาต์พุตความเร็วสูง

ใน LPC2148 มีขาพอร์ตจำนวนมาก และแต่ละขาพอร์ตยังสามารถกำหนดการเชื่อมต่อเพื่อทำงานกับโมดูลเพอริเฟอรัลภายใน LPC2148 ด้วย หลังจากเกิดการรีเซตขาพอร์ตทั้งหมดจะถูกกำหนดเป็นอินพุต และถ้ามีการเอ็นเอเบิลการดีบัก ขาพอร์ต JTAG จะทำงานในฟังก์ชันของ JTAG ทันทีเช่นเดียวกับขาพอร์ตติดตามการทำงานหรือ TRACE หากได้รับการเอ็นเอเบิลไว้ ขาพอร์ตนั้นๆก็จะทำงานในฟังก์ชันของการ TRACE

ในกรณีที่ขาพอร์ตได้รับการกำหนดให้ทำงานเป็นพอร์ตอินพุตเอาต์พุตดิจิตอล การทำงานของขาพอร์ตจะได้รับการควบคุมจากรีจิสเตอร์ GPIO ซึ่งสามารถเข้าถึงได้ในระดับไบต์ นอกจากนั้นยังสามารถเร่งการทำงานของ GPIO ได้ด้วยการย้ายไปทำงานบนบัสภายในเพื่อให้เวลาที่ใช้ในการทำงานของขาพอร์ตเร็วที่สุดเท่าที่เป็นไปได้ โดยการเขียนข้อมูลไปยังขาพอร์ตสามารถกระทำให้สำเร็จได้ภายในหนึ่งคำสั่ง



รูปที่ 2.11 การจัดสรรหน่วยความจำภายใน LPC2148

โมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล

ใน LPC2148 บรรจุโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอลที่มีความละเอียดในการแปลงสัญญาณ 10 บิต จำนวน 2 ชุดคือ ADC0 และ ADC1 โดยADC0 มี 6 ช่องและ ADC1 มี 8 ช่อง มีคุณสมบัติการทำงานที่ควรทราบดังนี้

- กระบวนการแปลงสัญญาณใช้วิธีซักเซสทีฟ แอ็ปพร็อกซิเมชั่น
- สามารถรับสัญญาณอินพุตได้ในย่าน 0V ถึง Vref (โดย Vref มีค่า 2.0V<Vref<Vdda)
- มีอัตราการสุ่มสัญญาณสูงถึง 400,000 ตัวอย่างต่อวินาที
- ทุกครั้งที่มีการแปลงสัญญาณข้อมูลที่ได้ จะส่งไปยังรีจิสเตอร์เก็บค่าทันทีเพื่อลดการอินเตอร์รัปต์

โมดูลแปลงสัญญาณดิจิตอลเป็นอะนาลอก

ใน LPC2148 บรรจุโมดูลแปลงสัญญาณดิจิตอลเป็นอะนาลอกที่มีความละเอียดในการแปลงสัญญาณ 10 บิต จำนวน 1 ชุดและมีเพียง 1 ช่องเอาต์พุต แรงดันอะนาลอกเอาต์พุตสูงสุดจะเท่ากับแรงดันอ้างอิงหรือ VREF มีคุณสมบัติการทำงานที่ควรทราบดังนี้

- ความละเอียดในการแปลงสัญญาณ 10 บิต
- มีวงจรบัฟเฟอร์ที่เอาต์พุต
- รองรับการทำงานในโหมดประหยัดพลังงาน
- สามารถเลือกความเร็วในการแปลงสัญญาณได้

โมดูลติดต่อพอร์ต USB 2.0

LPC2148 บรรจุโมดูลติดต่อพอร์ต USB 2.0 สามารถทำงานเป็นอุปกรณ์ USB ติดต่อกับโฮสต์ ซึ่งโดยปกติเป็นคอมพิวเตอร์ ด้วยอัตราการถ่ายทอดข้อมูลสูงถึง 12 เมกะบิตต่อวินาที โดยในโมดูลUSB นี้ประกอบด้วยรีจิสเตอร์เชื่อมต่อ (register interface), ส่วนจัดการเชื่อมต่ออนุกรม (serial interface engine), บัฟเฟอร์ของเอ็นด์พอยต์ (endpoint buffer memory) และตัวควบคุมการเข้าถึงหน่วยความจำโดยตรง (DMA controller) ส่วนจัดการเชื่อมต่ออนุกรมทำการถอดรหัสขบวนข้อมูล USB แล้วเขียนข้อมูลนั้นไปยังบัฟเฟอร์ของเอ็นด์พอยต์ที่ต้องการสถานะของการทำงานไม่ว่าจะเป็นการถ่ายทอดข้อมูลสมบูรณ์หรือผิดพลาดก็ตาม จะแสดงสถานการณ์ทำงานผ่านรีจิสเตอร์แสดงสถานะและเกิดการอินเตอร์รัปต์ขึ้นหากมีการเอ็นเอเบิลไว้ ตัวควบคุมการเข้าถึงหน่วยความจำโดยตรง หรือ DMA controller จะทำหน้าที่ถ่ายทอดข้อมูลระหว่างบัฟเฟอร์ของเอ็นด์พอยต์ไปยังหน่วยความจำแรมของ USB ได้โดยตรง ทำให้สามารถถ่ายทอดข้อมูลได้ด้วยความเร็วสูง

คุณสมบัติเด่นของโมดูลติดต่อพอร์ต USB 2.0 ของLPC2148 มีดังนี้

- ทำงานเข้ากันได้กับการติดต่อพอร์ต USB2.0 แบบความเร็วเต็มที่หรือ full-speed
- รองรับเอ็นด์พอยต์แบบฟิสิคัล32 เอ็นด์พอยต์ หรือ แบบลอจิกคัล 16 เอ็นด์พอยต์

- รองรับเอ็นด์พอยต์ควบคุม, บัลก์, อินเตอร์รัปต์ และไอโซโครนัส

- เลือกขนาดแพ็กเก็ตของเอ็นด์พอยต์ได้สูงสุดด้วยกระบวนการทางซอฟต์แวร์ในขณะทำงาน

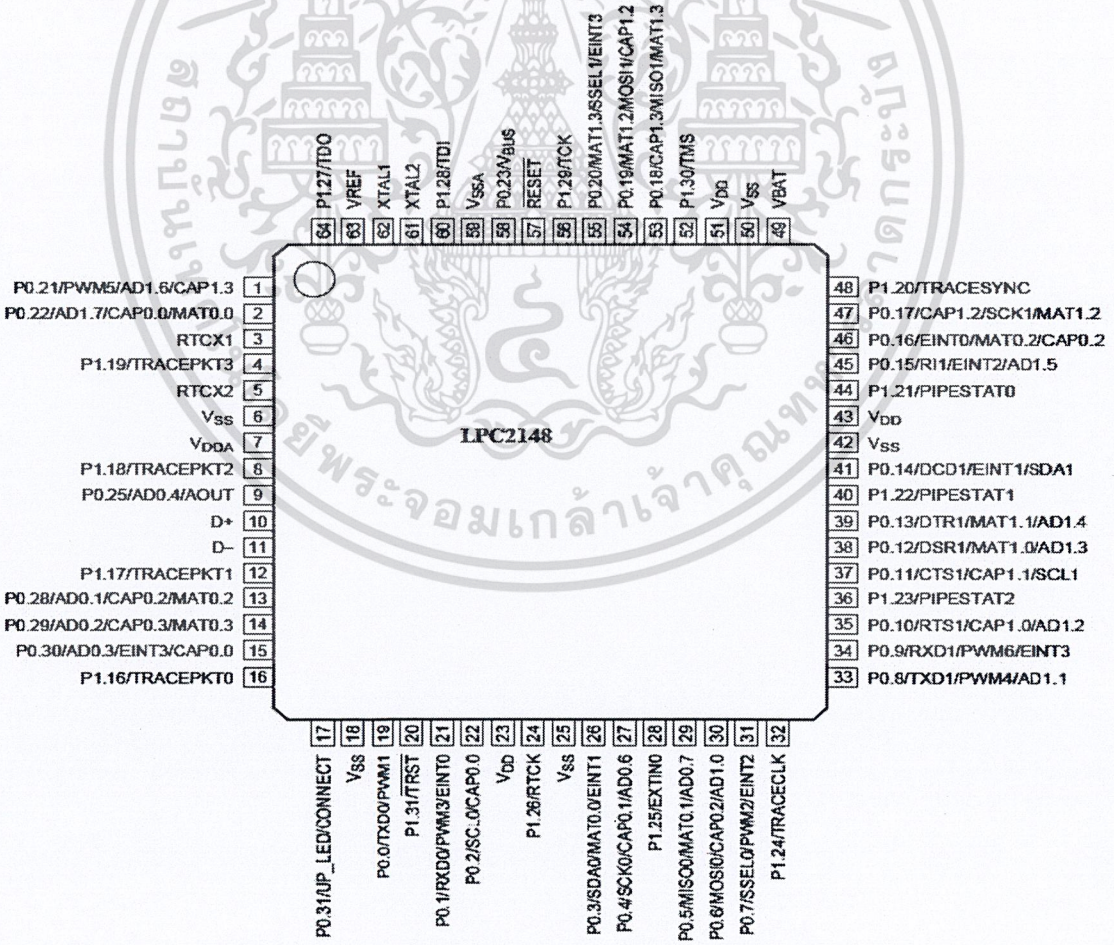
- ขนาดบัฟเฟอร์ของข้อมูลที่ใช้ในงานจะขึ้นอยู่กับชนิดของเอ็นด์พอยต์ และขนาดของแพ็กเก็ตสูงสุด

- แสดงสภาวะการเชื่อมต่อพอร์ต USB ด้วย LED ไม่ว่าจะเป็นการเชื่อมต่อด้วยซอฟต์แวร์ (SoftConnect) และภาวการณ์ต่ออย่างสมบูรณ์ (GoodLink) โดยใช้ขาพอร์ตเดียวกันแสดงการทำงานของทั้ง 2 ฟังก์ชัน

- รองรับการทำงานแบบใช้พลังงานจากบัส

- รองรับการถ่ายทอดข้อมูลแบบ DMA และเลือกการควบคุมการถ่ายทอดข้อมูลจากซีพียูหรือจากDMA ได้

- สามารถเพิ่มบัฟเฟอร์ 2 เท่าเพื่อทำงานกับเอ็นด์พอยต์แบบบัลก์และไอโซโครนัส



รูปที่ 2.12 แสดงการจัดขาสัญญาณของ LPC2148

โมดูลสื่อสารพอร์ตอนุกรม

LPC2148 มีโมดูลสื่อสารพอร์ตอนุกรมหรือ UART (Universal Asynchronour Receiver Transmitter) 2 ชุดคือ UART0 และ UART1 โดย UART0 สามารถใช้ในการดาวน์โหลดโปรแกรมผ่านกระบวนการ ISP ด้วยในขณะที่ UART1 มีการจัดขาสัญญาณพอร์ตอนุกรมแบบสมบูรณ์ภายในโมดูล UART มีส่วนกำเนิดอัตราบอดแยกกันอิสระ และสามารถกำหนดอัตราบอดได้สูงถึง 115,200 บิตต่อวินาทีเมื่อใช้คริสตอลความถี่สูงกว่า 2MHz

คุณสมบัติเด่นของโมดูล UART ของ LPC2148 มีดังนี้

- มีบัฟเฟอร์ FIFO (First In First Out: เข้าก่อนส่งออกก่อน) สำหรับและส่งข้อมูลขนาด 16 ไบต์
- ตัวรับข้อมูล FIFO มีจุดกระตุ้นที่ 1, 4, 8 และ 14 ไบต์
- มีตัวกำเนิดอัตราบอดภายใน
- สามารถควบคุมการดำเนินไปของการส่งข้อมูลได้ทางซอฟต์แวร์
- ใน UART1 มีการจัดสัญญาณพอร์ตอนุกรมครบถ้วนทั้ง DTR, DSR, CTS, RTS, DCD และ RI

โมดูลติดต่ออุปกรณ์ระบบบัส I^2C

ใน LPC2148 มีโมดูลติดต่ออุปกรณ์ระบบบัส I^2C ถึง 2 ชุดนับเป็นความสามารถพิเศษที่น่าสนใจของไมโครคอนโทรลเลอร์ในสมัยใหม่ ขาสัญญาณที่ใช้กับโมดูลนี้มี 2 ขา คือ ขาสัญญาณนาฬิกาอนุกรม หรือ SCI และขาสัญญาณข้อมูลอนุกรมหรือ SDA อัตราการถ่ายทอดข้อมูลสูงสุดคือ 400 กิโลบิตต่อวินาที

โมดูล I^2C ใน LPC2148 สามารถกำหนดให้ทำงานเป็นอุปกรณ์มาสเตอร์หรือสเลฟได้ รวมไปถึงสามารถโปรแกรมความถี่สัญญาณนาฬิกาได้ รองรับการทำงานแบบมัลติมาสเตอร์ รวมทั้งสามารถสื่อสารกับอุปกรณ์ที่มีอัตราการถ่ายทอดข้อมูลไม่เท่ากัน

โมดูลติดต่ออุปกรณ์ระบบบัส SPI และ SSP

LPC2148 มีตัวควบคุมการติดต่ออุปกรณ์แบบ SPI 1 ชุด สามารถที่จะสื่อสารข้อมูลอนุกรม 2 ทิศทางและติดต่อได้คราวละ 1 คู่คือ หนึ่งอุปกรณ์มาสเตอร์และหนึ่งอุปกรณ์สเลฟ โดยมีอัตราการถ่ายทอดข้อมูลสูงสุดเท่ากับ 1/8 ของความถี่สัญญาณนาฬิกา เพื่อให้ LPC2148 สามารถติดต่ออุปกรณ์ที่ใช้การสื่อสารข้อมูลอนุกรมให้ครบถ้วนสมบูรณ์ จึงบรรจุโมดูลติดต่ออุปกรณ์อนุกรมแบบ SSP(Synchronous Serial Peripheral) ที่สามารถใช้ติดต่ออุปกรณ์อนุกรมแบบ SPI,SSI (Synchronous Serial Interface) 4 สายหรือ ไมโครไวร์ (Microwire bus) ทำให้สามารถทำงานกับอุปกรณ์เพอริเฟอรัลที่ใช้การติดต่อแบบบัส SPI ของ Freescale, SSI 4 สายของ Texas Instrument และไมโครไวร์ของ National Semiconductor ดังนั้นจึงอาจมองได้ว่า LPC2148 มีโมดูลติดต่ออุปกรณ์ SPI จำนวน 2 ชุด ในกรณีที่กำหนดให้โมดูล SSP ทำงานในแบบ SPI โมดูล SSP สามารถกำหนดเฟรมข้อมูลได้ตั้งแต่ 4 ถึง 16 บิตและมี FIFO 8 เฟรมสำหรับการรับและส่งข้อมูล

ไทเมอร์/เคาน์เตอร์

ไทเมอร์/เคาน์เตอร์ใน LPC2148 มี 2 ชุดคือ ไทเมอร์ 0 และไทเมอร์ 1 แต่ละตัวมีขนาด 32 บิต สามารถกำหนดให้ทำงานได้หลายหน้าที่ทั้งเป็นไทเมอร์, เคาน์เตอร์โมดูลตรวจจับสัญญาหรือแคปเจอร์ และโมดูลเปรียบเทียบสัญญาณ สัญญาณนาฬิกาของไทเมอร์จะใช้สัญญาณนาฬิกาเพอริเฟอรัลหรือ peripheral clock (PCLK) และ สัญญาณนาฬิกาจากภายนอกในการกำหนดจังหวะการทำงานและเพิ่มค่าของไทเมอร์ สามารถสรุปคุณสมบัติทางเทคนิคที่สำคัญของไทเมอร์ได้ดังนี้

1. มีไทเมอร์/เคาน์เตอร์ 32 บิต จำนวน 2 ชุด พร้อมทั้งสเกลเลอร์ขนาด 32 บิต

2. รับสัญญาณนาฬิกาเพื่อเพิ่มค่าของไทเมอร์จากสัญญาณนาฬิกาเพอริเฟอรัลหรือและสัญญาณนาฬิกาจากภายนอก

3. ในไทเมอร์ 1 ชุดมีอินพุตสำหรับตรวจจับสัญญาณหรือแคปเจอร์ 4 ช่อง สามารถดักจับค่าของไทเมอร์เมื่อมีการเปลี่ยนแปลงลอจิกที่อินพุตเกิดขึ้น และทำให้เกิดการอินเตอร์รัปต์หากได้รับการเอ็นเอเบิลไว้

4. มีรีจิสเตอร์เก็บค่าเปรียบเทียบขนาด 32 บิต จำนวน 4 ตัวที่กำหนดให้ไทเมอร์ทำงานต่อเนื่องไปเมื่อค่าการนับตรงกันหรือเมื่อเกิดอินเตอร์รัปต์จากการที่ค่าของไทเมอร์เท่ากับค่าที่กำหนด

5. หยุดการทำงานของไทเมอร์เมื่อค่าการนับตรงกันหรือเมื่อเกิดอินเตอร์รัปต์จากการที่ค่าของไทเมอร์เท่ากับค่าที่กำหนด

6. รีเซ็ตไทเมอร์เมื่อค่านับตรงกันหรือเมื่อเกิดอินเตอร์รัปต์จากการที่ค่าของไทเมอร์เท่ากับค่าที่กำหนด

7. มีเอาต์พุตสำรับสร้างสัญญาณเมื่อค่าการนัยตรงกันหรือเมื่อเกิดอินเตอร์รัปต์จากการที่ค่าของไทเมอร์เท่ากับค่าที่กำหนด 4 ช่องต่อไทเมอร์ 1 ชุด โดยสัญญาณเอาต์พุตที่เกิดสามารถกำหนดได้ดังนี้

- กำหนดเป็นลอจิก “0”
- กำหนดเป็นลอจิก “1”

วอตช์ด็อกไทเมอร์

เป็นอีกส่วนประกอบหนึ่งที่ไมโครคอนโทรลเลอร์สมัยใหม่ต้องมี วอตช์ด็อกไทเมอร์มีประโยชน์อย่างยิ่งในการป้องกันไม่ให้ไมโครคอนโทรลเลอร์หยุดทำงานอันเนื่องมาจากการทำงานที่ผิดพลาด โดยสามารถเอ็นเอเบิลได้ทางซอฟแวร์ แต่จะต้องรีเซตการทำงานด้วยกระบวนการทางฮาร์ดแวร์ กำหนดตาบเวลาของวอตช์ด็อกไทเมอร์ได้ด้วยการกำหนดค่าให้แก่ไทเมอร์ขนาด 32 บิต คาบเวลาของวอตช์ด็อกไทเมอร์ใน LPC2148 มีค่าเท่ากับ $T_{PCLK} \times 256 \times 4$ ถึง $T_{PCLK} \times 2^{32} \times 4$

