การควบคุมเสถียรภาพของอินเวิร์ทเพนดูลัม

ROTATIONAL INVERTED PENDULUM

โดย

นายธานินทร์ วรรณะ

นายประเสริฐ ซิ้มเจริญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

การควบคุมเสถียรภาพของอินเวิร์ทเพนดูลัม

ROTATIONAL INVERTED PENDULUM

โดย

นายธานินทร์ วรรณะ

นายประเสริฐ ชิ้มเจริญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชาวิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมเสถียรภาพของอินเวิร์ทเพนดูลัม

ROTATIONAL INVERTED PENDULUM

ผู้จัดทำ นายธานินทร์ วรรณะ

นายประเสริฐ ซิ้มเจริญ

.....อาจารย์ที่ปรึกษา

(รองศาสตราจารย์ ดร.จงกล งามวิวิทย์)

.....อาจารย์ที่ปรึกษา

(อาจารย์สุมิตร พนาอุดมทรัพย์)

การควบคุมเสถียรภาพของอินเวิร์ทเพนดูลัม

ROTATIONAL INVERTED PENDULUM

โดย

นายธานินทร์ วรรณะ

รหัสประจำตัวนักศึกษา 37014169

นายประเสริฐ ซิ้มเจริญ

รหัสประจำตัวนักศึกษา 37014239

อาจารย์ที่ปรึกษา

รองศาสตราจารย์ ดร.จงกล งามวิวิทย์

อาจารย์สุมิตร พนาอุดมทรัพย์

บทคัดย่อ

ปริญญานิพนธ์นี้ได้กล่าวถึงวิธีคิดและออกแบบอินเวิร์ทเพนดูลัมที่จะทำให้ก้านลูกตุ้มทรงตัวได้อย่างสมดุลย์ โดยเราเลือกใช้การควบคุมแบบ PID

โครงสร้างทางเครื่องกลโดยใช้ชุดเฟืองส่งแรงไปยังแขนเพื่อควบคุมตำแหน่งของก้านลูกตุ้ม ส่วนผลการทดลองเราได้แสดงในปริญญานิพนธ์ฉบับนี้ด้วย

ABSTRACT

This thesis describes the design of control algorithms inorder to control the vertical balance of rotatoinal inverted pendulum. The PID control is used for control this system.

The gearbox and arm is used inorder to improved the pendulums energy transfer and made it robust.

The experiment results also are shown in this thesis.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดีก็เพราะได้รับความเมตตาจาก รองศาสตราจารย์ ดร.จงกล งามวิวิทย์ และอาจารย์สุมิตร พนาอุดมทรัพย์ อาจารย์ที่ปรึกษาที่ได้ให้ความกรุณาแนะนำแก่ผู้จัดทำตลอดมา พร้อมทั้ง ผู้ช่วยศาสตราจารย์ พรสุข รติโรจน์อนันท์ ผู้ให้อุปกรณ์ในการทำงาน และคณาจารย์ทุกท่านที่ได้ประสิทธิประสาทวิชาความรู้แก่ผู้จัดทำ ผู้จัดทำขอกราบขอบพระคุณเป็นอย่างสูงมา ณ โอกาสนี้ด้วย

ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ ของผู้จัดทำที่ได้อุปการะผู้จัดทำ ทั้งยังเป็นผู้ให้กำลังใจแก่ผู้จัดทำตลอดมา

ขอขอบคุณเพื่อนๆที่ให้ใช้อุปกรณ์ ให้ความรู้ ให้คำปรึกษา และให้กำลังใจแก่ผู้จัดทำมาโดยตลอด

ผู้จัดทำ
นายธานินทร์ วรรณะ
นายประเสริฐ ซิ้มเจริญ

สารบัญ

บทที่1

บทนำ

การควบคุมเสถียรภาพของอินเวิร์ทเพนดูลัมเป็นชุดควบคุมก้านลูกตุ้ม(Pendulum)บนแขน ซึ่งเดิมเป็นระบบที่ไม่เสถียร การที่จะทำระบบให้เสถียรนั้น เราต้องมีการควบคุม

1.1 จุดประสงค์

1. ศึกษาทฤษฎีของการควบคุมเสถียรภาพและทดลองภายใต้การควบคุมแบบ PID

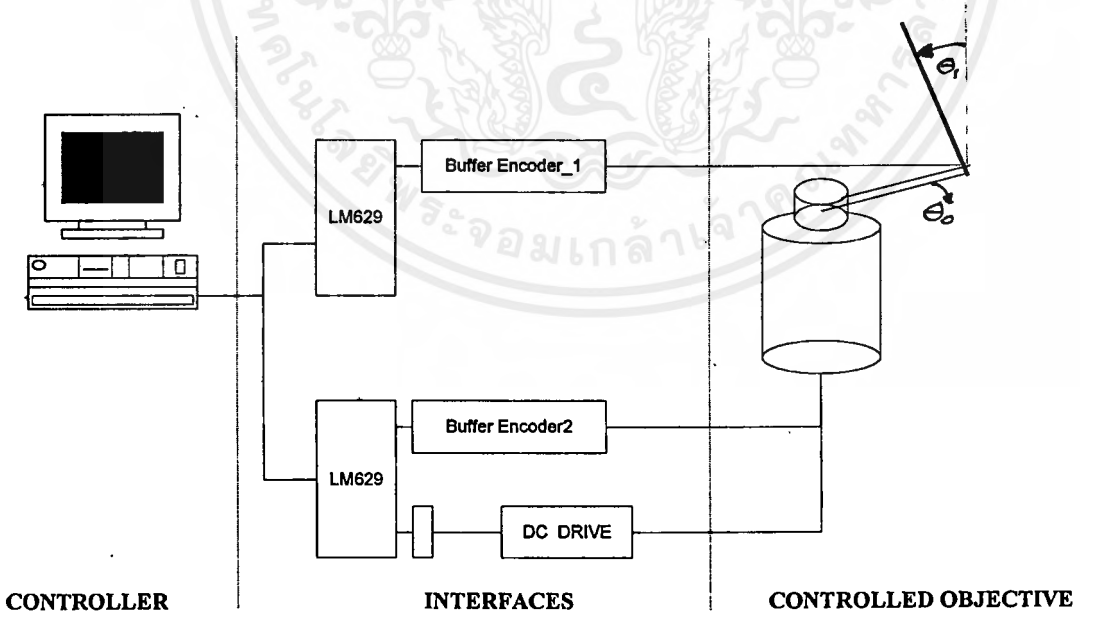
2.นำความรู้ที่ได้รับจากการศึกษาและทดลองไปเป็นพื้นฐานในการศึกษาพร้อมทั้งการประยุกต์ใช้งานในอนาคต

3 ทราบถึงปัญหาที่เกิดขึ้นและแนวทางแก้ไข

1.2 ประเภทของอินเวิร์ทเพนดูลัม

ที่ใช้ทำการศึกษามีสองประเภทใหญ่ๆคือ แบบรถ (cart and stick) แบบประยุกต์ซึ่งแบบประยุกต์ที่ได้ทำการค้นคว้าคือแบบหมุน ซึ่งมีข้อดีคือประหยัดเนื้อที่ และประมาณการคำนวณเป็นเชิงเส้นได้

1.3 โครงสร้างระบบโดยรวม



รูปที่ 1.1 การจัดระบบของควบคุมการตั้งของลูกตุ้ม

1.4 หลักการทั่วไป

ส่วนประกอบทั้งหมดได้แก่ ชุดลูกตุ้ม (rotational inverted pendulum) วงจรติดต่อคอมพิวเตอร์(interface circuit) และชุดควบคุม (controller) เราควบคุมการตั้งของก้านลูกตุ้มด้วยตำแหน่งแขนซึ่งจะยึดติดกับเพลาของมอเตอร์ โดยแกนมอเตอร์จะมี ตัววัดตำแหน่งแบบแสง(optical encoder) ติดอยู่เพื่อวัดมุมของแขน (θ_0)เพลาของมอเตอร์ และแขนจะหมุนได้รอบส่วนตัวมอเตอร์ จะยึดแน่นกับฐาน ทางด้านก้านลูกตุ้ม จะวัดถูกมุม(θ_1)ได้ โดยตัววัดตำแหน่งแบบแสง เช่นเดียวกัน โดยระบบทั้งหมดจะต้องรักษาสภาพสมดุลย์ของก้านลูกตุ้มไว้ ($\theta_1 \Longrightarrow 0$)

บทที่ 2
ส่วนประกอบที่สำคัญของระบบ

ระบบทั้งหมดแบ่งออกเป็น 3 ส่วนได้แก่

1. โครงสร้างทางเครื่องกล
2. วงจรอิเล็กทรอนิคส์และการติดต่อกับเครื่องคอมพิวเตอร์
3. ส่วนควบคุมและประมวลผล

2.1 โครงสร้างทางเครื่องกล

เมื่อเราได้หัวข้อของโครงงาน เราได้เริ่มศึกษารูปแบบต่างๆ ของอินเวิร์ทเพนดูลัม เปรียบเทียบข้อดี ข้อเสีย และความเป็นไปได้ เราได้เลือกอินเวิร์ทเพนดูลัมแบบหมุนได้ ซึ่งข้อดีคือ ชุดอินเวิร์ทเพนดูลัมดังกล่าวสามารถหมุนได้รอบจึงไม่มีการสิ้นสุดของระยะทาง และใช้พื้นที่น้อยในการติดตั้งขณะทำงาน พร้อมทั้งความง่ายของโครงสร้างทางเครื่องกลด้วย

ส่วนอุปกรณ์สำคัญที่ใช้ได้แก่ มอเตอร์กระแสตรง และ เอนโค้ดเดอร์ เราจึงต้องหาอุปกรณ์ดังกล่าว แล้ววัดขนาด น้ำหนัก และรูปแบบต่างๆ แล้วจึงโครงสร้าง เพื่อให้มีความเหมาะสม เป็นระเบียบ แข็งแรง และ สวยงาม

จากการศึกษาคุณสมบัติของโลหะต่างพบว่า อลูมิเนียม มีน้ำหนักเบา ทนทาน ไม่เป็นสนิม และง่ายต่อการทำชิ้นงาน ดังนั้นเราจึงเลือกอลูมิเนียมในการทำโครงสร้างทั้งหมด พร้อมทั้งเลือกน้อตคุณภาพดีแบบน้อตหัวหกเหลี่ยมแข็งพิเศษ เพื่อใช้ในการประกอบชิ้นงานเข้าด้วยกัน

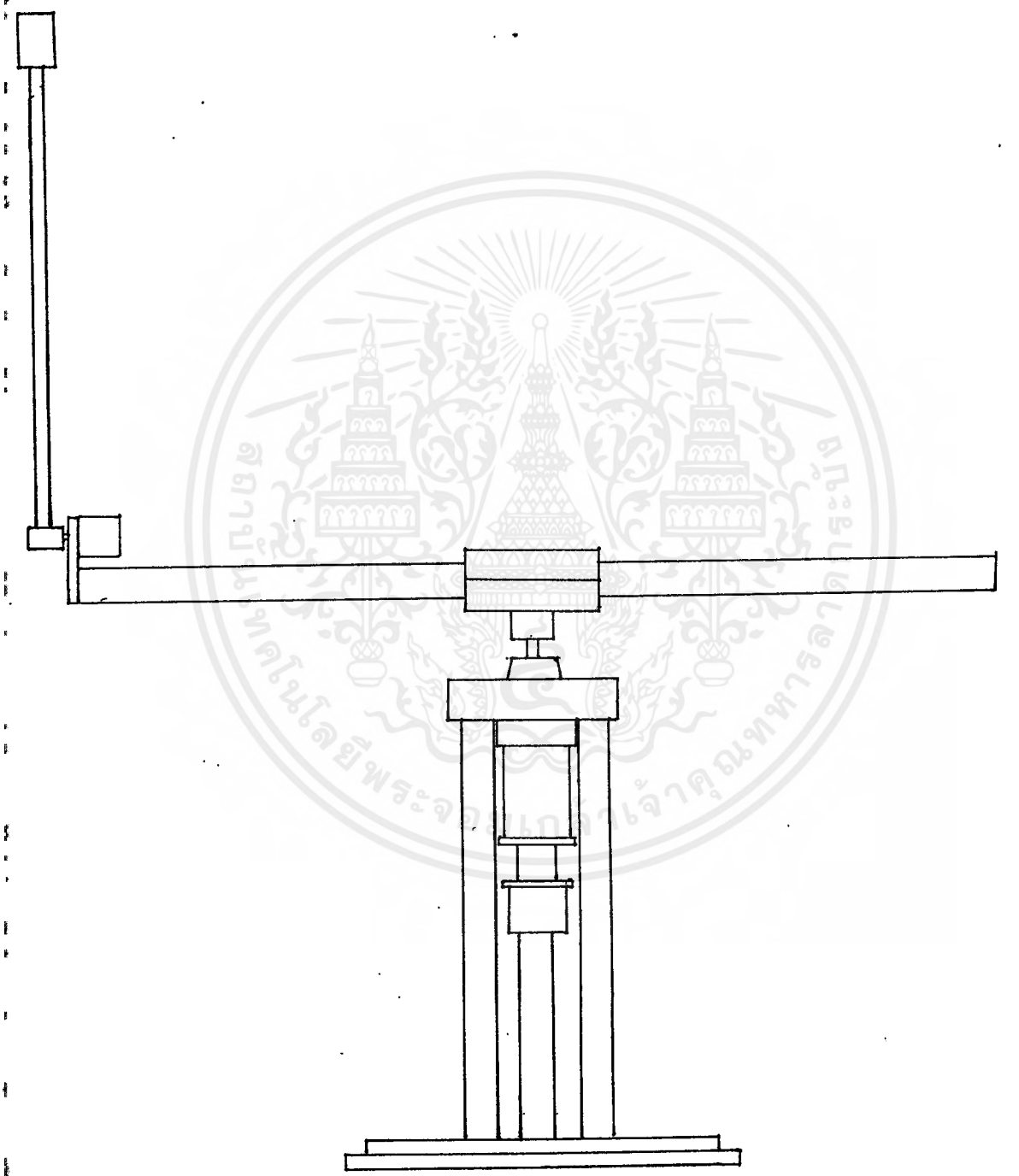
ในการทำและประกอบชิ้นงานเราเน้นที่ความแข็งแรง ความเป็นไปได้ และความสวยงาม จึงเสียเวลาพอสมควรแต่เมื่อชิ้นงานออกมาและประกอบเสร็จก็เป็นที่น่าพอใจ ดังรูปที่ 2.1จะเห็นได้ว่าเราออกแบบความยาวของก้านลูกตุ้ม ให้น้อยกว่าความสูงของฐานทั้งหมด เพื่อป้องกันลูกตุ้มกระแทกกับพื้นซึ่งจะเป็นอันตรายต่อเอนโค้ดเดอร์ตัวที่สองอย่างมาก

