เตียงเคลื่อนย้ายผู้ป่วยแบบกันกระเทือน

VIBRATED PROTECTIVE BED FOR AMBULANCE

โดย

นางสาววิภาวดี สาระบุตร

นายวิสุทธิ์ พรเจริญประเสริฐ

นางสาววีรภัทรา พูลสวัสดิ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมแมคคาทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

ปริญญานิพนธ์ปีการศึกษา 2551

ภาควิชาวิศวกรรมระบบควบคุม สาขาวิศวกรรมแมคคาทรอนิกส์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เตียงขนย้ายผู้ป่วยแบบกันกระเทือน

VIBRATED PROTECTIVE BED FOR AMBULANCE

ผู้จัดทำ 1. นางสาววิภาวดี สาระบุตร

2. นายวิสุทธิ์ พรเจริญประเสริฐ

3. นางสาววีรภัทรา พูลสวัสดิ์

.....อาจารย์ที่ปรึกษา

(อาจารย์นพดล มณีรัตน์)

เตียงขนย้ายผู้ป่วยแบบกันกระเทือน

โดย
น.ส. วิภาวดี สาระบุตร
นายวิสุทธิ์ พรเจริญประเสริฐ
นางสาววีรภัทรา พูลสวัสดิ์

อาจารย์ที่ปรึกษา
ดร. นพดล มณีรัตน์

ปีการศึกษา 2551

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ นำเสนอทฤษฎีและการออกแบบระบบควบคุมสำหรับควบคุมการสั่นสะเทือนเนื่องจากการขนย้ายผู้ป่วยที่บาดเจ็บจากสาเหตุต่างๆ ที่ไม่สามารถได้รับการกระทบกระเทือนอย่างรุนแรง เช่น ผู้ป่วยที่มีอาการบาดเจ็บกระดูกสันหลังหรือทุภพลภาพ ซึ่งในการขนย้ายผู้ป่วยโดยรถพยาบาลในสภาพพื้นผิวต่างๆ ของถนน ที่อาจทำให้เกิดการสั่นไหวของเตียงไปตามแรงรถขณะขับเคลื่อน

โดยได้ทำการศึกษากรณีต่างๆ ที่เป็นสาเหตุทำให้เตียงเกิดการสั่นสะเทือน แล้วนำมาพิจารณาเพื่อนำมาควบคุมและแก้ไขระบบให้ผู้ป่วยได้รับความกระทบกระเทือนน้อยที่สุด โดยใช้โช๊คเป็นตัวรองรับแรงสั่นสะเทือนซึ่งสามารถปรับได้ 2 ระดับเพื่อให้เหมาะสมกับสภาพพื้นผิวของถนนและมี dsPIC30F2010 เป็นไมโครคอนโทรลเลอร์ซึ่งเป็นตัวประมวลผลและควบคุมโช๊คอัพให้ทำการปรับไปยังสภาพถนนที่เหมาะสมในการสั่นสะเทือนที่ระดับต่างๆ โดยมี Linear Potentiometer เป็นเซนเซอร์ตรวจจับระยะการสั่นที่เปลี่ยนแปลงไป และใช้ตัวควบคุมชนิดพีไอดี (PID Controller) ในการควบคุมระบบให้มีเสถียรภาพที่ดีขึ้น

VIBRATED PROTECTIVE BED FOR AMBULANCE

By

Ms. Wiphawadee Sarabute

Mr. Wisut Porncharoenprasert

Ms. Weeraphathra Phunsawat

Advisor

Dr. Noppadol Maneerat

Academic Year 2008

ABSTRACT

This thesis presents theory and implementation procedures of how to reduce vibration of emergency bed which is transporting a patient who has a bad injury of backbones while being on the vehicle. A simulation of emergency bed is done using PID controller to make system more stable. The performance of emergency bed control system provides good absorption to make patient comfortable by using shock absorber and it can adjust 2 levels for each surface of the road. It has been found that DsPIC30F2010 is using for controlling shock absorber to a suitable level of vibration and also use linear potentiometer to detect position of shock absorber when it is vibrating to reduce vibration of the road.

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์ฉบับนี้ สามารถเสร็จลุล่วงไปได้ด้วยดี เพราะได้รับความช่วยเหลือเป็นอย่างดีจาก ดร.นพดล มณีรัตน์ ที่ได้กรุณาให้คำปรึกษาแนะนำที่ดีมาตลอดตั้งแต่ต้น รวมทั้งอุปกรณ์ที่จำเป็น และความช่วยเหลืออื่นๆ ที่เป็นประโยชน์ต่อโครงงาน ผู้จัดทำรู้สึกซาบซึ้งและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณเพื่อนๆ ทุกคนที่คอยให้กำลังใจ สนับสนุนอุปกรณ์ที่ขาดเหลือ กระตุ้นเตือน รวมทั้งคอยถามไถ่ความคืบหน้าของโครงงานอยู่เสมอ

สุดท้ายนี้ผู้จัดทำขอขอบพระคุณบิดา มารดา และครอบครัว ที่คอยเป็นกำลังใจที่ดีตลอดมา รวมถึงการสนับสนุนในเรื่องของงบประมาณที่ขาดเหลือ ตลอดจนเป็นแรงบันดาลใจที่ดีที่สุดที่ทำให้โครงงานนี้เสร็จสมบูรณ์ได้

ผู้จัดทำ
นางสาววิภาวดี สาระบุตร
นายวิสุทธิ์ พรเจริญประเสริฐ
นางสาววีรภัทรา พูลสวัสดิ์

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญภาพ

สารบัญภาพ(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

ในปัจจุบันการขนย้ายผู้ป่วยอาจมีความเสี่ยงในการทำให้ผู้ป่วยเกิดการบาดเจ็บมากขึ้น เนื่องจากการเดินทาง อาจทำให้ผู้ป่วยที่มีปัญหาด้านกระดูกสันหลังหรือปัญหาที่ไม่สามารถเกิดกระทบกระเทือนรุนแรงได้ จนทำให้ผู้ป่วยพิการหรือเสียชีวิตจากการเดินทางได้

โดยโครงงานนี้ได้นำเสนอการออกแบบระบบควบคุมที่สามารถลดการสั่นสะเทือนของเตียงในขณะขนย้ายผู้ป่วยได้ โดยการศึกษาระบบควบคุมชนิดพีไอดี (PID Controller) ธรรมชาติของการสั่นสะเทือน และ อุปกรณ์การดูดซับการสั่น เพื่อนำมาประยุกต์ใช้กับเตียงผู้ป่วย โดยนำอุปกรณ์การดูดซับการสั่นมาประกอบติดกับเตียงเพื่อลดการสั่นสะเทือน และนำระบบควบคุมชนิดพีไอดี มาพิจารณาการสั่นแต่ละแบบ เพื่อปรับระดับของอุปกรณ์การดูดซับการสั่น ว่าระดับเท่าใดจึงจะเหมาะสมกับการใช้งานในสภาวะนั้นๆ

1.1 วัตถุประสงค์

1) เพื่อศึกษาการใช้ระบบควบคุมชนิดพีไอดี (PID Controller)
2) เพื่อศึกษารูปแบบการสั่น
3) เพื่อศึกษาหาระดับที่เหมาะสมในการใช้อุปกรณ์ดูดซับการสั่น
4) เพื่อให้สามารถควบคุมการสั่นที่จะเกิดขึ้นให้อยู่ในระดับที่รับได้

1.2 ขอบเขตการทำงาน

1) เตียงขนย้ายผู้ป่วยสามารถรองรับแรงสั่นสะเทือนได้ในค่าที่กำหนดมาได้
2) เมื่อระบบมีการสั่นสะเทือนจากภายนอกมายังเตียง เตียงสามารถที่จะกลับมาอยู่ในสภาพที่คงเดิมหรือใกล้เคียงกับสภาพเดิมมากที่สุด
3) สามารถเขียนโปรแกรมควบคุมให้เตียงสามารถรักษาระดับเมื่อเกิดการสั่นจากภายนอกได้

1.3 วิธีดำเนินการ

1) กำหนดวัตถุประสงค์และขอบเขตการวิจัย

2) ศึกษาวรรณกรรมและงานวิจัยที่เกี่ยวข้อง

3) ออกแบบระบบกันสะเทือนที่ใช้กับเตียงขนย้ายผู้ป่วย

4) ศึกษาสมการการเคลื่อนที่และแบบจำลองทางคณิตศาสตร์ของระบบกันสะเทือน

5) ศึกษาการควบคุมชนิดพีไอดี (PID Controller)

7) ศึกษาเซนเซอร์ที่สามารถนำมาใช้ในระบบกันสะเทือนของเตียง

6) ศึกษาตัวไมโครคอนโทรลเลอร์ที่นำมาใช้ควบคุมในระบบ

7) สรุปและวิจารณ์

1.4 ผลที่คาดว่าจะได้รับ

1) เมื่อมีการขนย้ายผู้ป่วยไปยังบริเวณต่าง ๆ สามารถทำให้ผู้ป่วยนั้นลดแรงกระแทกและลดการบาดเจ็บขณะขนย้ายได้ดีขึ้น

2) ระบบควบคุมของระบบกันสะเทือนให้ผลตอบสนองที่เหมาะสมในการขนย้ายผู้ป่วย ณ บริเวณที่มีสภาพถนนต่างๆ กัน

บทที่ 2
ทฤษฎีและความรู้ที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงทฤษฎีเบื้องต้นของวงจร และอุปกรณ์ต่างๆ ที่นำมาใช้ในโครงงาน นอกจากนี้จะกล่าวถึงการออกแบบโครงงาน การติดต่อระหว่างไมโครคอนโทรลเลอร์กับอุปกรณ์ต่างๆ เช่น เซนเซอร์โพเทนทิออมิเตอร์แบบเชิงเส้น (Linear Potentiometer) โช๊คอัพ (Shock Absorber) รวมทั้งการเชื่อมต่ออุปกรณ์อื่นๆ ด้วย

2.1 ทฤษฎีพื้นฐาน

โครงงานนี้สามารถแบ่งเป็นส่วนประกอบได้หลายส่วนอันได้แก่

- ส่วนรองรับแรงสั่นสะเทือน
- ส่วนควบคุมกระบวนการทำงานและประมวลผล
- ส่วนตรวจจับและวัดระยะทางในการเคลื่อนที่ของเตียง

2.1.1 ส่วนรองรับแรงสั่นสะเทือน

ระบบกันสะเทือนที่ได้นำมาประยุกต์ใช้ในการรองรับแรงสั่นสะเทือนของเตียงเมื่อทำการขนย้ายผู้ป่วยนั้นได้จำลองมาจากระบบกันสะเทือนในรถยนต์ โดยสาเหตุที่ต้องมีระบบกันสะเทือนเนื่องจากไม่สามารถสร้างถนนหรือพื้นผิวให้เรียบได้ตามต้องการ หลุมบ่อเพียงเล็กน้อยก็มีผลทำให้เกิดการสั่นสะเทือนเมื่อมีการขนย้ายผู้ป่วยเกิดขึ้น โดยระบบกันสะเทือนที่จะกล่าวถึงต่อไปนี้จะประกอบไปด้วยสปริงและโช๊คอัพหรือตัวหน่วง โดยที่สปริงจะต้านการกระแทกอย่างแรงเมื่อเกิดการสั่นขึ้นลง ส่วนโช๊คอัพหรือตัวหน่วงจะเป็นตัวลดการสั่นขึ้นลงของสปริงให้หยุดลงโดยเร็ว

2.1.1.1 สปริง

ชนิดของสปริงที่ใช้ในระบบกันสะเทือนในรถยนต์ทั่วไปจะมีรูปร่างและลักษณะแตกต่างกันและการติดตั้งเข้ากับโครงรถแตกต่างกัน เช่น

- แหนบ (Leaf Spring)
- คอยล์สปริง (Coil Spring)
- ทอร์ชั่นบาร์ (Torsion Bar)

การทำงานของสปริงเหล่านี้มีลักษณะเหมือนกันคือ สปริงจะเป็นตัวสะสมพลังงานที่เกิดจากการกระแทกของรถยนต์ โดยการยืดตัว หดตัว โค้งงอหรือบิด แล้วปล่อยออกมาเมื่อมันกลับคืนรูปร่างเดิม โดยผู้จัดทำโครงงานได้เลือกใช้สปริงชนิดคอยล์สปริง

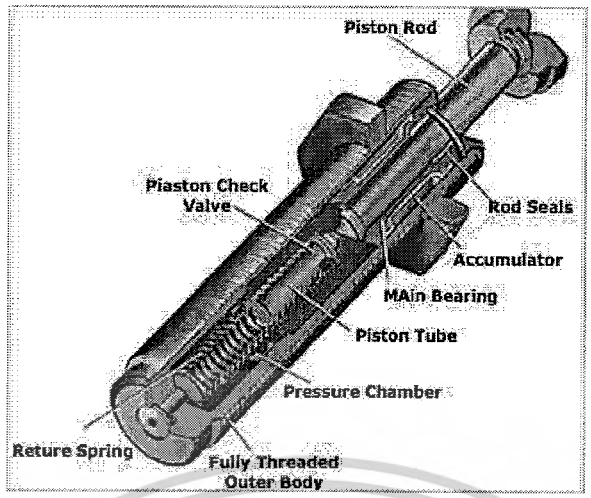
2.1.1.2 คอยล์สปริง (Coil Spring)

มีลักษณะเป็นเส้นลวดสปริงนำมาขดเป็นวงกลมซ้อนๆกันเป็นรูปทรงกระบอก รับน้ำหนักในแนวแกนทรงกระบอก คอยล์สปริงจะสะสมพลังงานในรูปการยืดและหดแล้วปล่อยพลังงานออกมาเมื่อกลับสู่สภาพเดิม คอยล์สปริงจะถูกยึดเข้ากับโครงรถที่ปลายด้านบนและด้านล่างยึดติดกับเพลาซึ่งรองไว้ด้วยยางที่ปลายทั้งสองด้าน

รูปที่ 2.1 คอยล์สปริง (Coil Spring)

2.1.1.3 โช๊คอัพ (Shock Absorber)

ในสมัยก่อนโช๊คอัพที่สร้างขึ้นมาจะอาศัยแรงเสียดทานของจอยท์ (Joint) แต่ปัจจุบันนี้ใช้โช๊คอัพไฮดรอลิกแทน ที่นิยมใช้คือโช๊คอัพกระบอก ลักษณะเป็นทรงกระบอกมีลูกสูบต่อกับก้านสูบ ทางด้านปิดของทรงกระบอกจะติดเข้ากับเพลาหรือคานยึดล้อ ส่วนทางด้านก้านสูบจะยึดติดกับโครงรถ นอกจากนี้จะมีวาวล์ที่ลูกสูบควบคุมให้น้ำมันในกระบอกสูบของโช๊คอัพถ่ายเทไปมาได้ระหว่างด้านบนและด้านล่างของลูกสูบ ซึ่งจะเปิดให้น้ำมันไหลผ่านลูกสูบขึ้นไปด้านบนได้เพียงทางเดียว ที่เหนือลูกสูบจะเป็นช่องว่างที่มีขนาดเล็กกว่าด้านล่าง เมื่อลูกสูบถูกกดลงน้ำมันจากด้านล่างจะถูกอัดให้ไหลผ่านวาวล์ขึ้นไปด้านบนของลูกสูบ พอลูกสูบถูกดึงขึ้นช่องว่างด้านล่างของลูกสูบจะขยายใหญ่ขึ้น น้ำมันจากด้านบนลูกสูบจะไหลผ่านวาวล์อีกอันหนึ่งซึ่งจะเปิดให้น้ำมันไหลลงทางเดียว เนื่องจากการถ่ายเทของน้ำมันต้องไหลผ่านช่องเล็กๆ ที่ลูกสูบจึงไหลไปได้อย่างช้าๆ โช๊คอัพจึงสามารถต้านทานการกระเด้งของสปริงให้ช้าลงและหยุดได้เร็วขึ้น



รูปที่ 2.2 โช๊คอัพ (Shock Absorber)

2.1.2 ส่วนควบคุมกระบวนการทำงานและประมวลผล

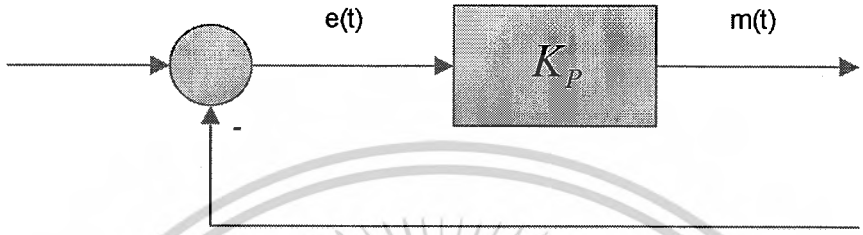
ในโครงงานนี้ได้ใช้ไมโครคอนโทรลเลอร์ dsPIC30F2010 เป็นอุปกรณ์ที่ใช้ควบคุมการทำงานของอุปกรณ์ภายในระบบ เช่น มอเตอร์ของโช๊ค เซนเซอร์โพเทนทิออมิเตอร์แบบเชิงเส้นในการวัดระยะทาง โดยที่ส่วนควบคุมกระบวนการทำงานนั้นผู้จัดทำได้เลือกใช้ตัวควบคุมชนิดพีไอดี (PID Controller)

2.1.2.1 ส่วนควบคุมกระบวนการทำงาน

ในบทนี้จะกล่าวถึงทฤษฎีพื้นฐานในการออกแบบตัวควบคุมเตียงเคลื่อนย้ายผู้ป่วยป้องกันการกระเทือน โดยจะแบ่งการอธิบายออกเป็นสองส่วน ส่วนแรกจะอธิบายถึงทฤษฎีพื้นฐานของตัวควบคุมแต่ละชนิดรวมทั้งตัวควบคุมชนิดพีไอดี ซึ่งข้อดีของตัวควบคุมชนิดพีไอดี คือ ช่วยลดผลกระทบจากสิ่งรบกวนที่มีต่อระบบได้ สามารถกำจัดค่าความคลาดเคลื่อนเชิงสถิตย์ (Steady-State Errors) ออกไปได้หมด และช่วยปรับปรุงการตอบสนองเชิงพลวัตของระบบให้ดีขึ้น และในส่วนที่สองจะอธิบายถึงทฤษฎีพื้นฐานของการออกแบบการป้อนกลับด้วยวิธีซีเกลอร์-นิโคลส์ (Ziegler-Nichols Method)

2.1.2.2 การควบคุมแบบสัดส่วน (Proportional Control Action)

เป็นปฏิกิริยาควบคุมซึ่งค่อนข้างจะสม่ำเสมอและเป็นเชิงเส้น ระหว่างการเปลี่ยนแปลงของอินพุทและเอาท์พุท สามารถแสดงความสัมพันธ์ในรูปแบบของแผนผังรูปภาพ ดังรูปที่ 2.3



รูปที่ 2.3 การควบคุมแบบสัดส่วน

ความสัมพันธ์ของสัญญาณควบคุม (เอาท์พุทของตัวควบคุม) m(t) กับสัญญาณค่าความคลาดเคลื่อน c(t) คือ

$$m(t) = K_p e(t) \tag{2.1}$$

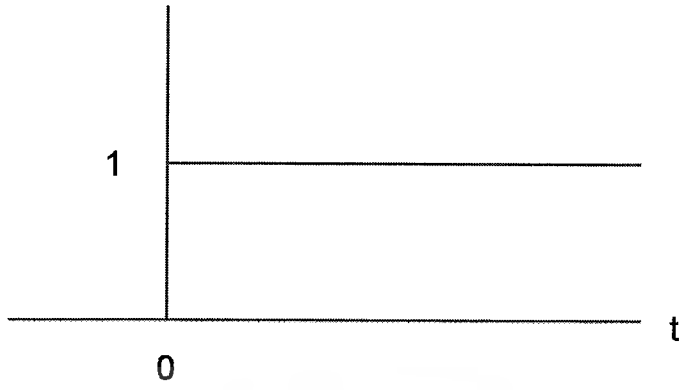
หรือ

$$\frac{M(s)}{E(s)} = K_p \tag{2.2}$$

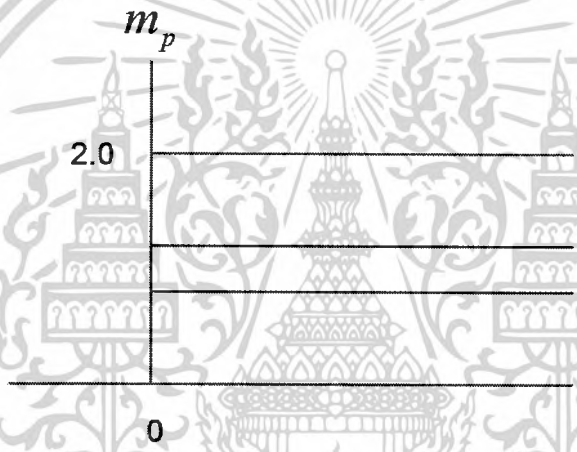
$$PB = \left(\frac{1}{K_p}\right) \times 100\% \tag{2.3}$$

โดยที่ K_p จะอยู่ในเทอมของ Proportional Sensitivity หรือ Gain

Proportional Band (PB) เป็นการเปลี่ยนแปลงของอินพุท เพื่อที่จะทำให้เกิดการเปลี่ยนแปลงของเอาท์พุทมากที่สุดในการควบคุมแบบสัดส่วนดังรูปที่ 2.4 และ 2.5

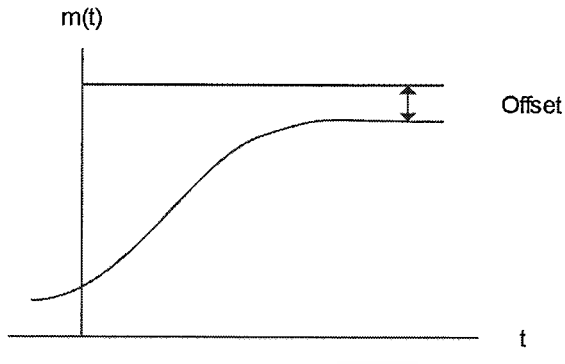


รูปที่ 2.4 ผลตอบสนองของการควบคุมแบบสัดส่วน

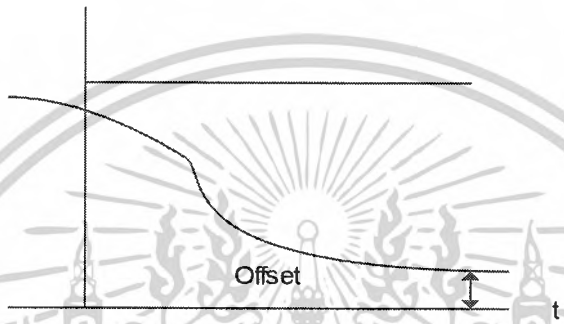


รูปที่ 2.5 ผลตอบสนองของการควบคุมแบบสัดส่วน

การเกิด Offset เป็นคุณลักษณะของระบบควบคุมแบบสัดส่วน ทั้งนี้เนื่องจากการทำงานของระบบควบคุมแบบสัดส่วนนั้น ไม่สามารถควบคุมระบบที่มีโหลด (Load) เปลี่ยนแปลงได้ดีเท่าที่ควร และในกรณีที่โหลด (Load) คงที่แต่เปลี่ยนค่าระดับ Set Point ที่ควบคุมไปก็เช่นเดียวกัน จะเกิดมี Offset ขึ้น โดยที่ Offset ก็คือ ค่าความแตกต่างอินพุทและเอาท์พุทที่สภาวะคงที่เมื่อเป้าหมายคงที่นั่นเอง ดังรูปที่ 2.6 (ก) และ (ข) ที่แสดงถึง Offset ที่เกิดขึ้นของความสัมพันธ์ระหว่างตัวแปรควบคุมกับอัตราการเปิดวาล์ว



(ก)



(ข)

รูปที่ 2.6 แสดงความสัมพันธ์ระหว่างตัวแปรควบคุมกับอัตราการเปิดวาล์ว

(ก) แสดงค่าความแตกต่างระหว่างอินพุทและเอาท์พุทที่มีค่าน้อยเกินไป

(ข) แสดงค่าความแตกต่างระหว่างอินพุทและเอาท์พุทที่มีค่ามากเกินไป

เราสามารถลดค่าการเกิด Offset ได้โดย

1. เพิ่มอัตราการขยายแบบสัดส่วน
2. เพิ่มค่าสัญญาณจัดการที่สภาวะเริ่มต้น (m_0) คือ $m_T = (K_p * e) + m_0$
3. เปลี่ยนค่าเป้าหมาย

2.1.2.3 การควบคุมแบบอินทิกรัล (Integral Control Action)

เป็นการควบคุมซึ่งค่าเอาท์พุทเป็นสัดส่วนโดยตรงกับค่าอินทิกรัลเชิงเวลาของอินพุทโดย จะมีความสัมพันธ์ระหว่างเอาท์พุทของตัวควบคุม m(t) และค่าความคลาดเคลื่อน e(t) ดังสมการ (2.4) และ (2.5)

$$\frac{dm(t)}{dt} = K_i e(t) \qquad (2.4)$$

หรือ

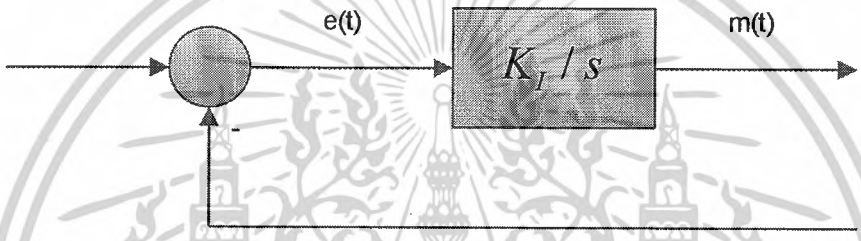
$$m(t) = K_i \int_0^t e(t)dt \qquad (2.5)$$