ระบบเวลานาฬิกาจริงหรือเรียลไทม์คล็อก

นี่คืออีกคุณสมบัติเด่นของ LPC2148 มี่ช่วยให้ไมโครคอนโทรลเลอร์ตัวนี้มีความสมบูรณ์แบบ โดยระบบเวลานาฬิกาจริงนี้จะทำงานเสมือนหนึ่งมีส่วนจัดการเวลาอิสระ ที่แยกออกมาจากไทเมอร์ ถ้าเทียบกับไมโครคอนโทรลเลอร์เบอร์อื่นๆ ซึ่งไม่มีระบบเวลาตามเวลาจริงอยู่หากต้องการใช้ก็ต้องเพิ่มไอซีหรือโมดูลเข้าไป และเมื่อต้องการอ่านค่าเวลาจะต้องมีโปรแกรม เข้ามาจัดการ ซึ่งอาจต้องใช้เวลาและหน่วยความจำโปรแกรมส่วนหนึ่ง แต่สำหรับ LPC2148 สามารถทำได้สะดวกกว่าเนื่องจากการกำหนดให้ระบบทำงานติดต่อรีจิสเตอร์ไม่กี่ตัว จากนั้นก็เอ็นเอเบิลให้ทำงานระบบเวลานาฬิกาจริงก็จะทำงานไปอย่างอัตโนมัติเมื่อต้องการอ่านค่าเวลาก็เพียงอ้างถึงรีจิสเตอร์ที่เกี่ยวข้องก็จะทราบค่าเวลา ระบบเวลานาฬิกาจริงนี้ให้ข้อมูลเวลาอย่างครบถ้วน คือ ค่าเวลาในหน่วยวินาที,นาที,ชั่วโมง(หรือนาฬิกา),วันที่,วันในสัปดาห์,เดือนและปี การทำงานจะใช้คริสตอลความถี่ 32.768 KHz แยกต่างหาก และต้องการแบตเตอรี่สำรอง +3V เพื่อรักษาค่าเวลาและเลี้ยงให้โมดูลทำงานอย่างต่อเนื่องแม้ว่าจะปลดไฟเลี้ยงของไมโครคอนโทรลเลอร์แล้วก็ตาม

โมดูลสัญญาณ PWM

LPC 2148 มีโมดูลสร้างสัญญาณ PWM มากถึง 6 ช่อง โดยใช้ไทเมอร์หลักของระบบเป็นหลัก และในสัญญาณนาฬิกาเพอริเฟอรัลในการเพิ่มค่าของไทเมอร์ และใช้รีจิสเตอร์เก็บค่าเปรียบเทียบ (Match register : MR) เป็นตัวกำหนดคาบเวลาของสัญญาณ PWM โดยรีจิสเตอร์ MR ตัวหนึ่งใช้ควบคุมไซเกิลของสัญญาณ และรีจิสเตอร์ MR อีกตัวหนึ่งใช้กำหนดขอบขาของสัญญาณ ดังนั้นหากสัญญาณ PWM มีตำแหน่งการเริ่มต้นและสิ้นสุดของสัญญาณในตำแหน่งที่แตกต่างกันไปในแต่ละช่วงเวลา ก็ต้องรีจิสเตอร์ MR เพิ่มตามตำแหน่งที่ต้องการ

สัญญาณ PMW ที่สร้างขึ้นนี้สามารถกำหนดขอบขาของสัญญาณได้ทั้งขอบขาขึ้นและลงจึงสามารถนำไปสร้างสัญญาณนาฬิกา PMW เพื่อควบคุมมอเตอร์ 3 เฟสที่กำหนดได้ทั้งความกว้างและเฟสของสัญญาณรีจิสเตอร์เปรียบเทียบ 2 ตัว จะถูกใช้ในการกำหนดขอบขาของสัญญาณ PMW โดยรีจิสเตอร์ MR ตัวหนึ่งใช้ควบคุมไซเกิลของสัญญาณ และรีจิสเตอร์ MR อีกตัวหนึ่งใช้กำหนดตำแหน่งขอบขาของสัญญาณ ดังนั้นหากสัญญาณ PMW มีตำแหน่งการเริ่มต้นหรือสิ้นสุดของสัญญาณที่แตกต่างกันไปในแต่ละช่วงเวลา ก็ต้องใช้รีจิสเตอร์ MR เพิ่มตามตำแหน่งที่ต้องการ

คุณสมบัติทางเทคนิคโดยสรุปของโมดูลสร้างสัญญาณ PWM มีดังนี้

- มีรีจิสเตอร์เปรียบเทียบ (MR) 7 ตัว เพื่อใช้ในการกำหนดขอบขาเดี่ยวของสัญญาณได้ 6 ตำแหน่งหรือขอบขาคู่ 3 คู่และอีกกำหนดไซเกิลของสัญญาณ
- รีจิสเตอร์ MR มีการทำงานเมื่อค่าของการนับเท่ากับค่าใน MR ดังนี้
- ยังคงให้ไมเทอร์ทำการนับหรือเพิ่มค่าต่อไป
- หยุดการทำงานของไทเมอร์

- รีเซตค่าของไทเมอร์

- สามารถกำหนดตำแหน่งของขอบขาสัญญาณได้อย่างอิสระ

- ใช้ไทเมอร์/เคาน์เตอร์ขนาด 32 บิต ในการทำงานร่วมกับสเกลเลอร์ 32 บิต

- มีเอาต์พุตสำหรับส่งสัญญาณ PWM รวม 6 ช่อง

ส่วนควบคุมระบบ ในส่วนนี้เป็นองค์ประกอบที่สำคัญ และเกี่ยวข้องกับการทำงานของทุกๆ โมดูลในLPC2148

- วงจรคริสตอลออสซิลเลเตอร์

เป็นวงจรกำเนิดสัญญาณนาฬิกาหลักของระบบหรือ f_{osc} ต่อกับคริสตอลภายนอกค่าตั้งแต่ 1MHz ถึง 25 MHz ในกรณีที่ไม่มีการใช้งานเฟสล็อกลูป (PLL) สัญญาณนาฬิกาของซีพียู (CCLK) จะมีค่าความถี่เท่ากับ f_{osc}

- วงจรเฟสล็อกลูป (Phase Lock Loop: PLL)

วงจร PLL จะรับสัญญาณนาฬิกาอินพุตในย่าน 10MHz ถึง 25 MHz มาคูณกับตัวคูณค่าเพื่อสร้างสัญญาณนาฬิกาใหม่สำหรับซีพียูในช่วงความถี่ 10 MHz ถึง 60 MHz ด้วยการทำงานของวงจรควบคุมออสซิลเลเตอร์ด้วยกระแสไฟฟ้าหรือ CCO (Current Controlled Oscillator) ตัวคูณสามารถกำหนดได้ตั้งแต่ 1 ถึง 32 แต่ในทางปฏิบัติไม่ควรใช้ค่าเกิน 6 ทั้งนี้เนื่องจาก LPC2148 มีขอบเขตการทำงานจำกัดไว้สูงสุดที่ 60 MHz การกำหนดให้วงจร PLL ทำงานสามารถกำหนดด้างซอฟต์แวร์

- รีเซ็ตและเวกอัปไทเมอร์

มี 2 สาเหตุ คือ รีเซตจากสัญญาณภายนอกที่ขา RESET และรีเซตจากการทำงานในวอตช์ด็อกไทเมอร์ เมื่อเกิดการรีเซตเวกอัปไทเมอร์จะเริ่มทำงานเพื่อควบคุมการให้วงจรรีเซตภายใน LPC2148 ทำงานต่อไปจนกว่าสัญญาณรีเซตจากภายนอกจะถูกนำออกไป จากนั้นวงจรรีเซตภายในก็จะหยุดทำงาน วงจรกำเนิดสัญญาณนาฬิกาจึงเริ่มทำงาน ซีพียูและเพอริเฟอรัลทั้งหมดจึงเริ่มเข้าสู่สถานะความพร้อมเพื่อเริ่มการทำงานต่อไป

- วงจรตรวจจับแรงดันไฟเลี้ยงต่ำกว่าที่กำหนดหรือวงจรตรวจจับบราวด์เอาต์

LPC2148 มีการตรวจสอบแรงดันไฟเลี้ยง 2 ช่วง ช่วงแรก ถ้าแรงดันไฟเลี้ยงที่ขา V_{DD} ลดลงต่ำกว่า 2.9 V วงจรตรวจจับบราวด์เอาต์ (Brownout Dectector: BOD) จะสร้างสัญญาณอินเตอร์รัปต์ส่งไปยังตัวควบคุมเวกเตอร์อินเตอร์รัปต์อาจตรวจสอบได้โดยการวนอ่านค่าของรีจิสเตอร์ที่เกี่ยวข้องแทน

- ตัวแบ่งสัญญาณบัส VPB

ทำหน้าที่ 2 อย่าง หน้าที่แรกคือ จัดสรรสัญญาณนาฬิกาเพอริเฟอรัล PCLK ลงไปในบัส VPB โดยปกติจะมีความถี่ต่ำกว่าสัญญาณนาฬิกาซีพียู 0.5 ถึง 0.25 เท่า โดยปกติหลังจากการรีเซตตัวแบ่งสัญญาณ VPB จะกำหนดให้ไบอัส VPB ทำงานด้วยอัตรา 0.25 เท่าของสัญญาณนาฬิกาซีพียู

หน้าที่ที่สอง คือ จัดการพลังงานบนบัส หากอุปกรณ์เพอริเฟอรัลใดไม่มีการใช้งาน ก็จะทำให้อุปกรณ์เพอรอเฟอรัลนั้นๆทำงานในโหมดประหยัดพลังงาน

- ส่วนติดต่ออุปกรณ์ทดสอบและดีบัก

LPC2148 รองรับการตรวจสอบการทำงานหรือดีบักผ่านทางพอร์ต JTAG รวมถึงการติดตามการทำงานผ่านทางพอร์ต TRACE โดยใช้พอร์ต P1 เป็นทางผ่านสัญญาณ ดังนั้นการเลือกใช้พอร์ต JTAG หรือ TRACE การเชื่อมต่อกับอุปกรณ์เพอริเฟอรัลอื่นๆควรใช้ขาพอร์ตP0 ทำงานแทน

2.11.11 ตัวอย่างความสามารถของ ARM7

1. เครื่องเล่น MP3

ARM7 32-bit RISC processor core used, the frequency of 60 MHz Philips PNX0102 Decoder chips, TFBGA180 packaging technology, dual-core architecture design (based on ARM+DSP)

2. iPod

3. Nintendo DS

- The technical data:
- ARM9 CPU with 67 MHz
- ARM7 CPU with 22 MHz
- 4MB main storage (RAM)
- 656KB video RAM
- 802.11 wireless LAN
- Screen: 256x192 pixel
- Touchscreen
- GBA module slot
- Stereo sound

4. อื่นๆ อีกมากมาย เช่น โทรศัพท์มือถือ,Pocket PC, หุ่นยนต์ และอุปกรณ์อิเล็กทรอนิกส์ต่างๆ

2.12 พอร์ตอนุกรม RS-232

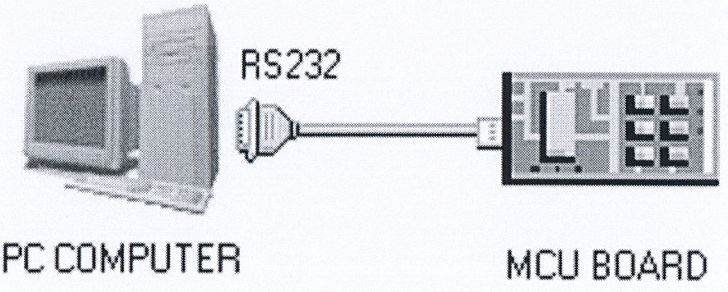
RS-232 ย่อมาจาก Recommended Standard-232 เป็นมาตรฐานการเชื่อมต่อข้อมูลแบบอนุกรม กำหนดโดย EIA (Electronics Industry Association) หรือ สมาคมผู้ประกอบการอุตสาหกรรมอิเล็กทรอนิกส์ของอเมริกา ใช้กับการสื่อสารแบบจุดต่อจุด โดยใช้สายเชื่อมต่อ DB แบบ 25 และ 9 เข็ม ที่ไม่ประสานจังหวะระหว่างคอมพิวเตอร์กับอุปกรณ์ต่อพ่วง มีการทำงานแบบ

สองทางพร้อมกัน (Full-duplex) โดยอาจใช้สายสัญญาณอื่นร่วมเพื่อทำแฮนด์เชค (Hand-shake) ทั้ง นี้มาตรฐาน RS-232 จำกัดความยาวสายไว้ที่ 50 ฟุต (หรือประมาณ 15 เมตร) สำหรับการส่งสัญญาณที่ความเร็ว 19,200 บิตต่อวินาที โดยที่ความยาวสายจะต้องสั้นลงถ้าต้องการสื่อสารที่ความเร็วสูงขึ้น โดย มีจุดเริ่มต้นจากความต้องการที่จะกำหนดมาตรฐานการเชื่อมต่อระหว่างคอมพิวเตอร์กับโมเด็มในสมัยนั้น ตัวมาตรฐานจะกำหนดสิ่งที่เกี่ยวข้องกับการเชื่อมต่อนี้ด้วยกันทั้งหมด 4 หัวข้อหลักๆ ด้วยกันคือ

1. คุณสมบัติทางไฟฟ้าของสัญญาณ
2. คุณสมบัติทางกลของการเชื่อมต่อ ซึ่งหมายถึงตัวคอนเน็กเตอร์นั่นเอง
3. หน้าที่การทำงานของวงจรสำหรับแลกเปลี่ยนข้อมูล
4. มาตรฐานการเชื่อมต่อสำหรับระบบสื่อสารเฉพาะอย่าง

มาตรฐาน RS-485 กำหนดโดยสมาคมผู้ประกอบการอุตสาหกรรมอิเล็กทรอนิกส์หรือ EIA เป็นมาตรฐานการเชื่อมต่อสัญญาณแบบอนุกรม (Serial Communication) มีลักษณะการเชื่อมต่อเป็นแบบหลายจุด (Multi-point) หรือ Multi-drop สายสัญญาณที่ใช้มีทั้งแบบที่เป็น 2 สายและแบบที่เป็น 4 สาย การต่อแบบหลายจุดนี้ทำให้สามารถมองสายสัญญาณเป็นบัสนำสัญญาณได้ (Signal Bus) จำนวนคอมพิวเตอร์หรืออุปกรณ์ที่สามารถอยู่บน RS-485 บัสหนึ่งถูกกำหนดไว้ที่ 32 ตัว ในกรณีที่ต้องการเพิ่มจะต้องมีตัวทวนสัญญาณ (Signal Repeater) หรือใช้ตัวส่ง-รับสัญญาณที่มีอิมพิแดนซ์ (ความต้านทานเสมือน) สูงขึ้น ซึ่งเราอาจเพิ่มจำนวนจุดเชื่อมต่อขึ้นได้ถึง 128 จุด ความยาวของสายสัญญาณตามมาตรฐาน RS-485 นี้สามารถยาวได้ถึง 1.2 กม เช่นเดียวกับมาตรฐาน RS-422 แต่การสื่อสารจะเป็นแบบสองทางไม่พร้อมกัน (Half Duplex) มีเพียงคอมพิวเตอร์หรืออุปกรณ์ตัวเดียวเท่านั้นที่สา มารถส่งสัญญาณออกได้ ณ เวลาหนึ่งๆ ส่วนที่เหลือจะเป็นผู้รับสัญญาณ หรือผู้ฟัง

2.12.1 การใช้งานพอร์ตอนุกรม RS-232



รูปที่ 2.13 แสดงการใช้งานพอร์ตอนุกรม RS-232

การสื่อสารแบบอนุกรม นับว่ามีความสำคัญ ต่อการใช้งาน ไมโครคอนโทรลเลอร์มาก เพราะสามารถใช้แป้นพิมพ์ และจอภาพของ PC เป็น อินพุต และ เอาต์พุต ในการติดต่อ หรือ ควบคุม ไมโครคอนโทรลเลอร์ ด้วยสัญญาณอย่างน้อย เพียง 3 เส้นเท่านั้น คือ

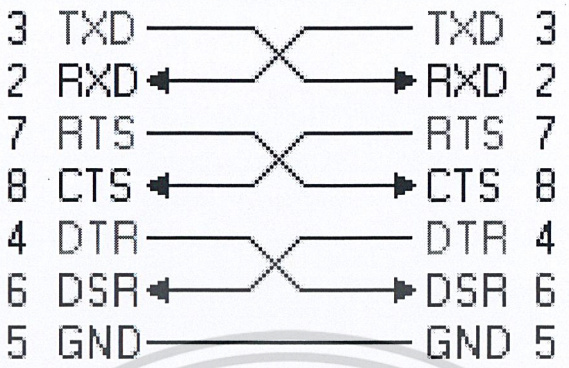
- สายส่งสัญญาณ TX
- สายรับสัญญาณ RX
- และสาย GND

โดยปกติพอร์ตอนุกรม RS-232C จะสามารถต่อสายได้ยาว 50 ฟุตโดยประมาณ ขึ้นอยู่กับ ชนิดของ สายสัญญาณ, ระยะทาง, และ ปริมาณ สัญญาณ รบกวน

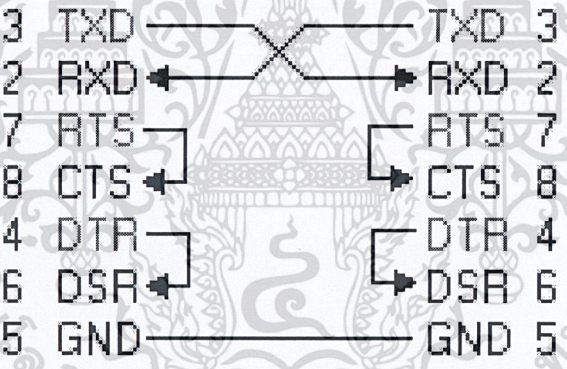
ตารางที่ 2.2 ตารางแสดงการจัดขาของคอนเน็กเตอร์ อนุกรมแบบ DB9 และหน้าที่การใช้งานต่างๆ

PIN	Description	Type
1	Data Carrier Detect (DCD)	Input
2	Received Data (RXD)	Input
3	Transmitted Data (TXD)	Output
4	Data Terminal Ready (DTR)	Output
5	Signal Ground (GND)	Input
6	Data Set Ready (DSR)	Input
7	Request To Send (RTS)	Output
8	Clear to Send (CTS)	Input
9	Ring Indicator (RI)	Input

การต่ออุปกรณ์ภายนอกผ่าน DB9 แบบ 3 เส้น



รูปที่ 2.14 แสดงการเชื่อมต่ออุปกรณ์อุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์ด้วยสาย DB9