ลักษณะและการประกอบอุปกรณ์ (ดูภาคผนวก ก)

ฐาน เราใช้แผ่นอลูมิเนียมขนาด 270 × 270 × 10 มม. และตัดฐานข้างล่างด้วยแผ่นอคิลิกขนาด 300 × 300 × 10 มม. ยึดแผ่นกับเสาโดยน้อต

เสา เป็นอลูมิเนียมตันขนาด ϕ 25 × 320 มม. ทำรูขันน้อตตรงกลางสามเสายึดให้แน่นกับฐานแท่นยึดมอเตอร์

แท่นยึดมอเตอร์ เป็นอลูมิเนียมขนาด ϕ 130 $\times$ 30 มม. ยึดตัวมอเตอร์ให้แข็งแรงแล้วจึงนำมายึดกับเสาอีกครั้งหนึ่งอย่างแน่นหนาโดยส่วนของเพลามอเตอร์จะสูงเหนือแท่นยึดมอเตอร์ขึ้นมา ส่วนที่ปลายมอเตอร์จะมีเอนโค้ดเดอร์ตัวที่สองติดอยู่



รูปที่ 2.1 แสดงโครงสร้างทางเครื่องกล มาตราส่วน 1:5

ตัวยึดเพลามอเตอร์กับแขน ใช้อลูมิเนียมขนาดϕ 95 × 50 มม.โดยยึดเพลาของมอเตอร์แบบบีบรัดจะมีความแข็งแรงมากโดยส่วนนี้สามารถขันปรับน้อตตัวบนเพื่อปรับความยาวของแขนได้

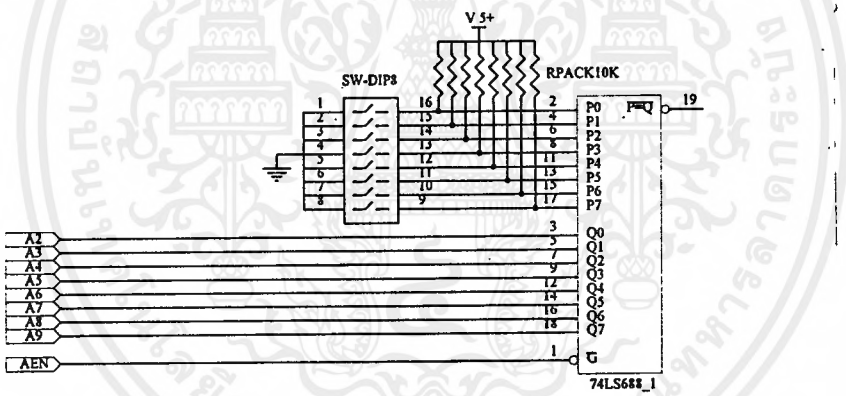
แขน เป็นอลูมิเนียมสี่เหลี่ยมจตุรัสกลวงบางขนาด 25 × 25 × 700 มม. โดยแขนนี้จะมีน้ำหนักเบามากและแข็งแรงจะส่งแรงจากเพลามอเตอร์ไปยังตำแหน่งฐานของก้านลูกตุ้ม

ส่วนยึดฐานของก้านลูกตุ้ม ที่ปลายของแขนจะมียึดแขนกับฐานก้านลูกตุ้มจะมีเอนโค้ดเดอร์ติดอยู่ เพื่อวัดมุมเปลี่ยนแปลง

ก้านลูกตุ้ม เป็นแท่งอลูมิเนียมกลมกลวงขนาด ϕ 10 × 40 มม. น้ำหนักเบาที่ปลายของก้านลูกตุ้มจะมีลูกตุ้มติดอยู่

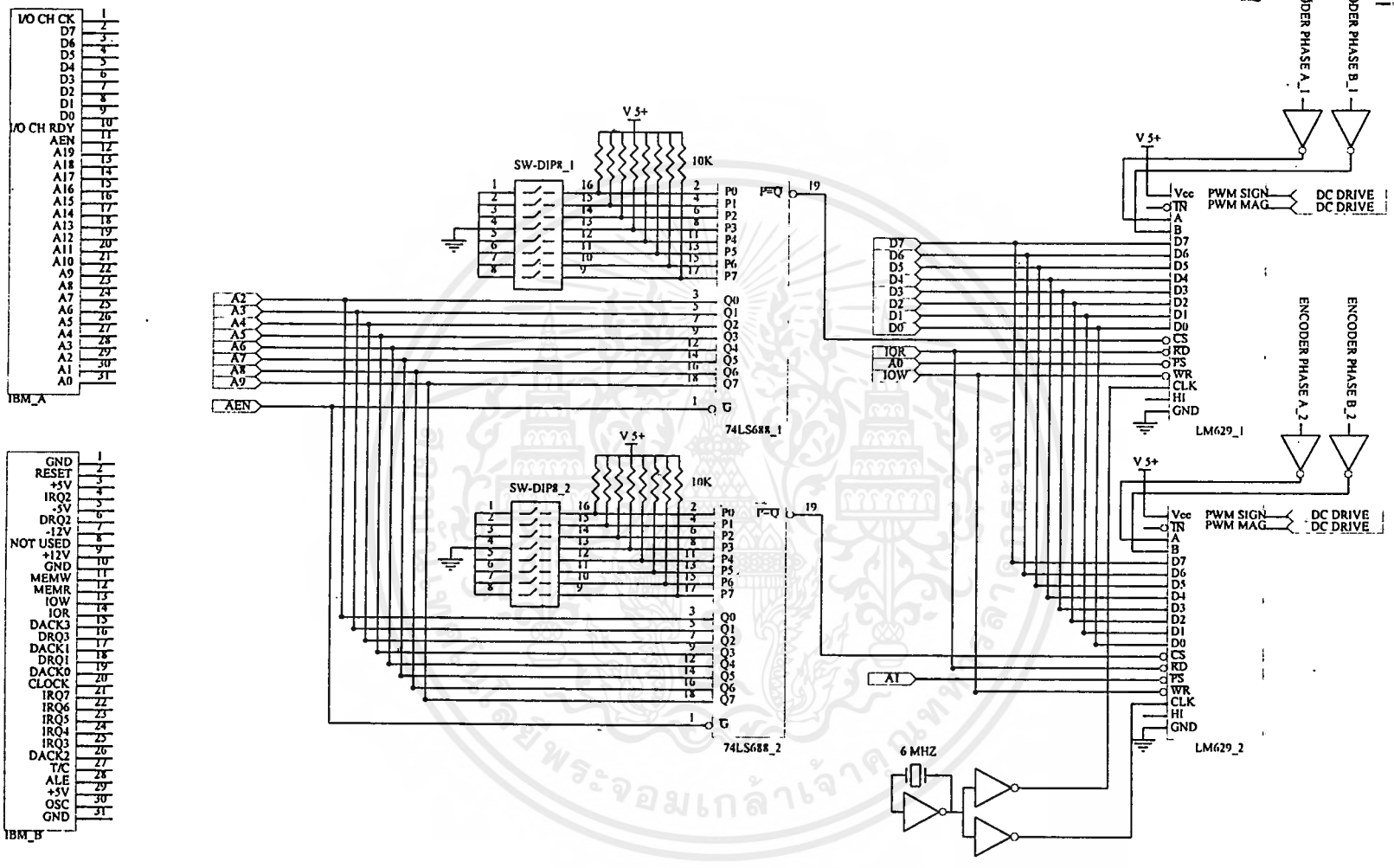
2.2การติดต่อกับเครื่องคอมพิวเตอร์

การติดต่อกับเครื่องคอมพิวเตอร์นั้นใช้การติดต่อพอร์ตแแบบตายตัว(fix decode port)ดังนั้นอุปกรณ์ IC 74LS688 จึงถูกนำมาใช้งานร่วมดิปสวิทช์ในการดีโค้ดพอร์ตดังวงจรที่แสดงในรูปที่ 2.2



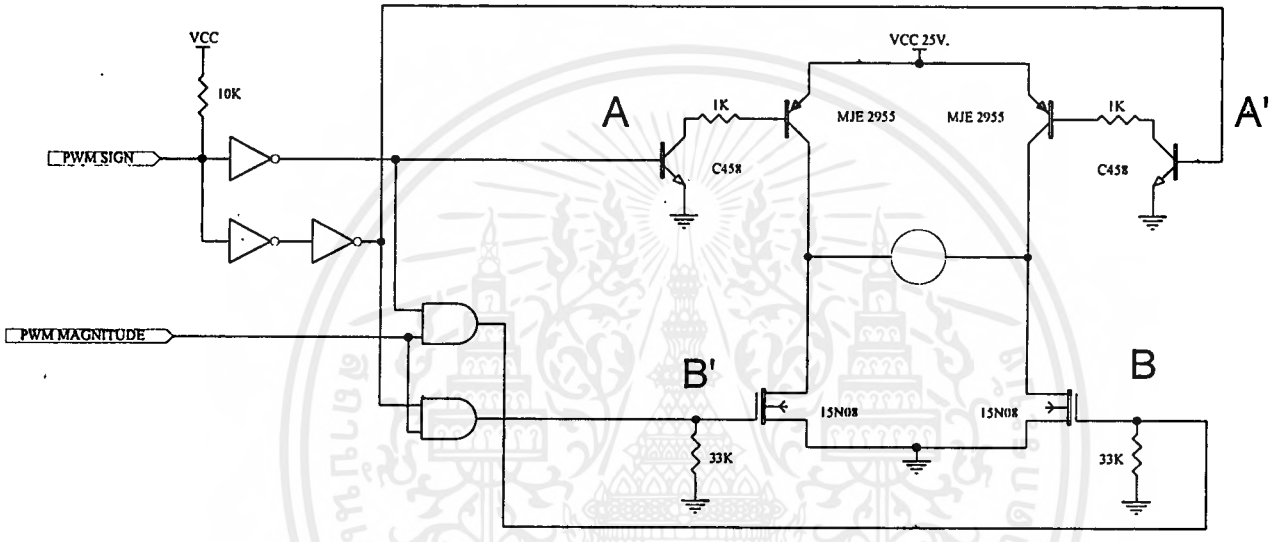
รูปที่ 2.2 แสดงวงจรการทำงานของ IC 74LS688 ร่วมกับดิปสวิทช์

การกำหนดแอดเดรสจะถูกกำหนดโดยขา A0-A9 (รายละเอียดดูที่ภาคผนวก)จากรูปถ้าขาสัญญานดิปสวิทช์ตรงกับขาสัญญาน IC 74LS688 จะทำให้เกิดการเลือกการทำงานของชิป LM629 แต่ละตัวดังนั้นเมื่อรวมการทำงานแล้วจะได้อุปกรณ์ที่เรียกว่าการ์ดอินเตอร์เฟส(interface card)[1] ซึ่งเป็นอุปกรณ์ที่ควบคุมการทำงานของอินเวิร์ทเพนดูลัมและถูกกำหนดการทำงานโดยคอมพิวเตอร์ส่วนบุคคล(personal computer)วงจรการ์ดอินเตอร์เฟสรวมแสดงในรูปที่ 2.3 หน้าถัดไป



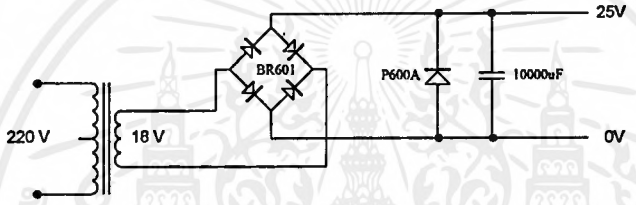
รูปที่ 2.3 แสดงวงจรการ์ดอินเตอร์เฟส

Title	INTERFACE CARD		
Size: B	Number: 2		Revision
Date: 7-Apr-1998		Sheet of RIP	
File: C:\NINPROJ\INTERFAC.SCH		Drawn By: Prasert & Tanin	



รูปที่ 2.4 แสดงวงจรขับมอเตอร์กระแสตรง

Title: DC DRIVE			
Size: B	Number:		Revision:
Date: 7-Apr-1998		Sheet of RIP	
File: C:\NINPRO\DCDRIVE.SCH		Drawn By:Prasert & Tanin	



รูปที่ 2.5 แสดงแหล่งจ่ายไฟตรง 25 โวลท์

Title	POWER SUPPLY		
Size B	Number		Revision
Date:	8-Apr-1998	Sheet of	RIP
File:	C:\WINPRO\SUPPLY25.SCH	Drawn By:	Prasert & Tanin

บนการ์ดอินเตอร์เฟสนั้นมีไอซีพิเศษตัวหนึ่งชื่อว่า LM629 ซึ่งมีคุณสมบัติดังนี้

- มีรีจิสเตอร์ตำแหน่ง ความเร็ว และความเร่ง ขนาด 32 บิต
- เป็นฟิลเตอร์ดิจิตอลแบบ PID
- สามารถกำหนดตำแหน่งได้
- เอาท์พุทเป็น PWM ซึ่งมีทั้งขนาดและทิศทาง
- สามารถทำงานได้ทั้งโหมดการควบคุมตำแหน่งและโหมดการควบคุมความเร็ว
- ติดต่อกับหน่วยประมวลหลักขนาด 8 บิต
- มีหน่วยรับสัญญานการป้อนกลับจากเอนโค้ดเดอร์

* รายละเอียดการทำงาน อ้างที่ภาคผนวก จ *

การทำงานของอุปกรณ์ควบคุมมอเตอร์กระแสตรง

จากรูปที่ 2.4 วงจรแสดงวงจรขับมอเตอร์อธิบายได้ดังนี้

เมื่อสัญญาน PWM SIGN มีสัญญานลอจิกเป็น "1" A ก็เป็น "1" ด้วยการเคลื่อนที่จะไปทางขวา และถูกควบคุมความความเร็วโดย PWM MAGNITUDE ผ่านทาง A′ ในทางตรงข้ามถ้าสัญญาน PWM SIGN มีสัญญานลอจิกเป็น "0" B ก็เป็น "0" ด้วยการเคลื่อนที่จะไปทางซ้าย และถูกควบคุมความความเร็วโดย PWM MAGNITUDE ผ่านทาง B′และสัญญานควบคุมทั้งหมดเป็นสัญญาณลอจิกทั้งสิ้น