โดยที่ K_i เป็นค่าคงที่ที่สามารถปรับได้

ฟังก์ชันถ่ายโอน (Transfer Function) ของตัวควบคุมแบบอินทิกรัลคือ

$$\frac{M(s)}{E(s)} = \frac{K_i}{s} \qquad (2.6)$$

โดยสามารถแสดงในรูปของ แผนผังบล็อก ได้ดังรูปที่ 2.7



รูปที่ 2.7 แผนผังบล็อกของการควบคุมแบบอินทิกรัล

ในการควบคุมแบบอินทิกรัลนั้นค่าเอาท์พุทของตัวควบคุม m(t) จะเปลี่ยนแปลงตามค่าความผิดพลาด e(t) ดังนั้น ถ้าความผิดพลาดซึ่งได้เกิดขึ้น ทำให้ระบบได้ค่าที่ผิดไป จากค่าที่ต้องการแล้ว อุปกรณ์ควบคุมจะจัดการกับค่าความผิดพลาดโดยเร็ว (โดยลดให้ค่า Error นี้หมดไป) เมื่อตัวแปรควบคุมอยู่ที่ค่าเป้าหมายแล้วอุปกรณ์ควบคุมส่วนสุดท้าย (Final Element Control) จะยังไม่ทำงาน ซึ่งแสดงให้เห็นว่า ระบบอยู่ในสภาวะคงที่แล้วนั่นเอง ดังนั้นในการควบคุมแบบอินทิกรัลจะไม่ทำให้เกิดค่า Offset ขึ้นมา

2.1.2.4 การควบคุมแบบเดริเวอทีฟ (Derivative Control Active)

เป็นการควบคุมที่ค่าเอาท์พุทเป็นสัดส่วนกับอัตราการเปลี่ยนแปลงของอินพุท โดยมีความสัมพันธ์ดังสมการที่ 2.7 และ 2.8

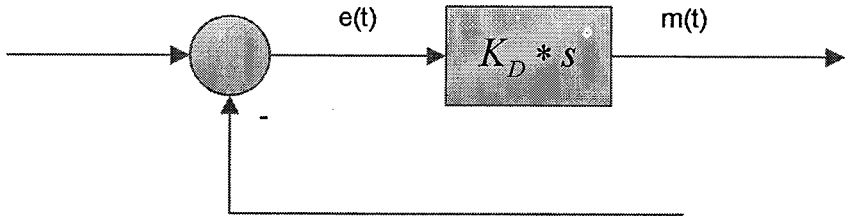
$$m(t) = K_D \times \frac{de(t)}{dt} \qquad (2.7)$$

หรือ

$$\frac{M(s)}{E(s)} = K_D s \qquad (2.8)$$

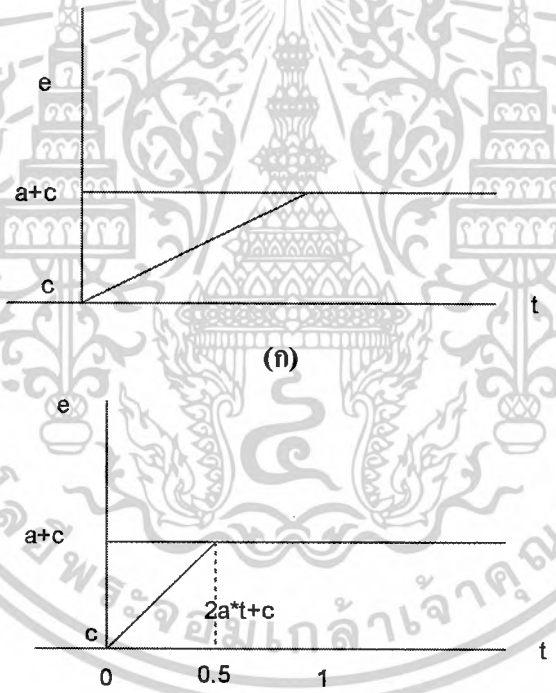
โดยที่ K_D เป็นค่าคงที่ที่สามารถปรับค่าได้

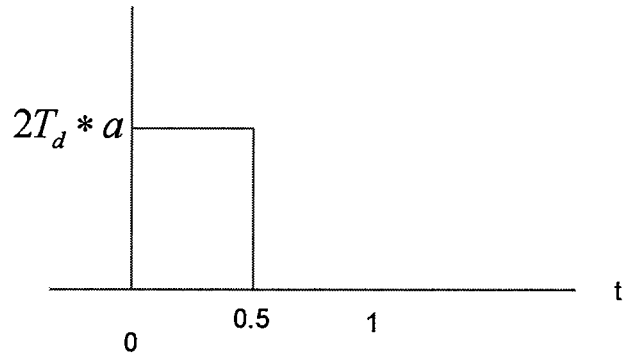
แผนผังบล็อก ของตัวควบคุมแบบเดริเวอทีฟแสดงได้ดังรูปที่ 2.8



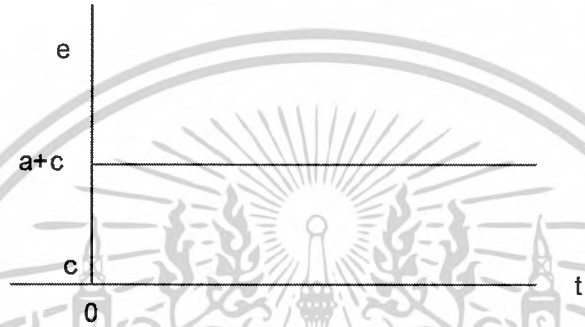
รูปที่ 2.8 แผนผังบล็อกของตัวควบคุมแบบเดริเวอทีฟ

เมื่อเพิ่มการควบคุมแบบ Derivative ไปในเครื่องควบคุมจะเป็นการบวกมุมนำในเครื่องควบคุม เพื่อชดเชยมุมตาม ในฟังก์ชันถ่ายโอน ซึ่งขบวนการส่วนใหญ่ จะมีค่ามุมเป็นแบบมุมตาม ดังรูปที่ 2.9





(ง)



(จ)

(ฉ)

รูปที่ 2.9 ผลตอบสนองสัญญาณแบบขั้นบันไดของการควบคุมแบบเดริเวอทีฟ

(ก) ผลตอบสนองสัญญาณที่ยังไม่มีการควบคุมแบบเดริเวอทีฟ

(ข) ผลตอบสนองสัญญาณที่มีการควบคุมแบบเดริเวอทีฟ

(ค) ผลตอบสนองสัญญาณที่ยังไม่มีการควบคุมแบบเดริเวอทีฟ

(ง) ผลตอบสนองสัญญาณที่มีการควบคุมแบบเดริเวอทีฟ

(จ) ผลตอบสนองสัญญาณที่ยังไม่มีการควบคุมแบบเดริเวอทีฟ

(ฉ) ผลตอบสนองสัญญาณที่มีการควบคุมแบบเดริเวอทีฟ

2.1.2.5 การควบคุมแบบพีไอ (Proportional and Integral Control Active)

เป็นการควบคุมที่ค่าเอาท์พุท เป็นสัดส่วนเชิงเส้นกับผลรวมของค่าอินพุท และค่าอินทิกรัลเชิงเวลาของอินพุท โดยสามารถแสดงความสัมพันธ์ได้ดังสมการที่ 2.9 และ 2.10

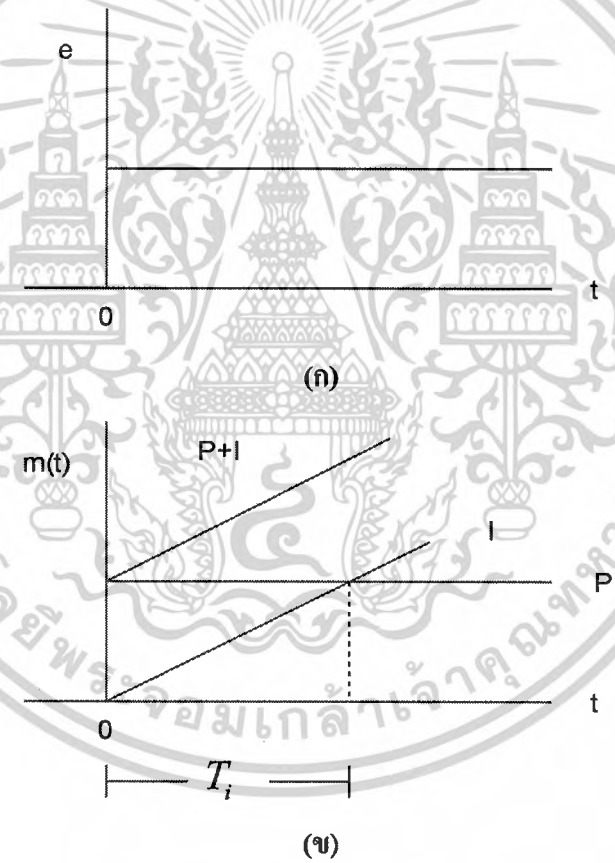
$$m(t) = \left[K_p e(t)\right] + \left(\frac{K_p}{T_i}\right)\int_0^t e(t)dt \qquad (2.9)$$

หรือ

$$\frac{M(s)}{E(s)} = K_p\left(1 + \frac{1}{T_i s}\right) \qquad (2.10)$$

โดยที่ K_p เป็นค่าของ Proportional Sensitivity หรือ Gain

T_i เป็นค่าของ Integral Time



รูปที่ 2.10 การควบคุมแบบพีไอ

(ก) แสดงความสัมพันธ์ระหว่างความต่างศักย์กับเวลาก่อนการปรับปรุงระบบ

(ข) แสดงความสัมพันธ์ระหว่างสัญญาณควบคุมกับเวลา ภายหลังการชดเชยด้วยตัวควบคุมแบบ PI Controller

ข้อดีของการควบคุมแบบพีไอคือตัวควบคุมอินทิกรัลจะกำจัด Offset ของตัวควบคุมแบบสัดส่วนให้หมดไป

2.1.2.6 การควบคุมแบบพีดี (Proportional and Derivative Control Action)

เป็นการควบคุมซึ่งค่าเอาท์พุทเป็นสัดส่วนโดยตรงกับผลรวมของค่าอินพุทกับผลคูณคาบเวลากับ อัตราการเปลี่ยนแปลงของอินพุท โดยสามารถแสดงด้วยสมการดังต่อไปนี้

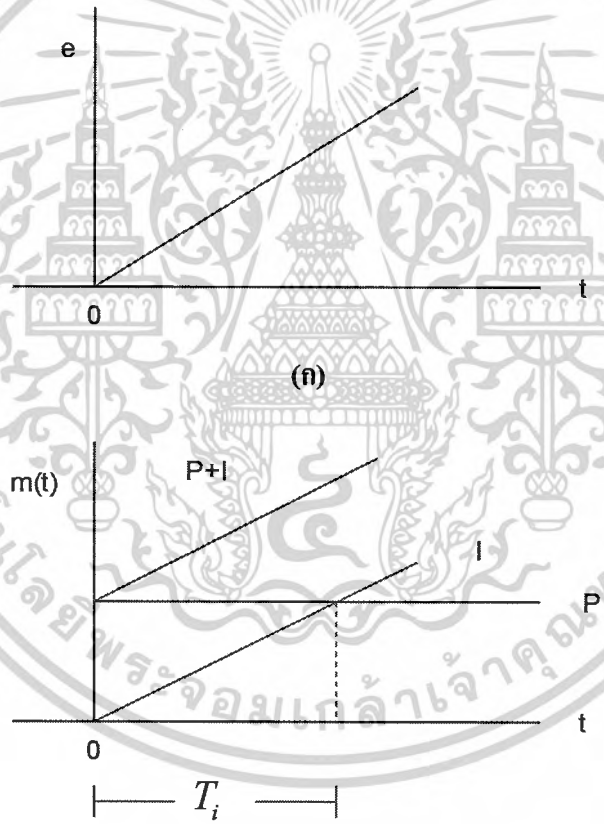
$$m(t) = \left[K_p e(t)\right] + \left[K_p T_D \frac{de(t)}{dt}\right] \qquad (2.11)$$

หรือ

$$\frac{M(s)}{E(s)} = K_p(1 + T_D S) \qquad (2.12)$$

โดยที่ K_p เป็นค่าของ Proportional Sensitivity หรือ Gain

T_D เป็นค่าของ Derivative Time



(ข)

รูปที่ 2.11 การควบคุมแบบพีดี

(ก) ความสัมพันธ์ระหว่างความต่างศักย์กับเวลาก่อนการปรับปรุงระบบ

(ข) แสดงความสัมพันธ์ระหว่างสัญญาณควบคุมกับเวลา ภายหลังการชดเชยด้วยตัวควบคุมแบบ PD Contoller

ข้อดีของการควบคุมแบบพีดีคือ เมื่อมีสัญญาณเข้าเป็นเชิงเส้น (Ramp) จะมีผลตอบสนองทางเวลาได้เปรียบกว่าการควบคุมแบบสัดส่วนเพียงอย่างเดียว

2.1.2.7 ผลตอบสนองของระบบต่อสัญญาณอินพุทชนิดต่างๆ

ในที่นี้จะพิจารณาผลตอบสนองของระบบอันดับหนึ่งและอันดับสอง ดังนี้

2.1.2.7.1 ผลตอบสนองต่อระบบอันดับหนึ่ง (Response of First-order System)

พิจารณาระบบอันดับหนึ่งต่อไปนี้

$$a_1 \frac{dy(t)}{dt} + a_0 y(t) = hx(t) = c \qquad (2.13)$$

โดยที่

$y(t)$ = เอาท์พุทหรือตัวแปรตาม

$x(t)$ = อินพุท

t = เวลา

a_0, a_1, b, c = ค่าคงที่

ถ้าพิจารณา ณ สถานะเริ่มต้น (Initial Steady State) ของระบบจะได้ความสัมพันธ์ดังสมการ 2.14

$$a_0 y(0) = bx(0) + c \qquad (2.14)$$

เราสามารถหาความสัมพันธ์ระหว่างค่าเริ่มต้นของ x และ y ได้ดังนี้

$$a_1 \frac{dy(t)}{dt} - a_0 y(0) + a_0 y(t) = bx(t) - hx(0) + c - c \qquad (2.15)$$

$$a_1 \frac{dy(t)}{dt} - a_0 \left[y(t) - y(0) \right] = b[x(t) - x(0)] \qquad (2.16)$$

ให้ $Y(t) = y(t) - y(0)$ (2.17)

$X(t) = x(t) - x(0)$ (2.18)

จะได้ว่า

$$a_1 \frac{dy(t)}{dt} - a_0 Y(t) = bX(t) \qquad (2.19)$$

แต่ในความเป็นจริงแล้วค่า $\frac{dy(t)}{dt}$ ก็คือ ค่าความแตกต่างระหว่างค่า ณ เวลาใดๆ กับค่า ณ จุดเริ่มต้น จึงได้ว่า

$$\frac{dy(t)}{dt} = \frac{dY(t)}{dt} \qquad (2.20)$$

ดังนั้นสมการ (2.20) จะกลายเป็น

$$a_1 \frac{dY(t)}{dt} + a_0 Y(t) = bX(t) \qquad (2.21)$$

ให้ τ = $\frac{a_1}{a_0}$ = ค่าคงที่ของเวลา

K = $\frac{b}{a_0}$ = อัตราการขยายสัญญาณ ณ สถานะคงตัว (Steady State Gain)

จะได้ว่า

$$\tau \frac{dY(t)}{dt} + Y(t) = KX(t) \qquad (2.22)$$

$$Y(s) = \left(\frac{K}{\tau s + 1}\right) X(s) \qquad (2.23)$$

- ผลตอบสนองต่อสัญญาณสเต็ป (Step Response)

ให้ $\triangle x$ = ขนาดของสัญญาณสเต็ป (Magnitude)

$u(t)$ = ฟังก์ชันสัญญาณหนึ่งหน่วย (Unit Step Function)

จะได้ว่า $x(t) = \triangle x \cdot u(t)$; u(t) =1

$$x(s) = \frac{\triangle x}{s} \qquad (2.24)$$

แทนสมการ (2.24) ลงในสมการ (2.23) จะได้ว่า

$$Y(s) = \left(\frac{K}{\tau s + 1}\right)\left(\frac{\triangle x}{s}\right) \qquad (2.25)$$

อินเวอร์สลาปลาซจะได้

$$Y(t) = K \triangle x[u(t) - e^{-t\tau}] \qquad (2.26)$$

- ผลตอบสนองเมื่อมีการหน่วงเวลา (Response with Time Delay)

ในกระบวนการบางอย่างอาจมีการหน่วงเวลาเอาท์พุท (Time Delay or Transportation Lag or Dead Time) โดนมีรูปแบบสมการดังนี้

$$Y(s) = \left(\frac{Ke^{-t_0 s}}{\tau s + 1}\right) X(s) \qquad (2.27)$$

เรียกสมการ (2.27) ว่า “ระบบอันดับหนึ่งที่มีการหน่วงเวลา (First Order Plus Dead Time: FOPDT)” ถ้าสัญญาณอินพุทของระบบมีความแตกต่างกัน สมการของการหน่วงเวลาจะแตกต่างกันไปดังนี้

สัญญาณสเต็ป $$Y(t)=\left[K\Delta x\times u(t-t_0)\right]\left[1-e^{-(t-t_0)/\tau}\right] \quad (2.28)$$

สัญญาณเชิงเส้น $$Y(t)=u(t-t_0)\left[\frac{K_r\tau e^{-(t-t_0)/\tau}}{\tau}\right]+\left[K_r(t-t_0-\tau\right] \quad (2.29)$$

สัญญาณรูปคลื่น

$$Y(t)=u(t-t_0)\left\{\left(\frac{KA\omega t}{1+\tau^2\omega^2}\right)e^{-(t-t_0)/\tau}+\left(\frac{KA}{1+\tau^2\omega^2}\right)\sin\left(\omega\left(t-t_0\right)+\theta\right)\right\} \quad (2.30)$$

2.1.2.7.2 ผลตอบสนองต่อระบบอันดับสอง (Response of Second-Order System)

ผลตอบสนองของระบบอันดับสองนี้จะมีความแตกต่างกันขึ้นอยู่กับค่ารากของสัดส่วนในฟังก์ชันถ่ายโอนว่ามีค่าเท่าไหร่ ถ้า

- ถ้าค่ารากเป็นจำนวนจริง ผลตอบสนองจะเรียกว่า “ระบบที่มีกระบวนการความหน่วง มาก (Over Damped)”
- ค่ารากเป็นจำนวนเชิงซ้อน ผลตอบสนองจะเรียกว่า “ระบบที่มีความหน่วงน้อย (Under Damped)” ซึ่งมักจะเกิดในกระบวนการที่มีอันดับสูงกว่าอันดับสอง

ระบบอันดับสองสามารถแสดงความสัมพันธ์ระหว่างตัวแปรเข้าและตัวแปรออกในรูปแบบสมการทั่วไปได้ ดังนี้

$$a_2\left(\frac{d^2y(t)}{dt^2}\right)+a_1\frac{dy}{dt}+a_0y(t)=bx(t)+c \quad (2.31)$$

โดยที่ y(t) = ตัวแปรออก

x(t) = ตัวแปรเข้า

a_0,a_1,a_2,b,c = ค่าคงที่

สมมุติให้สมการ ณ สถานะเริ่มต้นมีค่าเป็น

$$a_0y(0)=bx(0)+c \quad (2.32)$$

นำสมการ (2.31) ลบด้วยสมการ (2.32) จะได้ว่า

$$a_2\left(\frac{d^2Y(t)}{dt^2}\right)+a_1\frac{dY(t)}{dt}+a_0Y(t)=bX(t) \qquad (2.33)$$

โดยที่ $Y(t) = y(t) - y(0)$ (2.34)

$X(t) = x(t) - x(0)$ (2.35)

แปลงสมการ(2.11)ใหม่จะได้

$$\tau^2\left(\frac{d^2Y(t)}{dt^2}\right)+2\zeta\tau\frac{dY(t)}{dt}+Y(t)=KX(t) \qquad (2.36)$$

$$Y(s)=\left(\frac{K}{\tau^2s^2+2\zeta\tau s+1}\right)X(s) \qquad (2.37)$$

ให้ คุณสมบัติของเวลา (Characteristic Time : τ) = $\left(\frac{a_2}{a_0}\right)^{\frac{1}{2}}$

อัตราความหน่วง (Damping Ration : ζ) = $\frac{a_1}{2\sqrt{a_0a_1}}$

อัตราของกระบวนการ (Steady State Gain : K) = $\frac{b}{a_0}$

หาค่ารากของ $\tau^2s^2+2\zeta\tau s+1=0$ จะได้ความสัมพันธ์ดังนี้

$$r_{1,2}=\frac{-\zeta\pm\sqrt{\zeta^2-1}}{\tau} \qquad (2.38)$$

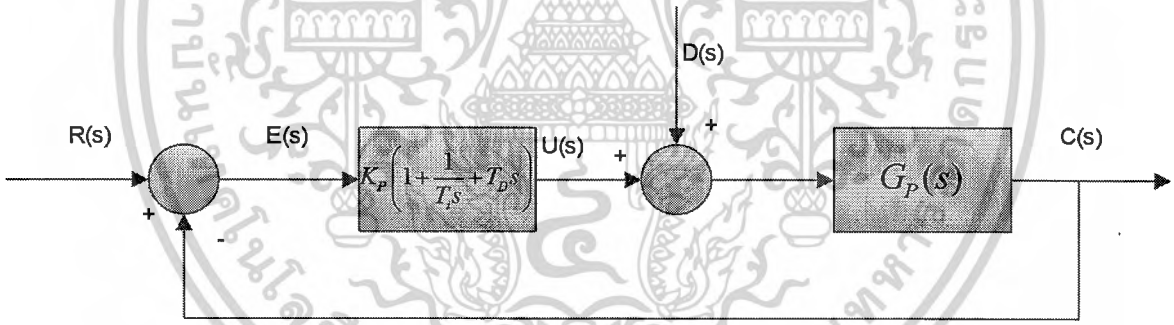
อัตราความหน่วงที่มีค่าแตกต่างกันจะทำให้เกิดผลตอบสนองที่แตกต่างกันซึ่งสามารถแสดงได้ดังตารางที่ 2.1

ζ	ผลตอบสนอง (Response)
$\zeta > 1$	กระบวนการที่มีความหน่วงมาก (Over Damped)
$\zeta = 1$	กระบวนการที่มีความหน่วงวิกฤติ (Critically Damped)
$0 < \zeta < 1$	กระบวนการที่มีความหน่วงน้อย (Under Damped)
$\zeta = 0$	กระบวนการที่ไม่มีความหน่วง (Undamped)
$-1 < \zeta < 0$	กระบวนการที่ไม่เสถียร (Unstable)
$\zeta \leq -1$	กระบวนการที่หนีออก (Run – Away)

ตารางที่ 2.1 ผลการตอบสนองเมื่อค่าของอัตราความหน่วงที่แตกต่างกัน

2.1.2.8 ทฤษฎีพื้นฐานการควบคุมพีไอดี

การควบคุมแบบพีไอดีนั้น ได้นำมาใช้เพื่อปรับปรุงระบบของเตียงที่ต้องการควบคุมให้ระบบมีผลตอบสนองชั่วขณะให้ดีขึ้น พร้อมๆ กับการทำให้ระบบมีเสถียรภาพมากขึ้น รูปแบบโดยทั่วไปของการควบคุมแบบพีไอดีสามารถเขียนเป็นบล็อกไดอะแกรมได้ดังต่อไปนี้



รูปที่ 2.12 โครงสร้างของระบบควบคุมแบบพีไอดี

จากโครงสร้างของระบบควบคุมพีไอดีแสดงได้ดังรูปที่ 2.12 นั้น จะมีสัญญาณควบคุมคือ

K_p คือ อัตราขยายของตัวควบคุมแบบพี

K_i คือ อัตราขยายของตัวควบคุมไอซึ่งเท่ากับ $\frac{K_p}{T_i}$

K_d คือ อัตราขยายของตัวควบคุมแบบดีซึ่งเท่ากับ $K_p T_d$

$E(s)$ คือ ค่าความผิดพลาดระหว่างสัญญาณเข้ากับสัญญาณออก

ฟังก์ชันถ่ายโอนของคอนโทรลเลอร์ คือ

$$G_c(s) = K_p + \frac{K_i}{s} + k_d s \qquad (2.39)$$

หรือ

$$G_c(s) = K_p(1 + \frac{1}{\tau_i s} + \tau_d s) \qquad (2.40)$$

2.1.2.9 วิธีซีเกลอร์-นิโคลส์ (Ziegler-Nichols Method) เพื่อใช้ในการปรับค่าอัตราขยายของตัวควบคุมชนิดพีไอดี

การออกแบบตัวควบคุมหรือตัวชดเชยโดยวิธีพล็อตทางเดินรากนั้นจะต้องทราบถึงแบบจำลองคณิตศาสตร์ของระบบซึ่งในบางระบบอาจมีสมการหรือแบบจำลองที่ยุ่งยาก จึงอาจหลีกเลี่ยงปัญหาเหล่านั้นโดยใช้วิธีการประมาณค่าพารามิเตอร์ของตัวควบคุมจากผลตอบสนองซึ่งไม่จำเป็นต้องหาแบบจำลองทางคณิตศาสตร์ของระบบ วิธีในการประมาณผลตอบสนองวิธีหนึ่งคือการออกแบบตัวควบคุมชนิดพีไอดีด้วยวิธีการ วิธีซีเกลอร์-นิโคลส์

เมื่อผลตอบสนองของมีการแกว่งเกิดขึ้น วิธีการหาค่าพารามิเตอร์ของตัวควบคุมมีดังนี้

1. เพิ่มตัวควบคุมชนิดพีแล้วปรับค่า K จนได้กราฟที่มีการแกว่งเท่าๆกัน (K ตัดแกนจินตภาพ)
2. ค่า K ที่ปรับได้ คือ ค่า K_u นำค่าที่ได้ไปแทนหาค่าในตาราง (2.2)