รูปที่ 2.15 แสดงการเชื่อมต่ออุปกรณ์ภายนอกผ่าน DB9 แบบ Null modem

การทำงานของขาสัญญาณ DB9

TXD เป็นขาที่ใช้ส่งข้อมูล

RXD เป็นขาที่ใช้รับข้อมูล

DTR แสดงสภาวะพอร์ตว่าเปิดใช้งาน

DSR ตรวจสอบว่าพอร์ต ที่ติดต่อด้วย เปิดอยู่หรือไม่

เมื่อเปิดพอร์ตอนุกรม ขา DTR จะ ON เพื่อให้อุปกรณ์ได้รับทราบว่าต้องการติดต่อด้วยใน ขณะเดียวกันก็จะตรวจสอบขา DSR ว่าอุปกรณ์พร้อมหรือไม่

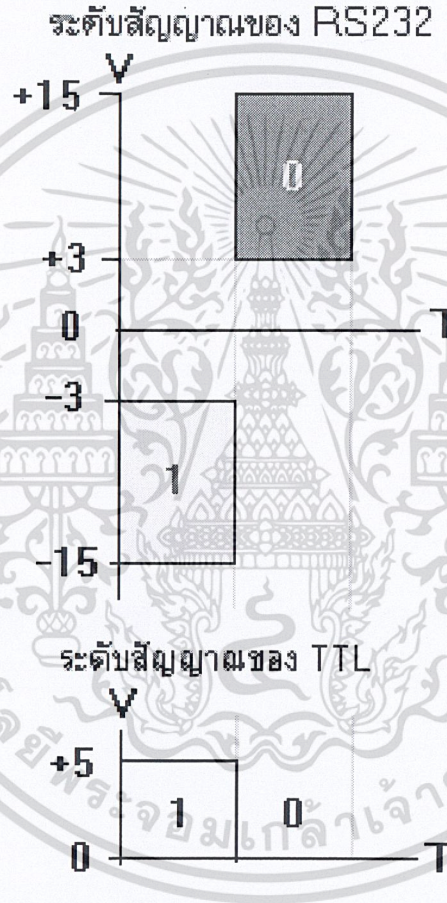
RTS แสดงสภาวะพอร์ตว่าต้องการส่งข้อมูล

CTS ตรวจสอบว่าพอร์ตที่ติดต่ออยู่ ต้องการส่งข้อมูลหรือไม่

เมื่อต้องการส่งข้อมูลขา RTS จะ ON และจะส่งข้อมูลออกที่ขา TXD เมื่อส่งเสร็จก็จะ OFF ในขณะเดียวกันก็จะตรวจสอบขา CTS ว่าอุปกรณ์ต้องการที่จะส่งข้อมูลหรือไม่

GND ขา ground

ระดับสัญญาณของ RS232



รูปที่ 2.16 ระดับสัญญาณของ RS232C และระดับสัญญาณของ TTL

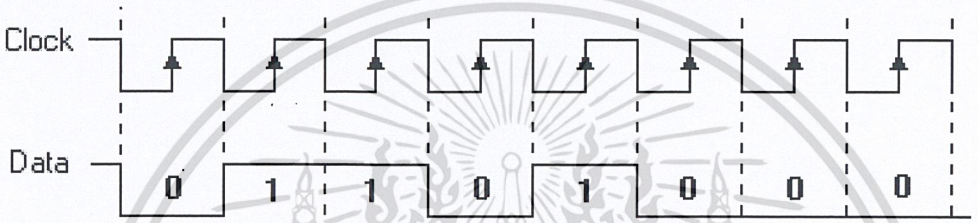
สัญญาณรบกวนที่เกิดขึ้น ในสายนำสัญญาณ มักจะมีแรงดันเป็นบวก เมื่อเทียบกับกราวด์ เพื่อป้องกันสัญญาณรบกวนนี้ จึงออกแบบแรงดัน ของโลจิก "1" เป็นลบ คืออยู่ในช่วง -3V ถึง -15V ส่วนแรงดัน ของโลจิก "0" อยู่ในช่วง +3V ถึง +15V และเหตุที่ ระดับสัญญาณ ของ RS232 อยู่ในช่วง +15V ถึง -15V ก็เพื่อให้ต่อสายสัญญาณไปได้ไกลขึ้น ดังนั้นจึงจำเป็นจะต้องมีวงจรเปลี่ยนระดับแรงดันของ RS232 มาเป็นระดับแรงดันของ TTL

อัตราการส่งข้อมูล (Baud rate)

คือความเร็วของการรับ-ส่งข้อมูล เป็นจำนวนบิดต่อวินาทีเช่น 300, 1,200, 2,400, 4,800, 9,600 ,14,400 ,19,200, 38,400 ,56,000 เป็นต้น การเลือกอัตราการส่งข้อมูลขึ้นอยู่กับ ชนิดของสายสัญญาณ, ระยะทาง,และปริมาณสัญญาณรบกวน

2.12.2 การสื่อสารแบบซิงโครนัส (Synchronous)

การรับส่งข้อมูล จะมีสัญญาณนาฬิกา ซึ่งเป็นตัวกำหนด จังหวะเวลา การส่งข้อมูล ร่วมอยู่ด้วยอีกเส้นหนึ่ง ใช้คู่กับสัญญาณข้อมูล ตัวอย่างเช่น การส่งสัญญาณจากคีย์บอร์ด

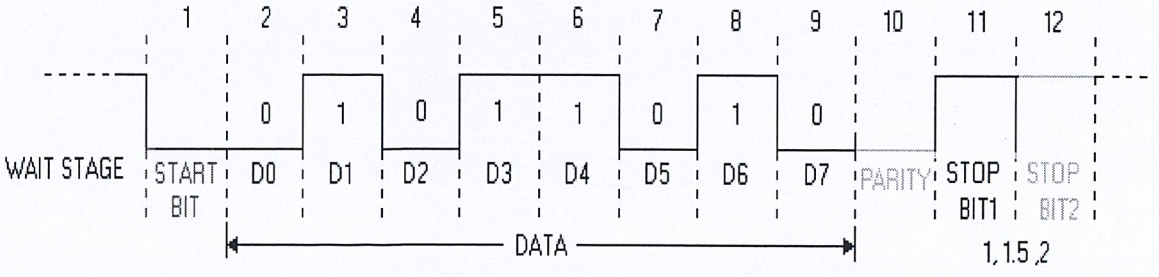


รูปที่ 2.17 แสดงตัวอย่างการรับส่งข้อมูลแบบซิงโครนัส

2.12.3 การสื่อสารแบบอะซิงโครนัส (Asynchronous)

การรับส่งข้อมูล โดยที่ไม่จำเป็นต้อง มีสัญญาณนาฬิการ่วมด้วย แต่จะใช้ให้ตัวส่งและตัวรับ มีอัตราส่งข้อมูล ที่เท่ากันรูปแบบข้อมูลแบบอะซิงโครนัส ประกอบด้วย 4 ส่วนคือ

1. บิตเริ่มต้น (Start bit) มีขนาด1 บิต
2. บิตข้อมูล (Data) มีขนาด 5,6,7 หรือ 8 บิต
3. บิตตรวจสอบพาริตี้ (Parity bit) มีขนาด 1 บิตหรือไม่มี
4. บิตหยุด (Stop bit) มีขนาด 1, 1.5, 2 บิต



รูปที่ 2.18 แสดงตัวอย่างการรับส่งข้อมูลแบบอะซิงโครนัส

- เมื่อไม่มีการส่งข้อมูล ขา data จะมีสถานะเป็นโลจิก "1" หรือ สถานะหยุดรอ (Waiting stage)
- เมื่อเริ่มต้นส่งข้อมูลจะให้ขา data เป็นโลจิก "0" เป็นจำนวน 1 บิต เรียกว่าบิตเริ่มต้น (Start bit)
- จากนั้นก็จะเริ่มต้นส่งข้อมูล โดยส่งบิตต่ำไปก่อน (LSB)
- แล้วตามด้วยพาริตี้บิต (จะมีหรือไม่ก็ได้ ขึ้นอยู่กับการติดตั้งค่า ของทั้งสองฝ่าย)
- สุดท้ายตามด้วยโลจิก "1" อย่างน้อย 1 บิต (มีขนาด 1, 1.5, หรือ 2 บิต) เพื่อแสดงว่าสิ้นสุดข้อมูล

การรับและส่งข้อมูลแบบอนุกรมยังแบ่งออกเป็นลักษณะการใช้งานได้ 3 แบบคือ

1. แบบซิมเพลกซ์ (Simplex) เป็นการส่ง หรือรับข้อมูล แบบทิศทางเดียว เท่านั้น

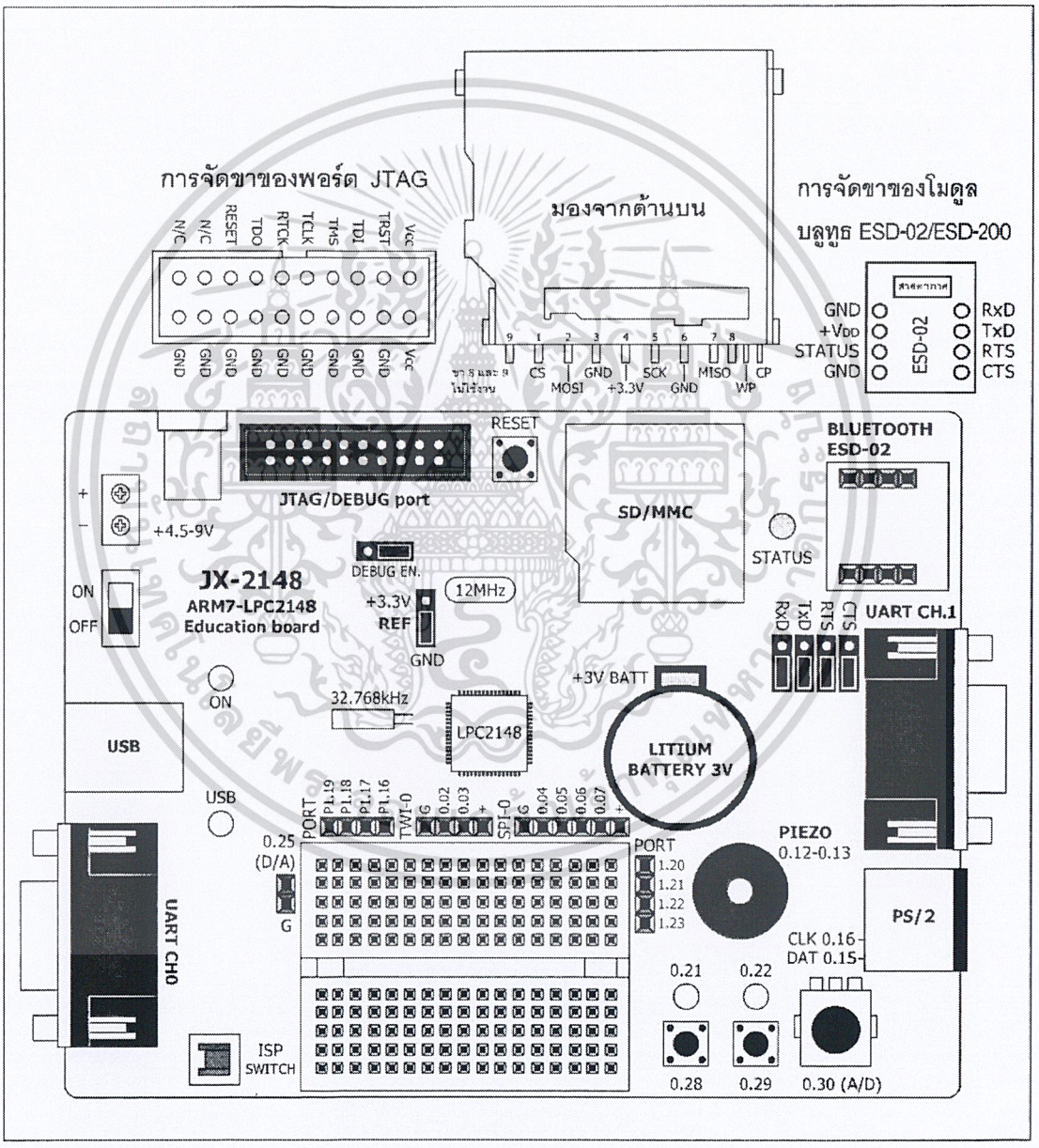
2. แบบฮาล์ฟดูเพลกซ์ (Half Duplex) เป็นการส่งและรับข้อมูลแบบสลับกัน คือเมื่อด้านหนึ่งส่ง อีกด้าน หนึ่ง เป็นฝ่ายรับ สลับกัน ไม่สามารถรับ-ส่งในเวลาเดียวกันได้

3. แบบฟลูดูเพลกซ์ (Full Duplex) สามารถรับ-ส่งข้อมูลในเวลาเดียวกันได้

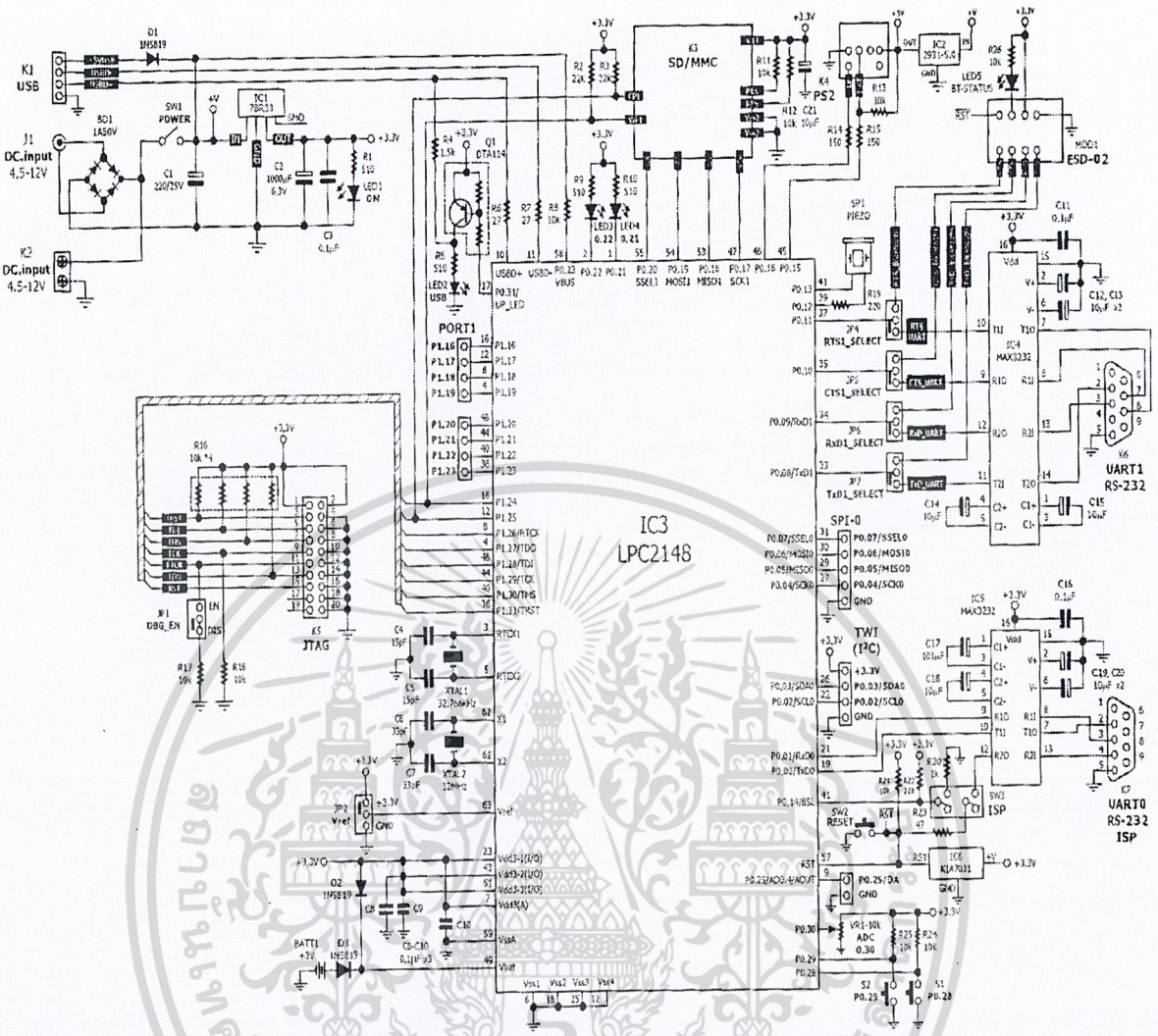
บทที่ 3

การคำนวณและการสร้าง

3.1 ศึกษาวงจรของบอร์ด JX-2148



รูปที่ 3.1 แสดงลักษณะและองค์ประกอบของบอร์ด JX-2148



รูปที่ 3.2 แสดงผังวงจรของบอร์ด JX-2148

P0.00/TxD0 และ **P0.01/RxD0** ต่อกับวงจรเชื่อมต่อพอร์ทอนุกรม UART0

P0.02/SCL0 และ **P0.03/SDA0** ต่อกับจุดต่ออุปกรณ์ระบบบัส 2 สาย(I²C)

P0.04 ถึง **P0.07** ต่อกับจุดต่ออุปกรณ์ระบบบัส SPI

P0.08/TxD1, **P0.09/RxD1**, **P0.10** และ **P0.11** ต่อกับวงจรเชื่อมต่อพอร์ตอนุกรม UART และโมดูลบลูทูธ ESD-02 โดยขาพอร์ต P0.10 ต่อกับขา CTS และ P0.11 ต่อกับขา RTS มีจั๊มเปอร์สำหรับเลือกการทำงานของขาพอร์ตทั้งสี่ ซึ่งในการใช้งานต้องเลือกต่อเข้ากับอุปกรณ์ในแบบเดียวกัน

P0.12 และ **P0.13** ต่อกับบัซเซอร์

P0.15 และ **P0.16** ต่อกับแจ๊ก PS/2 โดย P0.15 เป็นขาข้อมูล P0.16 เป็นขาสัญญาณนาฬิกา

P0.17 ถึง **P0.20** เป็นขาพอร์ตของโมดูล SSP ต่อกับซ็อกเก็ต SD

P0.21 และ **P0.22** ต่อกับ LED

P0.25 ต่อกับจุดต่อ D/A เป็นขาเอาต์พุตของโมดูลแปลงสัญญาณดิจิตอลเป็นอะนาลอก

P0.28 และ **P0.29** ต่อกับสวิตช์กดติดปล่อยดับพร้อมตัวต้านทานพูลอัป

P0.30 ต่อกับตัวต้านทานปรับค่าได้ 10kΩ เพื่อมดสอบโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล

P0.31 ใช้ควบคุมการเชื่อต่อพอร์ต USB และต่อกับวงจรแสดงความพร้อมในการเชื่อมต่อ

P1.16 ถึง **P1.23** เป็นขาพอร์ตอิสระ

P1.24 และ **P1.25** ต่อกับซ็อกเก็ต SD เพื่อตรวจสอบการมีอยู่ของ SD การ์ดในซ็อกเก็ต