ความคงทนของวงจรนั้นทนได้ที่ประมาณ 50V. 3.5 A. ระบายร้อนด้วยแผ่นอลูมิเนียม

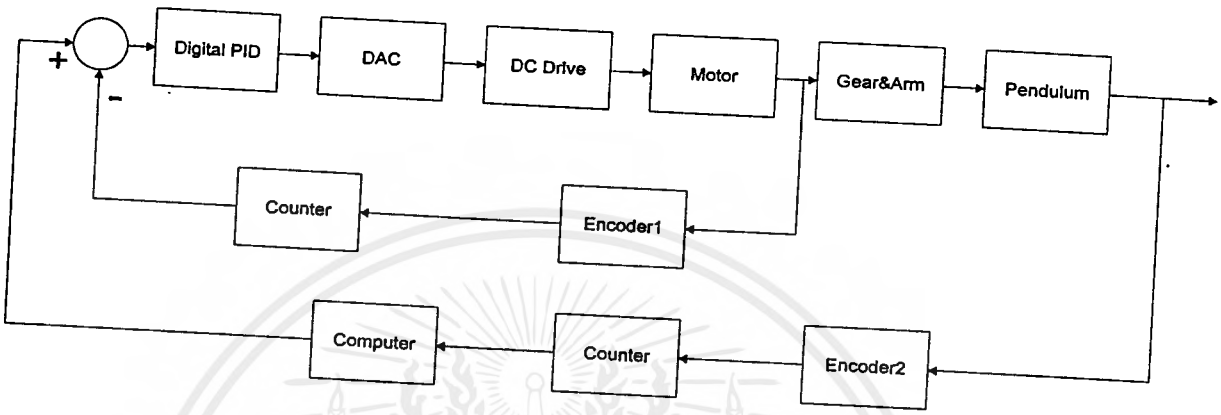
2.3 ส่วนควบคุม

เรานำคอมพิวเตอร์มารับค่าจากส่วนอินเตอร์เฟสแล้วนำค่าที่ได้มาคำนวน กำหนดค่าพารามิเตอร์ต่างๆแล้วจึงส่งค่าที่ได้ไปยังส่วนอินเตอร์เฟสอีกทีหนึ่ง เพื่อควบคุมระบบต่อไป

ส่วนควบคุมเราจะแสดงในภาคผนวก ข อันได้แก่ โฟลชาร์ท และโปรแกรม

บทที่ 3
แบบจำลองทางคณิตศาสตร์

จากรูปที่ 1.1 เราสามารถเขียนเป็น บล็อกไดอะแกรม (Block Diagram)ได้ดังนี้



รูปที่ 3.1 บล็อกไดอะแกรม ของระบบทั้งหมด

เราสามารถหา ฟังก์ชั่นถ่ายโอน (Transfer Function) ต่างๆได้ดังนี้

ชุดควบคุม ดิจิตอล PID (Digital PID)

$$G(s) = K_P + \frac{K_I}{s} + K_D s$$ พัลส์/พัลส์

ตัวแปลงสัญญาณดิจิตอลเป็นอะนาลอก(Digital to analog converter) G(s) =10 โวลท์/พัลส์

วงจรขับมอเตอร์กระแสตรง(DC Drive) G(s) =5 โวลท์/โวลท์

เฟืองและแขน(Gear & Arm) G(s) =3.7 เมตร/เรเดียน

ตัววัดตำแหน่งแบบแสงตัวที่ 1 (Optical Encoder) G(s) = 572.95 พัลส์/เรเดียน

ตัววัดตำแหน่งแบบแสงตัวที่ 2 (Optical Encoder) G(s) = 318.31 พัลส์/เรเดียน

วงจรนับ(Counter) G(s) = 4 พัลส์/พัลส์

มอเตอร์กระแสตรง (DC Motor)

จาก $$\frac{\Theta(s)}{E_a(s)} = \frac{K_m}{s(T_m s + 1)}$$

$K_m = K/(R_a b + KK_b)$: ค่าคงที่ของมอเตอร์(motor gain constant)

$T_m = R_a J/(R_a b + KK_b)$: ค่าคงที่ของเวลาของมอเตอร์(motor time constant)

K : ค่าที่แรงบิดของมอเตอร์ (motor torque constant)

K_b : ค่าคงที่แรงดันย้อนกลับ (back emf constant)

R_a : ความต้านอาร์มาเจอร์ (armature resistance)

b : แรงเสียดทาน(Equivalent visous-friction coefficient of the motor and load referred to the motor shaft)

J : โมเมนต์ของความเฉื่อย(Equivalent moment of inertia of the motor and load refered to the motor shaft

C : แรงเสียดทานของก้านลูกตุ้ม

จากการคำนวนและข้อมูลของมอเตอร์ (ดูภาคผนวก ค)

K = 0.6 kg.cm/A

K_b = 6.1 V/Krpm

R_a = 4 Ω

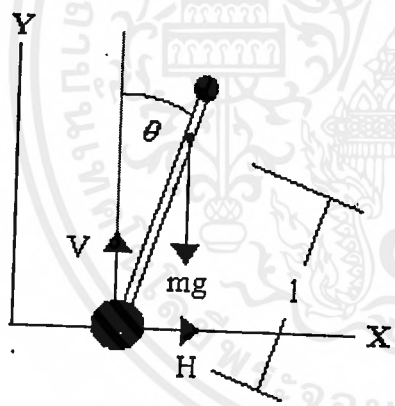
b ≅ 0

J = 0.00023415 kg.m2

เมื่อแทนค่าต่างๆ จะได้

$$\frac{\Theta(s)}{E(s)} = \frac{0.64}{s(s+390.7)}$$ เรเดียน/โวลท์

ลูกตุ้ม (Pendulum) [2],[3],[4]



รูปที่ 3.2 แสดงแบบจำลองของลูกตุ้ม

δ = ระยะของฐานก้านลูกตุ้มกับแนวอ้างอิง

l = ความยาวของก้านลูกตุ้มวัดที่จุดรวมน้ำหนัก

m = น้ำหนักของก้านลูกตุ้ม

θ = มุมของก้านลูกตุ้มที่เปลี่ยนไป

V = แรงในแนวตั้งที่ทำกับก้านลูกตุ้ม

H = แรงในแนวนอนที่ทำกับก้านลูกตุ้ม

จากกฎข้อ 2 ของนิวตัน

$$\sum F = Ma$$

จะได้

$$\frac{Jd^2\theta}{dt^2} + C\frac{d\theta}{dt} = Vl\sin\theta - Hl\cos\theta \qquad (1)$$

พิจารณาในแกน Y

$$\frac{md^2}{dt^2}(l\cos\theta) = V - mg \qquad (2)$$

พิจารณาในแกน X

$$\frac{md^2}{dt^2}(\delta + l\sin\theta) = H \qquad (3)$$

จาก

$$\frac{d^2}{dt^2}\cos\theta = -(\cos\theta)(\dot{\theta})^2 - (\sin\theta)(\ddot{\theta})^2$$

$$\frac{d^2}{dt^2}\sin\theta = -(\sin\theta)(\dot{\theta})^2 - (\cos\theta)(\ddot{\theta})^2$$

แทน (2) ,(3)ใน (1)

$$(J + ml^2)\ddot{\theta} + ml(\cos\theta)\ddot{\delta} = -C\dot{\theta} + mgl\sin\theta \qquad (5)$$

เราจะประมาณให้เป็นระบบเชิงเส้น(Linear System)

$$\sin\theta \approx \theta$$

$$\cos\theta \approx 1$$

เราเขียน (5) ใหม่เป็น

$$(J + ml^2)\ddot{\theta} + ml\ddot{\delta} = -C\dot{\theta} + mgl\theta \qquad (5)$$

เมื่อ C = 0 เราได้ Transfer Function ของลูกตุ้มเป็น

$$\frac{\Theta(s)}{\delta(s)} = \frac{mls^2}{(J + ml^2)s^2 - mgl}$$

m = 0.04809 kg

l = 0.347 m

g = 9.8066 m/s^2

J = 7.125×10^{-3} $kg.m^2$

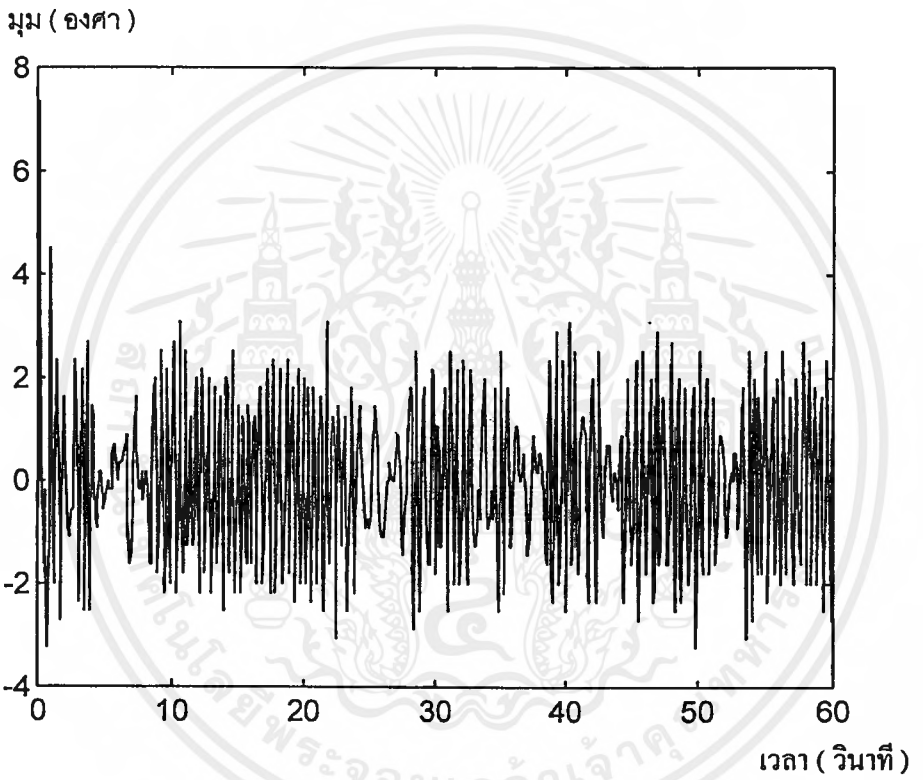
แทนค่า

$$\frac{\Theta(s)}{\delta(s)} = \frac{1.292s^2}{s^2 - 12.67}$$ เรเดียน/เมตร

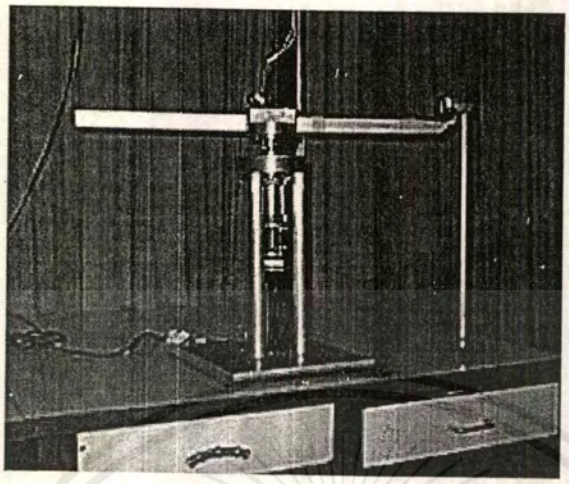
บทที่ 4

ผลการทดลอง

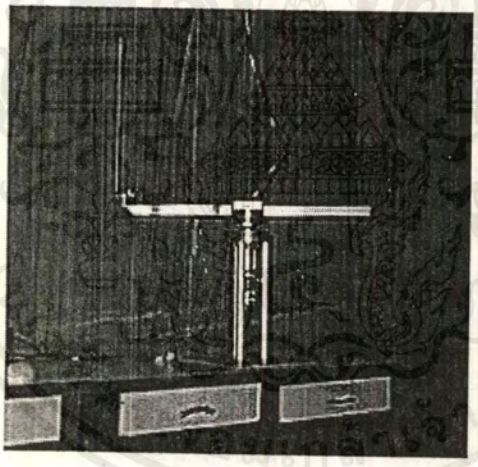
เมื่อทำการทดลองแล้วนำค่ามุมที่เปลี่ยนไปเทียบกับเวลาผลแสดงเป็นกราฟจะแกว่งในช่วงที่น้อยมากดังนั้นระบบจึงมีเสถียรภาพสามารถควบคุมได้ ดังรูป 4.1



รูปที่ 4.1 กราฟแสดงตำแหน่งของก้านลูกตุ้มขณะทดลอง



รูปที่ 4.2 แสดงชุดโครงงานในขณะยังไม่มีการควบคุม



รูปที่ 4.3 แสดงโครงงานในขณะที่ทำการควบคุมให้ทรงตัวแล้ว

บทที่ 5
สรุปและวิเคราะห์ผลการดำเนินงาน

5.1 สรุปการดำเนินงาน

จากการศึกษาและทดลองของโครงงานแบ่งเป็น 3 ส่วนคือ ส่วนโครงสร้างทางเครื่องกล ได้ออกแบบไว้อย่างเหมาะสม ซึ่งเป็นผลดีต่อระบบอย่างยิ่งเนื่องจากเกิดปัญหาน้อยมาก ด้านวงจรอิเล็กทรอนิกส์ และติดต่อกับคอมพิวเตอร์ เราเลือกใช้อุปกรณ์และออกแบบวงจร เพื่อง่ายต่อการติดตั้งการตรวจสอบข้อผิดพลาดและเสถียรภาพ ส่วนควบคุมคือโปรแกรมเราใช้โปรแกรมภาษาซีโครงสร้างโปรแกรมที่ตรงไปตรงมาทำงาน ง่ายไม่ซับซ้อน ง่ายต่อการตรวจสอบ การศึกษาและพัฒนาในอนาคต