ค่า / Type	K	T_I	T_D
P	$0.6k_u$	-	-
PI	$0.45k_u$	$\frac{1}{1.2}p_u$	-
PID	$0.6k_u$	$0.6k_u$	$\frac{1}{8}P_u$

ตารางที่ 2.2 สมการคำนวณค่าพารามิเตอร์ของแต่ละตัวควบคุมเมื่อผลตอบสนองมีการแกว่ง

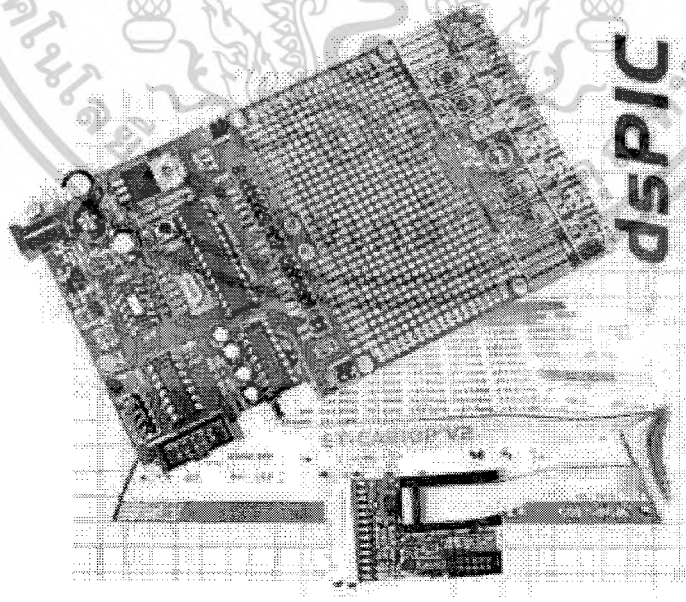
2.1.2.10 ส่วนควบคุมและประมวลผล

ในโครงงานนี้เราใช้ไมโครคอนโทรลเลอร์ dsPIC30F2010 เป็นตัวควบคุมการทำงานของเตียงและทำการประมวลผลในการที่จะวัดระยะทางที่เคลื่อนที่ไปของโช๊คอัพเมื่อเกิดการสั่นสะเทือน

2.1.2.11 หน้าที่ของส่วนควบคุมและประมวลผล

- รับและเก็บค่าของระยะทางที่ Linear Potentiometer นั้นได้เกิดการเปลี่ยนแปลงความต้านทานเมื่อได้ทำการลากเตียงไปยัง ณ จุดต่างๆ
- ส่งสัญญาณไปควบคุมการทำงานของวงจรตรวจจับวัดระยะการเคลื่อนที่และการสั่นสะเทือนของโช๊คอัพ
- ควบคุมการทำงานมอเตอร์ของโช๊คอัพและสามารถปรับค่าให้เหมาะสมเมื่อเกิดการสั่นสะเทือน ณ จุดต่างๆ

เนื่องจากส่วนควบคุมและประมวลผลจำเป็นต้องมีคุณสมบัติหลายอย่างเพื่อให้สอดคล้องกับการออกแบบ และการทำงาน ในโครงงานนี้จึงเลือกใช้ไมโครคอนโทรลเลอร์ dsPIC30F2010 โดยทางผู้จัดทำโครงงานได้เลือกใช้บอร์ดสำเร็จรูปของบริษัท ETT ที่สามารถลงโปรแกรมและลบในตัวได้ โดยมีลักษณะดังรูปที่ 2.13



รูปที่ 2.13 แสดงบอร์ดสำเร็จรูป dsPIC30F2010 ที่ใช้ในการทดลอง

2.1.2.12 คุณสมบัติเด่นของไมโครคอนโทรลเลอร์ dsPIC30F2010

- เป็นไมโครคอนโทรลเลอร์ที่ใช้ซีพียูแบบ RISC
- ความเร็วในการทำงานสูงถึง 30 ล้านคำสั่งต่อวินาที
- ชุดคำสั่งมีขนาด 24 บิต สามารถประมวลผลข้อมูลได้ 16 บิต
- หน่วยความจำโปรแกรมเป็นแบบแฟลช สามารถลบและเขียนใหม่ได้ไม่น้อยกว่า 100,000 ครั้ง สามารถป้องกันการอ่านได้ และสามารถโปรแกรมตัวเอง โดยกระบวนการทางซอฟแวร์
- มีอินเตอร์รัปต์เวกเตอร์จำนวนมาก จึงรองรับการตอบสนองสัญญาณอินเตอร์รัปต์ได้ดี
- มีวงจรตรวจจับแรงดันไฟเลี้ยงต่ำกว่ากำหนดแบบโปรแกรมได้
- Timer/Counter มีขนาด 16 บิต ไม่น้อยกว่า 3 ตัว ต่อการใช้งานร่วมกันเป็น Timer 32 บิตได้
- วงจรสามารถตรวจสอบการทำงานของวงจรกำเนิดสัญญาณนาฬิกาได้
- รองรับการโปรแกรมในวงจรแบบอนุกรม (ICSP : In-Circuit Serial Programming)
- สามารถเลือกโหมดในการใช้พลังงานได้
- Watchdog-Timer แบบโปรแกรมได้
- หน่วยความจำ EEPROM 1 กิโลไบท์
- ช่องควบคุม Motor PWM 6 ช่อง

2.1.2.13 ข้อดีและข้อเสียในการเลือกใช้ไมโครคอนโทรลเลอร์ dsPIC30F2010

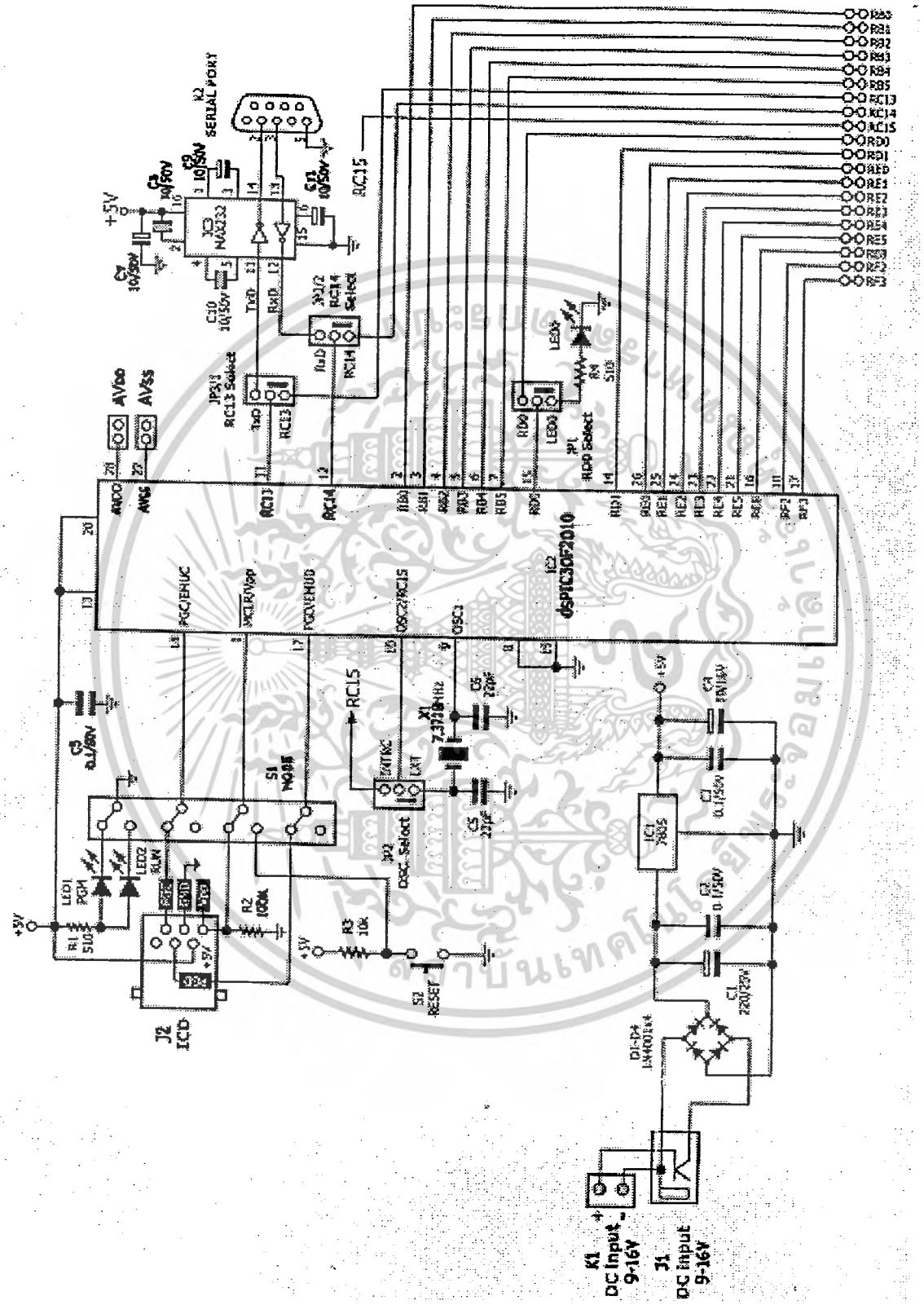
ในการเลือกใช้ไมโครคอนโทรลเลอร์ dsPIC30F2010 นั้นมีการพิจารณาถึงข้อดีและข้อเสียได้ดังนี้

ข้อดี

1. ความเร็วมากกว่า PIC แบบธรรมดา 3-4 เท่า
2. สามารถโปรแกรมข้อมูลได้ถึง 100,000 ครั้ง

ข้อเสีย

1. CPU มีความเร็วสูง เมื่อทำการโปรแกรมจะเกิดการเสียหายได้ง่าย



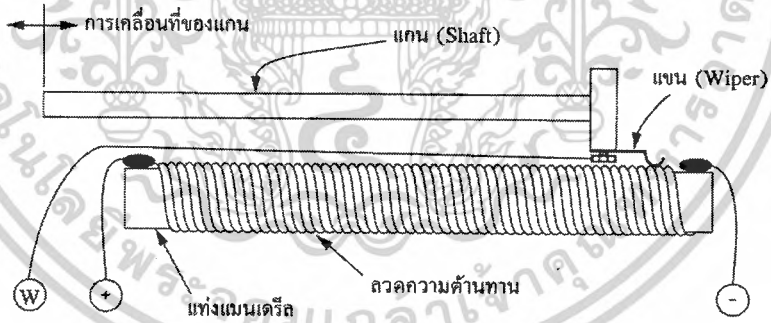
รูปที่ 2.14 วงจรบอร์ดสำเร็จรูปของ dsPIC30F2010

2.1.3 ส่วนตรวจจับและวัดระยะทางในการเคลื่อนที่ของเตียง

โครงงานนี้ได้เลือกใช้เซนเซอร์ในการตรวจจับและวัดระยะทางในการเคลื่อนที่เมื่อเกิดการสั่นสะเทือนของเตียงก็คือ เซนเซอร์วัดระยะทางโพเทนทิออมิเตอร์แบบเชิงเส้น (Linear Potentiometer)

2.1.3.1 โพเทนทิออมิเตอร์แบบเชิงเส้น (Linear Potentiometer)

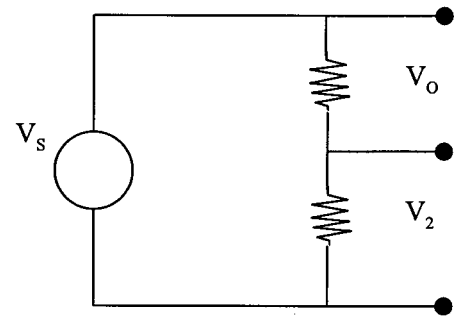
โพเทนทิออมิเตอร์ เป็นตัวต้านทานทางไฟฟ้าชนิดหนึ่งที่สามขั้วและสามารถปรับค่าได้ โครงสร้างทั่วไปของโพเทนทิออมิเตอร์แสดงได้ดังรูป ประกอบด้วยหน้าสัมผัส (Sliding Contact) ที่สามารถเลื่อนขึ้นหรือลงผ่านตามความยาวของลวดความต้านทานได้ โดยหน้าสัมผัสหรือบางครั้งนิยมเรียกว่า “ไวเปอร์ (Wiper)” นี้จะถูกต่อเชื่อมกับแท่งวัดซึ่งต่ออยู่กับวัตถุที่ต้องการตรวจสอบระยะขจัด โดยทั่วไปแล้วรูปแบบของหน้าสัมผัสของโพเทนทิออมิเตอร์มักมีการออกแบบแตกต่างกันออกไปซึ่งขึ้นอยู่กับปัจจัยและเงื่อนไขในการประยุกต์ใช้งาน แต่ไม่ว่าจะมีรูปแบบไหน วัสดุที่นำมาใช้ทำหน้าสัมผัสมักนิยมทำมาจากโลหะทองแดงผสมเป็นเสียส่วนมาก ที่เป็นเช่นนี้ก็เพราะว่าโลหะทองแดงผสมมีความยืดหยุ่นตัวสูงจำนำไปขึ้นรูปทรงแบบต่างๆ ได้ง่าย รวมทั้งยังมีคุณสมบัติในการนำไฟฟ้าได้ดีอีกด้วย ส่วนลวดความต้านทานของโพเทนทิออมิเตอร์นั้นนิยมทำมาจากลวดหรือแพลตตินัม นำมาพันรอบแกนที่เป็นฉนวนทางไฟฟ้า



รูปที่ 2.15 แสดงส่วนประกอบของโพเทนทิออมิเตอร์

2.1.3.2 การต่อโพเทนทิออมิเตอร์

เนื่องจากการเปลี่ยนแปลงความต้านทานเทียบกับการวัดของอุปกรณ์ประเภทนี้ค่อนข้างสูง ดังนั้นจะใช้วงจรแบ่งแรงดัน(Voltage Divider) ในการตรวจจับการเปลี่ยนแปลงความต้านทาน และสามารถนำมาปรับเทียบเป็นการขจัดได้ การใช้ Potentiometer ในวงจรจะมีลักษณะดังรูปที่ 2.16



รูปที่ 2.16 แสดงวงจรการต่อโพเทนทิออมิเตอร์

จากรูป เราพบว่า $$V_0 = IR_1 \quad (2.41)$$

และ $$V_2 = IR_2 \quad (2.42)$$

ดังนั้น $$-V_s + IR_1 + IR_2 = 0 \quad (2.43)$$

ซึ่งจะได้ $$I = \frac{R_s}{(R_1 + R_2)} \quad (2.44)$$

ดังนั้นเราจะได้ $$V_0 = \frac{R_1}{R_1 + R_2} V_s \quad (2.45)$$

แต่เนื่องจาก $R_1 + R_2$ คือความต้านทานของ Potentiometer, R_{total} ดังนั้น

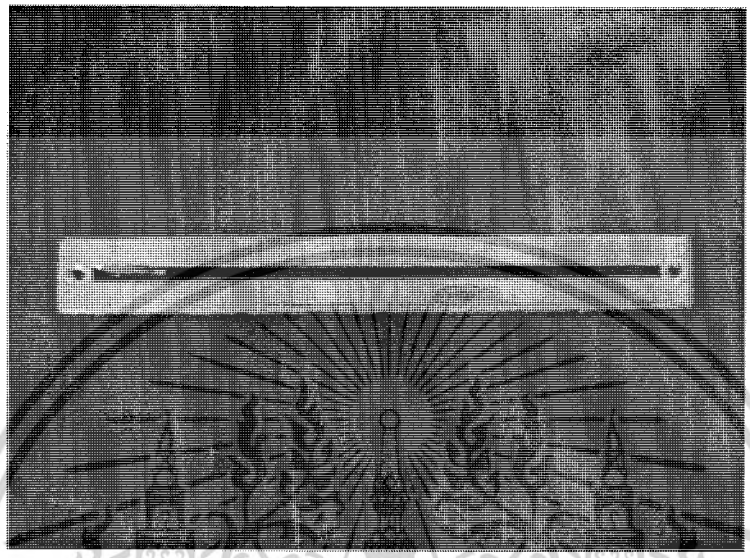
$$V_0 = \frac{R_1 V_s}{R_{total}} \quad (2.46)$$

หรือ $$R_1 = \frac{R_{total} V_0}{V_s} \quad (2.47)$$

ในการวัดเราจะวัดค่า V_0 ที่เป็น Output ส่วนค่า R_{total} และ Vs เป็นค่าที่เราทราบ ดังนั้น จากสมการนี้เราสามารถหาค่า R_1 ได้ เมื่อได้ค่า R_1 เราสามารถนำไปเปรียบเทียบกับระยะบน Potentiometer ได้ ซึ่งหากเราสมมุติว่า Potentiometer นี้ให้ค่าความต้านทานเชิงเส้นกับการขจัดเราจะสามารถหาการขจัดได้โดยการเปรียบสัดส่วน ถ้าเราให้ ΔR_0 เป็นการเปลี่ยนแปลงความต้านทานจากจุดที่มีการขจัดเป็นศูนย์ เราจะได้การขจัดดังสมการที่ 2.48

$$L_{at} = \frac{\Delta R_0 L_{total}}{R_{total}} \quad (2.48)$$

เมื่อ L_{total} คือการขจัดทั้งหมดของ Potentiometer นั้น ข้อควรระวังก็คือว่าที่ $L_0 = 0$ ไม่ได้หมายความว่า $R_0 = 0$ เพราะโดยปกติจะมีความต้านทานในอุปกรณ์นี้อยู่บ้าง แม้ว่าการขจัดจะเป็นศูนย์ก็ตาม อย่างไรก็ตาม ค่านี้สามารถเปรียบวัดได้ โดยมีลักษณะดังรูปที่ 2.17



รูปที่ 2.17 โพเทนทิออมิเตอร์แบบเชิงเส้น

บทที่ 3
การออกแบบและหลักการทำงาน

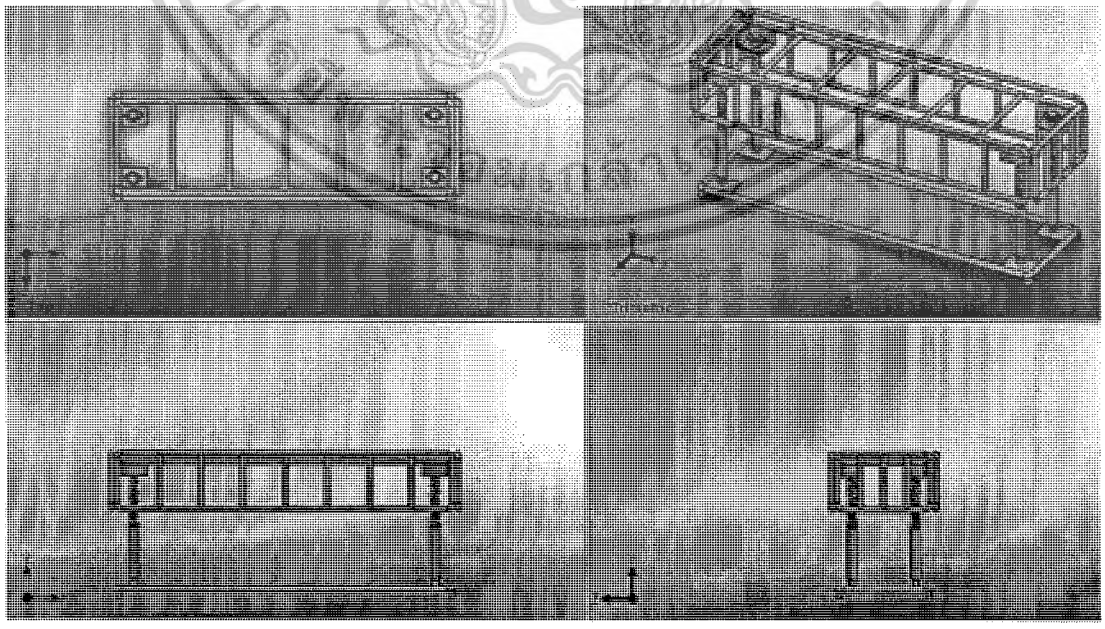
ในบทนี้จะกล่าวถึงการออกแบบ การทำงานของอุปกรณ์ และการเขียนโปรแกรมควบคุมในส่วนของไมโครคอนโทรลเลอร์ โดยจะแสดงเป็น โฟลว์ชาร์ต (Flowchart)

3.1 การออกแบบโครงงาน

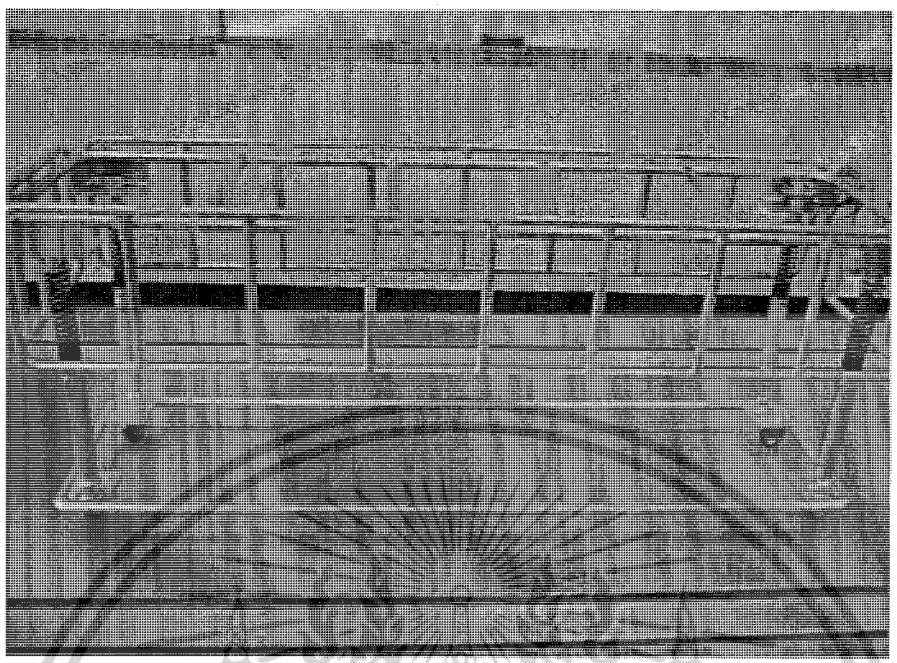
เพื่อการสะดวกในการใช้งาน การออกแบบจึงเป็นสิ่งสำคัญอย่างยิ่งที่ต้องคำนึงถึง ไม่ว่าจะเป็นขนาดของเตียง รูปแบบของเตียง การเลือกตำแหน่งในการติดเซนเซอร์ เป็นต้น

3.1.1 การออกแบบโครงสร้าง และการเลือกวัสดุที่ใช้ในการประกอบ

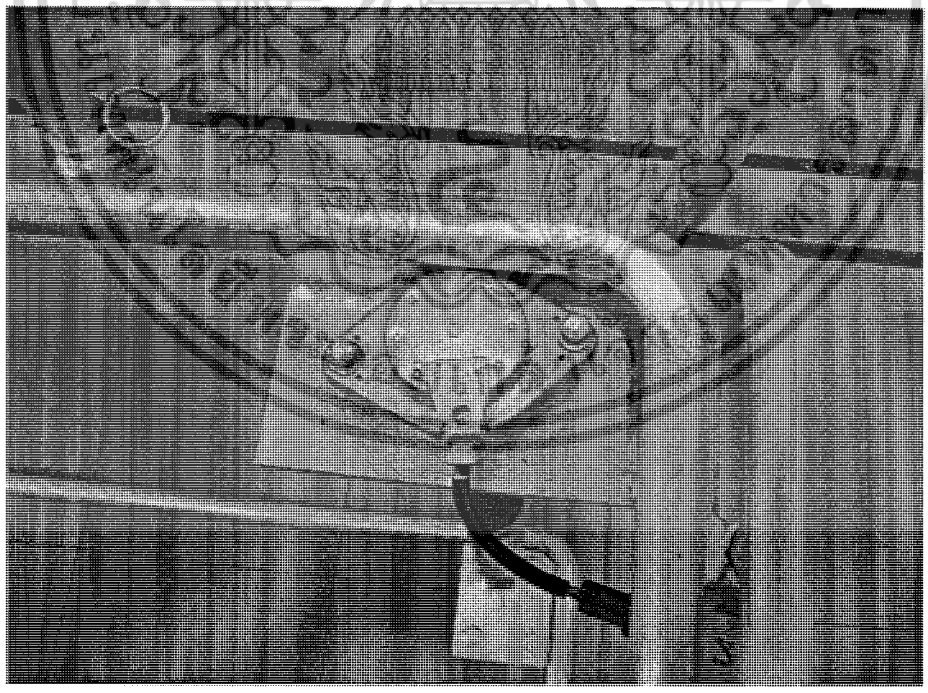
เนื่องจากเตียงขนย้ายผู้ป่วยนั้นจำเป็นที่จะต้องมีความสามารถในการรับน้ำหนัก ดังนั้นจึงจำเป็นที่จะต้องเลือกวัสดุที่มีความแข็งแรงทนทาน คือ เหล็ก โครงงานนี้ได้ทำการออกแบบโครงสร้างโดยใช้โปรแกรม Solid Works ดังรูปที่ 3.1 ซึ่งเตียงได้มีขนาดความกว้าง 50 เซนติเมตร ยาว 180 เซนติเมตร จากนั้นได้ทำการประกอบเตียงดังรูปที่ 3.2 โดยนำมาประกอบกับมอเตอร์ของโช๊คอัพดังรูปที่ 3.3 และรูปที่ 3.4 แสดงตำแหน่งของโช๊คอัพเมื่อได้ทำการประกอบเข้ากันกับเตียงและมอเตอร์ โดยที่สปริงของโช๊คได้ติดตั้งดังรูปที่ 3.5 แล้วนำหน้าสัมผัสของเซนเซอร์มายึดไว้กับโช๊คอัพดังรูปที่ 3.6