P1.26 ถึง **P1.31** ต่อกับคอนเน็กเตอร์ JTAG

การโปรแกรมจะกระทำผ่านโมดูล UART0 โดยมีสวิตช์กด SW3 เพื่อเลือกโหมดการโปรแกรมและรันในขณะที่โมดูล UART1 ใช้เชื่อมต่อพอร์ตอนุกรมและเชื่อมต่อโมดูลบลูทูธ ESD200 โดยมีจั๊มเปอร์สำหรับเลือกการติดต่อระหว่างอุปกรณ์ทั้งสอง

3.2 เขียนคำสั่ง ARM7 LPC2148 ด้วยโปรแกรม Keil uVision3

3.2.1 ประกาศการเรียกใช้ฟังก์ชัน (Header file)

```
#include "LPC214x.H"                    // LPC2148 MPU Register
#include <stdio.h>                      // For Used Function printf
#include <stdlib.h>
```

3.2.2 กำหนดค่าการทำงานของพอร์ตต่างๆ

```
#define LED_RX_ON   IOCLR0   |= 0x00200000
#define LED_RX_OFF  IOSET0   |= 0x00200000
#define LED_TX_ON   IOCLR0   |= 0x00400000
#define LED_TX_OFF  IOSET0   |= 0x00400000
#define PWM_OFF   IOCLR0     |= 0x02000000
#define PWM_ON    IOSET0     |= 0x02000000
#define Buzzer_OFF1   IOCLR0 |= 0x00001000
#define Buzzer_ON1  IOSET0   |= 0x00001000
#define Buzzer_OFF2   IOCLR0 |= 0x00002000
#define Buzzer_ON2  IOSET0   |= 0x00002000
```

3.2.3 ประกาศฟังก์ชันย่อยและกำหนดตัวแปรต่างๆ เช่น

```
void init_serial0 (void);               // Initil UART-0
int putchar (int ch);                   // Put Char to UART-0
```

```
int getchar (void);                    // Get Char From Uart-0
void init_serial1 (void);              // Initil UART-1
int putchar1 (int ch);                 // Put Char to UART-1
int getchar1 (void);                   // Get Char From Uart-1
void delay(unsigned long int);         // Delay Time Function
```

...เป็นต้น

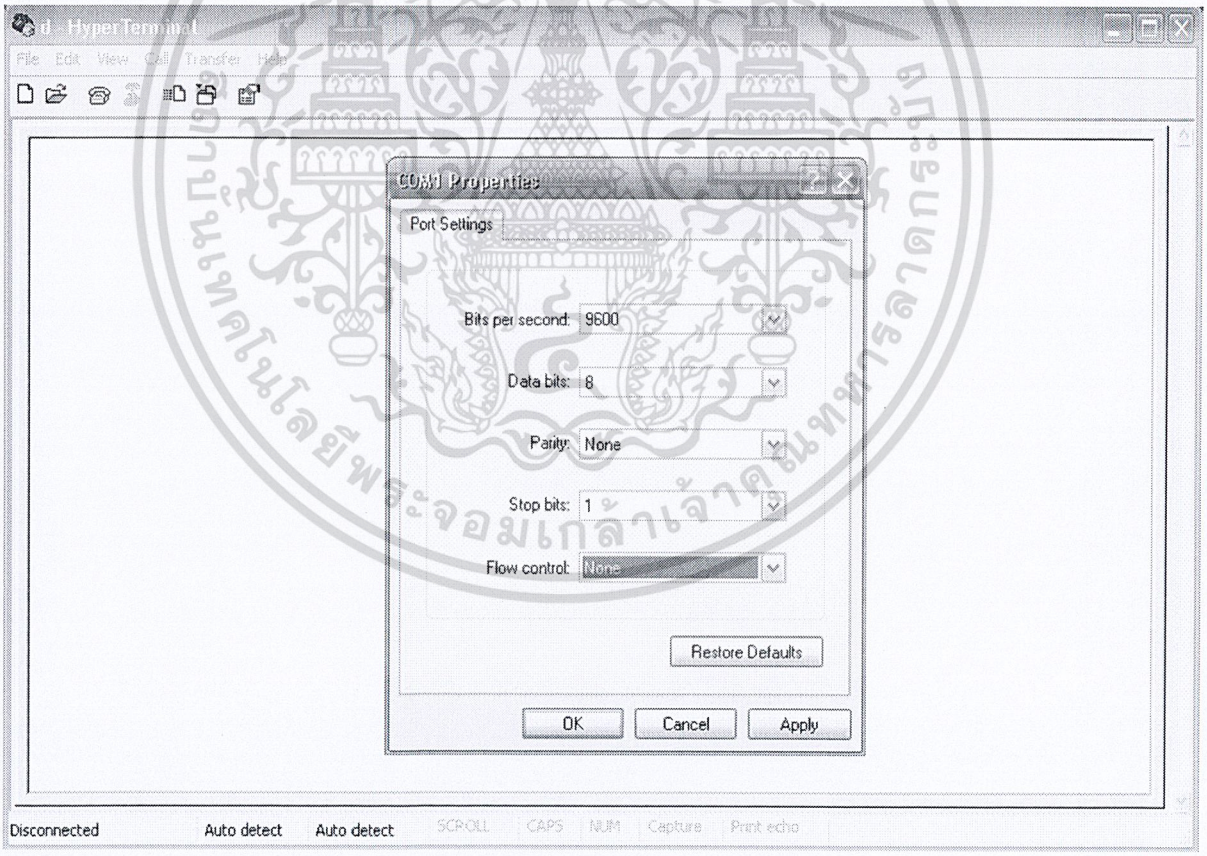
3.2.4 กำหนดฟังก์ชันหลักในการทำงาน (int main)

```
int main(void)
{
    while(1)                           // Loop Continue
    {
        if((IOPIN0 & 0x10000000) == 0)     Send = 1;
        else  if((IOPIN0 & 0x20000000) == 0)  Send = 0;
        if(Send == 1)
    {
    do                                 // Loop Read ADC1
    {
        val = AD0DR3;                  // Read A/D Data Register
    }
    while  ((val & 0x80000000) == 0);  // Wait ADC Conversion Complete
    val = (val >> 6) & 0x03FF;         // Shift ADC Result to Integer
    volt = val * 3.3 / 1023.0;         // Volt = ADC Result x [3.3V / 1024]
    PWMMR2 = 0x00000000+duty;
    // Update PWM2 (High Period = 0..1023 Cycle)
    PWMLER = 0x00000004;
    // Enable PWM Match2 Latch
    ratio = val*3000/1024;             //duty = val;
    sound (ratio,100) ;                //beep();
    printf ("%d\n",val);               // Display 3-Digit Result(0-3.3V)
    LED_TX_ON;
    else
    {
```

```
            LED_TX_OFF;
            PWMMR2 = 0x00000000+duty;
            // Update PWM2 (High Period = 0..1023 Cycle)
            PWMLER = 0x00000004;                    // Enable PWM Match2 Latch
            //        delay (150000);
        }

    }
```

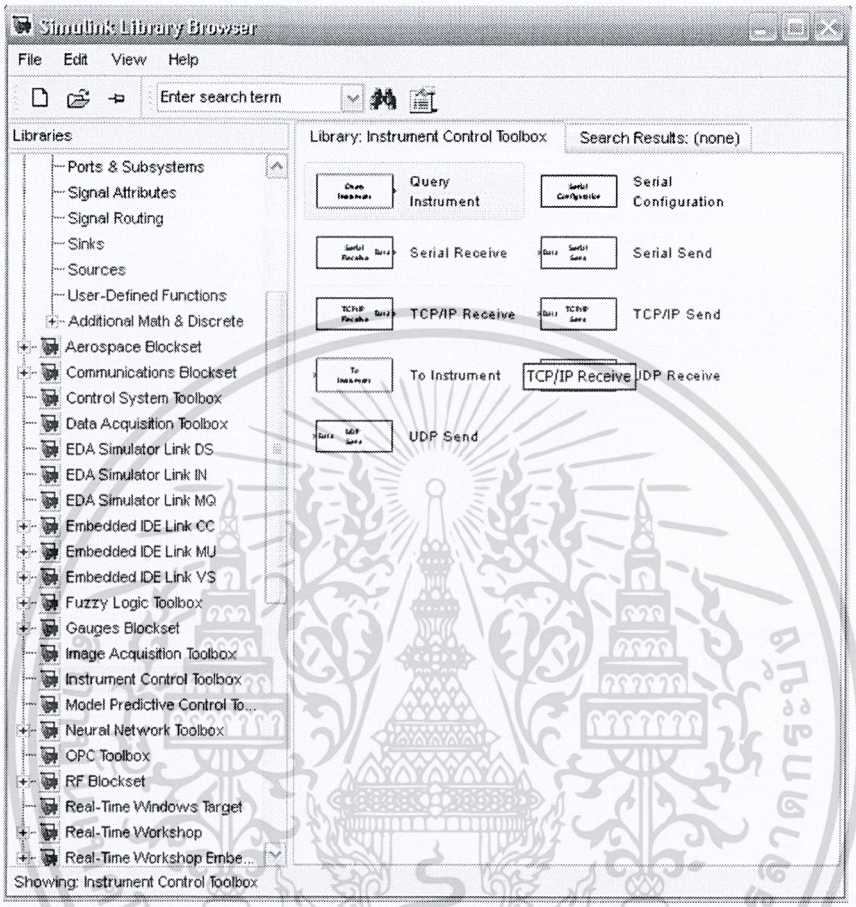
3.3 โปรแกรม Hyper Terminal ใช้ในการทดลองรับส่งค่าผ่านทางพอร์ตอนุกรม

โปรแกรม Hyper Terminal เป็นโปรแกรมสำเร็จที่มีอยู่ในคอมพิวเตอร์ใช้สำหรับทดสอบการรับส่งค่าผ่านทางพอร์ตอนุกรม



รูปที่3.3 เป็นภาพของโปรแกรม Hyper Terminal

3.4 Simulink Block สำหรับเชื่อมต่อกับอุปกรณ์ภายนอกผ่านทางพอร์ตอนุกรมตามเวลาจริง



รูปที่ 3.4 เป็นภาพของโปรแกรมซิมูลิงค์บล็อก

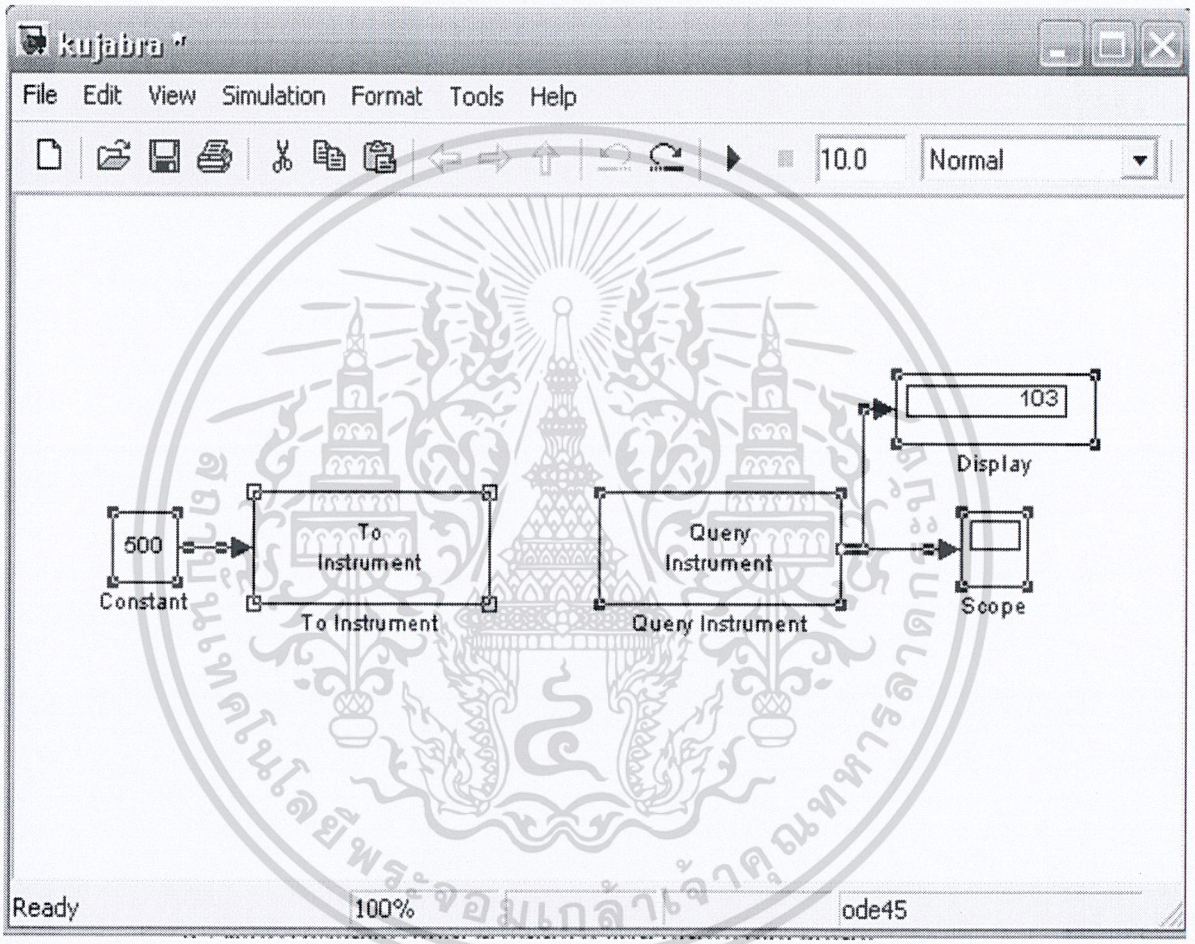
- To Instrument Block ใช้ในการส่งสัญญาณจากคอมพิวเตอร์สู่อุปการณ์ภายนอก

- Query Instrument ใช้ในการรับค่าจากอุปกรณ์ภายนอกมาสู่คอมพิวเตอร์

บทที่ 4

การทดลองและผลการทดลอง

4.1 การทดลองรับส่งค่าผ่านทาง Simulink Block



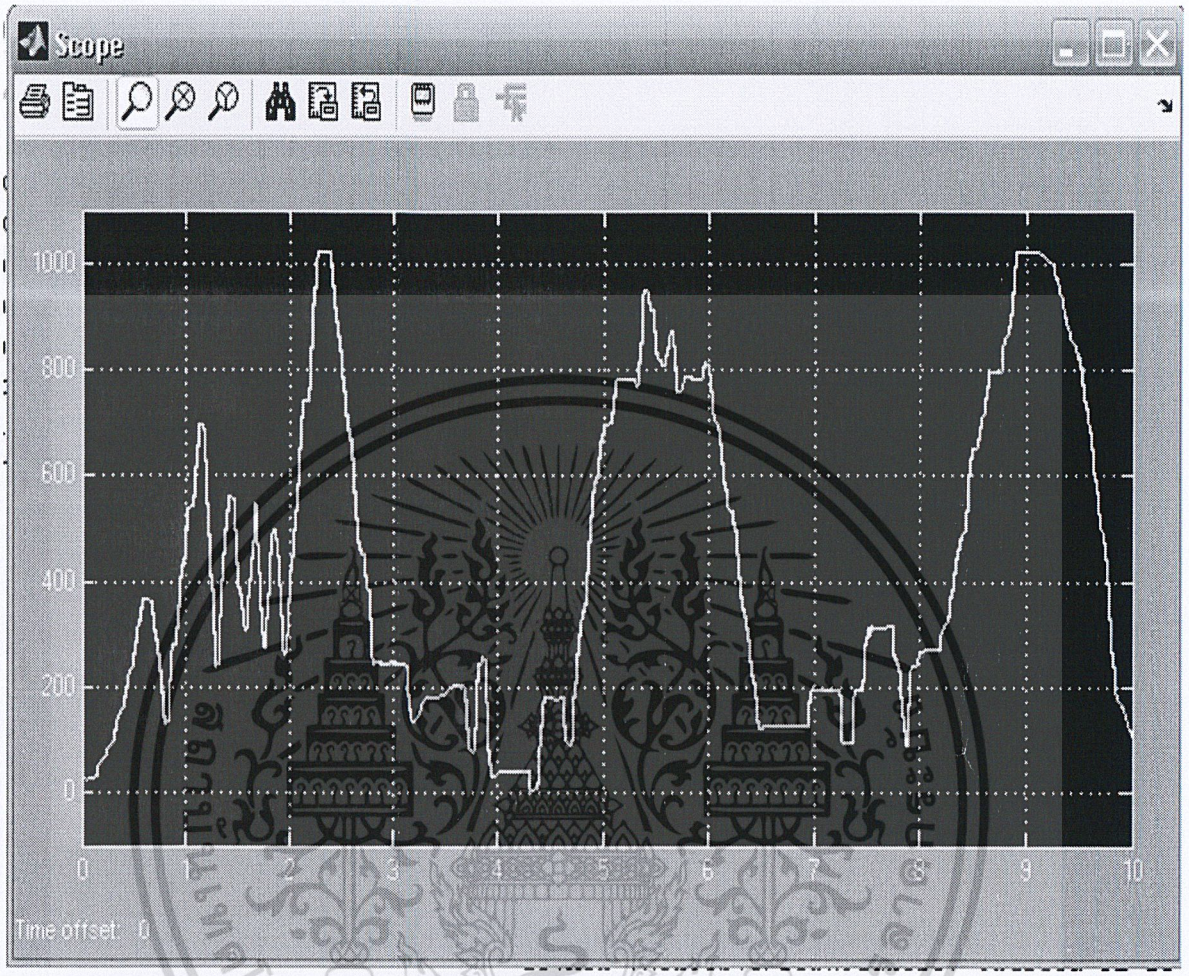
รูปที่ 4.1 แสดงภาพการรับส่งค่าผ่านทาง Simulink Block

4.2 ผลการทดลองอ่านค่าสัญญาณเอาต์พุตของอุปกรณ์ภายนอก

ตารางที่ 4.1 แสดงค่า Output ที่ได้จากการวัด เทียบกับค่าที่ได้จากการคำนวณ

Input	ค่าที่วัดได้(Volt)	จากการคำนวณ(Volt)
0	0.000	0.000
2	0.004	0.006
4	0.013	0.012
8	0.026	0.025
16	0.052	0.051
32	0.103	0.103
64	0.207	0.206
128	0.413	0.412
256	0.825	0.825
512	1.649	1.651
1023	3.280	3.300