5.2 ปัญหาที่เกิดขึ้นและแนวทางแก้ไข

ปัญหาหลักในการดำเนินงานแบ่งออกได้ดังนี้

5.2.1 โครงสร้างทางเครื่องกล ที่จุดต่อระหว่างเพลามอเตอร์และส่วนจับแขนควรเป็นวัสดุชนิดอื่นเพราะอลูมิเนียมค่อนข้างเสียรูปง่ายและอีกอย่างหนึ่งคือ มอเตอร์เป็นแบบมีชุดเฟืองทดผลจากการสึกหรอของเฟืองทำให้เกิด back lag เป็นผลเสียต่อเสถียรภาพของระบบอย่างยิ่ง จึงควรเปลี่ยนมอเตอร์เป็นแบบไม่มีเฟืองทดแทน

5.2.2 ส่วนวงจรอิเล็กทรอนิกส์และส่วนติดต่อคอมพิวเตอร์ เนื่องจากวงจรของเรายังใช้เป็นแผ่นปริ้นท์เอนกประสงค์ จึงเกิดความไม่ถาวรอาจเกิดปัญหาเล็กๆ น้อยๆ ดังนั้นควรเปลี่ยนเป็นการกัดปริ้นท์ถาวรให้เรียบร้อย ส่วนสายไฟและสายสัญญาณ ควรเปลี่ยนเป็นแบบถอดได้เพื่อความสะดวกในการทำงาน และการขนย้าย

5.2.3 ส่วนโปรแกรม ควรมีการแสดงผลขณะทดลองพร้อมทั้งเก็บข้อมูลต่าง ๆเพื่อวิเคราะห์เสถียรภาพและง่ายต่อการพัฒนาในอนาคต

5.3 แนวทางการพัฒนาในอนาคต

การพัฒนาในอนาคตนั้นสามารถพัฒนาได้หลายแบบเช่นลดความยาวของแกนเพื่อเพิ่มความยากของการทรงตัว แต่ในขั้นพัฒนาเพื่อเพิ่มประสิทธิภาพคือต่ออีกหนึ่งแกนกลายเป็นสองแกนซึ่งเป็นความยากมากขึ้นในการทำเพนดูลัม

ดังที่ได้กล่าวมาแล้วนั้นเป็นทางด้านโครงสร้างและด้านที่สามารถพัฒนาเพิ่มอีกคือตัวควบคุมอาจจะเลือกใช้การควบคุแบบฟัซซี่ , LQR , analog control

อีกสิ่งที่สามารถพัฒนาได้คือการ์ดอินเตอร์เฟสและวงจรขับเคลื่อนมอเตอร์กระแสตรงซึ่งอาจเลือกอุปกรณ์อิเล็กทรอนิกส์ที่มีประสิทธิภาพดีกว่า

เอกสารอ้างอิง

[1] ธานินทร์ ถาวรศาสนวงศ์, " การอินเตอร์เฟส ", สำนักพิมพ์ฟิสิกส์เซ็นเตอร์,พศ 2536

[2] KATSHUISA FURATA ," **STATE VARIABLE METHODS IN AUTOMATIC CONTROL** ", JOHN WILEY & SON CO. 1988

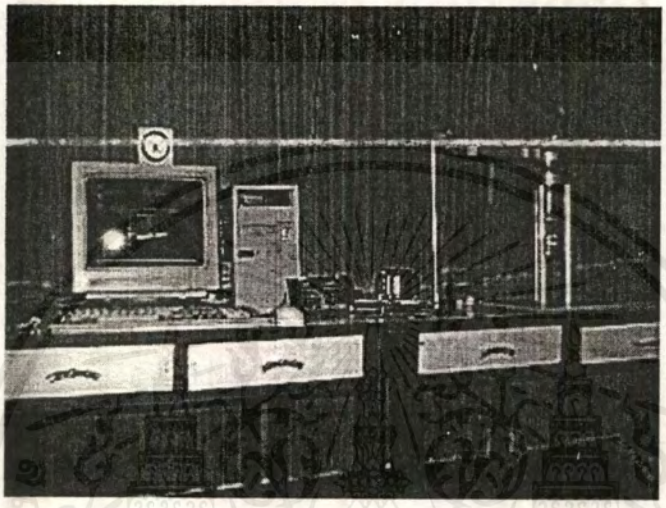
[3] KATSUHIKO OTAGA ," **MODERN CONTROL ENGINEERING** " ,SECOND EDITION,PRENTICE -HALL INTERNATIONAL,1990

[4] RAUL CORDONEZ , " **ADAPTIVE FUZZY CONTROL: EXPERIMENTS AND COMPARATIVE ANALYSES** ",IEEE TRANSATION ON FUZZY SYSTEMS,VOL. 5 NO.2, MAY 1997,PP 167-180

ภาคผนวก

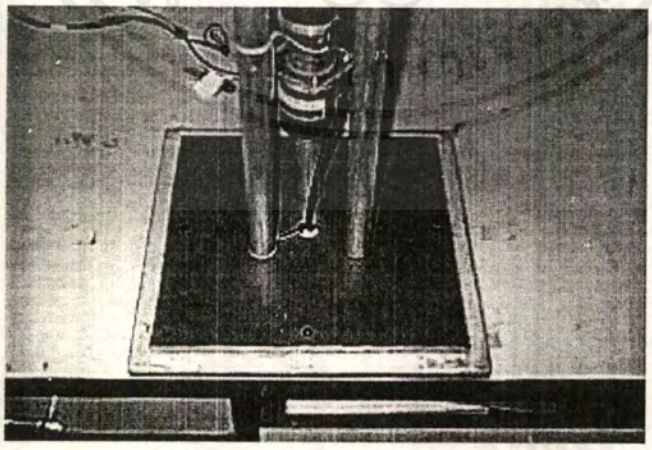
ภาคผนวก ก

ภาพแสดงส่วนประกอบของระบบ

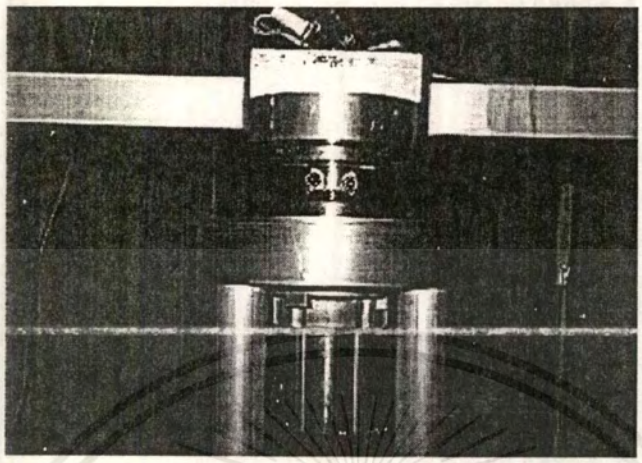


รูปที่ 1ก แสดงโครงงานทั้งหมด

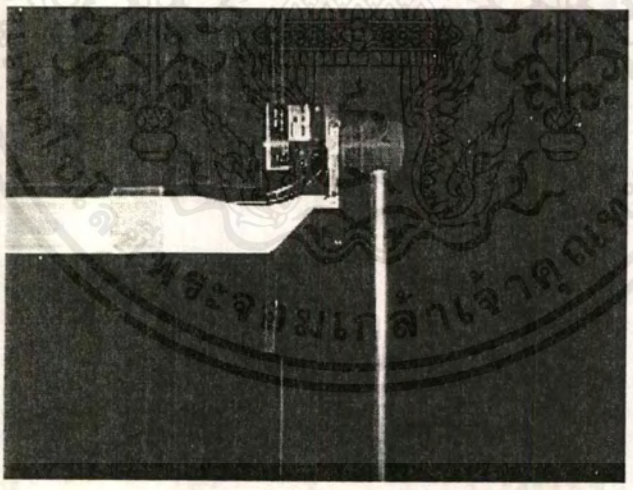
อุปกรณ์การทดลองจะประกอบไปด้วย คอมพิวเตอร์ส่วนตัว การ์ดอินเตอร์เฟส ชุดขับเคลื่อนมอเตอร์กระแสตรง อินเวิร์ทเพนดูลัม



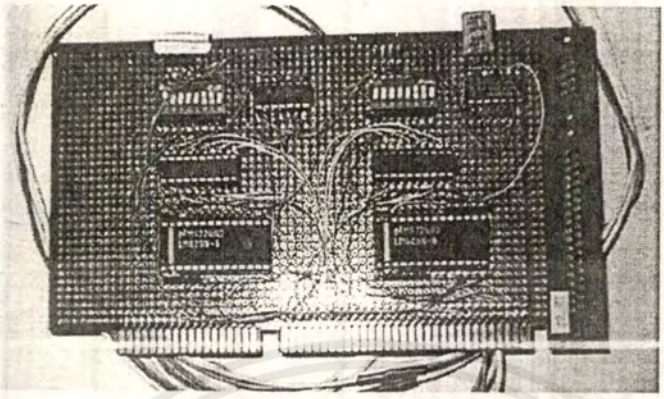
รูปที่ 2ก แสดงฐานรองรับพร้อมทั้งเอนโค้ดเดอร์ที่ติดตัวมอเตอร์



รูปที่ 3ก แสดงส่วนจับยึดแขน

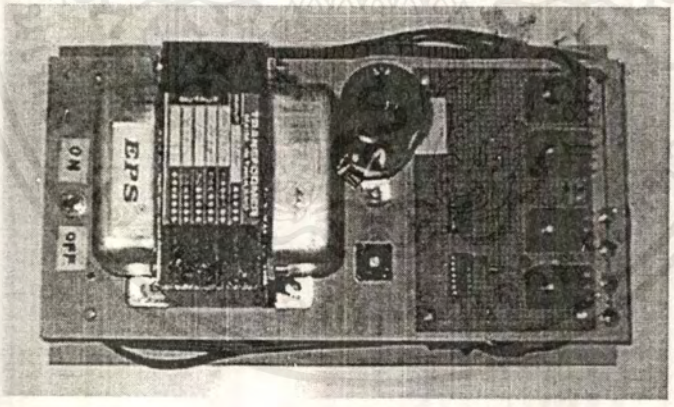


รูปที่ 4ก แสดงเอนโค้ดเดอร์วัดมุมที่เปลี่ยนไปของก้านลูกตุ้ม



รูป 5ก แสดงการ์ดอินเตอร์เฟส

ซึ่งในทางปฏิบัตินั้นจะไม่ใช้การเชื่อมสายโดยตะกั่วแต่จะใช้การพันสายแทนเพราะการพันสายทำให้มีข้อผิดพลาดน้อยกว่าและลดปัญหาฉีกขาดของสายไฟ



รูปที่ 6ก แสดงวงจรชุดขับเคลื่อนมอเตอร์กระแสตรงขนาด 25 โวลท์.ชุดขับเคลื่อนนั้นจะรวมแหล่งจ่ายเป็นชุดเดียวกันเพื่อความสะดวกในการเคลื่อนที่และใช้เต้าเสียบเพื่อสะดวกในการต่อมอเตอร์

ภาคผนวก ข

โปรแกรมควบคุมหลัก (Swing&ru.cpp)

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <bios.h>
#include<jerry.h>
#include<jerry2.h>
void SET_PID_FILTER(unsigned int, unsigned int, unsigned int, unsigned int);
void DECTECT_NEW_HOME1(void);
void DECTECT_NEW_HOME(void);
int  READ_REAL_POS_COUNT(void);
int  P2_count,P1_count;
long b,a;
long READ_REAL_VELO_COUNT(void);
void WAITING_FOR_STOP_MOTOR(void);
void main(void)
{
      P2_count = 0;
      clrscr();
      INIT_LM629_2();
      DECTECT_NEW_HOME();
      INIT_LM629();
      SET_PID_FILTER(4,7,285,10000);
      while((bioskey(1)&0xff) != 0x001b)
        {
              DECTECT_NEW_HOME1();
              P1_count = READ_REAL_POS_COUNT();
              P2_count = READ_REAL_POS_COUNT_2();
              a = ( 142.7657143 * (1+0.3) * P2_count );
              POSITION_OUT(a,1000,10000);
              printf(" position_pen(P2):%d\n " ,P2_count);
        }
```

```
    INIT_LM629();
}

void DECTECT_NEW_HOME1(void)
 {

    delay(3);
    WRITE_COM(DFH);       //define arm home command
 }
  void DECTECT_NEW_HOME(void)
    {
    WRITE_COM(RESET);     //reset command code
    WRITE_COM(DFH);       //define home command
    WRITE_COM(SIP);
    while(READ_REAL_POS_COUNT_2() < 4002)
    {
        int P2_count = READ_REAL_POS_COUNT_2();
        b=42880;
        SET_PID_FILTER(20,0,4000,1000);
        POSITION_OUT(b,500,1000);
            printf("\nPEN_POSITION = %d   ",P2_count);

    }

    INIT_LM629_2();
}
```

โปรแกรม Header File (Jerry.h)

```
#define ADDR_COM        0X27c
#define ADDR_DATA       0X27d
#define posConvertC2M   3.0041758E-5
#define posConvertM2C   1 //33287
#define veloConvertC2M  1.34297E-6
#define accConvertM2C   254.162
#define filterCommand   0x000F        //kp,ki,kd well be loaded
#define posCommand      0x002A
#define SIP                   0 x03   //set index position data n
#define LPEI                  0x1B    //load position error for
                              interrupt data 2
#define MSKI                  0x1C    //mask interrupts data 2
#define RSTI                  0 x1D   //reset interrupts data 2
#define LFIL                  0x1E    //funtion load parameter of
                              filter to lm629
#define UDF                   0x04    //update filter parameter
#define LTRJ                  0x1F    //load trajectory
#define STT                   0x01    //start trajectory
#define SBPA                  0x20    //set break point absolute
#define RDSIGS                0x0C    //read signal register
#define RDDP                  0x08    //read desired position
#define RDRP                  0x0A    //read real position
#define RDDV                  0x07 //read desired velocity


struct var_set
    {
        unsigned int Kp;
        unsigned int  Ki;
        unsigned int  Kd;
        unsigned int  Il;
        long Pos;
        long Vel;
```