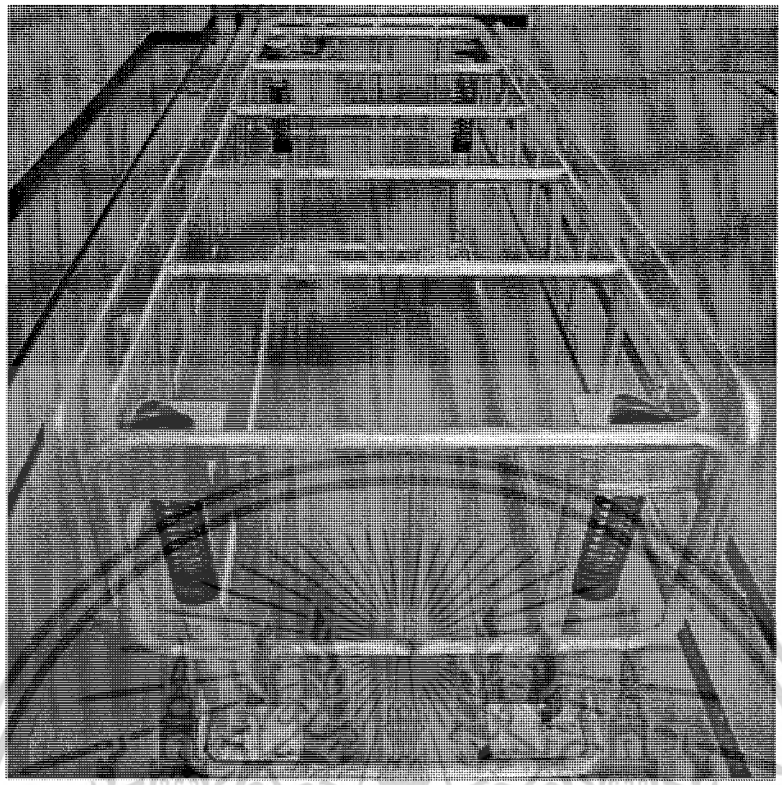
รูปที่ 3.1 การออกแบบโครงสร้างโดยใช้ โปรแกรม Solid Works



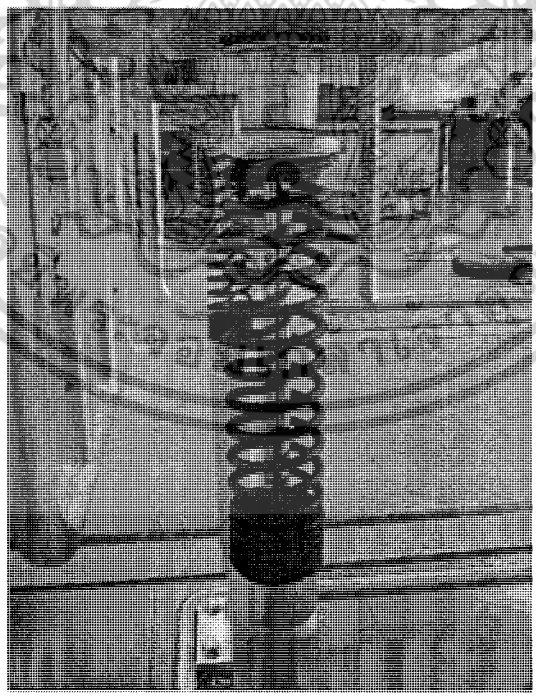
รูปที่ 3.2 เตียงขนย้ายผู้ป่วย



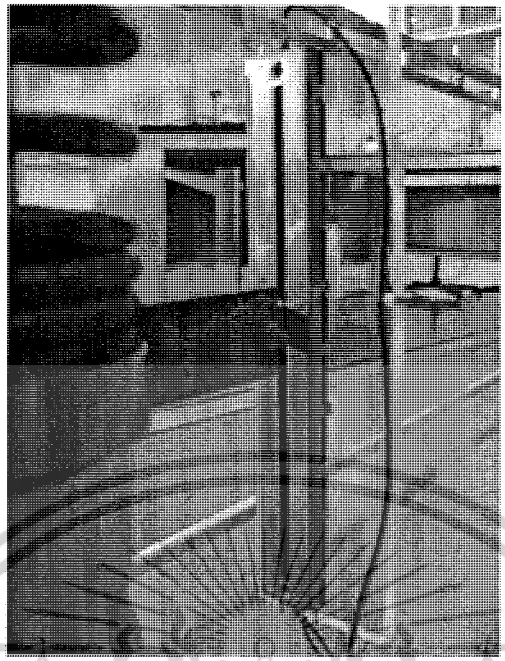
รูปที่ 3.3 แสดงมอเตอร์ของโช๊คอัพ



รูปที่ 3.4 แสดงตำแหน่งของโช๊คอัพ



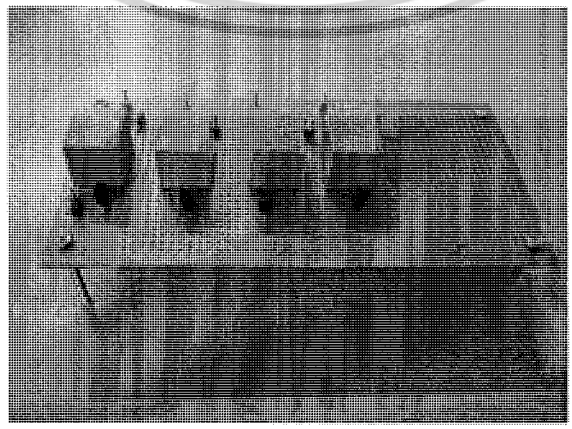
รูปที่ 3.5 สปริงของโช๊คอัพ



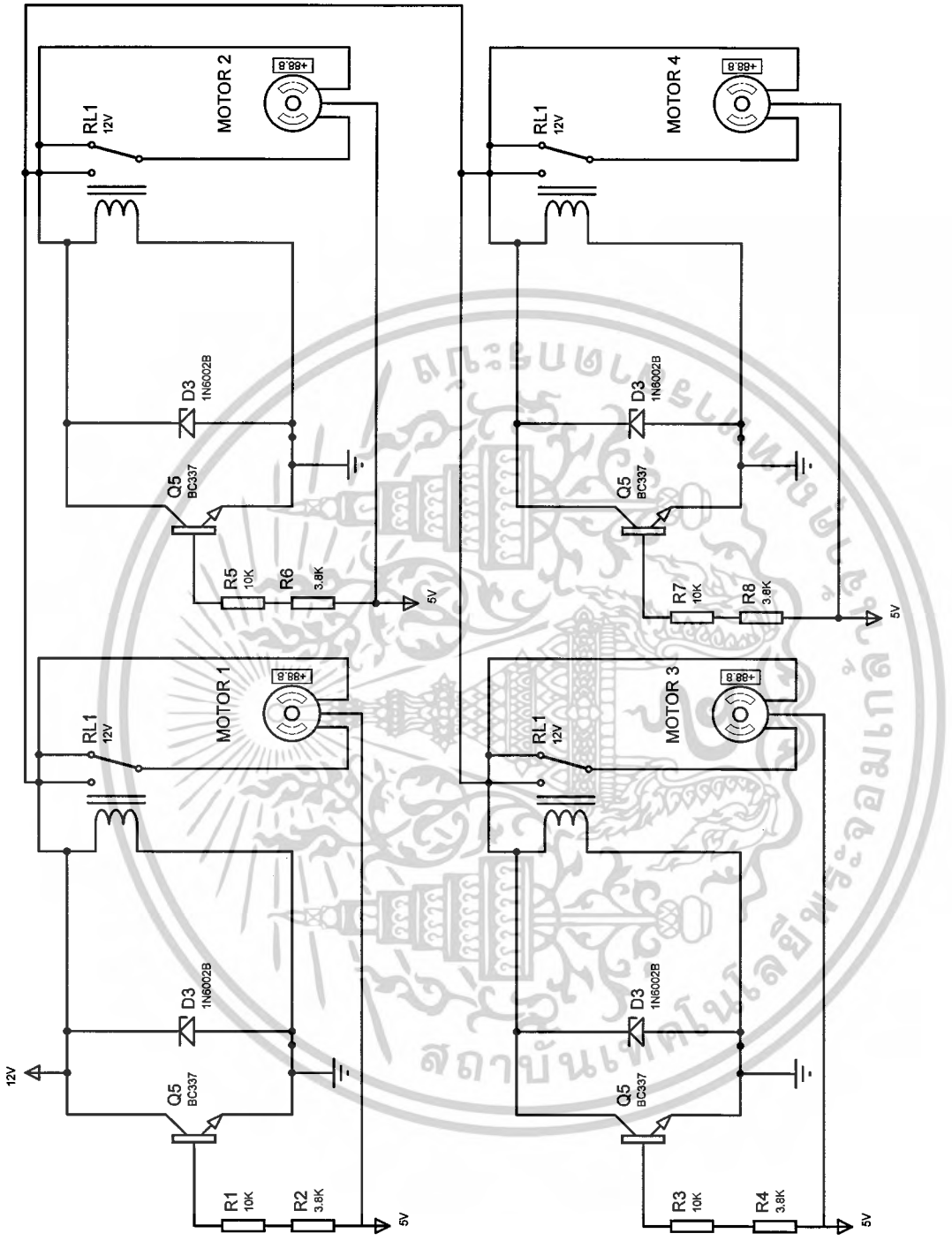
รูปที่ 3.6 การติดตั้งโพเทนทิออมิเตอร์แบบเชิงเส้นกับโช๊คอัพ

3.1.2 การเลือกใช้แรงดันไฟฟ้า

สำหรับโครงงานนี้ ใช้แรงดันไฟฟ้าทั้งหมด 2 ค่าด้วยกัน คือ 5 โวลต์ และ 12 โวลต์ โดยแรงดันไฟฟ้า 5 โวลต์ ใช้สำหรับไมโครคอนโทรลเลอร์และไอซีในวงจรไมโครคอนโทรลเลอร์ และแรงดันไฟฟ้า 12 โวลต์ ใช้สำหรับวงจรขับมอเตอร์ของโช๊คอัพ จึงทำเกิดความไม่สะดวกในการจ่ายไฟให้กับวงจร สามารถแก้ปัญหาด้วยการออกแบบวงจรจ่ายแรงดันไฟฟ้า 12 โวลต์ให้กับมอเตอร์ และได้เลือกใช้รีเลย์ (Relay) ขนาด 12 โวลต์ โดยใช้แบตเตอรี่แบบก้อนขนาด 12 โวลต์ เป็นตัวจ่ายกระแสให้กับมอเตอร์เพื่อแปลงแรงดันไฟฟ้าขนาด 12 โวลต์ เป็น 5 โวลต์ เพื่อนำไปใช้กับไมโครคอนโทรลเลอร์ ดังวงจรรูปที่ 3.7 และ 3.8



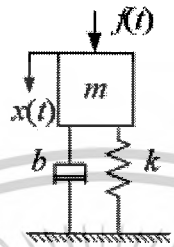
รูปที่ 3.7 วงจรขับมอเตอร์



รูปที่ 3.8 วงจรขับมอเตอร์

3.1.3 การจำลองระบบพลศาสตร์โดยใช้ State Space Approach

เนื้อหาในส่วนนี้เป็นการกล่าวถึงการจำลองระบบพลศาสตร์ที่ใช้ในระบบกันสะเทือนนี้ โดยมีแบบจำลองทางคณิตศาสตร์ ดังนี้



รูปที่ 3.9 แบบจำลองทางคณิตศาสตร์ของระบบกันสะเทือนของเตียง

มีแบบจำลองทางคณิตศาสตร์คือ

$$m\ddot{x} + b\ddot{y} + ky = f \qquad (3.1)$$

$$A = \begin{bmatrix} 0 & 1 \\ -k/m & -b/m \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1/m \end{bmatrix}, \text{C} = [\,1 \quad 0\,], \text{ และ } \text{D} = 0$$

โดย

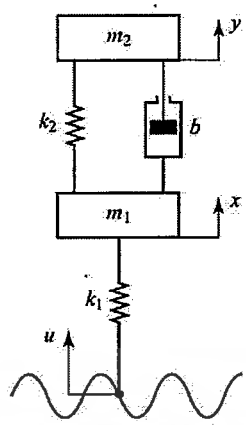
$f(t)$ = แรงที่มีผลกระทบกับระบบเตียง

$x(t)$ = ทิศทางการเคลื่อนที่ของมวล m

b = ค่าหน่วงของโช๊คอัพเตียง

k = ค่าคงที่สปริงของโช๊คอัพเตียง

m = ค่ามวลของคนรวมกับเตียงผู้ป่วย



รูปที่ 3.10 แบบจำลองทางคณิตศาสตร์ของระบบกันสะเทือนของรถ

มีแบบจำลองทางคณิตศาสตร์ คือ

$$m_2\ddot{y} + b\dot{y} + k_2 y = b\dot{x} + k_2 x \qquad (3.2)$$

โดย

m_1 = ค่ามวลของยางรถยนต์

m_2 = ค่ามวลของตัวรถ

k_1 = ค่าคงที่สปริงของยางรถยนต์

k_2 = ค่าคงที่สปริงของโช๊คอัพรถยนต์

b = ค่าหน่วงของโช๊คอัพรถยนต์

y = ทิศทางการเคลื่อนที่ของมวล m_2

x = ทิศทางการเคลื่อนที่ของมวล m_1

u = สัญญาณแทนความขรุขระของผิวถนน

3.1.4 การออกแบบระบบควบคุมชนิดพีไอดี เพื่อใช้ควบคุมมอเตอร์

เนื้อหาในส่วนนี้เป็นการกล่าวถึงการออกแบบระบบควบคุมของมอเตอร์ไฟฟ้ากระแสตรง โดยการนำ PID Controller ไปชดเชยเนื่องจากอาจเกิดการแกว่งของระบบ โดยในที่นี้จะกล่าวถึงการหาค่าพารามิเตอร์ของระบบควบคุมแบบ PID ซึ่งก็คือ K_p K_i และ K_d ซึ่งการหาค่าพารามิเตอร์ทั้ง 3 ค่านี้ก็คือการออกแบบระบบควบคุมนั่นเอง โดยมีฟังก์ชั่นถ่ายโอนของระบบ ดังสมการที่ (3.3)

Transfer Function ของเตียง

$$T(s) = \frac{3293s^2 + 227,000s + 1,300,000}{s} \qquad (3.3)$$

จากระบบสามารถหาค่า K_p K_i และ K_d ได้ดังนี้

K_p = 227,000

K_i = 1,300,000

K_d = 3,293

โดยในการออกแบบระบบเพื่อนำไปควบคุมรถนั้นได้นำตัวควบคุมชนิดพีดีมาชดเชย เนื่องจากอาจเกิดการแกว่งของระบบ โดยมีฟังก์ชั่นถ่ายโอนของระบบดังนี้

Transfer Function ของรถ

$$\frac{Y(s)}{U(s)} = \frac{(2.09\times10^8)s + 7.22\times10^9}{8750s^4 + 313500s^3 + (5.833\times10^7) + (2.09\times10^8)s + 5054} \qquad (3.4)$$

จากระบบสามารถหาค่า K_p และ K_d ได้ดังนี้

$K_P = 7.22\times10^9$

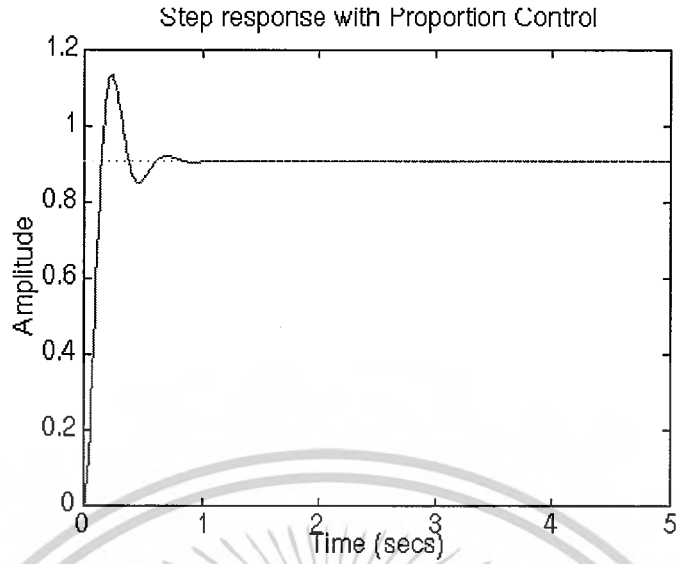
$K_d = 2.09\times10^8$

โดยที่

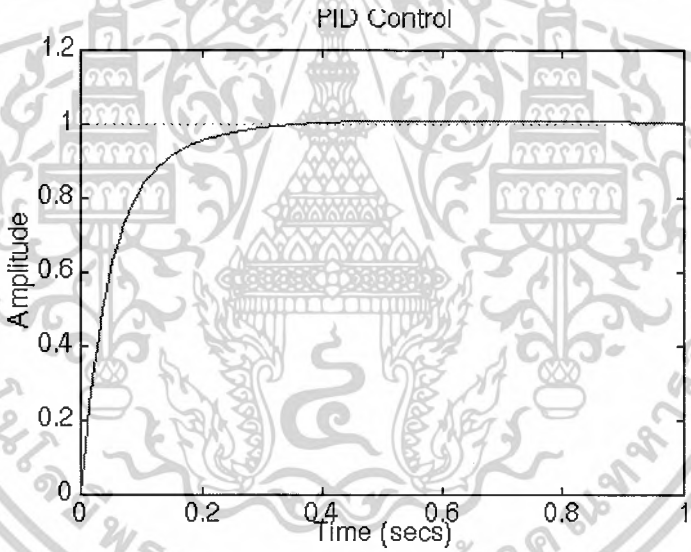
$K_1 = 190000$ n/m

$K_2 = 38000$ n/m

$B = 1100$ n/m·sec^{-1}



รูปที่ 3.11 ผลตอบสนองของระบบ ยังไม่ได้ชดเชยด้วย PID



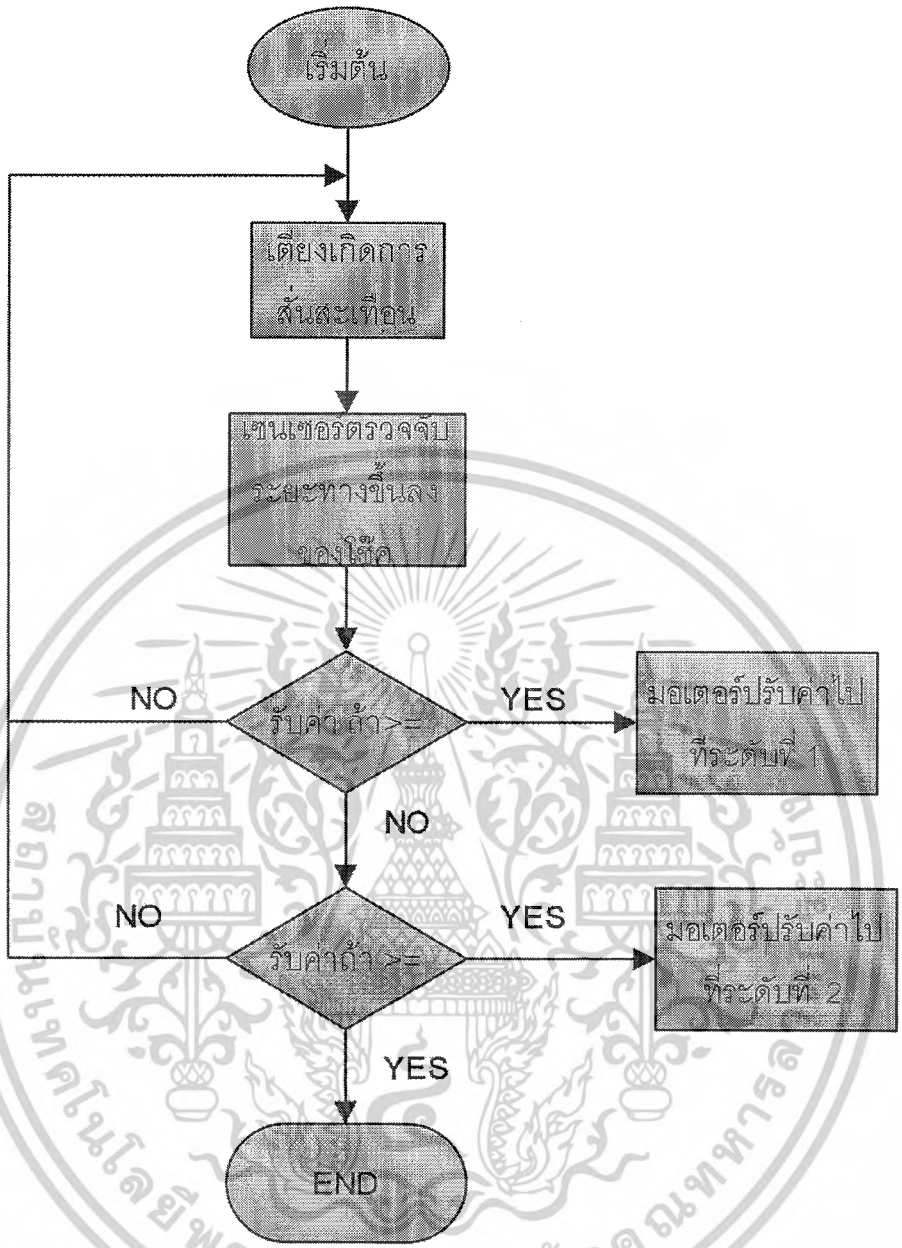
รูปที่ 3.12 ผลตอบสนองของระบบเมื่อทำการชดเชยด้วย PID Controller

จากรูปที่ 3.11 เมื่อทำการทดลองใช้ P-Controller ที่มี K_p= 100 ซึ่งพบว่าสามารถทำให้ความเร็วรอบของมอเตอร์ขึ้นไปได้ถึงเพียงแค่ประมาณ 0.9 เรเดียนต่อวินาทีซึ่งต่ำกว่าค่าที่ต้องการ และมี Overshoot สูงเกินค่าที่ยอมรับได้ เนื่องจากผลตอบสนองของระบบมีทั้ง Steady State Error และ Overshoot จึงต้องนำ PID Controller มาใช้ จากรูปที่ 3.12 หลังจากที่เพิ่มค่า K_d ขึ้นเป็น 10 จะพบว่าสามารถกำจัด Overshoot ของระบบลงได้ และทำให้ผลตอบชั่วครู่ของระบบดีขึ้นและยังเข้าเสถียรภาพโดยปราศจาก Steady State Error ได้เร็วขึ้น ทำให้ระบบควบคุมความเร็วของมอเตอร์นี้สามารถตอบสนองความต้องการที่เราตั้งไว้ได้เป็นที่เรียบร้อยแล้ว

3.2 ขั้นตอนการทำงานของอุปกรณ์

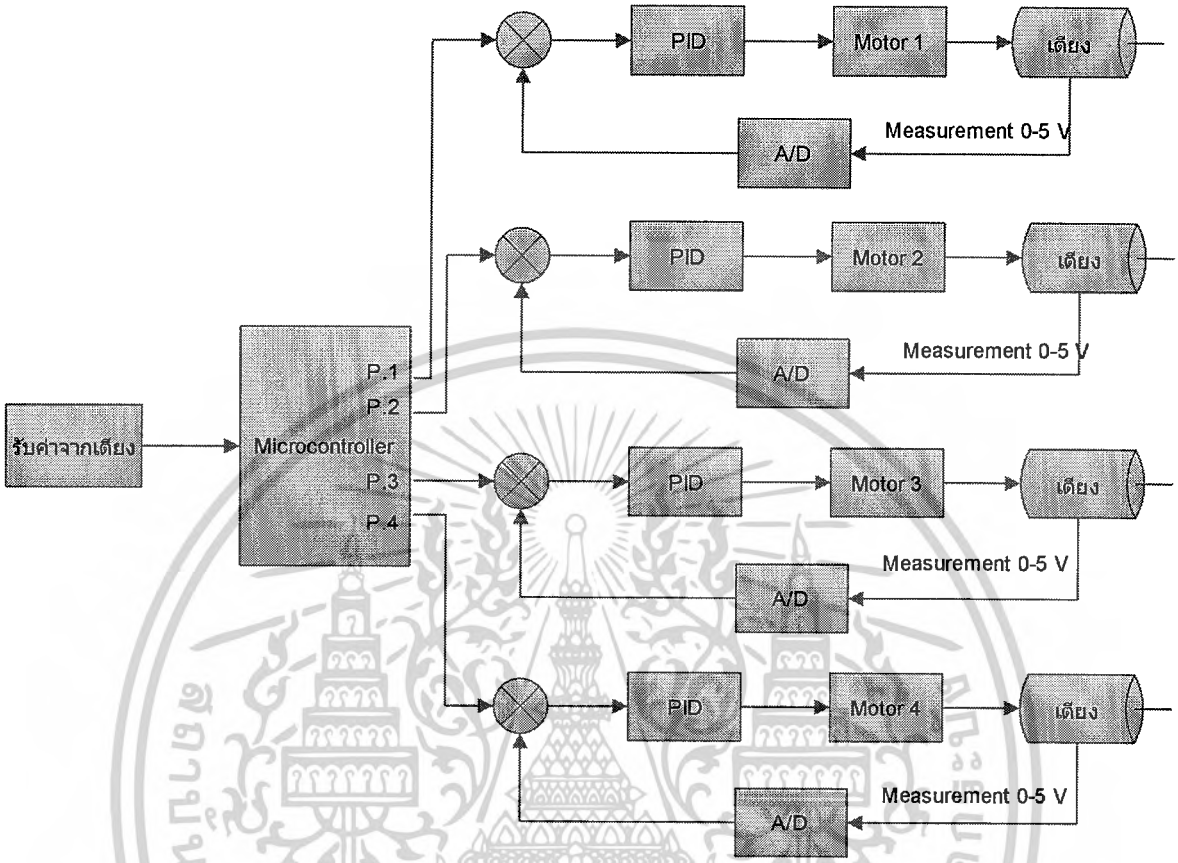
สำหรับขั้นตอนการทำงานของโครงงานนี้ มีแผนผังการทำงานดังรูปที่ 3.13 และมีลำดับขั้นตอนการทำงานดังต่อไปนี้