4.3 ผลการทดลองรับค่าจากอุปกรณ์ภายนอก และควบคุมการทำงานที่เวลาจริง



รูปที่ 4.2 แสดงการรับค่าจากอุปกรณ์ภายนอกที่ควบคุมความถี่ตามเวลาจริง

บทที่5
บทวิจารณ์และสรุปผล

5.1 สรุปผลการปฏิบัติงาน

โครงงานชุดเรียลไทม์เวอร์คชอพมีจุดประสงค์เพื่อรับสัญญาณจากระบบภายนอก เข้ามาแสดงผลในโมเดลของซิมูลิ้งค์และสามารถปรับค่าสัญญาณตามเวลาจริงได้ และยังสามารถส่งค่าไปสู่ระบบภายนอกได้อีกด้วย ซึ่งการรับส่งค่าสัญญาณนี้สามารถนำไปวิเคราะห์และยังนำไปใช้งานต่อได้อีกด้วย

5.2 ปัญหาในการทำงานและแนวทางแก้ไข

เป็นโครงงานที่ค่อนข้างยากต่อการศึกษาและเข้าใจ ทำให้เกิดความเข้าใจผิดในเนื้อหาหรือมีการศึกษาที่ไม่ตรงประเด็นทำให้เกิดการเสียเวลาเป็นอย่างมาก ควรแก้ไขโดยการปรึกษากับอาจารย์หรือผู้รู้ให้มากขึ้น

5.3 ผลที่ได้รับจากโครงงาน

จากการศึกษาทางทฤษฎีเพียงอย่างเดียวนั้นอาจไม่เพียงพอ เพราะเมื่อนำมาทดลองใช้งานจริงย่อมมีปัญหา และเงื่อนไขในสภาวะต่างๆมากมาย ซึ่งการที่จะประสบความสำเร็จนั้นบางครั้งต้องใช้ประสบการณ์ในการทำงานนั้นเป็นอย่างมาก หรือบางครั้งอาจได้จากการทดลองและนำมาวิเคราะห์ควบคู่กันไปกับแนวทางทฤษฎีที่ได้ศึกษา ซึ่งจากการทำโครงงานนี้ ทำให้ผู้ทำ ได้เรียนรู้สิ่งต่างๆมากมายขึ้น ซึ่งในบางส่วนอาจจะไม่ในตรงกับทฤษฎีเลย หรือบางสวมของทฤษฎีก็เป็นความรู้ใหม่ที่ผู้ทำไม่เคยศึกษามาก่อน รวมถึงปัญหาต่างๆที่จะต้องมีการแก้ไขให้ถูกลุล่วง ไปได้ทำให้สามารถนำไปใช้ในการทำงานจริง หลังจากจบการศึกษา

ภาคผนวก ก

โปรแกรม C-MEX S-Function

```
/*
 * File : times_t.c
 * Abstract:
 * An example C-file S-function for multiplying an input by 2,
 *      y = 3*u
 *
 * Real-Time Workshop note:
 * This file can be used as is (noninlined) with the Real-Time Workshop
 * C rapid prototyping targets, or it can be inlined using the Target
 * Language Compiler technology and used with any target. See
 * matlabroot/toolbox/simulink/blocks/tlc_c/timestwo.tlc
 * matlabroot/toolbox/simulink/blocks/tlc_ada/timestwo.tlc
 * the C and Ada TLC code to inline the S-function.
 *
 * See simulink/src/sfuntmpl_doc.c
 */

#define S_FUNCTION_NAME  times_t
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumSFcnParams(S, 0);
   if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
}
   if (!ssSetNumInputPorts(S, 1)) return;
   ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
```

```
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetNumSampleTimes(S, 1);
    ssSetSimStateCompliance(S, USE_DEFAULT_SIM_STATE);
    ssSetOptions(S,
                 SS_OPTION_WORKS_WITH_CODE_REUSE |
                 SS_OPTION_EXCEPTION_FREE_CODE |
                 SS_OPTION_USE_TLC_WITH_ACCELERATOR);
}
/* Function: mdlInitializeSampleTimes
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
    ssSetModelReferenceSampleTimeDefaultInheritance(S);
}
/* Function: mdlOutputs
 * Abstract:
 * y = 2*u
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T             i;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T            *y    = ssGetOutputPortRealSignal(S,0);
    int_T             width = ssGetOutputPortWidth(S,0);
    for (i=0; i<width; i++) {
        *y++ = 3.0 *(*uPtrs[i]);
```

```
    }
}
/* Function: mdlTerminate
*/
static void mdlTerminate(SimStruct *S)
{
}
 #ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

โปรแกรมคำสั่งของ Microcontroller ARM7

```
#include "LPC214x.H"                                    // LPC2148 MPU Register
#include <stdio.h>                                      // For Used Function printf
#include <stdlib.h>
#define LED_RX_ON   IOCLR0   |= 0x00200000
#define LED_RX_OFF  IOSET0   |= 0x00200000
#define LED_TX_ON   IOCLR0   |= 0x00400000
#define LED_TX_OFF  IOSET0   |= 0x00400000
#define PWM_OFF   IOCLR0     |= 0x02000000
#define PWM_ON    IOSET0     |= 0x02000000
#define Buzzer_OFF1   IOCLR0     |= 0x00001000
#define Buzzer_ON1  IOSET0       |= 0x00001000
#define Buzzer_OFF2   IOCLR0     |= 0x00002000
#define Buzzer_ON2  IOSET0       |= 0x00002000
/* pototype  section */
void init_serial0 (void);                               // Initil UART-0
int putchar (int ch);                                   // Put Char to UART-0
int getchar (void);                                     // Get Char From Uart-0
void init_serial1 (void);                               // Initil UART-1
int putchar1 (int ch);                                  // Put Char to UART-1
int getchar1 (void);                                    // Get Char From Uart-1
void delay(unsigned long int);                          // Delay Time Function
void sound(int freq,int time);
void beep(void);
void delay_sound(long c);
unsigned int val;                                       // ADC Result (HEX)
float volt;                                             // ADC Result Volt
```

```
char data_buff;
char isr_uart1,Send;
char databuff[20]= {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
char data;
int j,index_buff=0;
int Time_Pwm;
int Fyquency = 1023,duty = 0;
int ratio=0;
int counter,sec;
void isr_uart0 (void)__irq
{
 char msg;
 if(((msg=U0IIR)&0x01)==0)
 {
       switch(msg&0x0E)
     {
           case 0x04:
                 while(!(U0LSR&0x20));
                  //U0THR=U0RBR;    // Echo
                 data = U0RBR;
                 databuff[index_buff] = data;
                 index_buff++;
                 duty = atoi(databuff);
                  LED_RX_ON;
                 delay(100000);
                LED_RX_OFF;
                break;
           case 0x02: break;
           default: break;
```

```
        }
    }
    VICVectAddr=0;
}
int main(void)
{
    init_serial0();
    init_serial1();                                    // Initial UART0 = 9600,N,8,1
////////////////////   Initial ADC0 (ADCR=0x01210680) ////////////////////////////////
    AD0CR &= 0x00000000;
    PINSEL1|=0x10000000;
    AD0CR = 0x00210608;
    AD0CR |= 0x01000000;
    ////////////////////////////////////////////////////////////////////////////////
    ////////////////////////   I/O out pin p0.25 p0.21 p0.22  ////////////////////////
     IODIR0   = 0x02603000;
            LED_RX_OFF;
            LED_TX_OFF;
    ////////////////////////////////////////////////////////////////////////////////
            Send = 0;
  // Initial PWM2 (GPIO-0.7) By Set PINSEL0[15:14=10]
    PINSEL0 &= 0xFFFF3FFF;                             // Select PWM2 Pin Connect P0.7
    PINSEL0 |= 0x00008000;
    // Initial PWM2 = 28.8KHz (29.4912/1024=28.800 KHz)
    // Period = 34.722 uS
    PWMPR = 0;                                         // 35uS Period -> 28.80KHz
    //Initial PWM2 = Double Edge PWM
    // Set   Output By Match-0
    // Reset Output By Match-2
```

```
PWMPCR &= 0xFFFFFFFB;                    // PWMSEL2 = "1" = Single Edge Control
PWMPCR |= 0x00000400;                    // PWMENA2 = "1" = Enable PWM2
PWMMCR = 0x00000002;                     // On Match0 = Reset Counter
PWMMR0 = 0x00000400;      // Set PWM2 Rate(29.4912MHz/1024)
PWMMR2 = 0x00000200;      // Set Default PWM2 High Pulse (0..512 Cycle)
PWMLER = 0x00000005;      // Enable Shadow Latch For Match 0,2
PWMTCR = 0x00000002;      // Reset Counter and Prescaler
PWMTCR = 0x00000009;      // Enable Counter and PWM + Release Counter From Reset

    while(1)                             // Loop Continue
    {
    if((IOPIN0 & 0x10000000) == 0)      Send = 1;
    else  if((IOPIN0 & 0x20000000) == 0)  Send = 0;
     if(Send == 1)
     {
          do                             // Loop Read ADC1
    {
          val = AD0DR3;                  // Read A/D Data Register
    }
    while ((val & 0x80000000) == 0);     // Wait ADC Conversion Complete
          val = (val >> 6) & 0x03FF;     // Shift ADC Result to Integer
          volt = val * 3.3 / 1023.0;     // Volt = ADC Result x [3.3V / 1024]
    PWMMR2 = 0x00000000+duty;            // Update PWM2(High Period = 0..1023 Cycle)
    PWMLER = 0x00000004;                 // Enable PWM Match2 Latch
          //duty = val;
          ratio = val*3000/1024;
          sound(ratio,100) ;
          //beep();
          printf("%d\n",val);            // Display 3-Digit Result(0-3.3V)
```

```
                LED_TX_ON;
            }else
            {
            LED_TX_OFF;
           PWMMR2 = 0x00000000+duty;          // Update PWM2(High Period = 0..1023 Cycle)
           PWMLER = 0x00000004;               // Enable PWM Match2 Latch
           //          delay(150000);
            }
    }
}
/*****************************/
/* Initial UART0 = 9600,N,8,1 */
/* VPB(pclk) = 60.00 MHz     */
/*****************************/
void init_serial0 (void)
{
 PINSEL0 &= 0xFFFFFFF0;                     // Reset P0.0,P0.1 Pin Config
 PINSEL0 |= 0x00000001;                     // Select P0.0 = TxD(UART0)
 PINSEL0 |= 0x00000004;                     // Select P0.1 = RxD(UART0)
 U0LCR &= 0xFC;                             // Reset Word Select(1:0)
 U0LCR |= 0x03;                             // Data Bit = 8 Bit
 U0LCR &= 0xFB;                             // Stop Bit = 1 Bit
 U0LCR &= 0xF7;                             // Parity = Disable
 U0LCR &= 0xBF;                             // Disable Break Control
 U0LCR |= 0x80;                             // Enable Programming of Divisor Latches
// U0DLM:U0DLL = 60.00 MHz / [16 x Baud]
 //          = 60.00 MHz / [16 x 9600]
 //          = 390.6 = 391 = 0187H
 U0DLM = 0x01;                              // Program Divisor Latch(391) for 9600 Baud
```

```
 U0DLL = 0x87;
 U0LCR &= 0x7F;                          // Disable Programming of Divisor Latches
 U0FCR |= 0x01;                          // FIF0 Enable
 U0FCR |= 0x02;                                  // RX FIFO Reset
 U0FCR |= 0x04;                                  // TX FIFO Reset
 U0FCR &= 0x3F;
 ////////////////////////////////    enable RDA interrupt   ////////////////////////////
                U0IER = 0x01;                           // enable RDA interrupt
                VICProtection = 0;                      // disable protection
                VICIntSelect &= 0xFFFFFFBF;             // IRQ
                VICVectAddr0 = (unsigned)isr_uart0;
                VICVectCntl0 = 0x20 | 6;
                VICIntEnable = 0x00000040;              // enable
////////////////////////////////////////////////////////////////////////////////////////
}
/*****************************/
/* Write Character To UART0 */
/*****************************/
int putchar (int ch)
{
 if (ch == '\n')
 {
  while (!(U0LSR & 0x20));                      // Wait TXD Buffer Empty
  U0THR = 0x0D;                                 // Write CR
 }
 while (!(U0LSR & 0x20));                       // Wait TXD Buffer Empty
 return (U0THR = ch);                           // Write Character
}
/*****************************/
```

```
/* Read Character From UART0 */
/*****************************/
int getchar (void)
{
  while (!(U0LSR & 0x01));                    // Wait RXD Receive Data Ready
  return (U0RBR);                             // Get Receice Data & Return
}
/*****************************/
/* Initial UART1 = 9600,N,8,1 */
/* VPB(pclk) = 60.00 MHz      */
void init_serial1 (void)
{
  PINSEL0 &= 0xFFF0FFFF;                      // Reset P0.8,P0.9 Pin Config
  PINSEL0 |= 0x00010000;                      // Select P0.8 = TxD(UART1)
  PINSEL0 |= 0x00040000;                      // Select P0.9 = RxD(UART1)
  U1LCR &= 0xFC;                              // Reset Word Select(1:0)
  U1LCR |= 0x03;                              // Data Bit = 8 Bit
  U1LCR &= 0xFB;                              // Stop Bit = 1 Bit
  U1LCR &= 0xF7;                              // Parity = Disable
  U1LCR &= 0xBF;                              // Disable Break Control
  U1LCR |= 0x80;                              // Enable Programming of Divisor Latches
 // U1DLM:U1DLL = 60.00 MHz / [16 x Baud]
  //          = 60.00 MHz / [16 x 9600]
  //          = 390.6 = 391 = 0187H
  U1DLM = 0x01;                               // Program Divisor Latch(391) for 9600 Baud
  U1DLL = 0x87;
  U1LCR &= 0x7F;                              // Disable Programming of Divisor Latches
  U1FCR |= 0x01;                              // FIF0 Enable
  U1FCR |= 0x02;                              // RX FIFO Reset
```

```
 U1FCR |= 0x04;                          // TX FIFO Reset
 U1FCR &= 0x3F;
}
/*****************************/
/* Write character to UART1 */
/*****************************/
int putchar1 (int ch)
{
 if (ch == '\n')
 {
  while (!(U1LSR & 0x20));               // Wait TXD Buffer Empty
  U1THR = 0x0D;                          // Write CR
 }
 while (!(U1LSR & 0x20));                // Wait TXD Buffer Empty
 return (U1THR = ch);                    // Write Character
}
/*******************************/
/* Read character from UART1 */
/*******************************/
int getchar1 (void)
{
 while (!(U1LSR & 0x01));                // Wait RXD Receive Data Ready
 return (U1RBR);                         // Get Receice Data & Return
}
/************************/
/* Delay Time Function */
/*    1-4294967296     */
/************************/
void delay(unsigned long int count1)
```

```
{
 while(count1 > 0)
 {
        count1--;
        Time_Pwm++;
        //if(Time_Pwm < duty ) { Buzzer_ON1;   Buzzer_OFF2;    }
        //else  { Buzzer_OFF1;   Buzzer_OFF2; Buzzer_OFF1;      }

 }                                              // Loop Decrease Counter
}
 //-------------------------------------------------------------------------------------//
// Program                  : Library for gennerate sound
// Description              : Library for gennerate easy sound
// Frequency                : Crytal 12 MHz at PLL 5x(CCLK = 60 MHz),PCLK = 30 MHz
// Filename                 : sound.h
// C compiler     : Keil CARM Compiler
//-------------------------------------------------------------------------------------//
//-------------------------------------------------------------------------------------//
//--------------------------- Function delay 100 us per count ---------------------------//
//-------------------------------------------------------------------------------------//
void delay_sound(long c)                        // Delay 100 us per count @ CCLK 60 MHz
{
        long i,j;
        for (i = 0; i < c; i++ )
        for (j = 0; j < 660; j++ );
}
//-------------------------------------------------------------------------------------//
//--------------------------- Function generate sound ---------------------------------//
//-------------------------------------------------------------------------------------//
```

```
void sound(int freq,int time)
{
        int dt=0,m=0;                       // Keep value and
        //FIO0DIR |= 0x00003000;            // Set P0.13 and P0.12 @ output
   dt = 5000/freq;                          // Keep active logic delay
        time = (5*time)/dt;                 // Keep counter for generate sound
        for(m=0;m<time;m++)                 // Loop for generate sound(Toggle logic P0.12)
        {                                   //FIO0SET |= 0x00001000;  // P0.12 = '1'
          PWM_ON;
          Buzzer_ON1;
              Buzzer_OFF2;
     delay_sound(dt);                       // Delay for sound
              //FIO0CLR |= 0x00001000;     // P0.12 = '0'
              PWM_OFF;
              Buzzer_OFF1;
              Buzzer_OFF2;
     delay_sound(dt);                       // Delay for sound
        }
}
//-------------------------------------------------------------------------------//
//---------------------------- Function generate sound beep default ----------------------------//
//-------------------------------------------------------------------------------//
void beep(void)
{
        sound(600,100); // Generate sound default frequency
}
```

ภาคผนวก ข

1 Introduction

It is possible to compile C, C++, or Fortran code so that it is callable from MATLAB. This kind of program is called a Matlab Executable (MEX) external interface function, or more briefly a "MEX-function." MEX enables the high performance of C, C++, and Fortran while working within the MATLAB environment. We will discuss C/MEX functions, which also applies directly to C++/MEX. Fortran/MEX is quite different and we do not discuss it here.

Warning This is not a beginner's tutorial to MATLAB. Familiarity with C and MATLAB is assumed. Use at your own risk.

MEX is often more trouble than it is worth. MATLAB's JIT interpreter in recent versions runs M-code so efficiently that it is often times difficult to do much better with C. Before turning to MEX in an application, optimize your M-code (see my other article, "Writing Fast MATLAB Code"). MEX-functions are best suited to substitute one or two bottleneck M-functions in an application. If you replace all functions in an application with MEX, you might as well port the application entirely to C.

2 Getting Started

The following program demonstrates the basic structure of a MEX-function.

hello.c

```
#include "mex.h"  /* Always include this */

void mexFunction(int nlhs, mxArray *plhs[],      /* Output variables */
                 int nrhs, const mxArray *prhs[]) /* Input variables  */
{
    mexPrintf("Hello, world!\n");  /* Do something interesting */
    return;
}
```

Copy the code into MATLAB's editor (it has a C mode) or into the C editor of your choice, and save it as `hello.c`.

The next step is to compile. On the MATLAB console, compile `hello.c` by entering the command

```
>> mex hello.c
```

If successful, this command produces a compiled file called `hello.mexa64` (or similar, depending on platform). Compiling requires that you have a C compiler and that MATLAB is configured to use it. MATLAB will autodetect most popular compilers, including Microsoft Visual C/C++ and GCC. As a fallback, some distributions of MATLAB come with the Lcc C compiler. Run `mex -setup` to change the selected compiler and build settings.

Once the MEX-function is compiled, we can call it from MATLAB just like any M-file function:

```
>> hello
Hello, world!
```

Note that compiled MEX files might not be compatible between different platforms or different versions of MATLAB. They should be compiled for each platform/version combination that you need. It is possible to compile a MEX file for a target platform other than the host's using the `-<arch>` option, for example, `mex -win32 hello.c`.

MATLAB comes with examples of MEX in `matlab/extern/examples`. For detailed reference, also see `matrix.h` and the other files in `matlab/extern/include`.

Inputs and Outputs

Of course, a function like `hello.c` with no inputs or outputs is not very useful. To understand inputs and outputs, let's take a closer look at the line

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
```

Here "`mxArray`" is a type for representing a MATLAB variable, and the arguments are:

C/MEX	Meaning	M-code equivalent
`nlhs`	Number of output variables	`nargout`
`plhs`	Array of `mxArray` pointers to the output variables	`varargout`
`nrhs`	Number of input variables	`nargin`
`prhs`	Array of `mxArray` pointers to the input variables	`varargin`

These MEX variables are analogous to the M-code variables `nargout`, `varargout`, `nargin`, and `varargin`. The naming "lhs" is an abbreviation for left-hand side (output variables) and "rhs" is an abbreviation for right-hand side (input variables).