```
        long Acc;
    };

enum boolean{ TRUE, FALSE };
boolean exit_loop = FALSE;
char Dir,ch;
int  Direction;
int  Stop = 0;
/* PROTOTYPE */
void INIT_LM629(void);          //initial lm629
void WRITE_FILTER(void);        //write filter parameter to lm629
void POSITION_OUT(float,float,float);      //send position command to lm629
int   READ_REAL_POS_COUNT(void);    //read real position in counter unit
void SET_PID_FILTER(unsigned int, unsigned int, unsigned int, unsigned int);
//--->Sub function
void WAITBUSY(void);          //wait if lm629 busies
void WRITE_COM(unsigned char);          //write command to lm629
void WRITE_DATA_BYTE(unsigned char);    //write data 1 byte to lm629
void WRITE_DATA_2BYTES(int);   //write data 2 bytes to lm629
void WRITE_DATA_4BYTES(long);  //write data 4 bytes to lm629
void WRITE_DATA_WORD(int);     //write word data ONLY FILTER DATA
void WRITE_DATA_2WORDS(long);  //write 2 words data ONLY TRAJECTORY DATA
unsigned char READ_DATA_BYTE(void);     //read data from lm629
int  READ_DATA_2BYTES(void);   //read 2 BYTEs data from lm629
unsigned long READ_DATA_4BYTES(void);   //read 4 BYTEs daata from lm629

/* FUNCTION */
void WAITBUSY()//1
{
    while(0x01 & inp(ADDR_COM)); //wait util busy flag is zero
}
void WRITE_COM(unsigned char comm)//2
{
    WAITBUSY();
    outp(ADDR_COM,comm);
}
```

```
void WRITE_DATA_BYTE(unsigned char data)//3
{
    WAITBUSY();
    outp(ADDR_DATA,data);
}
void WRITE_DATA_2BYTES(int data)//4
{
    char *temp;
    temp=(char *)&data;
    WAITBUSY();
    outp(ADDR_DATA,*(temp+1));
    WAITBUSY();
    outp(ADDR_DATA,*(temp));
}
void WRITE_DATA_4BYTES(long data)//5
{
    char *temp;
    temp=(char *)&data;
    WAITBUSY();
    outp(ADDR_DATA,*(temp+3));
    WAITBUSY();
    outp(ADDR_DATA,*(temp+2));
    WAITBUSY();
    outp(ADDR_DATA,*(temp+1));
    WAITBUSY();
    outp(ADDR_DATA,*temp);
}
void WRITE_DATA_WORD(int data)//6
{
    char *temp;
    temp=(char *)&data;
    WAITBUSY();
    outp(ADDR_DATA,*(temp+1));
    outp(ADDR_DATA,*(temp));
}
void WRITE_DATA_2WORDS(long data)//7
```

```
{
        char *temp;
        temp=(char *)&data;
        WAITBUSY();
        outp(ADDR_DATA,*(temp+3));
        outp(ADDR_DATA,*(temp+2));
        WAITBUSY();
        outp(ADDR_DATA,*(temp+1));
        outp(ADDR_DATA,*temp);
}
unsigned char READ_DATA_BYTE(void)//8
{
        WAITBUSY();
        return inp(ADDR_DATA);
}
int READ_DATA_2BYTES(void)//9
{
        char temp1[1];
        int *temp2;
        WAITBUSY();
        temp1[1]=inp(ADDR_DATA);
        WAITBUSY();
        temp1[0]=inp(ADDR_DATA);
        temp2=(int *)temp1;
        return *temp2;
}
unsigned long READ_DATA_4BYTES(void)// 10
{
        char temp1[3];
        long *temp2;
        WAITBUSY();
        temp1[3]=inp(ADDR_DATA);  //byte 1
        WAITBUSY();
        temp1[2]=inp(ADDR_DATA);  //byte 2
        WAITBUSY();
        temp1[1]=inp(ADDR_DATA);  //byte 3
```

```
    WAITBUSY();
    temp1[0]=inp(ADDR_DATA);  //byte 4
    temp2=(long *)temp1;
    return *temp2;
}
void INIT_LM629(void)//12
{
    WRITE_COM(RESET);     //reset command code
    WRITE_COM(DFH);       //define home command
    WRITE_COM(SIP);       //set index position command
}
void POSITION_OUT(float pos,float velo,float accel)
{
    WRITE_COM(LTRJ);
    WRITE_DATA_WORD(posCommand);
    WRITE_DATA_2WORDS((long)(accel*accConvertM2C));
    WRITE_DATA_2WORDS((long)(velo*veloConvertM2C));
    WRITE_DATA_2WORDS((long)(pos*posConvertM2C));
    WRITE_COM(STT);
}
int  READ_REAL_POS_COUNT(void)                //read real position in counter unit
{
    WRITE_COM(RDRP);
    return READ_DATA_4BYTES();
}
void SET_PID_FILTER(unsigned int Kp, unsigned int Ki, unsigned int Kd, unsigned int Il)
{
    WRITE_COM(LFIL);        //send command code.
    WRITE_DATA_WORD(filterCommand);  //Kp, Ki, Kd will be loaded.
    WRITE_DATA_WORD(Kp);          //Kp parameter
    WRITE_DATA_WORD(Ki);          //Ki parameter
    WRITE_DATA_WORD(Kd);          //Kd parameter
    WRITE_DATA_WORD(Il);          //Il parameter
WRITE_COM(UDF);           //update filter parameter
}
```

โปรแกรม Header File (Jerry2.h)

```
#define ADDR_COM        0X27c
#define ADDR_DATA       0X27d
#define posConvertC2M   3.0041758E-5
#define posConvertM2C   1 //33287
#define veloConvertC2M  1.34297E-6
#define accConvertM2C   254.162
#define filterCommand   0x000F        //kp,ki,kd well be loaded
#define posCommand      0x002A
#define SIP                   0 x03   //set index position data n
#define LPEI                  0x1B    //load position error for interrupt data 2
#define MSKI                  0x1C    //mask interrupts data 2
#define RSTI                  0 x1D   //reset interrupts data 2
#define LFIL                  0x1E    //funtion load parameter of filter to lm629
#define UDF                   0x04    //update filter parameter
#define LTRJ                  0x1F    //load trajectory
#define STT                   0x01    //start trajectory
#define SBPA                  0x20    //set break point absolute
#define RDSIGS                0x0C    //read signal register
#define RDDP                  0x08    //read desired position
#define RDRP                  0x0A    //read real position
#define RDDV                  0x07 //read desired velocity


struct var_set
    {
            unsigned int  Kp;
            unsigned int  Ki;
            unsigned int  Kd;
            unsigned int  Il;
            long Pos;
            long Vel;
            long Acc;
```

```
        };
enum boolean{ TRUE, FALSE };
boolean exit_loop = FALSE;
char Dir,ch;
int  Direction;
int  Stop = 0;
/* PROTOTYPE */
void INIT_LM629_2(void);        //initial lm629
void WRITE_FILTER_2(void);      //write filter parameter to lm629
void POSITION_OUT_2(float,float,float);     //send position command to lm629
int  READ_REAL_POS_COUNT_2(void); //read real position in counter unit
void SET_PID_FILTER_2(unsigned int, unsigned int, unsigned int, unsigned int);
//--->Sub function
void WAITBUSY_2(void);          //wait if lm629 busies
void WRITE_COM_2(unsigned char);        //write command to lm629
void WRITE_DATA_BYTE_2(unsigned char);    //write data 1 byte to lm629
void WRITE_DATA_2BYTES_2(int);   //write data 2 bytes to lm629
void WRITE_DATA_4BYTES_2(long);  //write data 4 bytes to lm629
void WRITE_DATA_WORD_2(int);     //write word data ONLY FILTER DATA
void WRITE_DATA_2WORDS_2(long);  //write 2 words data ONLY TRAJECTORY DATA
unsigned char READ_DATA_BYTE_2(void);     //read data from lm629
int  READ_DATA_2BYTES_2(void);   //read 2 BYTEs data from lm629
unsigned long READ_DATA_4BYTES_2(void);   //read 4 BYTEs daata from lm629


/* FUNCTION */
void WAITBUSY_2()//1
{
      while(0x01 & inp(ADDR_COM)); //wait util busy flag is zero
}
void WRITE_COM_2(unsigned char comm)//2
{
      WAITBUSY();
      outp(ADDR_COM,comm);
}
void WRITE_DATA_BYTE_2(unsigned char data)//3
{
```

```
        WAITBUSY();
        outp(ADDR_DATA,data);
}
void WRITE_DATA_2BYTES_2(int data)//4
{
        char *temp;
        temp=(char *)&data;
        WAITBUSY();
        outp(ADDR_DATA,*(temp+1));
        WAITBUSY();
        outp(ADDR_DATA,*(temp));
}
void WRITE_DATA_4BYTES_2(long data)//5
{
        char *temp;
        temp=(char *)&data;
        WAITBUSY();
        outp(ADDR_DATA,*(temp+3));
        WAITBUSY();
        outp(ADDR_DATA,*(temp+2));
        WAITBUSY();
        outp(ADDR_DATA,*(temp+1));
        WAITBUSY();
        outp(ADDR_DATA,*temp);
}
void WRITE_DATA_WORD_2(int data)//6
{
        char *temp;
        temp=(char *)&data;
        WAITBUSY();
        outp(ADDR_DATA,*(temp+1));
        outp(ADDR_DATA,*(temp));
}
void WRITE_DATA_2WORDS_2(long data)//7
{
        char *temp;
```

```
        temp=(char *)&data;
        WAITBUSY();
        outp(ADDR_DATA,*(temp+3));
        outp(ADDR_DATA,*(temp+2));
        WAITBUSY();
        outp(ADDR_DATA,*(temp+1));
        outp(ADDR_DATA,*temp);
}
unsigned char READ_DATA_BYTE_2(void)//8
{
        WAITBUSY();
        return inp(ADDR_DATA);
}
int READ_DATA_2BYTES_2(void)//9
{
        char temp1[1];
        int *temp2;
        WAITBUSY();
        temp1[1]=inp(ADDR_DATA);
        WAITBUSY();
        temp1[0]=inp(ADDR_DATA);
        temp2=(int *)temp1;
        return *temp2;
}
unsigned long READ_DATA_4BYTES_2(void)// 10
{
        char temp1[3];
        long *temp2;
        WAITBUSY();
        temp1[3]=inp(ADDR_DATA);  //byte 1
        WAITBUSY();
        temp1[2]=inp(ADDR_DATA);  //byte 2
        WAITBUSY();
        temp1[1]=inp(ADDR_DATA);  //byte 3
        WAITBUSY();
        temp1[0]=inp(ADDR_DATA);  //byte 4
```