1. ทำการเคลื่อนย้ายเตียงไปยังจุดต่างๆ
2. เมื่อเตียงมีการเคลื่อนที่ไปตามพื้นถนนที่ขรุขระจะทำให้เกิดการสั่นสะเทือน
3. เซนเซอร์ทำการตรวจจับระยะทางที่เคลื่อนที่ขึ้นลงของโช๊คอัพแล้วจากนั้นไมโครคอนโทรเลอร์ทำการเก็บค่าไว้
4. ไมโครคอนโทรลเลอร์ทำการเก็บค่าจากระยะทางที่เปลี่ยนไปของเซนเซอร์ตามที่ได้กำหนดไว้
5. ไมโครคอนโทรลเลอร์ทำการกำหนดระดับของมอเตอร์ที่ควรใช้และทำการปรับระดับโช๊คอัพที่เหมาะสมกับการสั่นสะเทือนในขณะนั้นๆ โดยที่มอเตอร์ที่ใช้สามารถปรับระดับได้ 2 ระดับ
6. รับค่าจากระยะทางที่เปลี่ยนไปเรื่อยๆและปรับระดับของโช๊คอัพให้เหมาะสม จนกว่าเตียงจะหยุดนิ่ง เมื่อหยุดนิ่งแล้วโช๊คอัพจะปรับระดับให้เป็นระดับที่ 1 นั่นคือ สภาวะเริ่มต้นของระบบ



รูปที่ 3.13 แผนผังแสดงการทำงานของระบบ

3.3 การทำงานของเตียง



รูปที่ 3.14 แผนผังรูปการทำงานของเตียง

จากแผนผังดังรูปที่ 3.14 นำสมการที่เป็นดิจิตอลที่ได้จากการออกแบบตัวควบคุม PID Controller มาออกแบบระบบควบคุมมอเตอร์ 4 ตัว แล้วนั้น โดยบล็อก PID และ A/D คือ โปรแกรมที่เขียนอยู่ในไมโครคอนโทรลเลอร์ dsPIC30F2010 โดยที่ไมโครคอนโทรลเลอร์ทำการรับค่าจากเตียงเมื่อเกิดการสั่นสะเทือน และบล็อก PID คือ สมการ PID ของมอเตอร์แต่ละตัว แล้วจากนั้นไมโครคอนโทรลเลอร์ได้ทำการส่งผลลัพท์ที่ได้ไปยังบล็อกมอเตอร์ เพื่อทำการขับมอเตอร์เกียร์ตามที่ได้ทำการเซตค่าไว้ในแต่ระดับว่าควรจะปรับความแข็งอ่อนของโช๊คอัพเท่าใด บล็อก A/D ทำการรับค่าแรงดันจาก โพเทนทิออมิเตอร์ ซึ่งติดอยู่กับโช๊คป้อนกลับมาควบคุมให้โช๊คอัพให้ได้ความแข็งหรืออ่อนตามที่ควรจะเป็นในการรองรับแรงสั่นสะเทือน

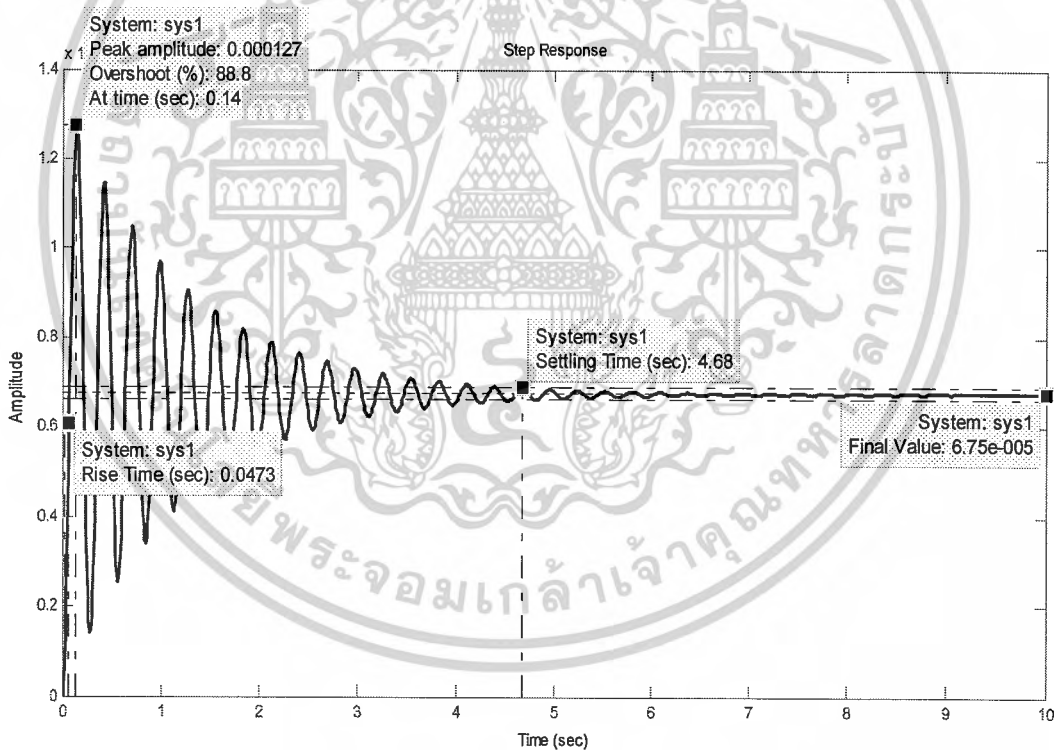
บทที่ 4

การทดลองและผลการทดลอง

ในบทนี้จะกล่าวถึงการทดลองรวมทั้งผลการทดลอง ซึ่งได้ทำการทดลอง ปรับค่า K ของโช๊คอัพโดยวิธีการของ Ziegler-Nichols โดยใช้โปรแกรม MATLAB โดยที่ผู้จัดทำโครงงานได้เลือกใช้โช๊คอัพที่สามารถปรับระดับความแข็งอ่อนได้ 2 ระดับ

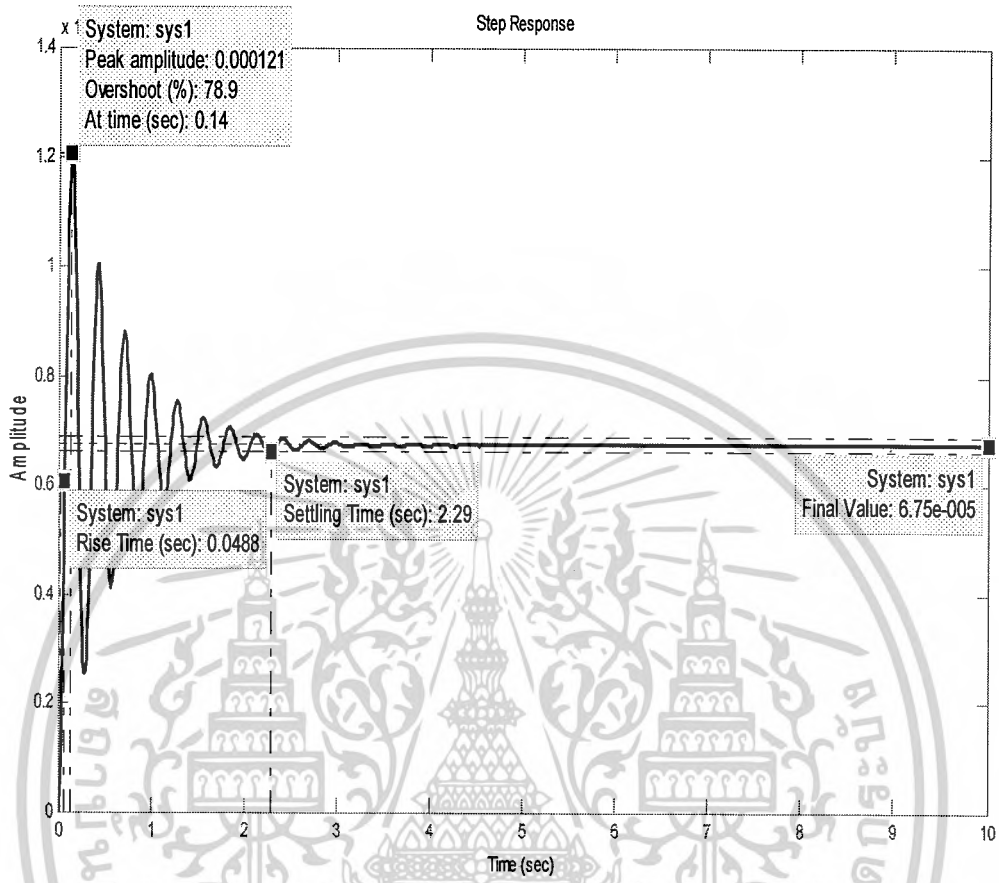
4.1 การทดลองระบบของเตียงเมื่อยังไม่ได้ใช้ระบบควบคุม PID Controller มาควบคุมระบบ

จากการทดลองที่ระดับของโช๊คอัพทั้งสองระดับนั้น ผลที่ได้ออกมาสามารถแสดงเป็นกราฟดังนี้



รูปที่ 4.1 แสดงกราฟผลของระบบเมื่อไม่มีการใช้ตัวควบคุม ของโช๊คอัพระดับที่ 1

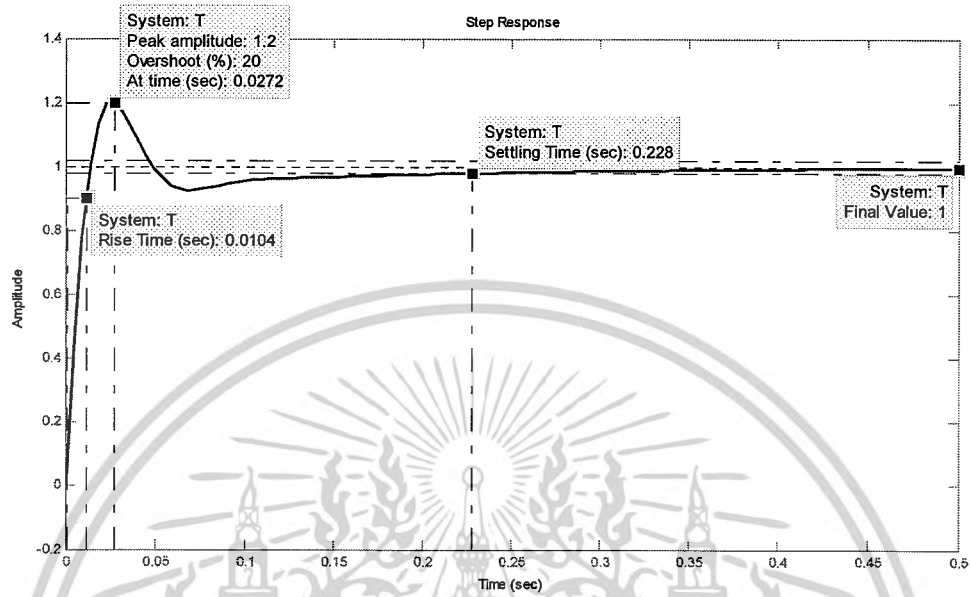
จากรูปที่ 4.1 เป็นการทดสอบระบบโดยไม่ได้ใช้ระบบควบคุม PID Controller มาควบคุม โดยใช้โปรแกรม MATLAB ในการทดสอบระบบ พบว่า โช๊คอัพระดับที่1 มีค่า Overshoot สูงถึง 88.8% ที่เวลา 0.14 วินาที และมีค่า Settling Time หรือเวลาที่เข้าสู่ระบบเสถียรภาพ เท่ากับ 4.68 วินาที



รูปที่ 4.2 แสดงกราฟผลของระบบเมื่อไม่มีการใช้ตัวควบคุม ของโช๊คอัพระดับที่ 2

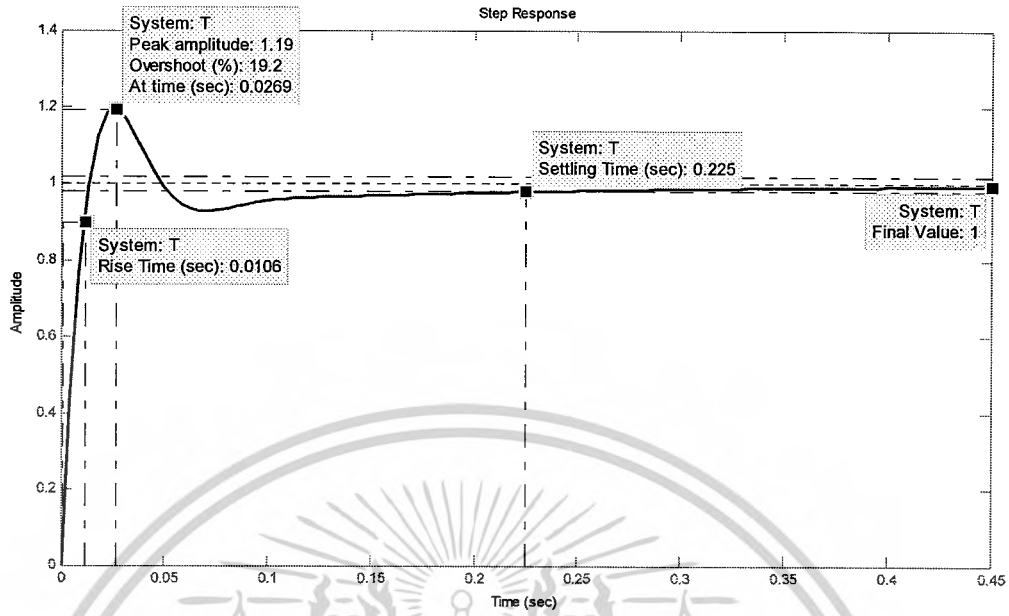
จากรูปที่ 4.2 เป็นการทดสอบระบบของโช๊คอัพระดับที่ 2 ที่ไม่มีการใช้ตัวควบคุมมาควบคุมระบบ พบว่า Overshoot ของระบบมีค่าเท่ากับ 78.9% ที่เวลา 0.14 วินาที และมีค่า Settling Time ที่เวลา2.29 วินาที

4.2 การทดลองปรับค่า K เมื่อใช้ตัวควบคุม PID Controller



รูปที่ 4.3 แสดงกราฟผลของระบบเมื่อมีการใช้ตัวควบคุมชนิดพีไอดีมาควบคุมโช๊คอัพระดับที่ 1

จากรูปที่ 4.3 เป็นกราฟแสดงผลของระบบที่ใช้ตัวควบคุม PID Controller มาควบคุมระบบ เมื่อทำการเปรียบเทียบกับระบบที่ไม่มีการใช้ตัวควบคุมมาควบคุมระบบ จะเห็นได้ว่า เมื่อมีควบคุมระบบโดยใช้ตัวควบคุมชนิดพีไอดีนั้น จะทำให้ระบบมีเสถียรภาพที่ดีขึ้น สามารถเปรียบเทียบได้จากค่าของ Overshoot มีค่าลดน้อยลงจนมีค่า Overshoot เท่ากับ 20% ที่เวลาเวลา 0.0272 วินาที แล้วเข้าสู่ Settling Time ที่เวลา 0.228 วินาที จะเห็นได้ว่า เวลาที่เข้าสู่ระบบเสถียรภาพนั้นเร็วขึ้นกว่าระบบที่ไม่ได้ใช้ตัวควบคุมมาควบคุมระบบ



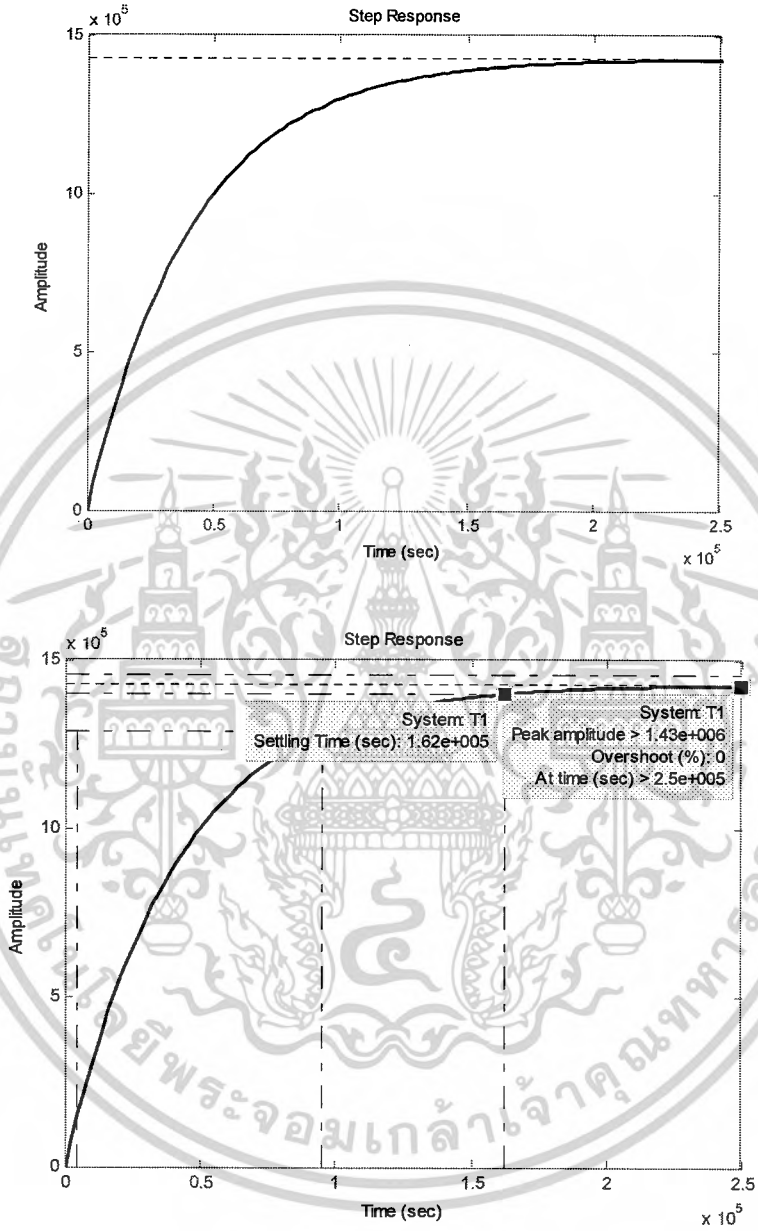
รูปที่ 4.4 แสดงกราฟผลของระบบเมื่อมีการใช้ตัวควบคุมชนิดพีไอดีมาควบคุมโช๊คอัพระดับที่ 2

จากรูปที่ 4.4 เป็นกราฟแสดงผลของระบบของโช๊คอัพระดับที่ 2 ได้ผลเช่นเดียวกับระดับที่ 1 นั่นคือระบบมีเสถียรภาพที่ดีขึ้น มากกว่าระบบที่ไม่มีตัวควบคุม โดยที่ค่า Overshoot มีค่าลดน้อยลงมากจนมีค่า Overshoot เท่ากับ 19.2% ที่เวลาเวลา 0.0269 วินาที แล้วเข้าสู่ Settling Time ที่เวลา 0.225 วินาที

จากระบบเตียงที่ได้ทำการทดสอบจะเห็นได้ว่า โช๊คอัพทั้งสองระดับที่สามารถปรับค่าได้ของเตียงขนย้ายผู้ป่วยแบบป้องกันการกระเทือนนั้น เมื่อยังไม่มีการปรับระบบใดๆ แล้วทำการป้อนสัญญาณหนึ่งหน่วยเข้าไปในระบบ จะพบว่าระบบเข้าสู่เสถียรภาพได้ช้ากว่าระบบที่มีตัวควบคุมมาปรับปรุงระบบเรียบร้อยแล้ว

เมื่อเปรียบเทียบระบบของโช๊คอัพระดับที่ 1 และโช๊คอัพระดับที่ 2 แล้วจะพบว่าโช๊คอัพระดับที่ 2 มีค่าความหน่วงและค่าคงที่ของสปริงมากกว่าโช๊คอัพระดับที่ 1 ดังนั้นโช๊คอัพระดับที่ 2 จึงสามารถเข้าสู่เสถียรภาพได้เร็วกว่าโช๊คอัพระดับที่ 1 แต่เนื่องจากระบบมีความสามารถในการปรับระดับได้เพียงสองระดับ จึงมีขอบเขตในการปรับระบบน้อยตามไปด้วย

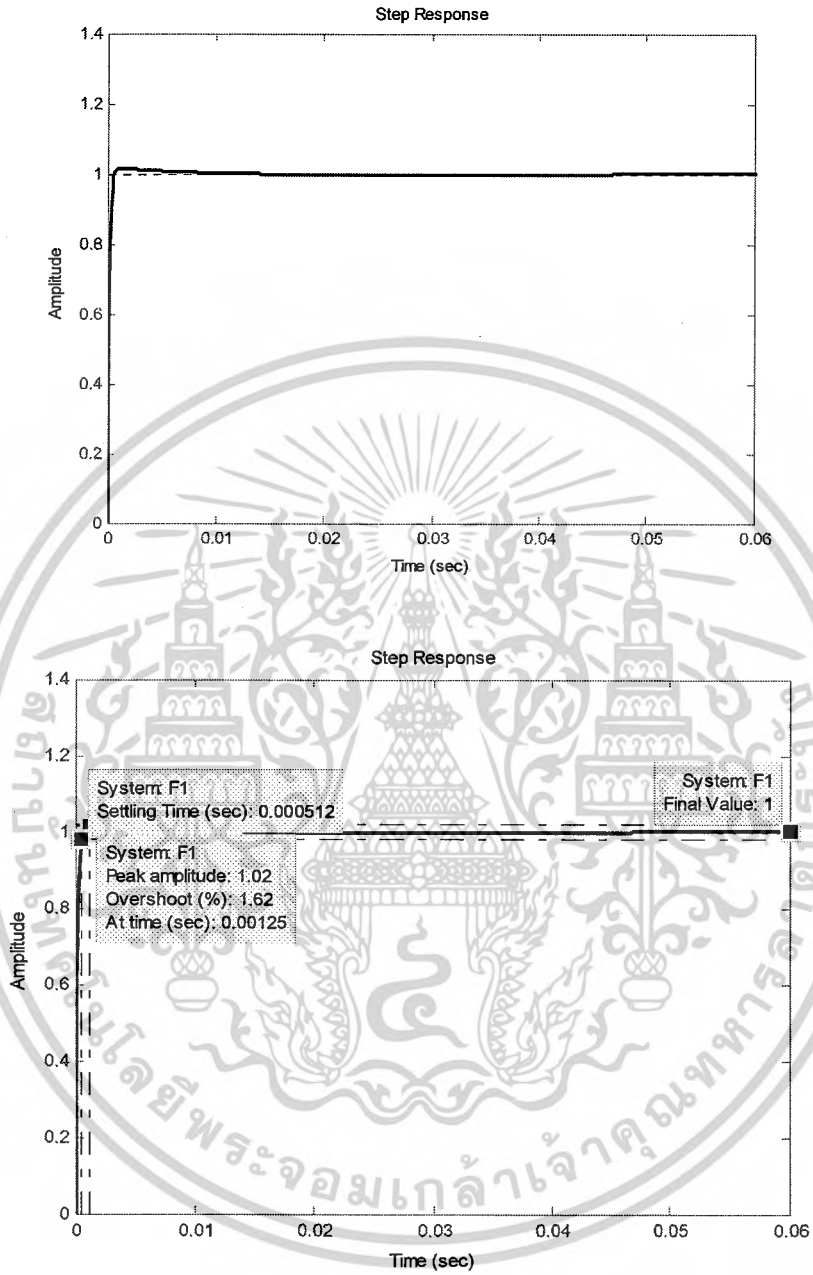
4.3 การทดลองระบบของรถเมื่อยังไม่ได้ใช้ระบบควบคุม PD Controller มาควบคุมระบบ



รูปที่ 4.5 แสดงผลกราฟระบบของรถเมื่อไม่มีการใช้ตัวควบคุม

จากรูปที่ 4.5 เป็นกราฟผลตอบสนองของระบบรถยนต์เมื่อระบบไม่มีการใช้ระบบควบคุมชนิด PD Controller มาควบคุมระบบซึ่งแสดงผลจากโปรแกรม MATLAB พบว่า ระบบรถยนต์ มีค่าOvershoot เท่ากับ 0% ที่เวลา 250000 วินาที และมีค่า Settling Time หรือเวลาที่เข้าสู่ระบบเสถียรภาพ เท่ากับ 162000 วินาที

4.4 การทดลองปรับค่า K ของรถเมื่อใช้ตัวควบคุม PD Controller



รูปที่ 4.6 แสดงกราฟผลระบบของรถเมื่อมีการใช้ตัวควบคุมแบบ PD Controller แล้ว

จากรูปที่ 4.6 เป็นกราฟผลตอบสนองของระบบรถยนต์เมื่อระบบควบคุม PD Controller มาควบคุมระบบโดยแสดงผลจากโปรแกรม MATLAB พบว่า ระบบรถยนต์ มีค่า Overshoot เท่ากับ 1.62% ที่เวลา 0.00125 วินาที และมีค่า Settling Time หรือเวลาที่เข้าสู่ระบบเสถียรภาพ เท่ากับ 0.000512 วินาที

การทดลองของระบบรถดังกล่าวจะเห็นได้ว่าเมื่อป้อนสัญญาณหนึ่งหน่วยเข้าไปในระบบ ระบบมีผลกระทบต่อสัญญาณหนึ่งหน่วยน้อยมาก เนื่องจากรถมีค่าความหน่วงที่มากหรือสามารถรับน้ำหนักได้มาก จึงทำให้เกิดผลกระทบต่อระบบน้อย จึงอาจเปลี่ยนสัญญาณและขนาดสัญญาณเป็นแบบอื่นๆ เพื่อให้เห็นค่าความแตกต่างได้ดีขึ้น