For example, suppose the MEX-function is called as

```
[X,Y] = mymexfun(A,B,C)
```

Then `nlhs` = 2 and `plhs[0]` and `plhs[1]` are pointers (type `mxArray*`) pointing respectively to `X` and `Y`. Similarly, the inputs are given by `rlhs` = 3 with `prhs[0]`, `prhs[1]`, and `prhs[2]` pointing respectively to `A`, `B`, and `C`.

The output variables are initially unassigned; it is the responsibility of the MEX-function to create them. If `nlhs = 0`, the MEX-function is still allowed return one output variable, in which case `plhs[0]` represents the `ans` variable.

The following code demonstrates a MEX-function with inputs and outputs.

`normalizecols.c`

```
/*  NORMALIZECOLS.C  Normalize the columns of a matrix
Syntax:      B = normalizecols(A)
       or    B = normalizecols(A,p)
The columns of matrix A are normalized so that norm(B(:,n),p) = 1. */
#include <math.h>
#include "mex.h"

#define IS_REAL_2D_FULL_DOUBLE(P) (!mxIsComplex(P) && \
mxGetNumberOfDimensions(P) == 2 && !mxIsSparse(P) && mxIsDouble(P))
#define IS_REAL_SCALAR(P) (IS_REAL_2D_FULL_DOUBLE(P) && mxGetNumberOfElements(P) == 1)

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    /* Macros for the ouput and input arguments */
    #define B_OUT       plhs[0]
```

```
    #define A_IN        prhs[0]
    #define P_IN        prhs[1]
    double *B, *A, p, colnorm;
    int M, N, m, n;

    if(nrhs < 1 || nrhs > 2)  /* Check the number of arguments */
        mexErrMsgTxt("Wrong number of input arguments.");
    else if(nlhs > 1)
        mexErrMsgTxt("Too many output arguments.");

    if(!IS_REAL_2D_FULL_DOUBLE(A_IN)) /* Check A */
        mexErrMsgTxt("A must be a real 2D full double array.");

    if(nrhs == 1) /* If p is unspecified, set it to a default value */
        p = 2.0;
    else  /* If P was specified, check that it is a real double scalar */
        if(!IS_REAL_SCALAR(P_IN))
            mexErrMsgTxt("P must be a real double scalar.");
        else
            p = mxGetScalar(P_IN);  /* Get p */

    M = mxGetM(A_IN);      /* Get the dimensions of A */
    N = mxGetN(A_IN);
    A = mxGetPr(A_IN);     /* Get the pointer to the data of A */
    B_OUT = mxCreateDoubleMatrix(M, N, mxREAL); /* Create the output matrix */
    B = mxGetPr(B_OUT);    /* Get the pointer to the data of B */

    for(n = 0; n < N; n++)    /* Compute a matrix with normalized columns */
    {
        for(m = 0, colnorm = 0.0; m < M; m++) colnorm += pow(A[m + M*n], p);
        colnorm = pow(fabs(colnorm),1.0/p);  /* Compute the norm of the nth column */

        for(m = 0; m < M; m++) B[m + M*n] = A[m + M*n]/colnorm;
    }

    return;
}
```

Much of the code is spent verifying the inputs. MEX provides the following functions to check datatype, dimensions, and so on:

C/MEX	Meaning	M-code equivalent
`mxIsDouble(A_IN)`	True for a double array	`isa(A,'double')`
`mxIsComplex(A_IN)`	True if array is complex	`~isreal(A)`
`mxIsSparse(A_IN)`	True if array is sparse	`issparse(A)`
`mxGetNumberOfDimensions(A_IN)`	Number of array dimensions	`ndims(A)`
`mxGetNumberOfElements(A_IN)`	Number of array elements	`numel(A)`

The `normalizecols.c` example simplifies input parsing by combining some of these checks into a macro `IS_REAL_2D_FULL_DOUBLE`. Notice how we check `nrhs==1` to see if the function was called as `normalizedcols(A)` or `normalizedcols(A,p)`.

Another approach to input parsing is to rename this MEX-function as "`normalizecolsmx.c`" and create an M-function wrapper:

normalizecols.m

```
function B = normalizecols(A,p)
% M-function wrapper for parsing the inputs
if nargin < 2
    if nargin < 1
        error('Not enough input arguments');
    end
    p = 2;   % p is unspecified, set default value
end

if ~isreal(A) || ndims(A) ~= 2 || issparse(A) || ~isa(A, 'double')
    error('A must be a real 2D full double array.');
elseif ~isreal(p) || ~isa(p, 'double') || numel(p) ~= 1
    error('P must be a real double scalar.');
end

normalizecolsmx(A, p);   % Call the MEX-function with the verified inputs
```

M-code is much more convenient for input parsing, especially for majestic calling syntaxes like property/value pairs or option structures.

The actual dimensions and data of array `A` are obtained by

```
M = mxGetM(A_IN);     /* Get the dimensions of A */
N = mxGetN(A_IN);
A = mxGetPr(A_IN);    /* Get the pointer to the data of A */
```

Array elements are stored in column-major format, for example, `A[m + M*n]` (where $0 \le$ `m` $\le$ `M` $- 1$ and $0 \le$ `n` $\le$ `N` $- 1$) corresponds to matrix element `A(m+1,n+1)`.

The output `M`×`N` array `B` is created with `mxCreateDoubleMatrix`:

```
B_OUT = mxCreateDoubleMatrix(M, N, mxREAL); /* Create the output matrix */
B = mxGetPr(B_OUT);   /* Get the pointer to the data of B */
```

A double scalar can be created by setting `M = N = 1`, or more conveniently with `mxCreateDoubleScalar`:

```
B_OUT = mxCreateDoubleScalar(Value); /* Create scalar B = Value */
```

4 Numeric Arrays

The previous section showed how to handle real 2D full double arrays. But generally, a MATLAB array can be real or complex, full or sparse, with any number of dimensions, and in various datatypes. Supporting all permutations of types is an overwhelming problem, but fortunately in many applications, supporting only one or a small number of input types is reasonable. MATLAB's Bessel functions do not support `uint8` input, but who cares?

4.1 Complex arrays

If `mxIsComplex(A_IN)` is true, then `A` has an imaginary part. MATLAB represents a complex array as two separate arrays:

```
double *Ar = mxGetPr(A_IN);  /* Real data      */
double *Ai = mxGetPi(A_IN);  /* Imaginary data */
```

And `Ar[m + M*n]` and `Ai[m + M*n]` are the real and imaginary parts of `A(m+1,n+1)`.

To create a 2-D complex array, use

```
B_OUT = mxCreateDoubleMatrix(M, N, mxCOMPLEX);
```

4.2 Arrays with more than two dimensions

For 2-D arrays, we can use `mxGetM` and `mxGetN` to obtain the dimensions. For an array with more than two dimensions, use

```
size_t K = mxGetNumberOfDimensions(A_IN);
const mwSize *N = mxGetDimensions(A_IN);
```

The dimensions of the array are `N[0]` $\times$ `N[1]` $\times \cdots \times$ `N[K-1]`. The element data is then obtained as

```
double *A = mxGetPr(A_IN);
```

or if `A` is complex,

```
double *Ar = mxGetPr(A_IN);
double *Ai = mxGetPi(A_IN);
```

The elements are organized in column-major format.

	C/MEX	M-code equivalent
3-D array	`A[n0 + N[0]*(n1 + N[1]*n2)]`	`A(n0+1,n1+1,n2+1)`
K-D array	`A[` $\sum_{k=0}^{K-1} \left(\prod_{j=0}^{k-1} \texttt{N[j]}\right) n_k$ `]`	`A(`n_0`+1,`n_1`+1,...,`n_{K-1}`+1)`

To create an `N[0]` $\times$ `N[1]` $\times \cdots \times$ `N[K-1]` array, use `mxCreateNumericArray`:

```
B_OUT = mxCreateNumericArray(K, N, mxDOUBLE_CLASS, mxREAL);
```

Or for a complex array, replace `mxREAL` with `mxCOMPLEX`.

4.3 Arrays of other numeric datatypes

Different kinds of MATLAB variables and datatypes are divided into classes.

Class Name	Class ID	Class Name	Class ID
`"double"`	`mxDOUBLE_CLASS`	`"int8"`	`mxINT8_CLASS`
`"single"`	`mxSINGLE_CLASS`	`"uint8"`	`mxUINT8_CLASS`
`"logical"`	`mxLOGICAL_CLASS`	`"int16"`	`mxINT16_CLASS`
`"char"`	`mxCHAR_CLASS`	`"uint16"`	`mxUINT16_CLASS`
`"sparse"`	`mxSPARSE_CLASS`	`"int32"`	`mxINT32_CLASS`
`"struct"`	`mxSTRUCT_CLASS`	`"uint32"`	`mxUINT32_CLASS`
`"cell"`	`mxCELL_CLASS`	`"int64"`	`mxINT64_CLASS`
`"function_handle"`	`mxFUNCTION_CLASS`	`"uint64"`	`mxUINT64_CLASS`

The functions `mxGetClassID`, `mxIsClass`, and `mxGetClassName` can be used to check a variable's class, for example,

```
switch(mxGetClassID(A_IN))
{
case mxDOUBLE_CLASS:   /* Perform computation for a double array */
    MyComputationDouble(A_IN);
    break;
case mxSINGLE_CLASS:   /* Perform computation for a single array */
    MyComputationSingle(A_IN);
    break;
default:               /* A is of some other class */
    mexPrintf("A is of %s class\n", mxGetClassName(A_IN));
}
```

For a double array, we can use `mxGetPr` to get a pointer to the data. For a general numeric array, use `mxGetData` and cast the pointer to the type of the array.

```
float *A = (float *)mxGetData(A_IN);                /* Get single data  */
signed char *A = (signed char *)mxGetData(A_IN);    /* Get int8 data    */
short int *A = (short int *)mxGetData(A_IN);        /* Get int16 data   */
int *A = (int *)mxGetData(A_IN);                    /* Get int32 data   */
int64_T *A = (int64_T *)mxGetData(A_IN);            /* Get int64 data   */
```

For a complex array, use `mxGetImagData` to obtain the imaginary part. Aside from the datatype, elements are accessed in the same way as with double arrays.

To create an array of a numeric datatype, use either `mxCreateNumericMatrix` (for a 2-D array) or generally `mxCreateNumericArray`:

```
mxArray* mxCreateNumericMatrix(int m, int n,
        mxClassID class, mxComplexity ComplexFlag)

mxArray* mxCreateNumericArray(int ndim, const int *dims,
        mxClassID class, mxComplexity ComplexFlag)
```

4.4 Sparse arrays

Sparse data is complicated. Sparse arrays are always 2-D with elements of double datatype and they may be real or complex. The following functions are used to manipulate sparse arrays.

C/MEX	Meaning
`mwIndex *jc = mxGetJc(A)`	Get the `jc` indices
`mwIndex *ir = mxGetIr(A)`	Get the `ir` indices
`mxGetNzmax(A)`	Get the capacity of the array
`mxSetJc(A, jc)`	Set the `jc` indices
`mxSetIr(A, ir)`	Set the `ir` indices
`mxSetNzmax(A, nzmax)`	Set the capacity of the array

See the example MEX-function `fulltosparse.c` in `matlab/extern/examples/refbook` to see how to create a sparse matrix.

5 Creating an Uninitialized Numeric Array

This trick is so effective it gets it own section. The array creation functions (e.g., `mxCreateDoubleMatrix`) initialize the array memory by filling it with zeros. This may not seem so serious, but in fact this zero-filling initialization is a significant cost for large arrays. Moreover, initialization is usually unnecessary.

Memory initialization is costly, and can be avoided.

The steps to creating an uninitialized array are:

1. Create an empty 0×0 matrix
2. Set the desired dimensions (`mxSetM` and `mxSetN` or `mxSetDimensions`)
3. Allocate the memory with `mxMalloc` and pass it to the array with `mxSetData`. Repeat with `mxSetImagData` if creating a complex array.

For example, the following creates an uninitialized `M×N` real double matrix.

```
mxArray *B;
B = mxCreateDoubleMatrix(0, 0, mxREAL);     /* Create an empty array          */
mxSetM(M);                                  /* Set the dimensions to M x N    */
mxSetN(N);
mxSetData(B, mxMalloc(sizeof(double)*M*N)); /* Allocate memory for the array */
```

This code creates an uninitialized `N[0]` $\times$ `N[1]` $\times \cdots \times$ `N[K-1]` complex single (float) array:

```
mxArray *B;
B = mxCreateNumericMatrix(0, 0, mxSINGLE_CLASS, mxREAL);  /* Create an empty array                    */
mxSetDimensions(B, (const mwSize *)N, K);                 /* Set the dimensions to N[0] x ... x N[K-1] */
mxSetData(B, mxMalloc(sizeof(float)*NumEl));              /* Allocate memory for the real part        */
mxSetImagData(B, mxMalloc(sizeof(float)*NumEl));          /* Allocate memory for the imaginary part   */
```

where above `NumEl = N[0]*N[1]*` $\cdots$ `*N[K-1]`.

Often it is useful to create an uninitialized array having the same dimensions as an existing array. For example, given `mxArray *A`, an uninitialized `int32` array of the same dimensions is created by

```
mxArray *B;
B = mxCreateNumericMatrix(0, 0, mxINT32_CLASS, mxREAL);              /* Create an empty array */
mxSetDimensions(B, mxGetDimensions(A), mxGetNumberOfDimensions(A)); /* Set the dimensions     */
mxSetData(B, mxMalloc(4*mxGetNumberOfElements(A)));                 /* Allocate memory       */
```

If you want to initialize `B` as a copy of `A`, just use `mxDuplicateArray`:

```
mxArray *B = mxDuplicateArray(A);  /* Create B as a copy of A */
```

Section 9.2 will explain `mxMalloc` and other memory allocation functions in more detail.

6 Calling a MATLAB function from MEX

6.1 mexCallMATLAB

MEX-functions are useful because they enable calling C code from MATLAB. The reverse direction is also possible: `mexCallMATLAB` enables a MEX-function to call a MATLAB command. The syntax is

```
int mexCallMATLAB(int nlhs, mxArray *plhs[], int nrhs,
  mxArray *prhs[], const char *functionName);
```

The first four arguments are the same as those for `mexFunction` (see section 3). The fifth argument is the MATLAB function to call. It may be an operator, for example `functionName = "+"`.

`callqr.c`

```
#include <string.h>
#include "mex.h"

void DisplayMatrix(char *Name, double *Data, int M, int N)
{   /* Display matrix data */
    int m, n;
    mexPrintf("%s = \n", Name);
    for(m = 0; m < M; m++, mexPrintf("\n"))
        for(n = 0; n < N; n++)
            mexPrintf("%8.4f ", Data[m + M*n]);
}

void CallQR(double *Data, int M, int N)
{   /* Perform QR factorization by calling the MATLAB function */
    mxArray *Q, *R, *A;
    mxArray *ppLhs[2];

    DisplayMatrix("Input", Data, M, N);
    A = mxCreateDoubleMatrix(M, N, mxREAL); /* Put input in an mxArray   */
    memcpy(mxGetPr(A), Data, sizeof(double)*M*N);

    mexCallMATLAB(2, ppLhs, 1, &A, "qr");   /* Call MATLAB's qr function */
    Q = ppLhs[0];
    R = ppLhs[1];
    DisplayMatrix("Q", mxGetPr(Q), M, N);
    DisplayMatrix("R", mxGetPr(R), M, N);

    mxDestroyArray(R);                      /* No longer need these      */
    mxDestroyArray(Q);
    mxDestroyArray(A);
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    #define M_IN   prhs[0]
    if(nrhs != 1 || mxGetNumberOfDimensions(M_IN) != 2 || !mxIsDouble(M_IN))
        mexErrMsgTxt("Invalid input.");

    CallQR(mxGetPr(M_IN), mxGetM(M_IN), mxGetN(M_IN));
}
```

```
>> M = round(rand(3)*3);
>> callqr(M)
Input =
   2.0000    2.0000    0.0000
   2.0000    1.0000    1.0000
   1.0000    2.0000    0.0000
Q =
  -0.6667    0.1617   -0.7276
  -0.6667   -0.5659    0.4851
  -0.3333    0.8085    0.4851
R =
  -3.0000   -2.6667   -0.6667
   0.0000    1.3744   -0.5659
   0.0000    0.0000    0.4851
```

It is possible during `mexCallMATLAB` that an error occurs in the called function, in which case the MEX-function is terminated. To allow the MEX-function to continue running even after an error, use `mexCallMATLABWithTrap`.

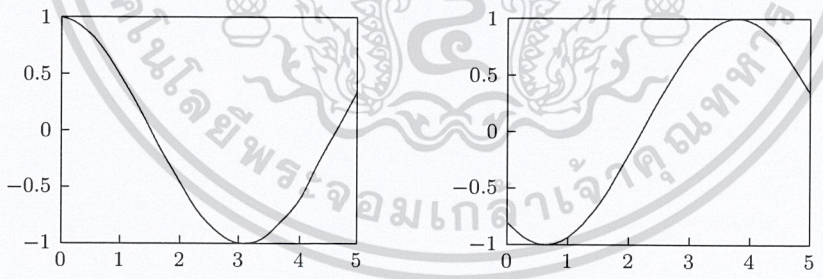
6.2 mexEvalString

Two related functions are `mexEvalString` and `mexEvalStringWithTrap`, which are MEX versions of MATLAB's `eval` command. They accept a `char*` string to be evaluated, for example

```
eval('x = linspace(0,5); for k = 1:200, plot(x, cos(x+k/20)); drawnow; end');
```

can be performed in MEX as

```
mexEvalString("x = linspace(0,5); for k = 1:200, plot(x, cos(x+k/20)); drawnow; end");
```



7 Calling a MATLAB function handle from MEX

This example solves the partial differential equation

$$\partial_t u - \partial_{xx} u = 0, \qquad u(0,t) = u(1,t) = 0,$$

and plots the solution at every timestep. It demonstrates passing a function handle to a MEX-function and allocating temporary work arrays with `mxMalloc`.