```
        temp2=(long *)temp1;
        return *temp2;
}
void INIT_LM629_2(void)//12
{
        WRITE_COM(RESET);      //reset command code
        WRITE_COM(DFH);        //define home command
        WRITE_COM(SIP);        //set index position command
}
void POSITION_OUT_2(float pos,float velo,float accel)
{
        WRITE_COM(LTRJ);
        WRITE_DATA_WORD(posCommand);
        WRITE_DATA_2WORDS((long)(accel*accConvertM2C));
        WRITE_DATA_2WORDS((long)(velo*veloConvertM2C));
        WRITE_DATA_2WORDS((long)(pos*posConvertM2C));
        WRITE_COM(STT);
}
int  READ_REAL_POS_COUNT_2(void)            //read real position in counter unit
{
        WRITE_COM(RDRP);
        return READ_DATA_4BYTES();
}
void SET_PID_FILTER_2(unsigned int Kp, unsigned int Ki, unsigned int Kd, unsigned int Il)
{
        WRITE_COM(LFIL);        //send command code.
        WRITE_DATA_WORD(filterCommand);   //Kp, Ki, Kd will be loaded.
        WRITE_DATA_WORD(Kp);            //Kp parameter
        WRITE_DATA_WORD(Ki);            //Ki parameter
        WRITE_DATA_WORD(Kd);            //Kd parameter
        WRITE_DATA_WORD(Il);             //Il parameter
        WRITE_COM(UDF);                 //update filter parameter
}
```

ภาคผนวก ค

ค่าจำเพาะของมอเตอร์

มอเตอร์กระแสตรง 24 V. ขนาด 20 W. Type SS32F

ยี่ห้อ SAWAMURA DENKI OGYO Co.,Ltd TOKYO , JAPAN

พิกัดแรงดันไฟฟ้า	V	24
พิกัดกระแสไฟฟ้า	A	1.4
ความเร็วรอบ	rpm	3000
พิกัดแรงบิด	kg . cm	0.65
กระแสขณะ no laod	A	0.24
ความเร็วรอบขณะ no laod	rpm	3800
ค่าคงที่แรงบิด (Kt)	kg . cm/A	0.6
แรงดันไฟฟ้าย้อนกลับ (Back EMF)	V/Krpm	6.1
ความต้านทานอาร์มาเจอร์ (Ra)	ohm	4
ความเหนี่ยวนำอาร์มาเจอร์ (La)	mH	4.4
โมเมนต์ความเฉื่อย (Jm)	kg . cm^2	0.11
Tm	msec	12
Te	msec	1.1

INSTRUCTION MANUAL

Rotary Encoder ORE Series

Thank you very much for using Sunx encoder.Please read this manual carefully and thoroughly for the optimum use of the encoder,and keep this manual in a convenient place for quick reference.

1.TYPE DESIGNATION

ORE- □ □ - □ □ □

- Body diameter
  - ·38: ϕ 38.8mm
  - ·48: ϕ 48.8mm
- Type of shaft
  - ·No mark:Standard shaft type
  - ·B:Hollow shaft type
- Resolution
  - ·Pulse number
- Output Mode
  - ·V:Voltage Output
  - ·C:Open Collector Output
  - ·L:Line Driver Output
- Supply voltage
  - ·05:5V
  - ·12:12V
  - ·24:24V

2.SPECIFICATIONS

Item \ Model No.	ORE-38-□	ORE-48-□
Detection Method	Optical Incremental System	
Shaft Diameter	Standard shaft type : 6 dia. Hollow shaft type : 5 dia. Hole	Standard shaft type : 6 dia. Hollow shaft type : 8 dia. Hole
Resolution	100,200,300,360,400 500,600,1,000,1024,1200 1500,1800,2000,2048	500,1000,1500 2000,2500,3600
Supply voltage	5V DC±5% { Voltage Output / Open Collector Output / Line Driver Output	12V DC±5% { Voltage Output / Open Collector Output 24V DC±5% { Open Collector Output
Consumption	●Max. 100mA under both of 5V and 12V and 24V of supply voltage (excluding external load current) for Voltage Output and Open Collector Output. ●Max. 200mA at 20mA of load current between each A-$\bar{A}$,B-$\bar{B}$ and Z-$\bar{Z}$ for Line Driver Output.	
Output Signal	•Voltage Output, Output resistance : 2kΩ,Current sink:Max. 20mA, Residual voltage : Max. 0.4V DC at 10mA current sink. •Open Collector Output,Applied Voltage : Max. 30V DC,Current sink : Max. 20mA,Residual voltage:Max. 0.4V DC at 20mA current sink. •Line Driver Output,Conformed to RS-422A of EIA Standards.	
Max. Response Frequency	100kHz	
Max. Operating Speed (Note3)	6,000r/min{rpm}(Note5)	
Signal Rise and Fall Time	Signal rise time : Max. 0.5 μs,Fall time:Max. 0.1 μs (For line driver output, both rise and fall time are 0.1 μs.)	
Power Indicator	Green LED (Light when power is supplied.)	———
Origin Indicator	Red LED (Light when Z Phase output is LOW.)	———
Ambient Temperature (Note7)	5V,12V type : −10 to +85°C , 24V type , −10 to +60°C Storage : −25 to +100°C	
Ambient humidity	35 to 85%RH(with no dew or ice condensation)	
Protection	———	IP65 for Standard shaft type
Allowable shaft Loading — Radial	19.6N{2kgf}	Standard shaft type : 53.9N{5.5kgf} Hollow shaft type : 29.4N{3kgf}
Allowable shaft Loading — Axial	9.8N{1kgf}	Standard shaft type : 39.2N{4kgf} Hollow shaft type : 19.6N{2kgf}

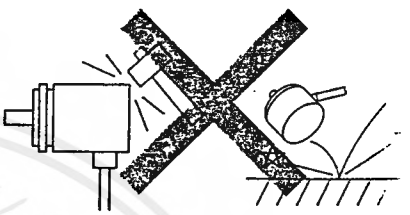
(Note1): The type with commutation signal is equipped with 7 core-shielded twist pair cabtyre cable 500mm.
(Note2): Electrical max. operating speed can be computed by the following:

$$\text{Electrical max. operating speed(rpm)} = \frac{100\times10^3\text{(Hz)}}{\text{Resolution (P/R)}}\times 60$$

(Note3): Note that the mechanical max. operating speed cannot exceed 6,000(rpm).
(Note4): If ambient temperature exceeds 70°C,max. operating speed, with which a continuous revolution is secured,becomes 3,000 rpm.(For ORE-48 standard shaft type only)
(Note5): At 25°C ambient temperature.
(Note6): At ambient atmosphere and shaft/coupling section.

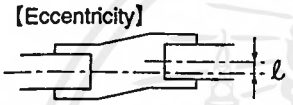
3.CAUTIONS

<INSTALLATION>

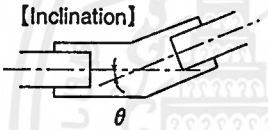


- Do not drop encoder on the floor or subject the cable to excessive stress.
- For the connection with the shaft of other equipment,use a coupling which is available as optional.(For standard shaft type only)
- Do not bend or deform the coupling.

【Eccentricity】



【Inclination】



Couplings

Model No.	Mounting tolerance		Weight(g)
	Eccentricity (mm)	Inclination(°)	
CP6-6	0.4 <0.2>	4 <2>	4
CP6-6A	0.8 <0.4>	8 <4>	43
CP6-6B	0.25 <0.15>	5 <3>	13

(Note) : If revolution is 2000 r/min or over, use figures in < >.

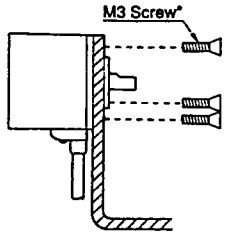
- When mount encoder using thread screws, clamping torque must be around 0.49N·m{5kgf·cm}.
- When wiring after mounted,do not give the cable a strain more than 29.4N{3kgf},nor stress on encoder or its shaft.
- Care should be taken that encoder is not exposed to water or oil, except the ORE-48 standard shaft type models.
- When rotating the encoder clockwise or counterclock-wise, pay attention to the mounting direction and addition-subtraction direction.
- When driving the encoder by the Z phase setting at the cam origin of the connecting device, be sure to mount the coupling in the Z phase mode.
- Avoid loose mounting screw by giving a proper screw-securing treatment.

Pioneer in Sensor System
SUNX
SUNX TRADING CO., LTD.
Meiji-seimei Bldg. 8th Fl., 2-27-4 Nishi-Gotanda, Shinagawa-Ku, Tokyo, 141, Japan
Phone : 03-3495-2601 FAX : 03-3495-2602 Telex : J24411 SUNXTRADCO
SUNX Limited
2431-1 Ushiyama-cho, Kasugai-shi, Aichi, 486, Japan
Phone : 0568-33-7211 FAX : 0568-33-2631

PRINTED IN JAPAN

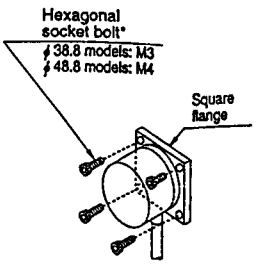
Standard shaft type

•Direct mounting



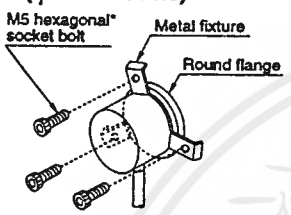
•Clamping torque:Approx. 0.49N·m {5kgf·cm}.
*: Please prepare M3 screws locally.

•Mounting by square flange



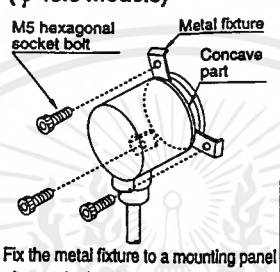
*: Please prepare hexagonal socket bolts locally.

•Mounting by round flange (φ38.8 models)



Engage the encoder to the flange after the flange is fixed by the metal fixture.
*: Please prepare M5 hexagonal socket bolts locally.

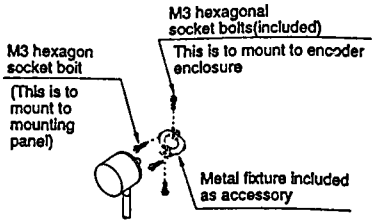
•Mounting by MS-ORE48N (φ48.8 models)



Fix the metal fixture to a mounting panel after ratcheting the metal fixture at encoder's concave part.
*: Please prepare M5 hexagonal socket bolts locally.

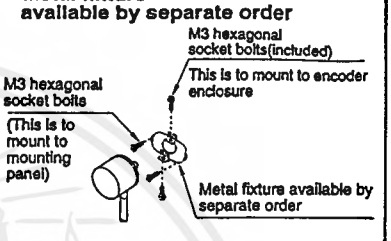
Hollow shaft type

•Metal fixture included as sccessory



①Align the metal fixture to the mounting panel.
②Align the metal fixture to the encoder.
③Tighten the encoder.

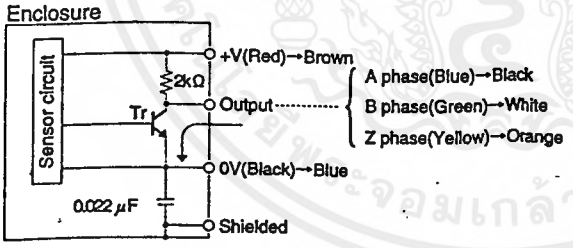
•Metal fixture available by separate order



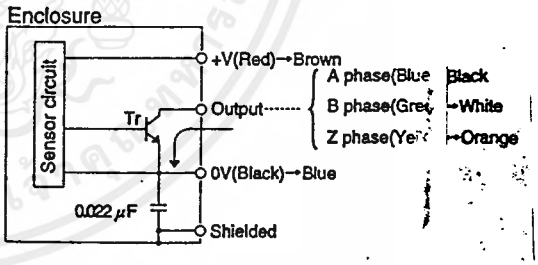
①Align the metal fixture to the encoder.
②Fix to the mounting panel with the metal fixture on.

4.WIRING DIAGRAM (Color of lead wires has been altered according to the revision of JIS standards)

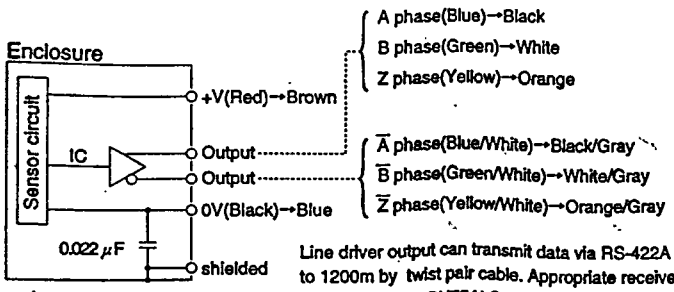
•Voltage output



•Open collector output



•Line driver output

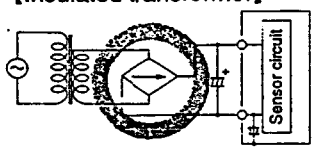


Line driver output can transmit data via RS-422A interface. Capability is extendable to 1200m by twist pair cable. Appropriate receivers are the following 6 types.
SN75157, SN75ALS193, SN75ALS195,
AM26LS32A, MC3486, μA9637A

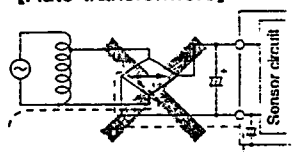
<WIRING>

●ORE series employs a line-bypass capacitor system to enhanse electrical noise resistance.If there is any equipment near the sensor such as supersonic welding machine which produces a high frequency noise,and in addition,if sensor mounting framework is electrically conductive,an insulation is required between the sensor and the framework.

【Insulated transformer】

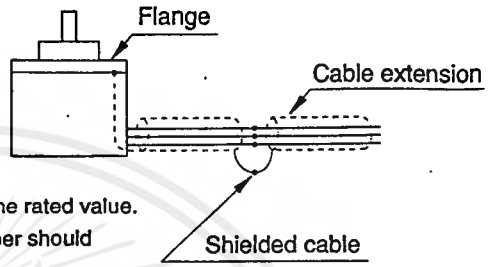


【Auto-transformers】



Beside,do not use a power source with an auto-trans former as it may resulf in a failure to danger if any fault happen at internal circuit.

●Do not connect the shielded cable of the encoder to circuit 0V or GND.Leave the shielded cable unconnected by taping it with insulation tape or similar material. When extending cables,leave the extended cable unconnected.



●Check voltage fluctuation so that it does not exceed the rated value.

●If the power source is subject to surge,a surge absorber should be connected.

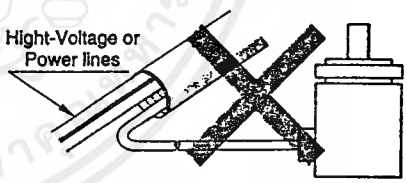
●Make the cable length as short as possible so as to avoid picking up electric noise.If a rather long cable must be used or the encoder must be installed in an electrically hazardous area, employ a Schmitt circuit or a similar device to shape the waveform.

●Do not use the encoder output signal for 0.1s immediately after the power is supplied.

●Make wiring connections after the connections are completely understood. Mis-wiring may cause a fault of an internal circuit.

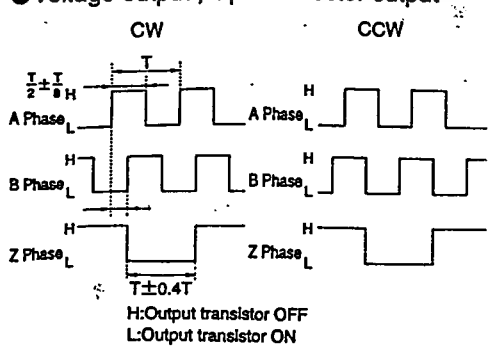
●To extend the cable ,it is recommended to use line driver output type encoders which are suitable for long distance transmission. For cable extension,use a twisted shield cable and a line receiver conforming to RS-422A in the receiving circuit. Take voltage droppage into consideration.

●Do not run sensor cables toghther with high-voltage lines, power lines or put them toghther in the same raceway. This warning should be strictly observed to prevent malfunctions caused by inductive interference.

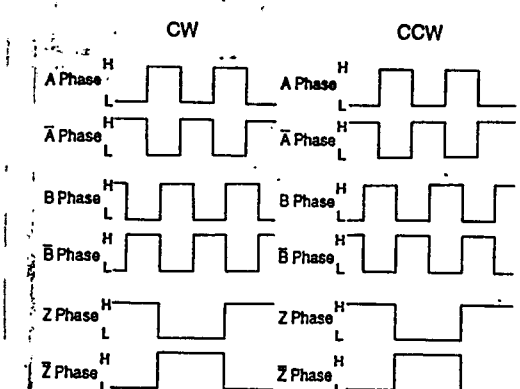