4.5 ตารางแสดงผลการทดลองเมื่อมีการทดลองปรับค่า R

V	R	ระยะทาง(cm)	ระดับของโช๊คอัพ
5	0k	0	ไม่เปลี่ยนแปลง
4.95	0.1k	0.1	ไม่เปลี่ยนแปลง
4.75	0.5k	0.5	ไม่เปลี่ยนแปลง
4.50	1k	1	เปลี่ยนแปลง
2.5	5k	5	เปลี่ยนแปลง
0	10k	10	เปลี่ยนแปลง

ตารางที่ 4.1 ตารางแสดงผลการทดลองเมื่อมีการปรับค่า R

จากตารางที่ 4.1 เป็นการแสดงผลการทดลองจะพบว่า เมื่อมีการรับค่าความต่างศักย์เข้ามาประมวลผล โดยที่ค่าความต่างศักย์ที่เรารับเข้ามา มีค่าความเปลี่ยนแปลงเกินไปกว่าที่กำหนดไว้ จะทำให้โช๊คอัพเกิดการเปลี่ยนระดับเกิดขึ้น แต่ถ้าระยะที่รับค่าของความต่างศักย์นั้นเปลี่ยนแปลงไปน้อยกว่าที่กำหนด ระบบจะไม่มีการเปลี่ยนแปลง จะคงอยู่ที่เป็นอยู่ต่อไป จนกว่าระบบจะหยุดนิ่งจึงการเปลี่ยนระดับของโช๊คอัพมาที่ระดับต่ำสุดนั้นคือระดับที่ 1

บทที่ 5

บทวิจารณ์และสรุป

5.1 สรุปผลการทดลอง

จากการทดลองการรองรับแรงสั่นสะเทือนโดยใช้ไมโครคอนโทรลเลอร์ dsPIC30F20120 ในการควบคุม โพเทนทิออมิเตอร์แบบเชิงเส้น ในการวัดระยะทางของโช๊คอัพที่เปลี่ยนแปลงเมื่อระบบเกิดการสั่นจะพบว่าระยะทางที่เคลื่อนที่ไปได้ของโพเทนทิออมิเตอร์นั้นจะแปรผันตรงกับค่าแรงดัน ซึ่งจะนำไปปรับระดับของโช๊คอัพ ซึ่งสามารถปรับระดับได้สองระดับ

เมื่อปรับค่าพารามิเตอร์ของโช๊คอัพโดยใช้โปรแกรม MATLAB จะพบว่าเมื่อเราใช้ตัวควบคุม PID Controller จะทำให้เกิดการเข้าสู่เสถียรภาพได้ดีกว่าระบบที่ไม่มีตัวควบคุม และการปรับระดับโช๊คอัพทั้ง 2 ระดับนี้จะเห็นได้ว่า ระดับที่สองนั้นสามารถทำให้เตียงนั้นรองรับแรงสั่นสะเทือนได้มากกว่าระดับแรก

5.2 ปัญหาที่พบและแนวทางแก้ไข

จากการศึกษาและทำโครงงานนี้จะพบว่าปัญหาที่เกิดขึ้นนั้นคือไม่สามารถนำตัวควบคุมชนิดพีไอดีเข้าไปควบคุมในส่วนของการเขียนโปรแกรมในไมโครคอนโทรลเลอร์เพื่อควบคุมมอเตอร์ให้เกิดเสถียรภาพที่ดีขึ้นได้ แต่ได้มีการทดลองปรับค่า K โดยการใช้โปรแกรม MATLAB เพื่อนำมาเปรียบเทียบผลระหว่างการใช้ตัวควบคุมแบบพีไอดีกับการที่ไม่ใช้ตัวควบคุม โดยวิธีที่ดีที่สุดในการที่จะทำให้ระบบของเตียงนั้นเกิดเสถียรภาพที่ดีขึ้นก็คือควรจะมีการนำตัวควบคุมชนิดพีไอดีเข้าไปควบคุมด้วย นอกจากนี้เมื่อทำการทดลองปรับค่า K ในการควบคุมระบบของรถโดยได้ทำการออกแบบโดยใช้ตัวควบคุมชนิดพีไอดีนั้นจะพบว่าสามารถทำได้ยากหรือไม่สามารถทำได้โดยใช้ตัวควบคุมแบบพีไอดีได้ จึงต้องใช้ตัวควบคุมชนิดพีดีแทน

5.3 ข้อเสนอแนะและแนวทางในการค้นคว้าพัฒนา

ตัวควบคุมที่ใช้ในปัจจุบันนั้นมีอยู่อีกหลายแบบ ควรทำการศึกษาเพิ่มเติมว่ามีตัวควบคุมแบบใดบ้างที่เหมาะสมกับระบบควบคุมของเตียง โดยอาจจะมีการนำตัวควบคุมชนิดฟัซซี (Fuzzy Control) เข้ามาช่วยในการออกแบบตัวควบคุมซึ่งจะทำให้ออกแบบได้ง่ายขึ้น นอกจากนี้อาจมีการหาโช๊คอัพไฟฟ้าที่สามารถปรับได้หลาย ๆ ระดับเพื่อทำให้เกิดความแตกต่างในการรองรับแรงสั่นสะเทือนซึ่งอาจจะก่อให้เกิดผลดียิ่งกว่าเดิม เนื่องจากสามารถปรับได้หลายระดับ

เอกสารอ้างอิง

[1] รัฐพงศ์ ปฏิกานัง. "การควบคุมแบบฟัซซีสำหรับระบบกันสะเทือนแบบบังคับโดยใช้แบบจำลองเต็มของยานพาหนะ." ปริญญานิพนธ์วิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมเครื่องกล คณะวิศวกรรมศาสตร์, มหาวิทยาลัยขอนแก่น.2548.

[2] กนก ตันติยวุฒิ, ชานนท์ สารพานิช. "ตัวควบคุมแบบพีไอดีชนิดหาค่าอัตโนมัติ." ปริญญานิพนธ์วิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมระบบควบคุม คณะวิศวกรรมศาสตร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง.2543.

[3] Katsuhuko Ogata. Modern Control Engineering. Fourth Edition-Prentice Hall, 1970

[4] ผศ.ดร.วรพงศ์ ตั้งศรีรัตน์. เซนเซอร์และทรานสดิวเซอร์. พิมพ์ครั้งที่ 4. กรุงเทพมหานคร : ส.ส.ท. 2550.

ภาคผนวก

ภาคผนวก ก

โปรแกรมควบคุม

```
---------------------------------------------------------------------------------------------------------
#include<p30f2010.h>
#include<adc10.h>
#include <stdio.h>

void Init_adc() //ฟังกชั่นใช้งาน ADC
{
    unsigned int Channel , PinConfig , Scanselect , Adcon3_reg , Adcon2_reg , Adcon1_reg
; //ประกาศค่าตัวแปรที่ใช้

    ADCON1bits.ADON = 0 ; //ปิดการทำงาน ADC

    Channel     =   ADC_CH0_POS_SAMPLEA_AN0 &
                    ADC_CH0_POS_SAMPLEA_AN1 &
                    ADC_CH0_POS_SAMPLEA_AN2 &
                    ADC_CH0_POS_SAMPLEA_AN3 &
                    ADC_CH0_NEG_SAMPLEA_NVREF ;  //กำหนดที่ Channel
                0 AN0-AN3 เป็น Positve และ กำหนดที่ Channel 0 NVREF เป็น
                Negative

                SetChanADC10(Channel); //เซต Channel ที่ใช้

    PinConfig   =       ENABLE_AN0_ANA &
                        ENABLE_AN1_ANA &
                        ENABLE_AN2_ANA &
                        ENABLE_AN3_ANA ;  //กำหนดพอร์ต AN0-AN3
                                          สามารถใช้งานได้

    Scanselect  =   SKIP_SCAN_AN4 &
                    SKIP_SCAN_AN5 ;     // ข้ามการ SCAN ที่ขา AN4-AN5

    Adcon3_reg  =   ADC_SAMPLE_TIME_10 &    // ทดลองรับค่าจำนวน10ครั้ง
```

```
                    ADC_CONV_CLK_SYSTEM & // เรียกใช้งานสัญญาณนาฬิกา
                                          ภายใน
                    ADC_CONV_CLK_2Tcy ;
        Adcon2_reg  =   ADC_VREF_AVDD_AVSS &   // อ้างอิงค่าความต่าง ศักย์ที่
Vdd และ Vss
                    ADC_SCAN_ON &      // ทำการ SCAN สำหรับฟังก์ชั่น ADC
                    ADC_ALT_BUF_OFF & // ปิดทำหาร alternate buffer
                    ADC_ALT_INPUT_OFF & // ปิดทำการ alternate input
                    ADC_CONVERT_CH0& // เลือกทำการ convert ที่ CH0
                    ADC_SAMPLES_PER_INT_1 ; // สุ่มอินพุต1ค่าระหว่างการ
อินเทอรัพท์

        Adcon1_reg  =   ADC_MODULE_ON & // เปิดการใช้งาน module ADC
                    ADC_IDLE_CONTINUE &// กำหนดให้ ADC ทำงานใน idle
mode ได้
                    ADC_FORMAT_SIGN_INT &   // กำหนดสัญญาณ Output
เป็นจำนวนเต็ม
                    ADC_CLK_MANUAL &// ADC Auto clock
                    ADC_SAMPLE_SIMULTANEOUS &// ADC สุ่มค่าตัวอย่าง
พร้อมๆกัน
                    ADC_AUTO_SAMPLING_ON ;// ADC สุ่มค่าตัวอย่าง
อัตโนมัติ

        OpenADC10(Adcon1_reg , Adcon2_reg , Adcon3_reg , PinConfig , Scanselect) ;   //
เปิดการทำงาน ADC module
        DisableIntADC ;// ปิดการใช้งานการอินเทอรัพท์ใน ADC
}
int diff_value(unsigned int value1,unsigned int value2)      // ฟังก์ชั่นหาค่าความต่างศักย์ที่รับมามี
ค่าเป็น+
{
        unsigned int ans ;
```

```
        if(value1<=value2)
        {
                ans = value2 - value1 ;
        }
        else
        {
                ans = value1 - value2 ;
        }
        return ans ;
}

int chk_value(unsigned int x) // ฟังก์ชั่นตรวจสอบค่าความต่างศักย์เพื่อแสดงผล
{
        unsigned int bits ;
        if(x <= 50)
        {
                bits = 1;
        }
        else
        {
                bits = 0;
        }
        return bits ;
}

void delay_led(unsigned long int count1) // ฟังก์ชั่น delay
{
      while(count1>0)
      {
        count1--;
      }
}
```

```
int main()
{
        Init_adc() ;
        unsigned int old_result[4] , result[4] , answer[4] , bas[4] ;
         unsigned int  i , j , k , l ;

while(1)
        {
        for(i=0;i<4;i++)
                {
                        ADCON1bits.SAMP = 1 ;           // เริ่มการสุ่มค่า
                        while(!ADCON1bits.SAMP) ;   // รอกระบวนการสุ่มค่า ทำให้เสร็จ
ก่อน
                        ConvertADC10() ;                // Convert ADC
                        while(BusyADC10());             // ตรวสอบให้แน่ใจว่า
กระบวนการสุ่มค่าสำเร็จแล้ว
                        result[i] = ReadADC10(i) ;      // รับค่ามาเก็บไว้
                        delay_led(500);                 // หน่วงเวลาก่อนรับค่าต่อไป
                }

                for (j=0;j<4;j++)
                {
                        answer[j] = diff_value(result[j],old_result[j])       ; // นำค่าที่ฟงก์ชั่น
diff_value กลับมาเก็บไว้
                }

        TRISEbits.TRISE0 = 0 ;
        TRISEbits.TRISE1 = 0 ;
        TRISEbits.TRISE2 = 0 ;
        TRISEbits.TRISE3 = 0 ; // กำหนดพอร์ต E เป็น Output
```

```
for(k=0;k<4;k++)
{
        bas[k] = chk_value(answer[k]) ; // นำค่าที่ฟงก์ชั่น chk_value กลับมาเก็บไว้
}

        LATEbits.LATE0 = bas[0] ;
        LATEbits.LATE1 = bas[1] ;
        LATEbits.LATE2 = bas[2] ;
        LATEbits.LATE3 = bas[3] ; // หาก bas[k] มีค่าเป็น 1 จะแสดงผลที่พอร์ต E ขา
นั้นๆ

            if(bas[0]==0||bas[1]==0||bas[2]==0||bas[3]==0)
            {
                delay_led(1000000); // กำหนดให้แสดงผลค้างไว้
            }
            else
            {
                ;
            }

for(l=0;l<4;l++)
{
        old_result[l] = result[l] ; // เก็บค่าที่รับเข้ามาเพื่อนำไปเปรียบเทียบกับค่าใหม่
ต่อไป
}

delay_led(500000); // เรียกการใช้งาน delay

}
}
```

ภาคผนวก ข

เอกสารคู่มือ dsPIC30F2010

dsPIC30F2010

28-pin dsPIC30F2010 Enhanced Flash 16-bit Digital Signal Controller

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

High-Performance Modified RISC CPU:

- Modified Harvard architecture
- C compiler optimized instruction set architecture
- 84 base instructions with flexible addressing modes
- 24-bit wide instructions, 16-bit wide data path
- 12 Kbytes on-chip Flash program space
- 512 bytes on-chip data RAM
- 1 Kbyte non-volatile data EEPROM
- 16 x 16-bit working register array
- Up to 30 MIPs operation:
  - DC to 40 MHz external clock input
  - 4 MHz-10 MHz oscillator input with PLL active (4x, 8x, 16x)
- 27 interrupt sources
- Three external interrupt sources
- 8 user selectable priority levels for each interrupt
- 4 processor exceptions and software traps

DSP Engine Features:

- Modulo and Bit-Reversed modes
- Two, 40-bit wide accumulators with optional saturation logic
- 17-bit x 17-bit single cycle hardware fractional/integer multiplier
- Single cycle Multiply-Accumulate (MAC) operation
- 40-stage Barrel Shifter
- Dual data fetch

Peripheral Features:

- High current sink/source I/O pins: 25 mA/25 mA
- Three 16-bit timers/counters; optionally pair up 16-bit timers into 32-bit timer modules
- Four 16-bit Capture input functions
- Two 16-bit Compare/PWM output functions
  - Dual Compare mode available
- 3-wire SPI™ modules (supports 4 Frame modes)
- I^2C™ module supports Multi-Master/Slave mode and 7-bit/10-bit addressing
- Addressable UART modules with FIFO buffers

Motor Control PWM Module Features:

- 6 PWM output channels
  - Complementary or Independent Output modes
  - Edge and Center Aligned modes
- 4 duty cycle generators
- Dedicated time base with 4 modes
- Programmable output polarity
- Dead time control for Complementary mode
- Manual output control
- Trigger for synchronized A/D conversions

Quadrature Encoder Interface Module Features:

- Phase A, Phase B and Index Pulse input
- 16-bit up/down position counter
- Count direction status
- Position Measurement (x2 and x4) mode
- Programmable digital noise filters on inputs
- Alternate 16-bit Timer/Counter mode
- Interrupt on position counter rollover/underflow

Analog Features:

- 10-bit Analog-to-Digital Converter (A/D) with:
  - 500 Ksps (for 10-bit A/D) conversion rate
  - Six input channels
  - Conversion available during Sleep and Idle
- Programmable Brown-out Detection and Reset generation

Special Microcontroller Features:

- Enhanced Flash program memory:
  - 10,000 erase/write cycle (min.) for industrial temperature range, 100K (typical)
- Data EEPROM memory:
  - 100,000 erase/write cycle (min.) for industrial temperature range, 1M (typical)
- Self-reprogrammable under software control
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Flexible Watchdog Timer (WDT) with on-chip low power RC oscillator for reliable operation
- Fail-Safe clock monitor operation
- Detects clock failure and switches to on-chip low power RC oscillator
- Programmable code protection
- In-Circuit Serial Programming™ (ICSP™)
- Selectable Power Management modes
  - Sleep, Idle and Alternate Clock modes

CMOS Technology:

- Low power, high speed Flash technology
- Wide operating voltage range (2.5V to 5.5V)
- Industrial and Extended temperature ranges
- Low power consumption

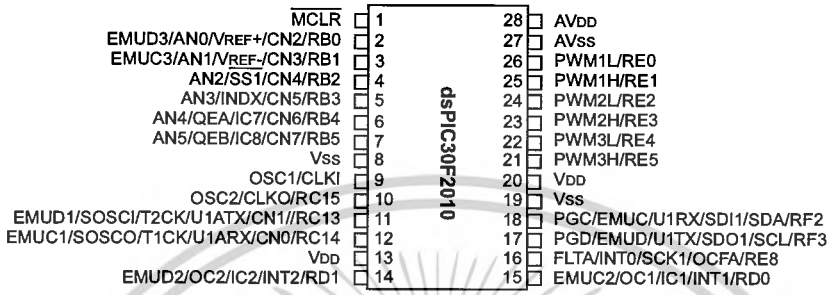
dsPIC30F Motor Control and Power Conversion Family*

Device	Pins	Program Mem. Bytes/ Instructions	SRAM Bytes	EEPROM Bytes	Timer 16-bit	Input Cap	Output Comp/Std PWM	Motor Control PWM	A/D 10-bit 500 Ksps	Quad Enc	UART	SPI™	I²C™	CAN
dsPIC30F2010	28	12K/4K	512	1024	3	4	2	6 ch	6 ch	Yes	1	1	1	–
dsPIC30F3010	28	24K/8K	1024	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	–
dsPIC30F4012	28	48K/16K	2048	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	1
dsPIC30F3011	40/44	24K/8K	1024	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	–
dsPIC30F4011	40/44	48K/16K	2048	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	1
dsPIC30F5015	64	66K/22K	2048	1024	5	4	4	8 ch	16 ch	Yes	1	2	1	1
dsPIC30F6010	80	144K/48K	8192	4096	5	8	8	8 ch	16 ch	Yes	2	2	1	2

* This table provides a summary of the dsPIC30F2010 peripheral features. Other available devices in the dsPIC30F Motor Control and Power Conversion Family are shown for feature comparison.

Pin Diagrams

28-Pin SDIP and SOIC



28-Pin QFN

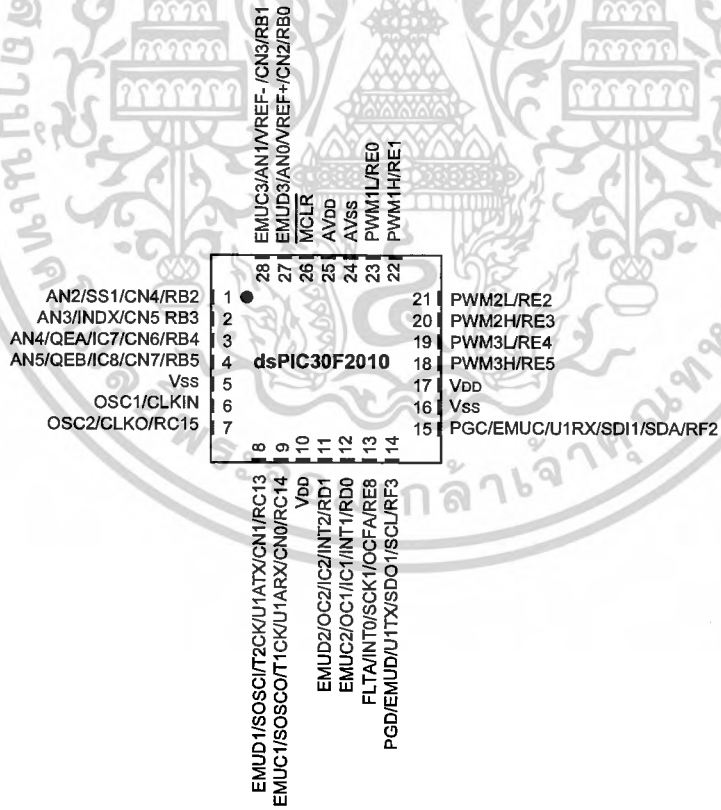


FIGURE 1-1: dsPIC30F2010 BLOCK DIAGRAM

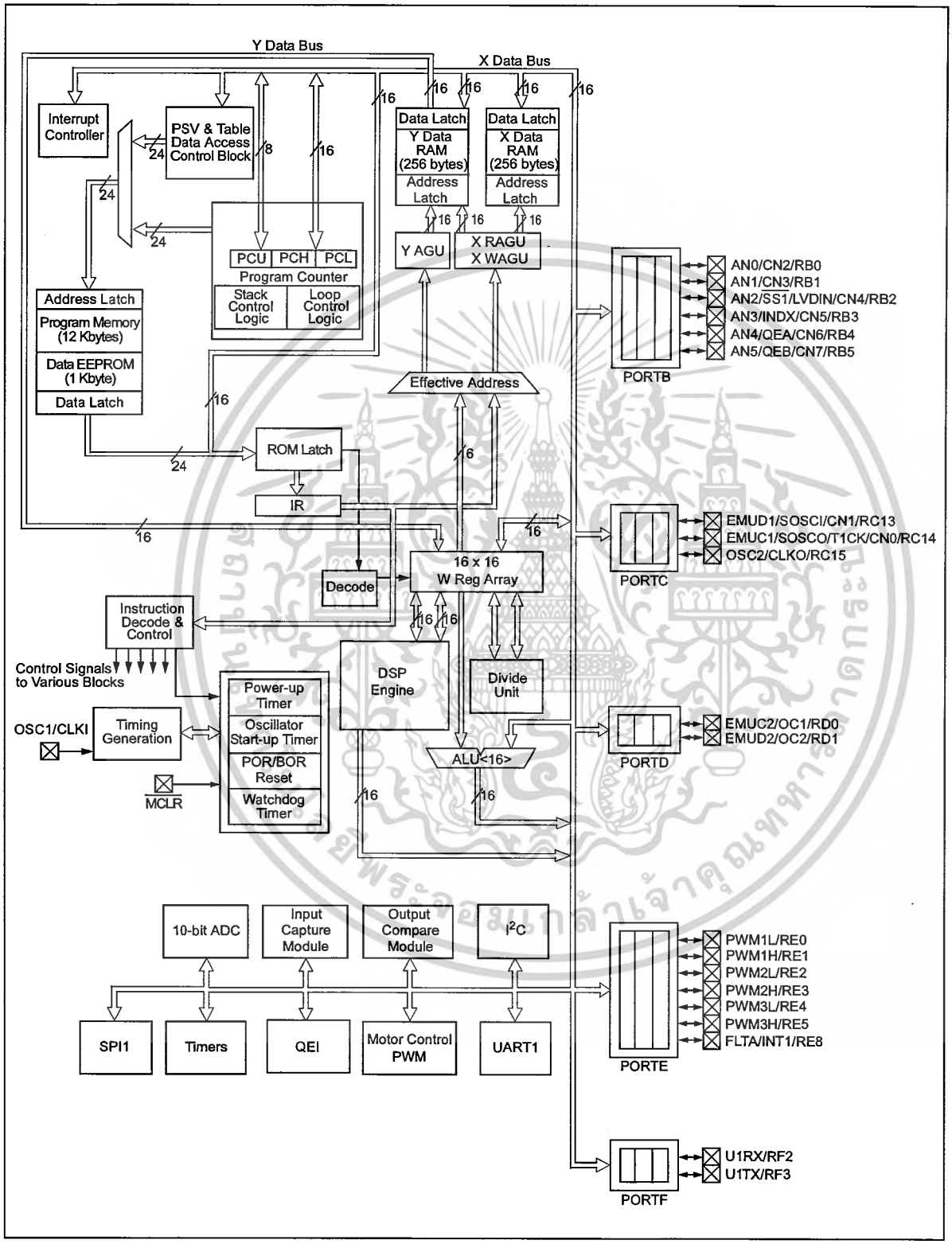


Table 1-1 provides a brief description of device I/O pinouts and the functions that may be multiplexed to a port pin. Multiple functions may exist on one port pin. When multiplexing occurs, the peripheral module's functional requirements may force an override of the data direction of the port pin.

TABLE 1-1: PINOUT I/O DESCRIPTIONS

Pin Name	Pin Type	Buffer Type	Description
AN0-AN5	I	Analog	Analog input channels.
AVDD	P	P	Positive supply for analog module.
AVSS	P	P	Ground reference for analog module.
CLKI CLKO	I O	ST/CMOS —	External clock source input. Always associated with OSC1 pin function. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. Optionally functions as CLKO in RC and EC modes. Always associated with OSC2 pin function.
CN0-CN7	I	ST	Input change notification inputs. Can be software programmed for internal weak pull-ups on all inputs.
EMUD EMUC EMUD1 EMUC1 EMUD2 EMUC2 EMUD3 EMUC3	I/O I/O I/O I/O I/O I/O I/O I/O	ST ST ST ST ST ST ST ST	ICD Primary Communication Channel data input/output pin. ICD Primary Communication Channel clock input/output pin. ICD Secondary Communication Channel data input/output pin. ICD Secondary Communication Channel clock input/output pin. ICD Tertiary Communication Channel data input/output pin. ICD Tertiary Communication Channel clock input/output pin. ICD Quaternary Communication Channel data input/output pin. ICD Quaternary Communication Channel clock input/output pin.
IC1, IC2, IC7, IC8	I	ST	Capture inputs. The dsPIC30F2010 has 4 capture inputs. The inputs are numbered for consistency with the inputs on larger device variants.
INDX QEA QEB	I I I	ST ST ST	Quadrature Encoder Index Pulse input. Quadrature Encoder Phase A input in QEI mode. Auxiliary Timer External Clock/Gate input in Timer mode. Quadrature Encoder Phase A input in QEI mode. Auxiliary Timer External Clock/Gate input in Timer mode.
INT0 INT1 INT2	I I I	ST ST ST	External interrupt 0 External interrupt 1 External interrupt 2
FLTA PWM1L PWM1H PWM2L PWM2H PWM3L PWM3H	I O O O O O O	ST — — — — — —	PWM Fault A input PWM 1 Low output PWM 1 High output PWM 2 Low output PWM 2 High output PWM 3 Low output PWM 3 High output
$\overline{MCLR}$	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low Reset to the device.
OCFA OC1-OC2	I O	ST —	Compare Fault A input (for Compare channels 1, 2, 3 and 4). Compare outputs.
OSC1 OSC2	I I/O	ST/CMOS —	Oscillator crystal input. ST buffer when configured in RC mode; CMOS otherwise. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. Optionally functions as CLKO in RC and EC modes.