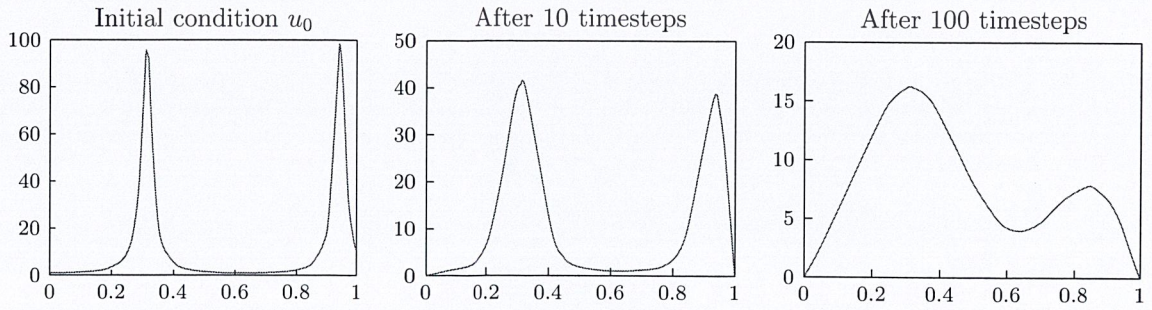
`heateq.c`

```
#include "mex.h"

#define IS_REAL_2D_FULL_DOUBLE(P) (!mxIsComplex(P) && \
mxGetNumberOfDimensions(P) == 2 && !mxIsSparse(P) && mxIsDouble(P))
#define IS_REAL_SCALAR(P) (IS_REAL_2D_FULL_DOUBLE(P) && mxGetNumberOfElements(P) == 1)

mxArray *g_PlotFcn;

void CallPlotFcn(mxArray *pu, mxArray *pt)
{   /* Use mexCallMATLAB to plot the current solution u */
    mxArray *ppFevalRhs[3] = {g_PlotFcn, pu, pt};

    mexCallMATLAB(0, NULL, 3, ppFevalRhs, "feval");  /* Call the plotfcn function handle */
    mexCallMATLAB(0, NULL, 0, NULL, "drawnow");      /* Call drawnow to refresh graphics */
}

void SolveHeatEq(mxArray *pu, double dt, size_t TimeSteps)
{   /* Crank-Nicolson method to solve u_t - u_xx = 0, u(0,t) = u(1,t) = 0 */
    mxArray *pt;
    double *u, *t, *cl, *cu, *z;
    double dx, lambda;
    size_t n, N = mxGetNumberOfElements(pu) - 1;

    pt = mxCreateDoubleMatrix(1, 1, mxREAL);
    u = mxGetPr(pu);
    t = mxGetPr(pt);
    u[0] = u[N] = 0.0;
    *t = 0.0;
    CallPlotFcn(pu, pt);                    /* Plot the initial condition */

    dx = 1.0/N;                             /* Method initializations */
    lambda = dt/(dx*dx);
    cl = mxMalloc(sizeof(double)*N);        /* Allocate temporary work arrays */
    cu = mxMalloc(sizeof(double)*N);
    z = mxMalloc(sizeof(double)*N);

    cl[1] = 1.0 + lambda;
    cu[1] = -lambda/(2.0*cl[1]);
    for(n = 2; n <= N-1; n++)
    {
        cl[n] = 1.0 + lambda + cu[n-1]*(lambda/2.0);
        cu[n] = -lambda/(2.0*cl[n]);
    }

    while(TimeSteps--)                      /* Main computation loop */
    {
```

```
        z[1] = ((1.0-lambda)*u[1] + (lambda/2.0)*u[2]) / cl[1];
        for(n = 2; n <= N-1; n++)
            z[n] = ((1.0-lambda)*u[n] + (lambda/2.0)*(u[n+1] + u[n-1] + z[n-1])) / cl[n];
        for(u[N-1] = z[N-1], n = N-2; n >= 1; n--)
            u[n] = z[n] - cu[n]*u[n+1];

        *t += dt;
        CallPlotFcn(pu, pt);              /* Plot the current solution */
    }

    mxFree(z);                            /* Free work arrays */
    mxFree(cu);
    mxFree(cl);
    mxDestroyArray(pt);
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray*prhs[])
{   /* MEX gateway */
    #define U0_IN           prhs[0]
    #define DT_IN           prhs[1]
    #define TIMESTEPS_IN    prhs[2]
    #define PLOTFCN_IN      prhs[3]
    #define U_OUT           plhs[0]

    if(nrhs != 4)                         /* Input checking */
        mexErrMsgTxt("Four input arguments required.");
    else if(nlhs > 1)
        mexErrMsgTxt("Too many output arguments.");
    else if(!IS_REAL_2D_FULL_DOUBLE(U0_IN) || !IS_REAL_SCALAR(DT_IN) || !IS_REAL_SCALAR(TIMESTEPS_IN))
        mexErrMsgTxt("Invalid input.");
    else if(mxGetClassID(PLOTFCN_IN) != mxFUNCTION_CLASS && mxGetClassID(PLOTFCN_IN) != mxCHAR_CLASS)
        mexErrMsgTxt("Fourth argument should be a function handle.");

    U_OUT = mxDuplicateArray(U0_IN);      /* Create output u by copying u0 */
    g_PlotFcn = (mxArray *)PLOTFCN_IN;
    SolveHeatEq(U_OUT, mxGetScalar(DT_IN), mxGetScalar(TIMESTEPS_IN));
    return;
}
```

In MATLAB, `heateq` can be used as

```
N = 100;
x = linspace(0,1,N+1);
u0 = 1./(1e-2 + cos(x*5).^2);       % Create the initial condition
plotfcn = @(u,t) plot(x, u, 'r');   % Create plotting function
heateq(u0, 1e-4, 100, plotfcn);     % Solve heat equation
```

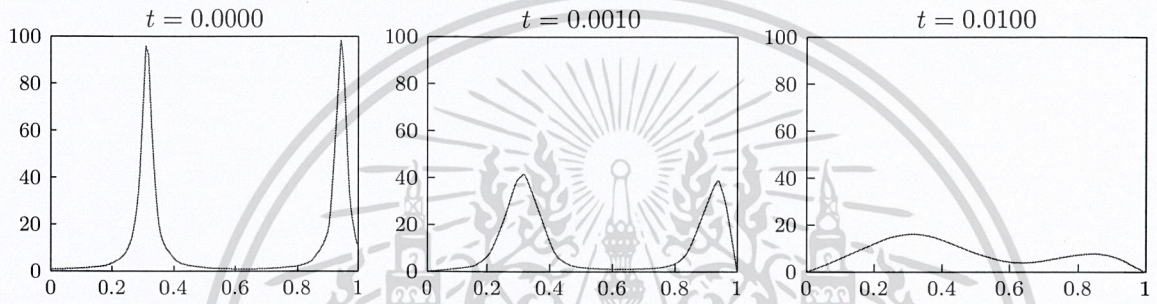


For more control over the plot function, we can write it as an M-function:

myplot.m

```
function myplot(u,t)
% Plotting function for heateq.c
plot(linspace(0,1,length(u)), u, 'r');
ylim([0,100]);                  % Freeze the y axis
title(sprintf('t = %.4f',t));  % Display t in the plot title
```

Calling `heateq(u0, 1e-4, 100, 'myplot')` produces



The actual figure is animated—try it to get the full effect.

In the example, the `plotfcn` function handle is called in `CallPlotFcn`:

```
mxArray *g_PlotFcn;

void CallPlotFcn(mxArray *pu, mxArray *pt)
{   /* Use mexCallMATLAB to plot the current solution u */
    mxArray *ppFevalRhs[3] = {g_PlotFcn, pu, pt};

    mexCallMATLAB(0, NULL, 3, ppFevalRhs, "feval");  /* Call the plotfcn function handle */
    mexCallMATLAB(0, NULL, 0, NULL, "drawnow");      /* Call drawnow to refresh graphics */
}
```

The trick is to pass the function handle to `feval`, which in turn evaluates the function handle. The first `mexCallMATLAB` call is equivalent to `feval(plotfcn, u, t)`. The second `mexCallMATLAB` calls `drawnow` to refresh graphics; this is necessary to watch the plot change in realtime during the computation. See section 9.3 for another example of calling a function handle.

8 Calling MATLAB from a non-MEX Program

We have discussed using `mexCallMATLAB` to call a MATLAB command from within a MEX-function. But is it possible to call MATLAB from a program that is *not* a MEX-function? The answer is yes, it is possible! But beware my approach is quite inefficient and roundabout.

The key is that MATLAB can be started to run a command, for example

```
matlab -r "x=magic(6);save('out.txt','x','-ascii');exit"
```

This starts MATLAB, creates a 6×6 magic matrix, saves it to out.txt, then promptly exits. More practically, you should start a script containing the actual commands.

```
matlab -r makemagic
```

Under UNIX, you can add the `-nodisplay` flag to hide the MATLAB window.

The following is a simple C program that calls MATLAB to create magic matrices of a specified size:

`makemagic.c`

```
/* Run as "makemagic N" to make an NxN magic matrix */
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    float v;
    int n, N  = (argc > 1) ? atoi(argv[1]) : 6;

    /* Write a MATLAB script */
    FILE *fp = fopen("makemagic.m", "wt");
    fprintf(fp, "x = magic(%d);\n"          /* Make an NxN magic matrix */
                "save magic.txt x -ascii\n" /* Save to magic.txt        */
                "exit;", N);                /* Exit MATLAB              */
    fclose(fp);

    /* Call MATLAB to run the script */
    printf("Calling MATLAB...\n");
    system("matlab -r makemagic");

    /* Read from the output file */
    fp = fopen("magic.txt", "rt");
    while(!feof(fp))
    {
        for(n = 0; n < N && fscanf(fp, "%f", &v) == 1; n++)
            printf("%4d ", (int)v);
        printf("\n");
    }
    fclose(fp);

    return 0;
}
```

9 Memory

9.1 Remembering variables between calls

C variables that are declared globally are remembered between calls. The following MEX-function counts the number of times it was called.

remember.c

```
#include "mex.h"

/* Count is a global variable, so it will be remembered between calls */
static int Count = 1;

void MyExit()
{
    mexPrintf("MyExit() called!\n");
    /* Do cleanup here ... */
    return;
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    mexAtExit(&MyExit); /* Register MyExit() to run when MEX-function is cleared */
    mexPrintf("Count=%d\n", Count);
    Count++;  /* Increment Count */
    return;
}
```

This MEX-function also demonstrates `mexAtExit`, which allows us to run a cleanup function when the MEX-function is cleared or when MATLAB exits.

```
>> remember
Count=1
>> remember
Count=2
>> remember
Count=3
>> clear remember
MyExit() called!
>> remember
Count=1
```

A MEX-function can be explicitly cleared by `clear` *function* or `clear all`.

9.2 Dynamic memory allocation

The MEX interface provides several functions for managing dynamic memory:

C/MEX	Meaning	Standard C equivalent
`mxMalloc(Size)`	Allocate memory	`malloc(Size)`
`mxCalloc(Num, Size)`	Allocate memory initialized to zero	`calloc(Num, Size)`
`mxRealloc(Ptr, NewSize)`	Change size of allocated memory block	`realloc(Ptr, NewSize)`
`mxFree(Ptr)`	Release memory allocated by one of the above	`free(Ptr)`

It is also possible to use the standard C `malloc`, etc., or C++ `new` and `delete` within a MEX-function. The advantage of `mxMalloc`, etc. is that they use MATLAB's internal memory manager so that memory is properly released on error or abort (Ctrl+C).

A useful function when allocating memory is `mxGetElementSize`, which returns the number of bytes needed to store one element of a MATLAB variable,

```
size_t BytesPerElement = mxGetElementSize((const mxArray *)A);
```

9.3 Persistent Memory

By default, memory allocated by `mxMalloc`, etc. is automatically released when the MEX-function completes. Calling `mexMakeMemoryPersistent(Ptr)` makes the memory persistent so that it is remembered between calls.

The following MEX-function wraps `feval` to remember function evaluations that have already been computed. It uses `mexMakeMemoryPersistent` to store a table of precomputed values.

pfeval.c

```
#include "mex.h"

#define INITIAL_TABLE_CAPACITY   64

/* Table for holding precomputed values */
static struct {
    double *X;     /* Array of x values                    */
    double *Y;     /* Array of corresponding y values      */
    int Size;      /* Number of entries in the table       */
    int Capacity; /* Table capacity                        */
} Table = {0, 0, 0, 0};

void ReallocTable(int NewCapacity)
{   /* (Re)allocate the table */
    if(!(Table.X = (double *)mxRealloc(Table.X, sizeof(double)*NewCapacity))
        || !(Table.Y = (double *)mxRealloc(Table.Y, sizeof(double)*NewCapacity)))
        mexErrMsgTxt("Out of memory");
    mexMakeMemoryPersistent(Table.X);  /* Make the table memory persistent */
    mexMakeMemoryPersistent(Table.Y);  /* Make the table memory persistent */
    Table.Capacity = NewCapacity;
}

void AddToTable(double x, double y)
{   /* Add (x,y) to the table */
    if(Table.Size == Table.Capacity)
        ReallocTable(2*Table.Capacity);
    Table.X[Table.Size] = x;
    Table.Y[Table.Size++] = y;
}

mxArray* EvaluateFunction(const mxArray *pFunction, const mxArray *px)
{   /* Evaluate function handle pFunction at px */
    const mxArray *ppFevalRhs[2] = {pFunction, px};
```

```
    mxArray *py;
    mexPrintf("Evaluating f(x = %g)\n", mxGetScalar(px));
    mexCallMATLAB(1, &py, 2, (mxArray **)ppFevalRhs, "feval");
    return py;
}

void MyExit()
{   /* Clean up */
    mxFree(Table.X);
    mxFree(Table.Y);
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    #define FCN_IN   prhs[0]
    #define X_IN     prhs[1]
    #define Y_OUT    plhs[0]
    double x, *y;
    int i;

    if(nrhs != 2)
        mexErrMsgTxt("Two input arguments required.");
    else if(mxGetClassID(FCN_IN) != mxFUNCTION_CLASS && mxGetClassID(FCN_IN) != mxCHAR_CLASS)
        mexErrMsgTxt("First argument should be a function handle.");
    else if(!mxIsDouble(X_IN) || mxGetNumberOfElements(X_IN) != 1)
            mexErrMsgTxt("X must be a real double scalar.");

    x = mxGetScalar(X_IN);
    mexAtExit(&MyExit); /* Register MyExit() to run when MEX function is cleared */

    if(!Table.X || !Table.Y)                /* This happens on the first call    */
        ReallocTable(INITIAL_TABLE_CAPACITY); /* Allocate precomputed values table */

    /* Search for x in the table */
    for(i = 0; i < Table.Size; i++)
        if(x == Table.X[i])
        {
            mexPrintf("Found precomputed value for x = %g\n", x);
            y = mxGetPr(Y_OUT = mxCreateDoubleMatrix(1, 1, mxREAL));
            *y = Table.Y[i];
            return;
        }

    /* x is not yet in the table */
    Y_OUT = EvaluateFunction(FCN_IN, X_IN);  /* Evaluate the function        */
    AddToTable(x, mxGetScalar(Y_OUT));       /* Make a new entry in the table */
    return;
}
```

As a simple example, here is `pfeval` applied to $f(x) = x^2$:

```
>> f = @(x) x.^2;  % Define the function to evaluate
>> pfeval(f, 4)
Evaluating f(x = 4)
ans =
    16
>> pfeval(f, 10)
Evaluating f(x = 10)
ans =
   100
```

```
>> pfeval(f, 4)
Found precomputed value for x = 4
ans =
    16
```

Remark: I do not recommend using `pfeval` in practice. It does not work correctly if called with more than one function handle. Additionally, it would be better to use a binary search on a sorted table and to do error checking when calling the function handle.

Persistent memory should be used in combination with `mexAtExit`. You should write a cleanup function that releases the persistent memory and use `mexAtExit` to register it. If you do not do this, persistent memory is never released, and MATLAB will leak memory!

9.4 Locking

A MEX-function is "unlocked" by default, meaning it may be cleared at any time. To prevent the MEX-function from being cleared, call `mexLock`. Call the function `mexUnlock` to unlock it again. If `mexLock` is called n times, `mexUnlock` must be called n times to unlock it. The `mexIsLocked` function checks whether the MEX-function is locked.

10 Non-Numeric Variables

There are specialized interface functions for handling non-numeric classes. Non-numeric variables are still represented with `mxArray` objects, and some functions like `mxDestroyArray` work on any class. Use `mxGetClassID` or `mxGetClassName` as described in section 4.3 to determine the class of a variable. You can alternatively use `mxIsLogical`, `mxIsChar`, `mxIsCell`, `mxIsStruct`, or `mxIsFunctionHandle`.

10.1 Logicals

Logicals are not so different from numeric classes. Logical elements are represented in C/MEX as type `mxLogical` (a typedef for `bool` or `unsigned char` depending on platform) and are arranged in column-major organization. The following functions are used to create and handle logical arrays.

Function	Description
`L = mxCreateLogicalScalar(Value)`	Create a logical `L = Value`
`L = mxCreateLogicalMatrix(M,N)`	Create an `M`×`N` logical matrix
`L = mxCreateLogicalArray(K,(const mwSize *)N)`	Create an `N[0]` × ··· × `N[K-1]` logical array
`mxLogical *Data = mxGetLogicals(L)`	Get pointer to the logical data
`mxIsLogicalScalar(L)`	True if `L` is logical and scalar
`mxIsLogicalScalarTrue(L)`	True if logical scalar `L` equals true

10.2 Char arrays

Char arrays (strings) are represented as UTF-16 `mxChar` elements. For convenience, there are functions to convert between char arrays and null-terminated C `char*` strings in the local codepage encoding.

Function	Description
`S = mxCreateString("My string")`	Create a 1×`N` string from a `char*` string
`S = mxCreateCharMatrixFromStrings(M, (const char **)Str)`	Create matrix from `Str[0]`,...,`Str[M-1]`
`S = mxCreateCharArray(K, (const mwSize *)N)`	Create an `N[0]` × ··· × `N[K-1]` char array
`mxGetString(S, Buf, BufLen)`	Read string `S` into `char*` string `Buf`

Warning: `mxGetString` will truncate the result if it is too large to fit in `Buf`. To avoid truncation, `BufLen` should be at least `mxGetNumerOfElements(S)*sizeof(mxChar) + 1`.

`stringhello.c`

```
/* A string version of the "Hello, world!" example */
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    plhs[0] = mxCreateString("Hello, world!");
    return;
}
```

The UTF-16 data may also be manipulated directly if you are determined to do so. Use `mxGetData` to get an `mxChar*` pointer to the UTF-16 data. For example, the following MEX-function creates the string "明天.txt" containing Chinese characters.