5.Output signal wave form

●Voltage output , Open collector output



●Line driver output



National Semiconductor

February 1995

LM628/LM629 Precision Motion Controller

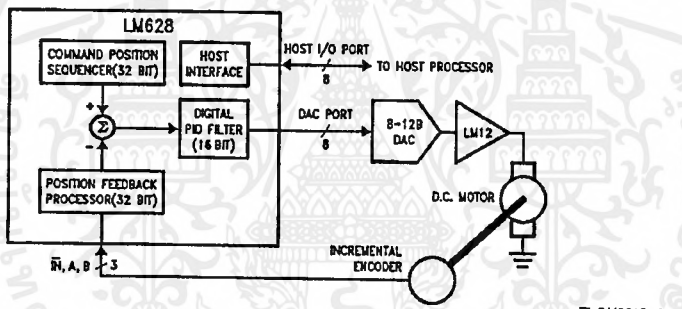
General Description

The LM628/LM629 are dedicated motion-control processors designed for use with a variety of DC and brushless DC servo motors, and other servomechanisms which provide a quadrature incremental position feedback signal. The parts perform the intensive, real-time computational tasks required for high performance digital motion control. The host control software interface is facilitated by a high-level command set. The LM628 has an 8-bit output which can drive either an 8-bit or a 12-bit DAC. The components required to build a servo system are reduced to the DC motor/actuator, an incremental encoder, a DAC, a power amplifier, and the LM628. An LM629-based system is similar, except that it provides an 8-bit PWM output for directly driving H-switches. The parts are fabricated in NMOS and packaged in a 28-pin dual in-line package or a 24-pin surface mount package (LM629 only). Both 6 MHz and 8 MHz maximum frequency versions are available with the suffixes -6 and -8, respectively, used to designate the versions. They incorporate an SDA core processor and cells designed by SDA.

Features

- 32-bit position, velocity, and acceleration registers
- Programmable digital PID filter with 16-bit coefficients
- Programmable derivative sampling interval
- 8- or 12-bit DAC output data (LM628)
- 8-bit sign-magnitude PWM output data (LM629)
- Internal trapezoidal velocity profile generator
- Velocity, target position, and filter parameters may be changed during motion
- Position and velocity modes of operation
- Real-time programmable host interrupts
- 8-bit parallel asynchronous host interface
- Quadrature incremental encoder interface with index pulse input
- Available in a 28-pin dual in-line package or a 24-pin surface mount package (LM629 only)

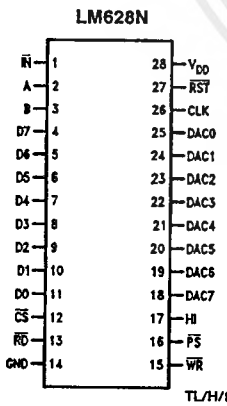
TRI-STATE® is a registered trademark of National Semiconductor Corporation.



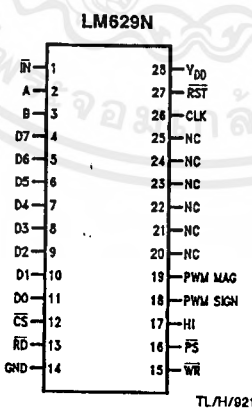
TL/H/9219–1

FIGURE 1. Typical System Block Diagram

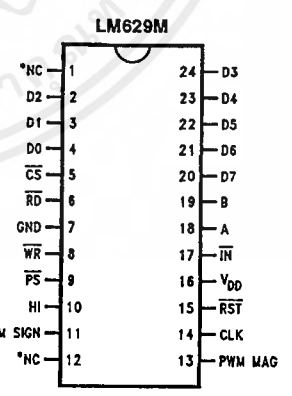
Connection Diagrams



TL/H/9219–2



TL/H/9219–3



*Do not connect.

TL/H/9219–21

Order Number LM629M-6, LM629M-8, LM628N-6, LM628N-8, LM629N-6 or LM629N-8
See NS Package Number M24B or N28B

RRD-B30M115/Printed in U. S. A.

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Voltage at Any Pin with Respect to GND	−0.3V to +7.0V
Ambient Storage Temperature	−65°C to +150°C
Lead Temperature	
28-pin Dual In-Line Package (Soldering, 4 sec.)	260°C
24-pin Surface Mount Package (Soldering, 10 sec.)	300°C
Maximum Power Dissipation ($T_A \leq 85°C$, Note 2)	605 mW
ESD Tolerance (C_{ZAP} = 120 pF, R_{ZAP} = 1.5k)	2000V

Operating Ratings

Temperature Range	$-40°C < T_A < +85°C$
Clock Frequency:	
LM628N-6, LM629N-6, LM629M-6	1.0 MHz < f_{CLK} < 6.0 MHz
LM628N-8, LM629N-8, LM629M-8	1.0 MHz < f_{CLK} < 8.0 MHz
V_{DD} Range	$4.5V < V_{DD} < 5.5V$

DC Electrical Characteristics (V_{DD} and T_A per Operating Ratings; f_{CLK} = 6 MHz)

Symbol	Parameter	Conditions	Tested Limits		Units
			Min	Max	
I_{DD}	Supply Current	Outputs Open		110	mA
INPUT VOLTAGES					
V_{IH}	Logic 1 Input Voltage		2.0		V
V_{IL}	Logic 0 Input Voltage			0.8	V
I_{IN}	Input Currents	$0 \leq V_{IN} \leq V_{DD}$	−10	10	μA
OUTPUT VOLTAGES					
V_{OH}	Logic 1	I_{OH} = −1.6 mA	2.4		V
V_{OL}	Logic 0	I_{OL} = 1.6 mA		0.4	V
I_{OUT}	TRI-STATE® Output Leakage Current	$0 \leq V_{OUT} \leq V_{DD}$	−10	10	μA

AC Electrical Characteristics

(V_{DD} and T_A per Operating Ratings; f_{CLK} = 6 MHz; C_{LOAD} = 50 pF; Input Test Signal $t_r = t_f$ = 10 ns)

Timing Interval	T#	Tested Limits		Units
		Min	Max	
ENCODER AND INDEX TIMING (See *Figure 2*)				
Motor-Phase Pulse Width	T1	$\frac{16}{f_{CLK}}$		μs
Dwell-Time per State	T2	$\frac{8}{f_{CLK}}$		μs
Index Pulse Setup and Hold (Relative to A and B Low)	T3	0		μs
CLOCK AND RESET TIMING (See *Figure 3*)				
Clock Pulse Width				
LM628N-6, LM629N-6, LM629M-6	T4	78		ns
LM628N-8, LM629N-8, LM629M-8	T4	57		ns
Clock Period				
LM628N-6, LM629N-6, LM629M-6	T5	166		ns
LM628N-8, LM629N-8, LM629M-8	T5	125		ns
Reset Pulse Width	T6	$\frac{8}{f_{CLK}}$		μs

AC Electrical Characteristics (Continued)

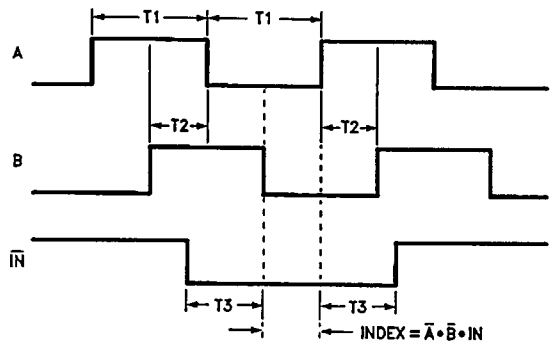
(V_{DD} and T_A per Operating Ratings; f_{CLK} = 6 MHz; C_{LOAD} = 50 pF; Input Test Signal $t_r = t_f$ = 10 ns)

Timing Interval	T#	Tested Limits		Units
		Min	Max	
STATUS BYTE READ TIMING (See *Figure 4*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Read Data Access Time	T10		180	ns
Read Data Hold Time	T11	0		ns
$\overline{RD}$ High to Hi-Z Time	T12		180	ns
COMMAND BYTE WRITE TIMING (See *Figure 5*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Busy Bit Delay	T13		(Note 3)	ns
$\overline{WR}$ Pulse Width	T14	100		ns
Write Data Setup Time	T15	50		ns
Write Data Hold Time	T16	120		ns
DATA WORD READ TIMING (See *Figure 6*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Read Data Access Time	T10		180	ns
Read Data Hold Time	T11	0		ns
$\overline{RD}$ High to Hi-Z Time	T12		180	ns
Busy Bit Delay	T13		(Note 3)	ns
Read Recovery Time	T17	120		ns
DATA WORD WRITE TIMING (See *Figure 7*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Busy Bit Delay	T13		(Note 3)	ns
$\overline{WR}$ Pulse Width	T14	100		ns
Write Data Setup Time	T15	50		ns
Write Data Hold Time	T16	120		ns
Write Recovery Time	T18	120		ns

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond the above Operating Ratings.

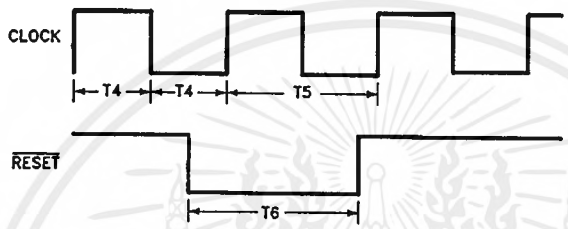
Note 2: When operating at ambient temperatures above 70°C, the device must be protected against excessive junction temperatures. Mounting the package on a printed circuit board having an area greater than three square inches and surrounding the leads and body with wide copper traces and large, uninterrupted areas of copper, such as a ground plane, suffices. The 28-pin DIP (N) and the 24-pin surface mount package (M) are molded plastic packages with solid copper lead frames. Most of the heat generated at the die flows from the die, through the copper lead frame, and into copper traces on the printed circuit board. The copper traces act as a heat sink. Double-sided or multi-layer boards provide heat transfer characteristics superior to those of single-sided boards.

Note 3: In order to read the busy bit, the status byte must first be read. The time required to read the busy bit far exceeds the time the chip requires to set the busy bit. It is, therefore, impossible to test actual busy bit delay. The busy bit is guaranteed to be valid as soon as the user is able to read it.



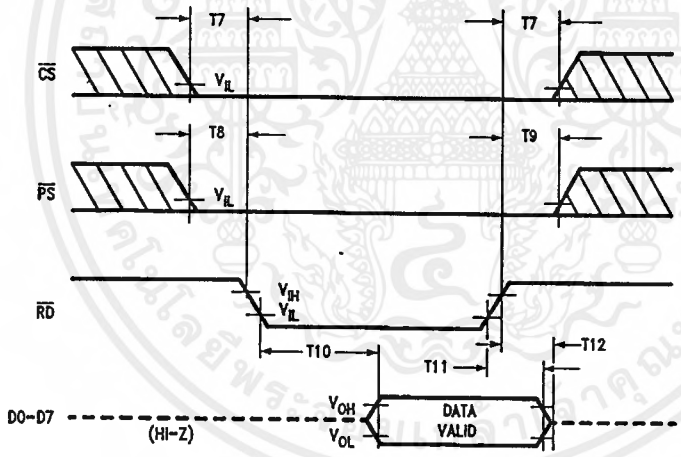
TL/H/9219–4

FIGURE 2. Quadrature Encoder Input Timing



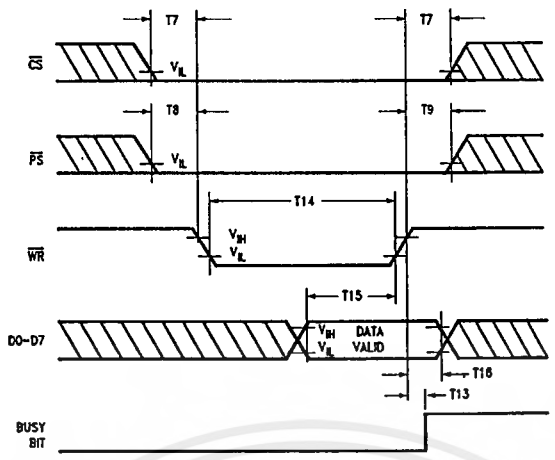
TL/H/9219–5

FIGURE 3. Clock and Reset Timing



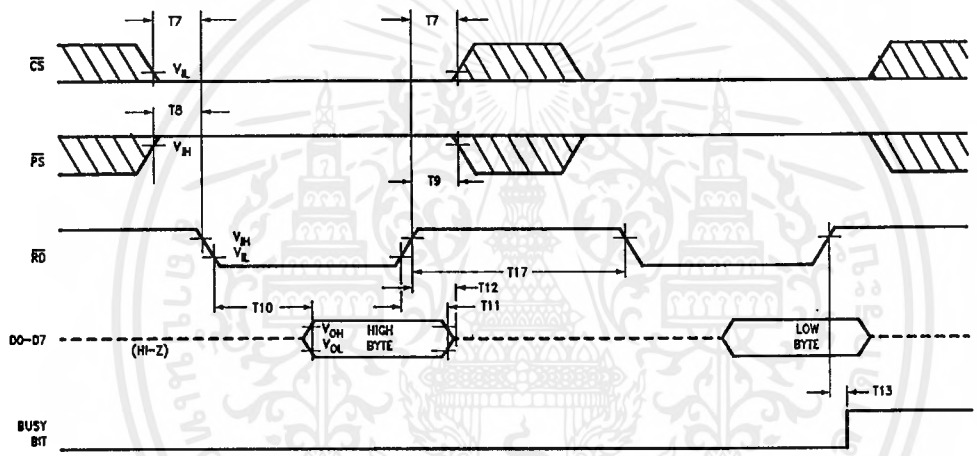
TL/H/9219–6

FIGURE 4. Status Byte Read Timing



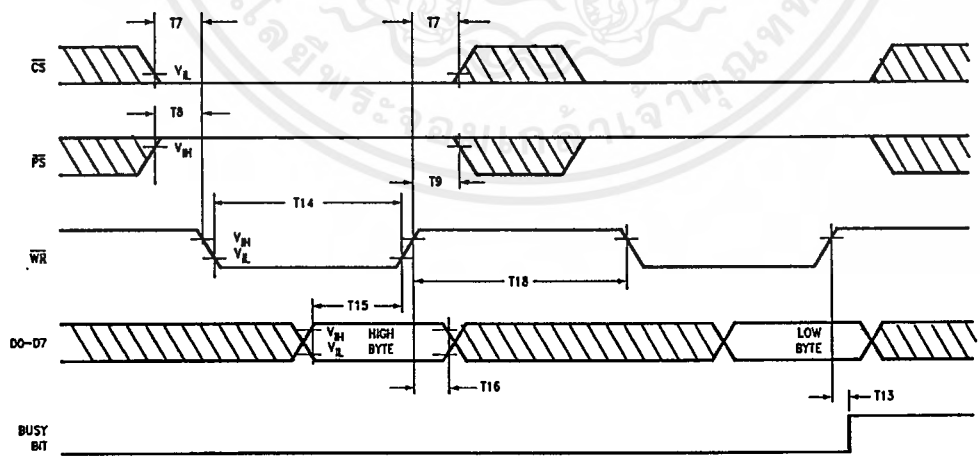
TL/H/9219-7

FIGURE 5. Command Byte Write Timing



TL/H/9219-8

FIGURE 6. Data Word Read Timing



TL/H/9219-9

FIGURE 7. Data Word Write Timing

Pinout Description

(See Connection Diagrams) Pin numbers for the 24-pin surface mount package are indicated in parentheses.

Pin 1 (17), Index ($\overline{IN}$) Input: Receives optional index pulse from the encoder. Must be tied high if not used. The index position is read when Pins 1, 2, and 3 are low.

Pins 2 and 3 (18 and 19), Encoder Signal (A, B) Inputs: Receive the two-phase quadrature signals provided by the incremental encoder. When the motor is rotating in the positive ("forward") direction, the signal at Pin 2 leads the signal at Pin 3 by 90 degrees. Note that the signals at Pins 2 and 3 must remain at each encoder state (See *Figure 9*) for a minimum of 8 clock periods in order to be recognized. Because of a four-to-one resolution advantage gained by the method of decoding the quadrature encoder signals, this corresponds to a maximum encoder-state capture rate of 1.0 MHz (f_{CLK} = 8.0 MHz) or 750 kHz (f_{CLK} = 6.0 MHz). For other clock frequencies the encoder signals must also remain at each state a minimum of 8 clock periods.

Pins 4 to 11 (20 to 24 and 2 to 4), Host I/O Port (D0 to D7): Bi-directional data port which connects to host computer/processor. Used for writing commands and data to the LM628, and for reading the status byte and data from the LM628, as controlled by $\overline{CS}$ (Pin 12), $\overline{PS}$ (Pin 16), $\overline{RD}$ (Pin 13), and $\overline{WR}$ (Pin 15).

Pin 12 (5), Chip Select ($\overline{CS}$) Input: Used to select the LM628 for writing and reading operations.

Pin 13 (6), Read ($\overline{RD}$) Input: Used to read status and data.

Pin 14 (7), Ground (GND): Power-supply return pin.

Pin 15 (8), Write ($\overline{WR}$) Input: Used to write commands and data.

Pin 16 (9), Port Select ($\overline{PS}$) Input: Used to select command or data port. Selects command port when low, data port when high. The following modes are controlled by Pin 16:

1. Commands are written to the command port (Pin 16 low),
2. Status byte is read from command port (Pin 16 low), and
3. Data is written and read via the data port (Pin 16 high).

Pin 17 (10), Host Interrupt (HI) Output: This active-high signal alerts the host (via a host interrupt service routine) that an interrupt condition has occurred.

Pins 18 to 25, DAC Port (DAC0 to DAC7): Output port which is used in three different modes:

1. LM628 (8-bit output mode): Outputs latched data to the DAC. The MSB is Pin 18 and the LSB is Pin 25.
2. LM628 (12-bit output mode): Outputs two, multiplexed 6-bit words. The less-significant word is output first. The MSB is on Pin 18 and the LSB is on Pin 23. Pin 24 is used to demultiplex the words; Pin 24 is low for the less-significant word. The positive-going edge of the signal on Pin 25 is used to strobe the output data. *Figure 8* shows the timing of the multiplexed signals.
3. LM629 (sign/magnitude outputs): Outputs a PWM sign signal on Pin 18 (11 for surface mount), and a PWM magnitude signal on Pin 19 (13 for surface mount). Pins 20 to 25 are not used in the LM629. *Figure 11* shows the PWM output signal format.