Legend: CMOS =CMOS compatible input or output Analog= Analog input
ST =Schmitt Trigger input with CMOS levels O= Output
I =Input P = Power

TABLE 1-1: PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Type	Buffer Type	Description
PGD PGC	I/O I	ST ST	In-Circuit Serial Programming data input/output pin. In-Circuit Serial Programming clock input pin.
RB0-RB5	I/O	ST	PORTB is a bidirectional I/O port.
RC13-RC14	I/O	ST	PORTC is a bidirectional I/O port.
RD0-RD1	I/O	ST	PORTD is a bidirectional I/O port.
RE0-RE5, RE8	I/O	ST	PORTE is a bidirectional I/O port.
RF2, RF3	I/O	ST	PORTF is a bidirectional I/O port.
SCK1 SDI1 SDO1 SS1	I/O I O I	ST ST — ST	Synchronous serial clock input/output for SPI™ #1. SPI #1 Data In. SPI #1 Data Out. SPI #1 Slave Synchronization.
SCL SDA	I/O I/O	ST ST	Synchronous serial clock input/output for I^2C. Synchronous serial data input/output for I^2C.
SOSCO SOSCI	O I	— ST/CMOS	32 kHz low power oscillator crystal output. 32 kHz low power oscillator crystal input. ST buffer when configured in RC mode; CMOS otherwise.
T1CK T2CK	I I	ST ST	Timer1 external clock input. Timer2 external clock input.
U1RX U1TX U1ARX U1ATX	I O I O	ST — ST —	UART1 Receive. UART1 Transmit. UART1 Alternate Receive. UART1 Alternate Transmit.
VDD	P	—	Positive supply for logic and I/O pins.
VSS	P	—	Ground reference for logic and I/O pins.
VREF+	I	Analog	Analog Voltage Reference (High) input.
VREF-	I	Analog	Analog Voltage Reference (Low) input.

Legend: CMOS = CMOS compatible input or output Analog = Analog input
ST = Schmitt Trigger input with CMOS levels O = Output
I = Input P = Power

2.0 CPU ARCHITECTURE OVERVIEW

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

This document provides a summary of the dsPIC30F2010 CPU and peripheral function. For a complete description of this functionality, please refer to the *dsPIC30F Family Reference Manual* (DS70046).

2.1 Core Overview

The core has a 24-bit instruction word. The Program Counter (PC) is 23 bits wide with the Least Significant (LS) bit always clear (see Section 3.1), and the Most Significant (MS) bit is ignored during normal program execution, except for certain specialized instructions. Thus, the PC can address up to 4M instruction words of user program space. An instruction pre-fetch mechanism is used to help maintain throughput. Program loop constructs, free from loop count management overhead, are supported using the DO and REPEAT instructions, both of which are interruptible at any point.

The working register array consists of 16x16-bit registers, each of which can act as data, address or offset registers. One working register (W15) operates as a software stack pointer for interrupts and calls.

The data space is 64 Kbytes (32K words) and is split into two blocks, referred to as X and Y data memory. Each block has its own independent Address Generation Unit (AGU). Most instructions operate solely through the X memory AGU, which provides the appearance of a single unified data space. The Multiply-Accumulate (MAC) class of dual source DSP instructions operate through both the X and Y AGUs, splitting the data address space into two parts (see Section 3.2). The X and Y data space boundary is device specific and cannot be altered by the user. Each data word consists of 2 bytes, and most instructions can address data either as words or bytes.

There are two methods of accessing data stored in program memory:

- The upper 32 Kbytes of data space memory can be mapped into the lower half (user space) of program space at any 16K program word boundary, defined by the 8-bit Program Space Visibility Page (PSVPAG) register. This lets any instruction access program space as if it were data space, with a limitation that the access requires an additional cycle. Moreover, only the lower 16 bits of each instruction word can be accessed using this method.
- Linear indirect access of 32K word pages within program space is also possible using any working register, via table read and write instructions. Table read and write instructions can be used to access all 24 bits of an instruction word.

Overhead-free circular buffers (modulo addressing) are supported in both X and Y address spaces. This is primarily intended to remove the loop overhead for DSP algorithms.

The X AGU also supports bit-reversed addressing on destination effective addresses, to greatly simplify input or output data reordering for radix-2 FFT algorithms. Refer to Section 4.0 for details on modulo and bit-reversed addressing.

The core supports Inherent (no operand), Relative, Literal, Memory Direct, Register Direct, Register Indirect, Register Offset and Literal Offset Addressing modes. Instructions are associated with predefined Addressing modes, depending upon their functional requirements.

For most instructions, the core is capable of executing a data (or program data) memory read, a working register (data) read, a data memory write and a program (instruction) memory read per instruction cycle. As a result, 3-operand instructions are supported, allowing C = A+B operations to be executed in a single cycle.

A DSP engine has been included to significantly enhance the core arithmetic capability and throughput. It features a high speed 17-bit by 17-bit multiplier, a 40-bit ALU, two 40-bit saturating accumulators and a 40-bit bidirectional barrel shifter. Data in the accumulator or any working register can be shifted up to 15 bits right or 16 bits left in a single cycle. The DSP instructions operate seamlessly with all other instructions and have been designed for optimal real-time performance. The MAC class of instructions can concurrently fetch two data operands from memory, while multiplying two W registers. To enable this concurrent fetching of data operands, the data space has been split for these instructions and linear for all others. This has been achieved in a transparent and flexible manner, by dedicating certain working registers to each address space for the MAC class of instructions.

The core does not support a multi-stage instruction pipeline. However, a single stage instruction pre-fetch mechanism is used, which accesses and partially decodes instructions a cycle ahead of execution, in order to maximize available execution time. Most instructions execute in a single cycle, with certain exceptions.

The core features a vectored exception processing structure for traps and interrupts, with 62 independent vectors. The exceptions consist of up to 8 traps (of which 4 are reserved) and 54 interrupts. Each interrupt is prioritized based on a user assigned priority between 1 and 7 (1 being the lowest priority and 7 being the highest) in conjunction with a predetermined 'natural order'. Traps have fixed priorities, ranging from 8 to 15.

2.2 Programmer's Model

The programmer's model is shown in Figure 2-1 and consists of 16x16-bit working registers (W0 through W15), 2x40-bit accumulators (AccA and AccB), STATUS register (SR), Data Table Page register (TBLPAG), Program Space Visibility Page register (PSVPAG), DO and REPEAT registers (DOSTART, DOEND, DCOUNT and RCOUNT), and Program Counter (PC). The working registers can act as data, address or offset registers. All registers are memory mapped. W0 acts as the W register for file register addressing.

Some of these registers have a shadow register associated with each of them, as shown in Figure 2-1. The shadow register is used as a temporary holding register and can transfer its contents to or from its host register upon the occurrence of an event. None of the shadow registers are accessible directly. The following rules apply for transfer of registers into and out of shadows.

- `PUSH.S` and `POP.S`
  W0, W1, W2, W3, SR (DC, N, OV, Z and C bits only) are transferred.
- `DO` instruction
  DOSTART, DOEND, DCOUNT shadows are pushed on loop start, and popped on loop end.

When a byte operation is performed on a working register, only the Least Significant Byte of the target register is affected. However, a benefit of memory mapped working registers is that both the Least and Most Significant Bytes can be manipulated through byte wide data memory space accesses.

2.2.1 SOFTWARE STACK POINTER/ FRAME POINTER

The dsPIC® devices contain a software stack. W15 is the dedicated software stack pointer (SP), and will be automatically modified by exception processing and subroutine calls and returns. However, W15 can be referenced by any instruction in the same manner as all other W registers. This simplifies the reading, writing and manipulation of the stack pointer (e.g., creating stack frames).

Note:	In order to protect against misaligned stack accesses, W15<0> is always clear.

W15 is initialized to 0x0800 during a Reset. The user may reprogram the SP during initialization to any location within data space.

W14 has been dedicated as a stack frame pointer as defined by the `LNK` and `ULNK` instructions. However, W14 can be referenced by any instruction in the same manner as all other W registers.

2.2.2 STATUS REGISTER

The dsPIC core has a 16-bit status register (SR), the LS Byte of which is referred to as the SR Low Byte (SRL) and the MS Byte as the SR High Byte (SRH). See Figure 2-1 for SR layout.

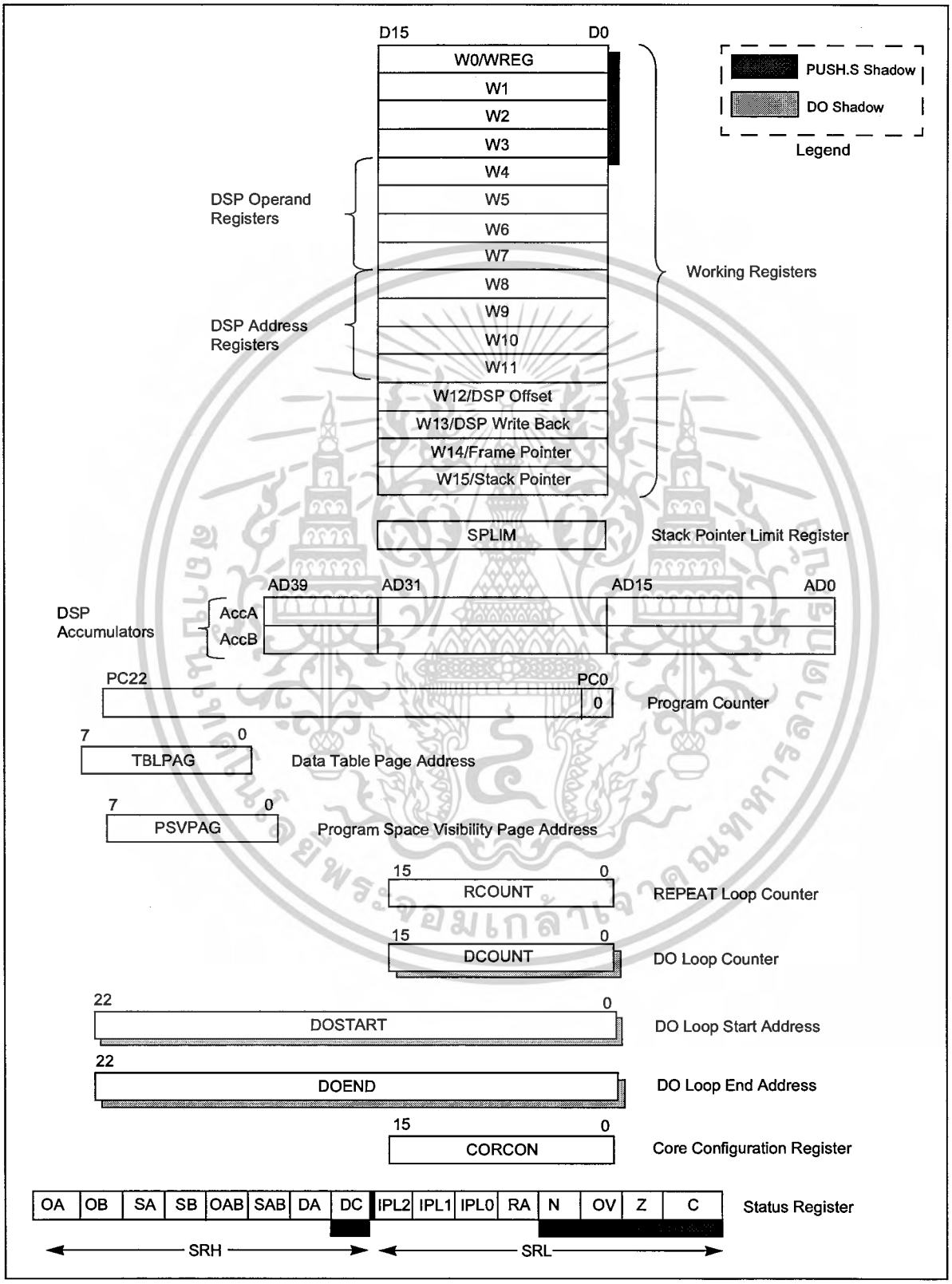
SRL contains all the MCU ALU operation status flags (including the Z bit), as well as the CPU Interrupt Priority Level status bits, IPL<2:0>, and the REPEAT active status bit, RA. During exception processing, SRL is concatenated with the MS Byte of the PC to form a complete word value which is then stacked.

The upper byte of the STATUS register contains the DSP Adder/Subtracter status bits, the DO Loop Active bit (DA) and the Digit Carry (DC) status bit.

2.2.3 PROGRAM COUNTER

The Program Counter is 23 bits wide. Bit 0 is always clear. Therefore, the PC can address up to 4M instruction words.

FIGURE 2-1: PROGRAMMER'S MODEL



2.3 Divide Support

The dsPIC devices feature a 16/16-bit signed fractional divide operation, as well as 32/16-bit and 16/16-bit signed and unsigned integer divide operations, in the form of single instruction iterative divides. The following instructions and data sizes are supported:

1. `DIVF` – 16/16 signed fractional divide
2. `DIV.sd` – 32/16 signed divide
3. `DIV.ud` – 32/16 unsigned divide
4. `DIV.sw` – 16/16 signed divide
5. `DIV.uw` – 16/16 unsigned divide

The 16/16 divides are similar to the 32/16 (same number of iterations), but the dividend is either zero-extended or sign-extended during the first iteration.

The divide instructions must be executed within a REPEAT loop. Any other form of execution (e.g. a series of discrete divide instructions) will not function correctly because the instruction flow depends on RCOUNT. The divide instruction does not automatically set up the RCOUNT value, and it must, therefore, be explicitly and correctly specified in the `REPEAT` instruction, as shown in Table 2-1 (`REPEAT` will execute the target instruction {operand value+1} times). The REPEAT loop count must be set up for 18 iterations of the `DIV/DIVF` instruction. Thus, a complete divide operation requires 19 cycles.

Note: The Divide flow is interruptible. However, the user needs to save the context as appropriate.

TABLE 2-1: DIVIDE INSTRUCTIONS

Instruction	Function
`DIVF`	Signed fractional divide: Wm/Wn → W0; Rem → W1
`DIV.sd`	Signed divide: (Wm+1:Wm)/Wn → W0; Rem → W1
`DIV.sw (or DIV.s)`	Signed divide: Wm/Wn → W0; Rem → W1
`DIV.ud`	Unsigned divide: (Wm+1:Wm)/Wn → W0; Rem → W1
`DIV.uw (or DIV.u)`	Unsigned divide: Wm/Wn → W0; Rem → W1

2.4 DSP Engine

The DSP engine consists of a high speed 17-bit x 17-bit multiplier, a barrel shifter, and a 40-bit adder/subtractor (with two target accumulators, round and saturation logic).

The DSP engine also has the capability to perform inherent accumulator-to-accumulator operations, which require no additional data. These instructions are ADD, SUB and NEG.

The DSP engine has various options selected through various bits in the CPU Core Configuration Register (CORCON), as listed below:

1. Fractional or integer DSP multiply (IF).
2. Signed or unsigned DSP multiply (US).
3. Conventional or convergent rounding (RND).
4. Automatic saturation on/off for AccA (SATA).
5. Automatic saturation on/off for AccB (SATB).
6. Automatic saturation on/off for writes to data memory (SATDW).
7. Accumulator Saturation mode selection (ACCSAT).

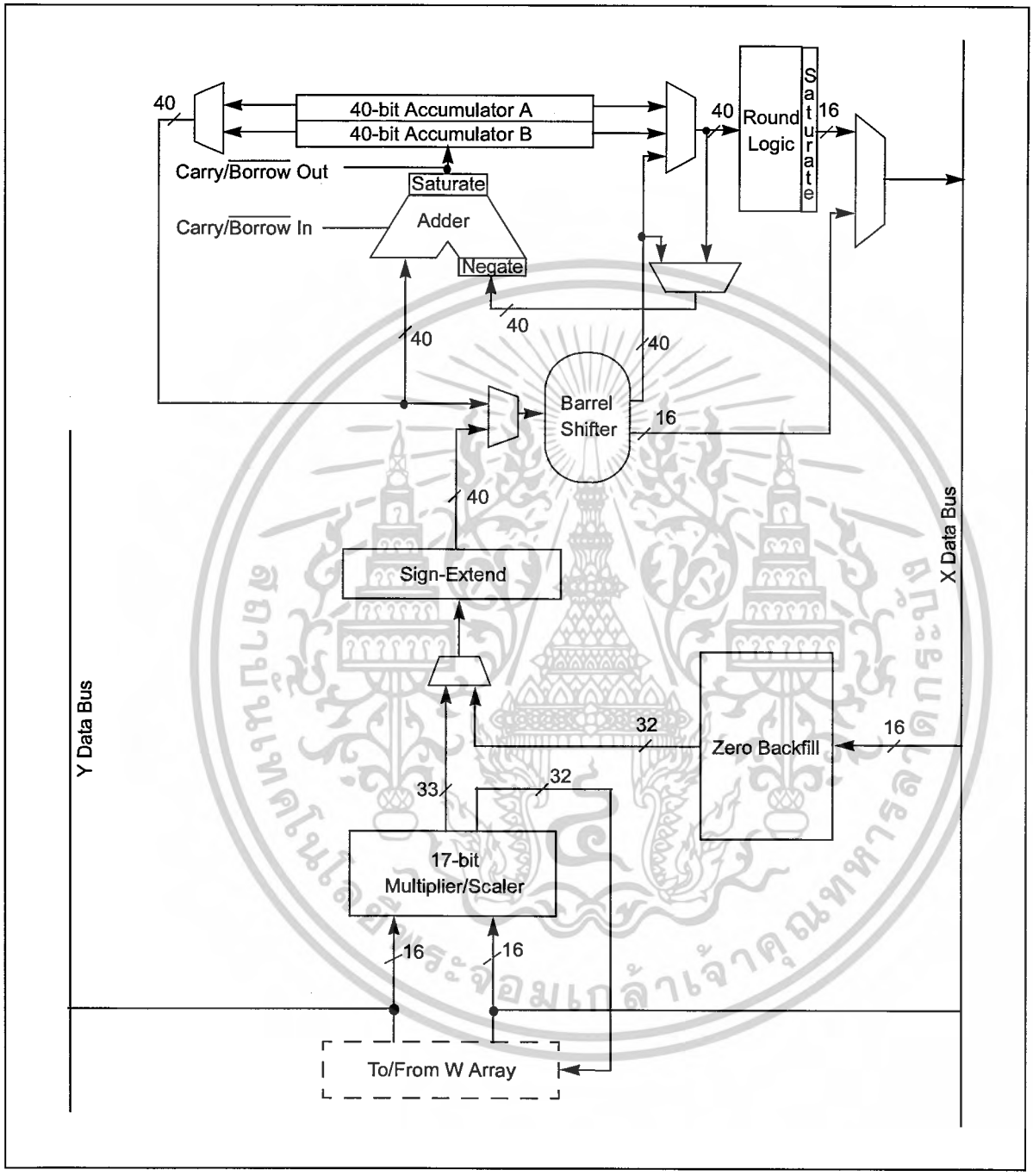
Note: For CORCON layout, see Table 4-2.

A block diagram of the DSP engine is shown in Figure 2-2.

TABLE 2-2: DSP INSTRUCTION SUMMARY

Instruction	Algebraic Operation	ACC WB?
CLR	A = 0	Yes
ED	$A = (x - y)^2$	No
EDAC	$A = A + (x - y)^2$	No
MAC	A = A + (x * y)	Yes
MAC	$A = A + x^2$	No
MOVSAC	No change in A	Yes
MPY	A = x * y	No
MPY.N	A = – x * y	No
MSC	A = A – x * y	Yes

FIGURE 2-2: DSP ENGINE BLOCK DIAGRAM



2.4.1 MULTIPLIER

The 17x17-bit multiplier is capable of signed or unsigned operation and can multiplex its output using a scaler to support either 1.31 fractional (Q31) or 32-bit integer results. Unsigned operands are zero-extended into the 17th bit of the multiplier input value. Signed operands are sign-extended into the 17th bit of the multiplier input value. The output of the 17x17-bit multiplier/ scaler is a 33-bit value, which is sign-extended to 40 bits. Integer data is inherently represented as a signed two's complement value, where the MSB is defined as a sign bit. Generally speaking, the range of an N-bit two's complement integer is -2^{N-1} to $2^{N-1} - 1$. For a 16-bit integer, the data range is -32768 (0x8000) to 32767 (0x7FFF), including 0. For a 32-bit integer, the data range is -2,147,483,648 (0x8000 0000) to 2,147,483,645 (0x7FFF FFFF).

When the multiplier is configured for fractional multiplication, the data is represented as a two's complement fraction, where the MSB is defined as a sign bit and the radix point is implied to lie just after the sign bit (QX format). The range of an N-bit two's complement fraction with this implied radix point is -1.0 to $(1-2^{1-N})$. For a 16-bit fraction, the Q15 data range is -1.0 (0x8000) to 0.999969482 (0x7FFF), including 0 and has a precision of 3.01518×10^{-5}. In Fractional mode, a 16x16 multiply operation generates a 1.31 product, which has a precision of 4.65661×10^{-10}.

The same multiplier is used to support the MCU multiply instructions, which include integer 16-bit signed, unsigned and mixed sign multiplies.

The MUL instruction may be directed to use byte or word sized operands. Byte operands will direct a 16-bit result, and word operands will direct a 32-bit result to the specified register(s) in the W array.

2.4.2 DATA ACCUMULATORS AND ADDER/SUBTRACTOR

The data accumulator consists of a 40-bit adder/ subtractor with automatic sign extension logic. It can select one of two accumulators (A or B) as its pre-accumulation source and post-accumulation destination. For the ADD and LAC instructions, the data to be accumulated or loaded can be optionally scaled via the barrel shifter, prior to accumulation.

2.4.2.1 Adder/Subtractor, Overflow and Saturation

The adder/subtractor is a 40-bit adder with an optional zero input into one side and either true or complement data into the other input. In the case of addition, the carry/borrow input is active high and the other input is true data (not complemented), whereas in the case of subtraction, the carry/borrow input is active low and the other input is complemented. The adder/subtractor generates overflow status bits SA/SB and OA/OB, which are latched and reflected in the status register.

- Overflow from bit 39: this is a catastrophic overflow in which the sign of the accumulator is destroyed.
- Overflow into guard bits 32 through 39: this is a recoverable overflow. This bit is set whenever all the guard bits are not identical to each other.

The adder has an additional saturation block which controls accumulator data saturation, if selected. It uses the result of the adder, the overflow status bits described above, and the SATA/B (CORCON<7:6>) and ACCSAT (CORCON<4>) mode control bits to determine when and to what value to saturate.

Six status register bits have been provided to support saturation and overflow; they are:

1. OA:
   AccA overflowed into guard bits
2. OB:
   AccB overflowed into guard bits
3. SA:
   AccA saturated (bit 31 overflow and saturation)
   or
   AccA overflowed into guard bits and saturated (bit 39 overflow and saturation)
4. SB:
   AccB saturated (bit 31 overflow and saturation)
   or
   AccB overflowed into guard bits and saturated (bit 39 overflow and saturation)
5. OAB:
   Logical OR of OA and OB
6. SAB:
   Logical OR of SA and SB

The OA and OB bits are modified each time data passes through the adder/Subtractor. When set, they indicate that the most recent operation has overflowed into the accumulator guard bits (bits 32 through 39). The OA and OB bits can also optionally generate an arithmetic warning trap when set and the corresponding overflow trap flag enable bit (OVATEN, OVBTEN) in the INTCON1 register (refer to Section 5.0) is set. This allows the user to take immediate action, for example, to correct system gain.

The SA and SB bits are modified each time data passes through the adder/subtractor, but can only be cleared by the user. When set, they indicate that the accumulator has overflowed its maximum range (bit 31 for 32-bit saturation, or bit 39 for 40-bit saturation) and will be saturated (if saturation is enabled). When saturation is not enabled, SA and SB default to bit 39 overflow and thus indicate that a catastrophic overflow has occurred. If the COVTE bit in the INTCON1 register is set, SA and SB bits will generate an arithmetic warning trap when saturation is disabled.

The overflow and saturation status bits can optionally be viewed in the Status Register (SR) as the logical OR of OA and OB (in bit OAB) and the logical OR of SA and SB (in bit SAB). This allows programmers to check one bit in the Status Register to determine if either accumulator has overflowed, or one bit to determine if either accumulator has saturated. This would be useful for complex number arithmetic which typically uses both the accumulators.

The device supports three Saturation and Overflow modes.

1. Bit 39 Overflow and Saturation:
   When bit 39 overflow and saturation occurs, the saturation logic loads the maximally positive 9.31 (0x7FFFFFFFFF) or maximally negative 9.31 value (0x8000000000) into the target accumulator. The SA or SB bit is set and remains set until cleared by the user. This is referred to as 'super saturation' and provides protection against erroneous data or unexpected algorithm problems (e.g., gain calculations).
2. Bit 31 Overflow and Saturation:
   When bit 31 overflow and saturation occurs, the saturation logic then loads the maximally positive 1.31 value (0x007FFFFFFF) or maximally negative 1.31 value (0x0080000000) into the target accumulator. The SA or SB bit is set and remains set until cleared by the user. When this Saturation mode is in effect, the guard bits are not used (so the OA, OB or OAB bits are never set).
3. Bit 39 Catastrophic Overflow
   The bit 39 overflow status bit from the adder is used to set the SA or SB bit, which remain set until cleared by the user. No saturation operation is performed and the accumulator is allowed to overflow (destroying its sign). If the COVTE bit in the INTCON1 register is set, a catastrophic overflow can initiate a trap exception.