`mingtian.c`

```
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    #define M_OUT   plhs[0]
    mxChar *Data;
    /* UTF-16LE data */
    unsigned short MingTian[] = {0xFEFF, 0x660E, 0x5929, 0x002E, 0x0074, 0x0078, 0x0074};
    int k, Size[] = {1, 7};

    M_OUT = mxCreateCharArray(2, (const int*)Size);
    Data = mxGetData(M_OUT);

    for(k = 0; k < 7; k++)
        Data[k] = MingTian[k];
}
```

Running MEX-function `mingtian` on the console will show non-ASCII characters as boxes (the data is there, but the console is limited in what it can display).

```
>> mingtian

ans =

□□.txt
```

So to see this string, we need to get it out of MATLAB. This script attempts to write a file with `mingtian`'s output as the filename:

```
Status = {'Succeeded', 'Failed'};
s = mingtian;
fid = fopen(s, 'w');
fprintf('Creating file with UTF-16 filename:  %s\n', Status{(fid == -1)+1});
fclose(fid);
fid = fopen(s, 'r');
fprintf('Opening file with UTF-16 filename:   %s\n', Status{(fid == -1)+1});
fclose(fid);
```

If successful, there should be a new file called 明天.txt in the current directory.

Remark: You don't need to use MEX to create exotic UTF-16 characters. The example above can be reproduced on the console as

```
>> s = char([65279, 26126, 22825,'.txt']);
>> fid = fopen(s, 'w'); Status = {'Succeeded', 'Failed'}; Status{(fid == -1)+1}, fclose(fid);

ans =
Succeeded
```

10.3 Cell arrays

A cell array is essentially an array of `mxArray` pointers. The functions `mxGetCell` and `mxSetCell` are used to access or change the `mxArray*` of a cell.

Function	Description
`C = mxCreateCellMatrix(M, N)`	Create an `M×N` cell array
`C = mxCreateCellArray(K, (const mwSize *)N)`	Create an `N[0]` × ⋯ × `N[K-1]` cell array
`mxArray *A = mxGetCell(C, i)`	Get contents of the `i`th cell, `A = C{i+1}`
`mxSetCell(C, i, A)`	Set contents of the `i`th cell, `C{i+1} = A`

The index `i` in `mxGetCell` and `mxSetCell` is zero based.

10.4 Structs

Like a cell array, a struct array is essentially an array of `mxArray` pointers, but with an additional dimension indexed by field names. Each field name has a corresponding field number; the fields are numbered in the order in which they were added to the struct. Fields may be referenced either by name or by number.

A struct array is manipulated as a whole by the following functions.

Function	Description
`mxGetNumberOfFields(X)`	Get the number of fields in struct `X`
`X = mxCreateStructMatrix(M, N, NumFields, (const char **)Str)`	Create an `M×N` struct
`X = mxCreateStructArray(K, N, NumFields, (const char **)Str)`	Create an `N[0]` × ⋯ × `N[K-1]` struct
`mxAddField(X, (const char *)Str)`	Add a new field to struct `X`
`mxRemoveField(X, k)`	Remove the `k`th field (if it exists)
`int k = mxGetFieldNumber(X, "myfield")`	Get the field number of a field name
`const char *Name = mxGetFieldNameByNumber(X, k)`	Get the field name of the `k`th field

In the creation functions `mxCreateStructMatrix` and `mxCreateStructArray`, the field names are given by null-terminated `char*` strings `Str[0]`, ..., `Str[NumFields-1]`.

Individual fields are accessed and changed with the following functions.

Function	Description
`mxArray *F = mxGetField(X, i, "myfield")`	Get the `i`th element's field `F = X(i+1).myfield`
`mxArray *F = mxGetFieldByNumber(X, i, k)`	Get the `i`th element's `k`th field
`mxSetField(X, i, "myfield", F)`	Set the `i`th element's field `X(i+1).myfield = F`
`mxSetFieldByNumber(X, i, k, F)`	Set the `i`th element's `k`th field

The index `i` above is zero based.

10.5 Function handles

Function handles are slippery creatures with very little support within MEX. Even in M-code, they have some surprising limitations:

```
>> f = @(x)cos(2*x); g = @(x)cos(2*x);
>> isequal(f,g)

ans =
     0
```

At least a function handle is equal to itself:

```
>> isequal(f,f)

ans =
     1
```

To identify an `mxArray*` as a function handle, use its class ID or `mxIsFunctionHandle`.

To execute a function handle, use `mexCallMATLAB` to call `feval`. For example, the following evaluates a function handle $y = f(x)$:

```
mxArray* EvaluateFunction(const mxArray *f, const mxArray *x)
{   /* Evaluate function handle by caling y = feval(f,x) */
    mxArray *y;
    const mxArray *ppFevalRhs[2] = {f, x};
    mexCallMATLAB(1, &y, 2, (mxArray **)ppFevalRhs, "feval");
    return y;
}
```

A function handle with multiple arguments can be evaluated similarly (see section 7 for an example).

Similarly, `mexCallMATLAB` may be used to perform other operations with function handles.

MATLAB function	Description
`y = feval(f,x)`	Evaluate a function handle
`functions(f)`	Get information about a function handle
`s = func2str(f)`	Convert function handle to string
`f = str2func(s)`	Convert string to function handle (see below)

In MATLAB 2009a and newer, it is possible to create a function handle in MEX by calling `str2func`. Some older versions of MATLAB have this command but do not support anonymous function creation.

```
mxArray* CreateFunctionFromString(const char *Str)
{   /* Create a function handle from a string */
    mxArray *f, *str2func = mxCreateString("str2func"), *s = mxCreateString(Str);
    const mxArray *ppFevalRhs[2] = {str2func, s};
    mexCallMATLAB(1, &f, 1, (mxArray **)ppFevalRhs, "feval");
    mexDestroyArray(s);
    return f;
}
```

For example, `f = CreateFunctionFromString("@(x) x^2")` creates the square function.

FFTs with FFTW

To perform an FFT within a MEX-function, you could use `mexCallMATLAB` to call MATLAB's `fft` command. However, this approach has the overhead that `fft` must allocate a new mxArray to hold the resulting FFT, as well as the overhead and nuisance of wrapping up the data in mxArray objects. It is more efficient to perform FFTs directly by calling the FFTW library.

11.1 A brief introduction to FFTW3

The FFTW3 library is available on the web at `www.fftw.org`. The library can perform FFTs of any size and dimension. It can also perform related trigonometric transforms.

To perform a transform, the type, size, etc. are specified to FFTW to create a *plan*. FFTW considers many possible algorithms and estimates which will be fastest for the specified transform. The transform itself is then performed by executing the plan. The plan may be executed any number of times. Finally, the plan is destroyed to release the associated memory.

Two common ways to store complex arrays are *split format* and *interleaved format*. Section 4.1 explained how complex MATLAB arrays are represented with two separate blocks of memory, one for the real part and the other for the imaginary part,

$$\text{Split format:} \quad r_0, r_1, r_2, \ldots, r_{N-1}, \quad \ldots \quad i_0, i_1, i_2, \ldots, i_{N-1}.$$

In FFTW, such a two-block organization is called *split format*. Another common way to arrange complex data is to interleave the real and imaginary parts into a single contiguous block of memory,

$$\text{Interleaved format:} \quad r_0, i_0, r_1, i_1, r_2, i_2, \ldots, r_{N-1}, i_{N-1}.$$

FFTW can handle both interleaved and split formats. Complex MATLAB arrays are always split format, so you must use split format to store a complex FFT output directly in a MATLAB array.

11.2 FFTW3 examples

To use FFTW3, we need to include `fftw3.h`. We also need to include option `-lfftw3` when calling the `mex` command to link the MEX-function with the FFTW3 library:

```
mex mymexfunction.c -lfftw3
```

Additional options may be necessary depending on how FFTW3 is installed on your system; see the `-l` and `-L` options in `help mex`.

It is helpful to define `DivideArray`, which we will use to normalize results after inverse transforms.

```
#include <fftw3.h>

/* Divide an array per-element (used for IFFT normalization) */
void DivideArray(double *Data, int NumEl, double Divisor)
{
    int n;
    for(n = 0; n < NumEl; n++)
        Data[n] /= Divisor;
}
```

We first consider transforms with the interleaved format since FFTW3's interface is simpler in this case. The following function computes the 1D FFT of length `N` on complex array `X` to produce complex output array `Y`.

```
/* FFT1DInterleaved  1D FFT complex-to-complex interleaved format
   Inputs:
   N      Length of the array
   X      Input array, X[2n] = real part and X[2n+1] = imag part of the nth element (n = 0, ..., N - 1)
   Sign   -1 = forward transform, +1 =  inverse transform

   Output:
   Y      Output array, Y[2n] = real part and Y[2n+1] = imag part of the nth element (n = 0, ..., N - 1)
*/
void FFT1DInterleaved(int N, double *X, double *Y, int Sign)
{
    fftw_plan Plan;
    if(!(Plan = fftw_plan_dft_1d(N, (fftw_complex *)X, (fftw_complex *)Y, Sign, FFTW_ESTIMATE)))
        mexErrMsgTxt("FFTW3 failed to create plan.");
    fftw_execute(Plan);
    fftw_destroy_plan(Plan);

    if(Sign == 1)  /* Normalize the result after an inverse transform */
        DivideArray(Y, 2*N, N);
}
```

Similarly, we can compute 2D FFTs as

```
/* FFT2DInterleaved  2D FFT complex-to-complex interleaved format */
/* X[2*(m + M*n)] = real part and X[2*(m + M*n)+1] = imag part of the (m,n)th element, and similarly for Y */
void FFT2DInterleaved(int M, int N, double *X, double *Y, int Sign)
{
    fftw_plan Plan;
    if(!(Plan = fftw_plan_dft_2d(N, M, (fftw_complex *)X, (fftw_complex *)Y, Sign, FFTW_ESTIMATE)))
        mexErrMsgTxt("FFTW3 failed to create plan.");
    fftw_execute(Plan);
    fftw_destroy_plan(Plan);

    if(Sign == 1)
        DivideArray(Y, 2*M*N, M*N);
}
```

Now we consider split format. Performing FFTs on split format arrays requires using FFTW3's more involved guru interface.

```
/* FFT1DSplit  1D FFT complex-to-complex split format
   Inputs:
   N      Length of the array
   XReal  Real part of the input array, XReal[n] = real part of the nth element
```

```
    XImag  Imaginary part of the input array, XImag[n] = imag part of the nth element
    Sign   -1 = forward transform, +1 =  inverse transform

    Output:
    YReal  Real part of the output array
    YImag  Imaginary part of the output array
*/
void FFT1DSplit(int N, double *XReal, double *XImag, double *YReal, double *YImag, int Sign)
{
    fftw_plan Plan;
    fftw_iodim Dim;

    Dim.n = N;
    Dim.is = 1;
    Dim.os = 1;

    if(!(Plan = fftw_plan_guru_split_dft(1, &Dim, 0, NULL,
        XReal, XImag, YReal, YImag, FFTW_ESTIMATE)))
        mexErrMsgTxt("FFTW3 failed to create plan.");

    if(Sign == -1)
        fftw_execute_split_dft(Plan, XReal, XImag, YReal, YImag);
    else
    {
        fftw_execute_split_dft(Plan, XImag, XReal, YImag, YReal);
        DivideArray(YReal, N, N);
        DivideArray(YImag, N, N);
    }

    fftw_destroy_plan(Plan);
}
```

Finally, here is a general function for the N-D FFT with split format:

```
/* FFTNDSplit  ND FFT complex-to-complex split format
    Inputs:
    NumDims  Number of dimensions
    N        Array of dimension sizes
    XReal    Real part of the input, an N[0] x N[1] x ... x N[NumDims-1] array in column-major format
    XImag    Imaginary part of the input
    Sign     -1 = forward transform, +1 =  inverse transform

    Output:
    YReal    Real part of the output array
    YImag    Imaginary part of the output array
*/
void FFTNDSplit(int NumDims, const int N[], double *XReal, double *XImag, double *YReal, double *YImag, int Sign)
{
    fftw_plan Plan;
    fftw_iodim Dim[NumDims];
    int k, NumEl;

    for(k = 0, NumEl = 1; k < NumDims; k++)
    {
        Dim[NumDims-k-1].n = N[k];
        Dim[NumDims-k-1].is = Dim[NumDims-k-1].os = (k == 0) ? 1 : (N[k-1] * Dim[NumDims-k].is);
        NumEl *= N[k];
    }

    if(!(Plan = fftw_plan_guru_split_dft(NumDims, Dim, 0, NULL,
        XReal, XImag, YReal, YImag, FFTW_ESTIMATE)))
```

```
        mexErrMsgTxt("FFTW3 failed to create plan.");

    if(Sign == -1)
        fftw_execute_split_dft(Plan, XReal, XImag, YReal, YImag);
    else
    {
        fftw_execute_split_dft(Plan, XImag, XReal, YImag, YReal);
        DivideArray(YReal, NumEl, NumEl);
        DivideArray(YImag, NumEl, NumEl);
    }

    fftw_destroy_plan(Plan);
    return;
}
```

Remark: To perform transforms in-place, simply set `Y = X` (or `YReal = XReal` and `YImag = XImag`) when calling the above functions.

Remark: The `DivideArray` function scales the result by $1/N$. It is often possible to absorb this scale factor elsewhere to avoid this computation.

Remark: There are many possibilities in FFTW3 beyond the scope of this document. It is possible to perform multiple FFTs in a single plan, which may be more efficient than performing multiple plans. Aside from complex-to-complex transforms, FFTW3 can also perform real-to-complex, complex-to-real, and real-to-real transforms. See `www.fftw.org/fftw3_doc` for more details.

12 Miscellaneous

There are a few other interface functions in MEX that we haven't discussed yet. They are mostly analogues of basic M-code commands.

C/MEX	Meaning	M-code equivalent
`mexPrint("Hello")`	Print a string	`disp('Hello')`
`mexPrintf("x=%d", x)`	Print a formatted string	`fprintf('x=%d', x)`
`mexWarnMsgTxt("Trouble")`	Display a warning message	`warning('Trouble')`
`mexErrMsgTxt("Abort!")`	Display a error message	`error('Abort!')`
`mexFunctionName()`	Get the MEX-function's name	`mfilename`
`mexGet(h,"Prop")`	Get a property on object `h`	`get(h,'Prop')`
`mexSet(h,"Prop",Value)`	Set a property on object `h`	`set(h,'Prop',Value)`
`mexGetVariable`	Copy variable from a workspace	`evalin(WS,'Var')`
`mexGetVariablePtr`	Get variable read-only pointer	—
`mexPutVariable`	Create variable in a workspace	`assignin`

There are also functions for manipulating MATLAB MAT files. An object of type `MATFile*` represents a handle to an open MAT file.

C/MEX	Meaning
`MATFile *mfp = matOpen("my.mat", Mode)`	Open a MAT file
`matClose(mfp)`	Close MAT file
`const char *Str = matGetDir(mfp, &Num)`	Get list of variable names in the file
`mxArray *V = matGetVariable(mfp, Name)`	Get the variable named `Name`
`mxArray *V = matGetNextVariable(mfp, &Name)`	Get the next variable and its name
`matGetNextVariableInfo`	Get header info about a variable
`matPutVariable(mfp, Name, V)`	Write variable `V` with name `Name`
`matPutVariableGlobal`	Write variable as global
`matDeleteVariable(mfp, Name)`	Delete the variable named `Name`

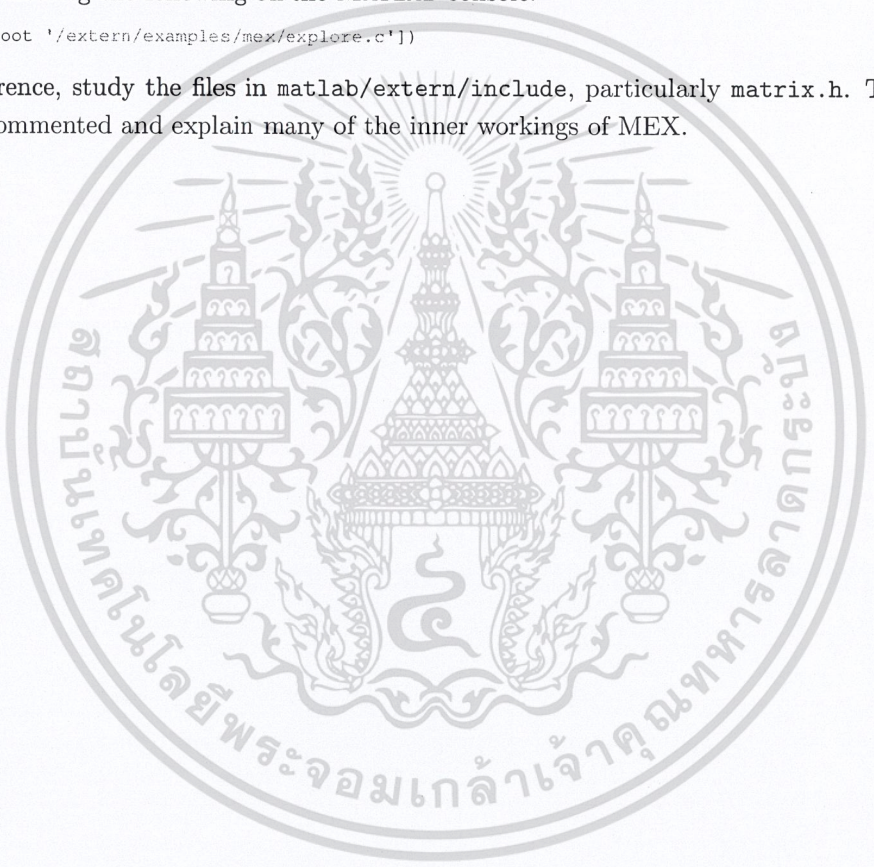
13 Further Reading

All MEX interface functions are documented on the web in online function references (search for example "matlab mxSetPr"). Information is otherwise limited, but there is MEX wisdom to be found scattered through forums and buried within the dark source code of existing MEX-functions.

For more MEX-function examples, study the files in `matlab/extern/examples`. For instance, the example "explore.c" shows how to read the data of a variety of different MATLAB variables. You can open this file by entering the following on the MATLAB console:

```
>> edit([matlabroot '/extern/examples/mex/explore.c'])
```

For detailed reference, study the files in `matlab/extern/include`, particularly `matrix.h`. These files are thoroughly commented and explain many of the inner workings of MEX.



หนังสืออ้างอิง

[1] THE MATH WORKS Ins, “C-MEX S-Function”, pp.1-28

[2]โอภาส ศิริครรชิตถาวร, “ปฏิบัติการ ARM7 LPC2148 ด้วยภาษาซี” ,หน้า1-92 และหน้า 111-144

[3] นคร ภักดีชาติ, โอภาส ศิริครรชิตถาวร, “คู่มือทดลองไมโครคอนโทรล เลอร์ 32 บิต ตระกูล ARM7 เบื้องต้นฉบับ LPC2148 กับ Sourcery G++ C คอมไพเลอร์”, หน้า1-96

[4] นิรุธ อำนวยศิลป์, “Visual C++ Version6.0”, มีนาคม 2549, หน้า 1-326 และหน้า 349-366