2.4.2.2 Accumulator 'Write Back'

The MAC class of instructions (with the exception of MPY, MPY.N, ED and EDAC) can optionally write a rounded version of the high word (bits 31 through 16) of the accumulator that is not targeted by the instruction into data space memory. The write is performed across the X bus into combined X and Y address space. The following addressing modes are supported:

1. W13, Register Direct:
   The rounded contents of the non-target accumulator are written into W13 as a 1.15 fraction.
2. [W13]+=2, Register Indirect with Post-Increment:
   The rounded contents of the non-target accumulator are written into the address pointed to by W13 as a 1.15 fraction. W13 is then incremented by 2 (for a word write).

2.4.2.3 Round Logic

The round logic is a combinational block, which performs a conventional (biased) or convergent (unbiased) round function during an accumulator write (store). The Round mode is determined by the state of the RND bit in the CORCON register. It generates a 16-bit, 1.15 data value which is passed to the data space write saturation logic. If rounding is not indicated by the instruction, a truncated 1.15 data value is stored and the LS Word is simply discarded.

Conventional rounding takes bit 15 of the accumulator, zero-extends it and adds it to the ACCxH word (bits 16 through 31 of the accumulator). If the ACCxL word (bits 0 through 15 of the accumulator) is between 0x8000 and 0xFFFF (0x8000 included), ACCxH is incremented. If ACCxL is between 0x0000 and 0x7FFF, ACCxH is left unchanged. A consequence of this algorithm is that over a succession of random rounding operations, the value will tend to be biased slightly positive.

Convergent (or unbiased) rounding operates in the same manner as conventional rounding, except when ACCxL equals 0x8000. If this is the case, the LS bit (bit 16 of the accumulator) of ACCxH is examined. If it is '1', ACCxH is incremented. If it is '0', ACCxH is not modified. Assuming that bit 16 is effectively random in nature, this scheme will remove any rounding bias that may accumulate.

The SAC and SAC.R instructions store either a truncated (SAC) or rounded (SAC.R) version of the contents of the target accumulator to data memory, via the X bus (subject to data saturation, see Section 2.4.2.4). Note that for the MAC class of instructions, the accumulator write back operation will function in the same manner, addressing combined MCU (X and Y) data space though the X bus. For this class of instructions, the data is always subject to rounding.

2.4.2.4 Data Space Write Saturation

In addition to adder/subtractor saturation, writes to data space may also be saturated, but without affecting the contents of the source accumulator. The data space write saturation logic block accepts a 16-bit, 1.15 fractional value from the round logic block as its input, together with overflow status from the original source (accumulator) and the 16-bit round adder. These are combined and used to select the appropriate 1.15 fractional value as output to write to data space memory.

If the SATDW bit in the CORCON register is set, data (after rounding or truncation) is tested for overflow and adjusted accordingly. For input data greater than 0x007FFF, data written to memory is forced to the maximum positive 1.15 value, 0x7FFF. For input data less than 0xFF8000, data written to memory is forced to the maximum negative 1.15 value, 0x8000. The MS bit of the source (bit 39) is used to determine the sign of the operand being tested.

If the SATDW bit in the CORCON register is not set, the input data is always passed through unmodified under all conditions.

2.4.3 BARREL SHIFTER

The barrel shifter is capable of performing up to 15-bit arithmetic or logic right shifts, or up to 16-bit left shifts in a single cycle. The source can be either of the two DSP accumulators or the X bus (to support multi-bit shifts of register or memory data).

The shifter requires a signed binary value to determine both the magnitude (number of bits) and direction of the shift operation. A positive value will shift the operand right. A negative value will shift the operand left. A value of 0 will not modify the operand.

The barrel shifter is 40 bits wide, thereby obtaining a 40-bit result for DSP shift operations and a 16-bit result for MCU shift operations. Data from the X bus is presented to the barrel shifter between bit positions 16 to 31 for right shifts, and bit positions 0 to 15 for left shifts.

NOTES:

3.0 MEMORY ORGANIZATION

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

3.1 Program Address Space

The program address space is 4M instruction words. It is addressable by a 24-bit value from either the 23-bit PC, table instruction Effective Address (EA), or data space EA, when program space is mapped into data space, as defined by Table 3-1. Note that the program space address is incremented by two between successive program words, in order to provide compatibility with data space addressing.

User program space access is restricted to the lower 4M instruction word address range (0x000000 to 0x7FFFFE), for all accesses other than TBLRD/TBLWT, which use TBLPAG<7> to determine user or configuration space access. In Table 3-1, Read/Write instructions, bit 23 allows access to the Device ID, the User ID and the configuration bits. Otherwise, bit 23 is always clear.

Note: The address map shown in Figure 3-1 is conceptual, and the actual memory configuration may vary across individual devices depending on available memory.

FIGURE 3-1: PROGRAM SPACE MEMORY MAP FOR dsPIC30F2010

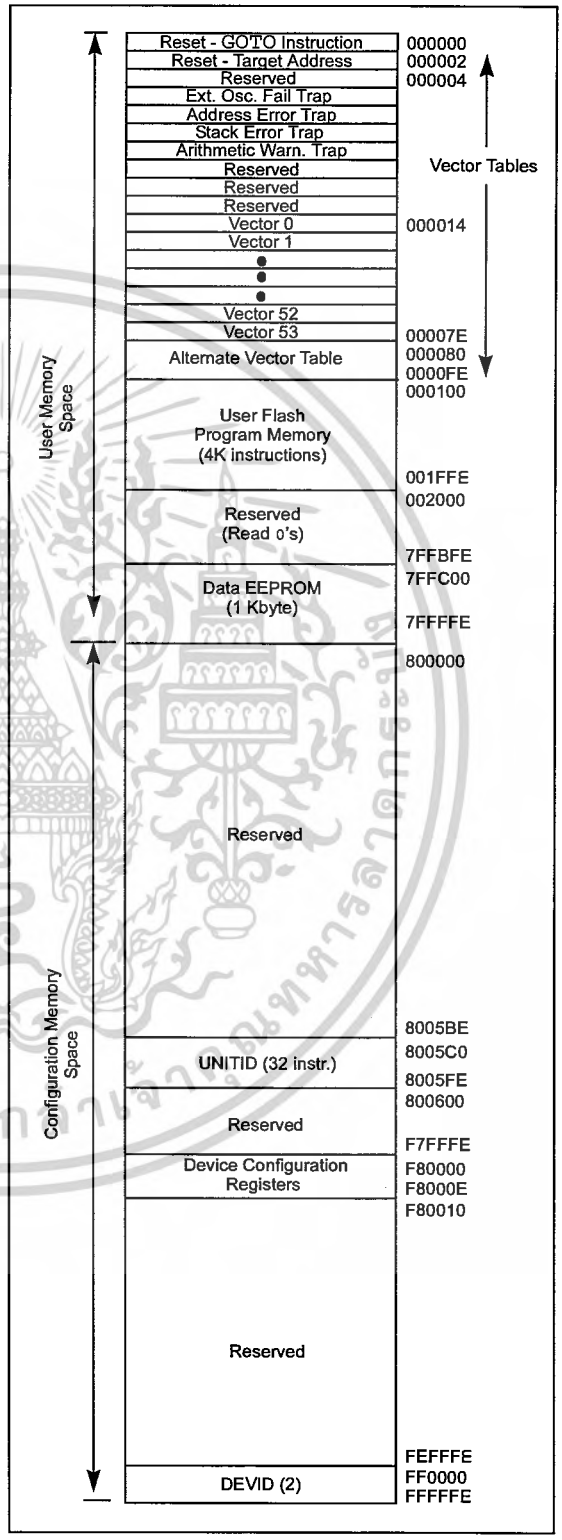
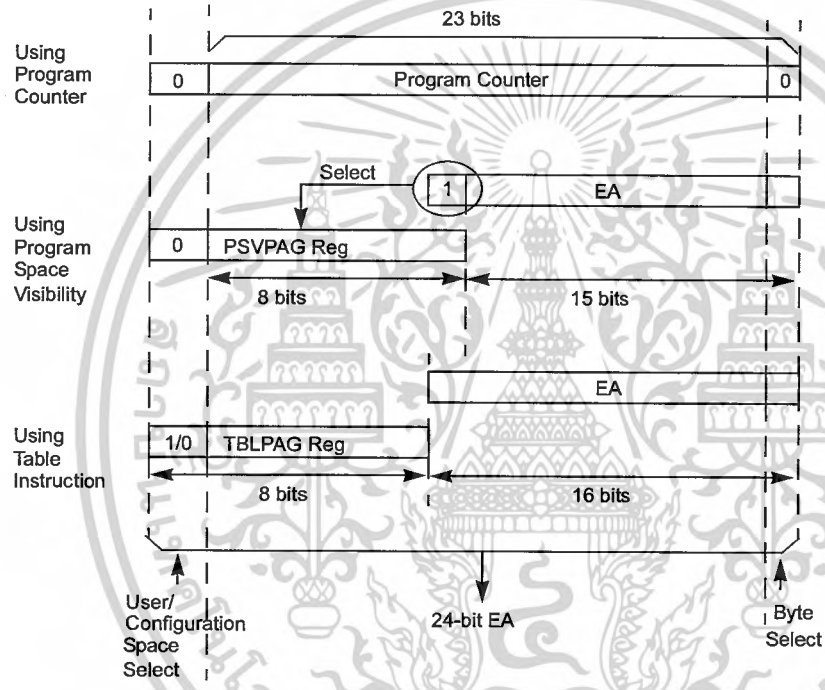


TABLE 3-1: PROGRAM SPACE ADDRESS CONSTRUCTION

Access Type	Access Space	Program Space Address				
		<23>	<22:16>	<15>	<14:1>	<0>
Instruction Access	User	0	PC<22:1>			0
TBLRD/TBLWT	User (TBLPAG<7> = 0)	TBLPAG<7:0>		Data EA <15:0>		
TBLRD/TBLWT	Configuration (TBLPAG<7> = 1)	TBLPAG<7:0>		Data EA <15:0>		
Program Space Visibility	User	0	PSVPAG<7:0>		Data EA <14:0>	

FIGURE 3-2: DATA ACCESS FROM PROGRAM SPACE ADDRESS GENERATION



Note: Program Space Visibility cannot be used to access bits <23:16> of a word in program memory.

3.1.1 DATA ACCESS FROM PROGRAM MEMORY USING TABLE INSTRUCTIONS

This architecture fetches 24-bit wide program memory. Consequently, instructions are always aligned. However, as the architecture is modified Harvard, data can also be present in program space.

There are two methods by which program space can be accessed; via special table instructions, or through the remapping of a 16K word program space page into the upper half of data space (see Section 3.1.2). The TBLRDL and TBLWTL instructions offer a direct method of reading or writing the LS Word of any address within program space, without going through data space. The TBLRDH and TBLWTH instructions are the only method whereby the upper 8 bits of a program space word can be accessed as data.

The PC is incremented by two for each successive 24-bit program word. This allows program memory addresses to directly map to data space addresses. Program memory can thus be regarded as two 16-bit word wide address spaces, residing side by side, each with the same address range. TBLRDL and TBLWTL access the space which contains the LS Data Word, and TBLRDH and TBLWTH access the space which contains the MS Data Byte.

Figure 3-2 shows how the EA is created for table operations and data space accesses (PSV = 1). Here, P<23:0> refers to a program space word, whereas D<15:0> refers to a data space word.

A set of Table Instructions are provided to move byte or word sized data to and from program space.

1. **TBLRDL:** Table Read Low
   Word: Read the LS Word of the program address;
   P<15:0> maps to D<15:0>.
   Byte: Read one of the LS Bytes of the program address;
   P<7:0> maps to the destination byte when byte select = 0;
   P<15:8> maps to the destination byte when byte select = 1.
2. **TBLWTL:** Table Write Low (refer to Section 6.0 for details on Flash Programming).
3. **TBLRDH:** Table Read High
   Word: Read the MS Word of the program address;
   P<23:16> maps to D<7:0>; D<15:8> always be = 0.
   Byte: Read one of the MS Bytes of the program address;
   P<23:16> maps to the destination byte when byte select = 0;
   The destination byte will always be = 0 when byte select = 1.
4. **TBLWTH:** Table Write High (refer to Section 6.0 for details on Flash Programming).

FIGURE 3-3: PROGRAM DATA TABLE ACCESS (LS WORD)

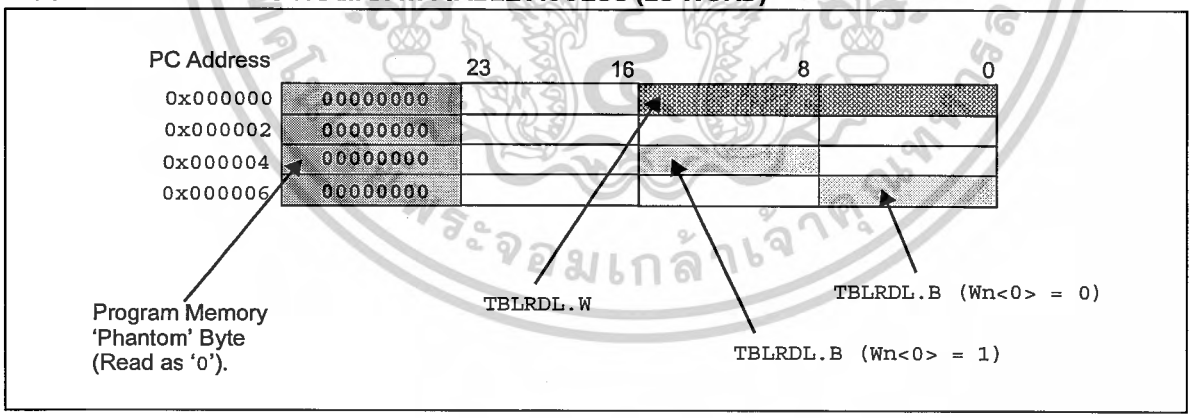
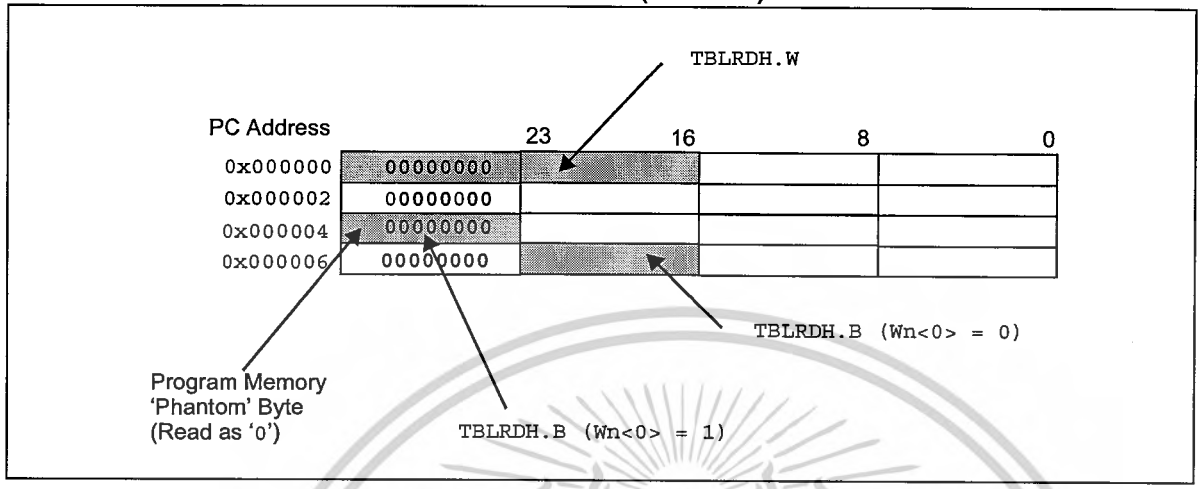


FIGURE 3-4: PROGRAM DATA TABLE ACCESS (MS BYTE)



3.1.2 DATA ACCESS FROM PROGRAM MEMORY USING PROGRAM SPACE VISIBILITY

The upper 32 Kbytes of data space may optionally be mapped into any 16K word program space page. This provides transparent access of stored constant data from X data space, without the need to use special instructions (i.e., `TBLRDL/H`, `TBLWTL/H` instructions).

Program space access through the data space occurs if the MS bit of the data space EA is set and program space visibility is enabled, by setting the PSV bit in the Core Control register (CORCON). The functions of CORCON are discussed in Section 2.4, DSP Engine.

Data accesses to this area add an additional cycle to the instruction being executed, since two program memory fetches are required.

Note that the upper half of addressable data space is always part of the X data space. Therefore, when a DSP operation uses program space mapping to access this memory region, Y data space should typically contain state (variable) data for DSP operations, whereas X data space should typically contain coefficient (constant) data.

Although each data space address, 0x8000 and higher, maps directly into a corresponding program memory address (see Figure 3-5), only the lower 16-bits of the 24-bit program word are used to contain the data. The upper 8 bits should be programmed to force an illegal instruction to maintain machine robustness. Refer to the Programmer's Reference Manual (DS70030) for details on instruction encoding.

Note that by incrementing the PC by 2 for each program memory word, the LS 15 bits of data space addresses directly map to the LS 15 bits in the corresponding program space addresses. The remaining bits are provided by the Program Space Visibility Page register, PSVPAG<7:0>, as shown in Figure 3-5.

Note:	PSV access is temporarily disabled during Table Reads/Writes.

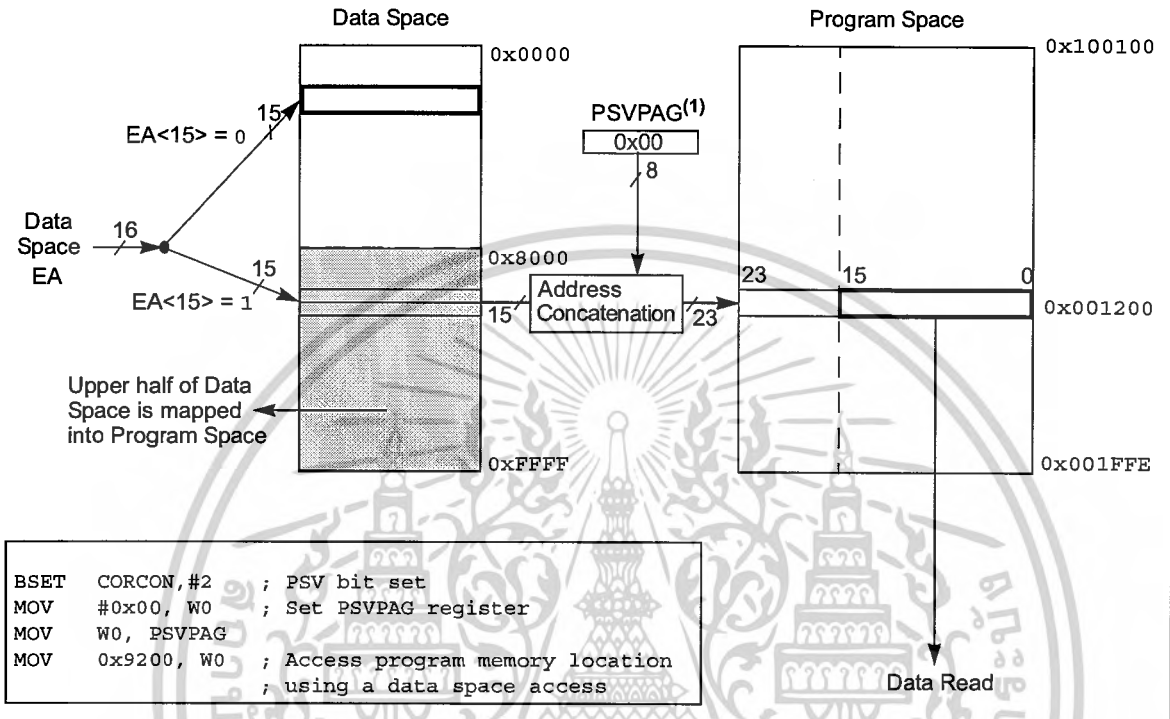
For instructions that use PSV which are executed outside a REPEAT loop:

- The following instructions will require one instruction cycle in addition to the specified execution time:
  - `MAC` class of instructions with data operand pre-fetch
  - `MOV` instructions
  - `MOV.D` instructions
- All other instructions will require two instruction cycles in addition to the specified execution time of the instruction.

For instructions that use PSV which are executed inside a REPEAT loop:

- The following instances will require two instruction cycles in addition to the specified execution time of the instruction:
  - Execution in the first iteration
  - Execution in the last iteration
  - Execution prior to exiting the loop due to an interrupt
  - Execution upon re-entering the loop after an interrupt is serviced
- Any other iteration of the REPEAT loop will allow the instruction, accessing data using PSV, to execute in a single cycle.

FIGURE 3-5: DATA SPACE WINDOW INTO PROGRAM SPACE OPERATION



Note: PSVPAG is an 8-bit register, containing bits <22:15> of the program space address (i.e., it defines the page in program space to which the upper half of data space is being mapped).

3.2 Data Address Space

The core has two data spaces. The data spaces can be considered either separate (for some DSP instructions), or as one unified linear address range (for MCU instructions). The data spaces are accessed using two Address Generation Units (AGUs) and separate data paths.

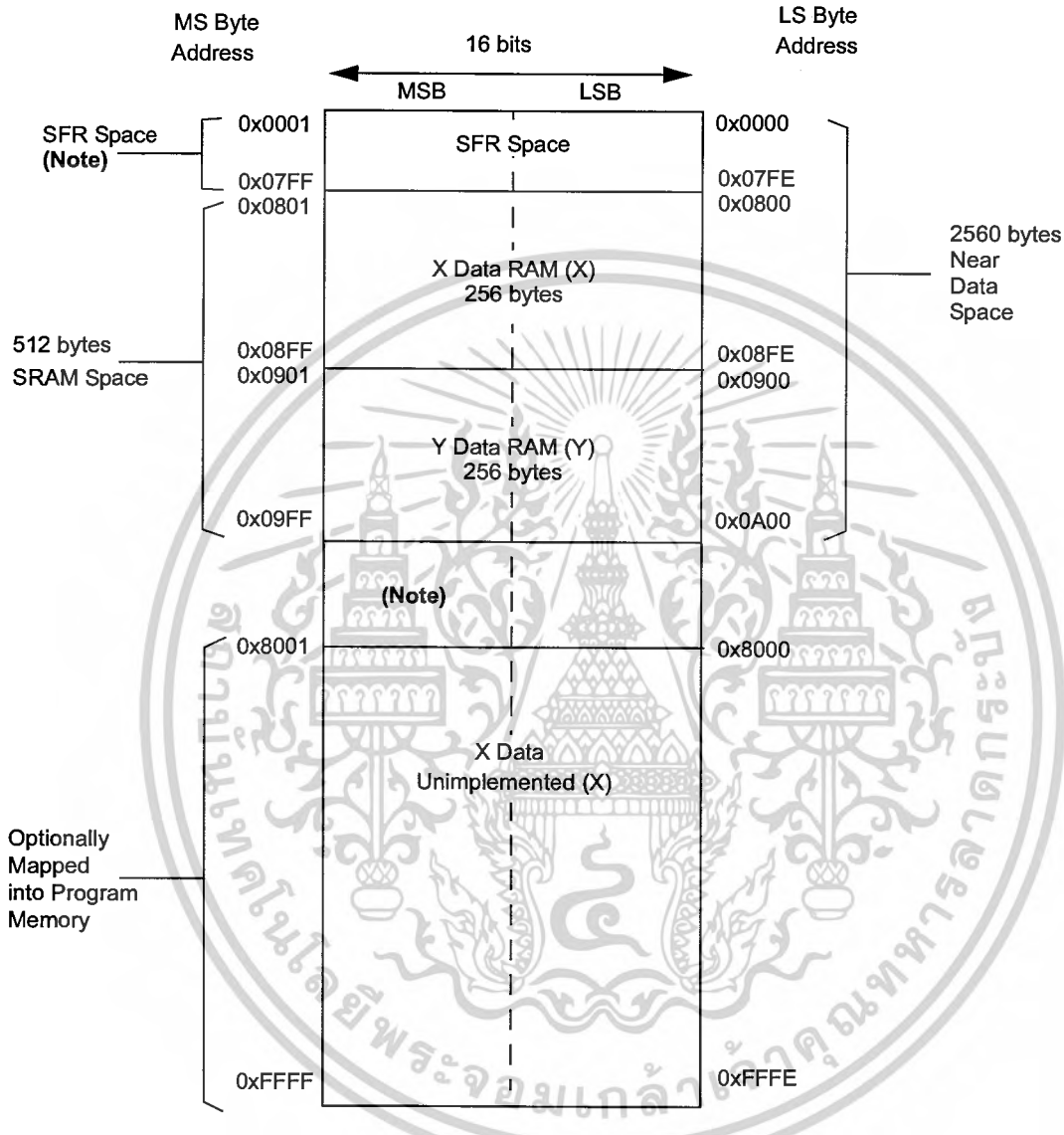
3.2.1 DATA SPACE MEMORY MAP

The data space memory is split into two blocks, X and Y data space. A key element of this architecture is that Y space is a subset of X space, and is fully contained within X space. In order to provide an apparent linear addressing space, X and Y spaces have contiguous addresses.

When executing any instruction other than one of the MAC class of instructions, the X block consists of the 256 byte data address space (including all Y addresses). When executing one of the MAC class of instructions, the X block consists of the 256 bytes data address space excluding the Y address block (for data reads only). In other words, all other instructions regard the entire data memory as one composite address space. The MAC class instructions extract the Y address space from data space and address it using EAs sourced from W10 and W11. The remaining X data space is addressed using W8 and W9. Both address spaces are concurrently accessed only with the MAC class instructions.

A data space memory map is shown in Figure 3-6.

FIGURE 3-6: DATA SPACE MEMORY MAP



Note: Unimplemented SFR or SRAM locations read as '0'.

FIGURE 3-7: DATA SPACE FOR MCU AND DSP (MAC CLASS) INSTRUCTIONS