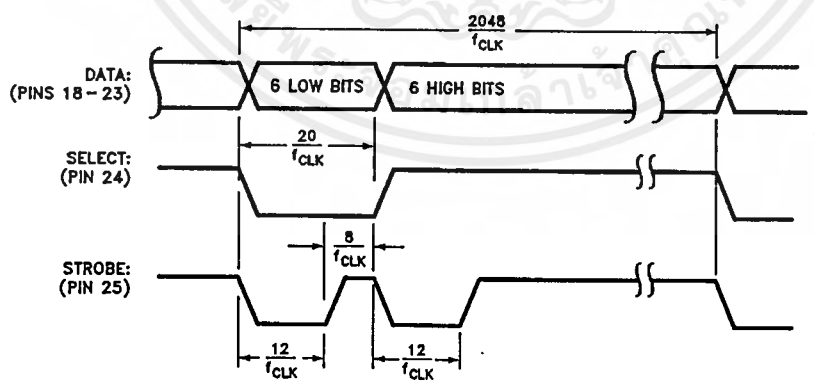
Pin 26 (14), Clock (CLK) Input: Receives system clock.

Pin 27 (15), Reset ($\overline{RST}$) Input: Active-low, positive-edge triggered, resets the LM628 to the internal conditions shown below. Note that the reset pulse must be logic low for a minimum of 8 clock periods. Reset does the following:

1. Filter coefficient and trajectory parameters are zeroed.
2. Sets position error threshold to maximum value (7FFF hex), and effectively executes command LPEI.
3. The SBPA/SBPR interrupt is masked (disabled).
4. The five other interrupts are unmasked (enabled).
5. Initializes current position to zero, or "home" position.
6. Sets derivative sampling interval to 2048/f_{CLK} or 256 μs for an 8.0 MHz clock.
7. DAC port outputs 800 hex to "zero" a 12-bit DAC and then reverts to 80 hex to "zero" an 8-bit DAC.

Immediately after releasing the reset pin from the LM628, the status port should read '00'. If the reset is successfully completed, the status word will change to hex '84' or 'C4' within 1.5 ms. If the status word has not changed from hex '00' to '84' or 'C4' within 1.5 ms, perform another reset and repeat the above steps. To be certain that the reset was properly performed, execute a **RSTI** command. If the chip has reset properly, the status byte will change from hex '84' or 'C4' to hex '80' or 'C0'. If this does not occur, perform another reset and repeat the above steps.

Pin 28 (16), Supply Voltage (V_{DD}): Power supply voltage (+5V).



TL/H/9219–10

FIGURE 8. 12-Bit Multiplexed Output Timing

Theory of Operation

INTRODUCTION

The typical system block diagram (See *Figure 1*) illustrates a servo system built using the LM628. The host processor communicates with the LM628 through an I/O port to facilitate programming a trapezoidal velocity profile and a digital compensation filter. The DAC output interfaces to an external digital-to-analog converter to produce the signal that is power amplified and applied to the motor. An incremental encoder provides feedback for closing the position servo loop. The trapezoidal velocity profile generator calculates the required trajectory for either position or velocity mode of operation. In operation, the LM628 subtracts the actual position (feedback position) from the desired position (profile generator position), and the resulting position error is processed by the digital filter to drive the motor to the desired position. Table I provides a brief summary of specifications offered by the LM628/LM629:

POSITION FEEDBACK INTERFACE

The LM628 interfaces to a motor via an incremental encoder. Three inputs are provided: two quadrature signal inputs, and an index pulse input. The quadrature signals are used to keep track of the absolute position of the motor. Each time a logic transition occurs at one of the quadrature inputs, the LM628 internal position register is incremented or decremented accordingly. This provides four times the resolution over the number of lines provided by the encoder. See *Figure 9*. Each of the encoder signal inputs is synchronized with the LM628 clock.

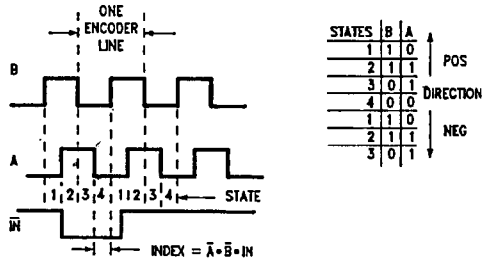
The optional index pulse output provided by some encoders assumes the logic-low state once per revolution. If the LM628 is so programmed by the user, it will record the absolute motor position in a dedicated register (the index register) at the time when all three encoder inputs are logic low.

If the encoder does not provide an index output, the LM628 index input can also be used to record the home position of the motor. In this case, typically, the motor will close a switch which is arranged to cause a logic-low level at the index input, and the LM628 will record motor position in the index register and alert (interrupt) the host processor. Permanently grounding the index input will cause the LM628 to malfunction.

TABLE I. System Specifications Summary

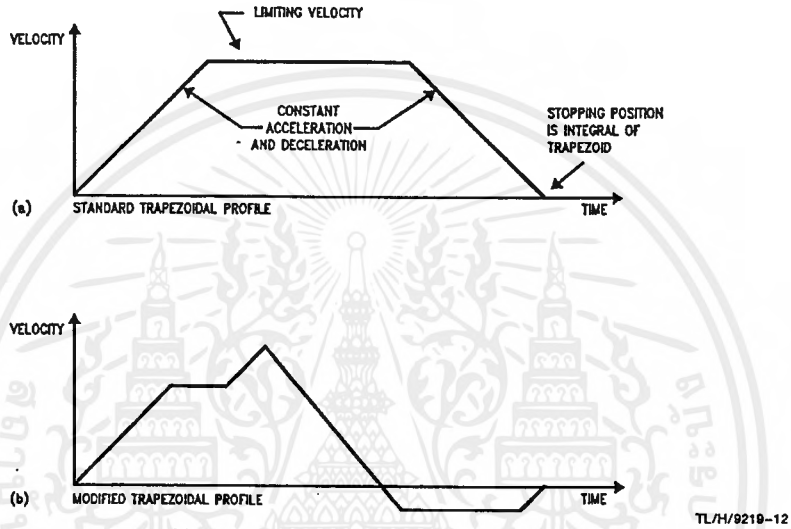
Position Range	−1,073,741,824 to 1,073,741,823 counts
Velocity Range	0 to 1,073,741,823/2^{16} counts/sample; ie, 0 to 16,383 counts/sample, with a resolution of $1/2^{16}$ counts/sample
Acceleration Range	0 to 1,073,741,823/2^{16} counts/sample/sample; ie, 0 to 16,383 counts/sample/sample, with a resolution of $1/2^{16}$ counts/sample/sample
Motor Drive Output	LM628: 8-bit parallel output to DAC, or 12-bit multiplexed output to DAC LM629: 8-bit PWM sign/magnitude signals
Operating Modes	Position and Velocity
Feedback Device	Incremental Encoder (quadrature signals; support for index pulse)
Control Algorithm	Proportional Integral Derivative (PID) (plus programmable integration limit)
Sample Intervals	Derivative Term: Programmable from $2048/f_{CLK}$ to $(2048 * 256)/f_{CLK}$ in steps of $2048/f_{CLK}$ (256 to 65,536 μs for an 8.0 MHz clock). Proportional and Integral: $2048/f_{CLK}$

Theory of Operation (Continued)



TL/H/9219–11

FIGURE 9. Quadrature Encoder Signals



TL/H/9219–12

FIGURE 10. Typical Velocity Profiles

VELOCITY PROFILE (TRAJECTORY) GENERATION

The trapezoidal velocity profile generator computes the desired position of the motor versus time. In the position mode of operation, the host processor specifies acceleration, maximum velocity, and final position. The LM628 uses this information to affect the move by accelerating as specified until the maximum velocity is reached or until deceleration must begin to stop at the specified final position. The deceleration rate is equal to the acceleration rate. At any time during the move the maximum velocity and/or the target position may be changed, and the motor will accelerate or decelerate accordingly. *Figure 10* illustrates two typical trapezoidal velocity profiles. *Figure 10 (a)* shows a simple trapezoid, while *Figure 10 (b)* is an example of what the trajectory looks like when velocity and position are changed at different times during the move.

When operating in the velocity mode, the motor accelerates to the specified velocity at the specified acceleration rate and maintains the specified velocity until commanded to stop. The velocity is maintained by advancing the desired position at a constant rate. If there are disturbances to the motion during velocity mode operation, the long-time average velocity remains constant. If the motor is unable to maintain the specified velocity (which could be caused by a locked rotor, for example), the desired position will continue to be increased, resulting in a very large position error. If this condition goes undetected, and the impeding force on the motor is subsequently released, the motor could reach a very high velocity in order to catch up to the desired position (which is still advancing as specified). This condition is easily detected; see commands LPEI and LPES.

All trajectory parameters are 32-bit values. Position is a signed quantity. Acceleration and velocity are specified as 16-bit, positive-only integers having 16-bit fractions. The integer portion of velocity specifies how many counts per sampling interval the motor will traverse. The fractional portion designates an additional fractional count per sampling interval. Although the position resolution of the LM628 is limited to integer counts, the fractional counts provide increased average velocity resolution. Acceleration is treated in the same manner. Each sampling interval the commanded acceleration value is added to the current desired velocity to generate a new desired velocity (unless the command velocity has been reached).

One determines the trajectory parameters for a desired move as follows. If, for example, one has a 500-line shaft encoder, desires that the motor accelerate at one revolution per second per second until it is moving at 600 rpm, and then decelerate to a stop at a position exactly 100 revolutions from the start, one would calculate the trajectory parameters as follows:

Theory of Operation (Continued)

let P = target position (units = encoder counts)

let R = encoder lines * 4 (system resolution)

then R = 500 * 4 = 2000

and P = 2000 * desired number of revolutions

P = 2000 * 100 revs = 200,000 counts (value to load)

P (coding) = 00030D40 (hex code written to LM628)

let V = velocity (units = counts/sample)

let T = sample time (seconds) = 341 μs (with 6 MHz clock)

let C = conversion factor = 1 minute/60 seconds

then V = R * T * C * desired rpm

and V = 2000 * 341E−6 * 1/60 * 600 rpm

V = 6.82 counts/sample

V (scaled) = 6.82 * 65,536 = 446,955.52

V (rounded) = 446,956 (value to load)

V (coding) = 0006D1EC (hex code written to LM628)

let A = acceleration (units = counts/sample/sample)

A = R * T * T * desired acceleration (rev/sec/sec)

then A = 2000 * 341E−6 * 341E-6 * 1 rev/sec/sec

and A = 2.33E−4 counts/sample/sample

A (scaled) = 2.33E−4 * 65,536 = 15.24

A (rounded) = 15 (value to load)

A (coding) = 0000000F (hex code written to LM628)

The above position, velocity, and acceleration values must be converted to binary codes to be loaded into the LM628. The values shown for velocity and acceleration must be multiplied by 65,536 (as shown) to adjust for the required integer/fraction format of the input data. Note that after scaling the velocity and acceleration values, literal fractional data cannot be loaded; the data must be rounded and converted to binary. The factor of four increase in system resolution is due to the method used to decode the quadrature encoder signals, see *Figure 9*.

PID COMPENSATION FILTER

The LM628 uses a digital Proportional Integral Derivative (PID) filter to compensate the control loop. The motor is held at the desired position by applying a restoring force to the motor that is proportional to the position error, plus the integral of the error, plus the derivative of the error. The following discrete-time equation illustrates the control performed by the LM628:

$$u(n) = kp*e(n) + ki\sum_{N=0}^{n} e(n) + kd[e(n') - e(n'-1)] \qquad \text{(Eq.1)}$$

where u(n) is the motor control signal output at sample time n, e(n) is the position error at sample time n, n' indicates sampling at the derivative sampling rate, and kp, ki, and kd are the discrete-time filter parameters loaded by the users.

The first term, the proportional term, provides a restoring force porportional to the position error, just as does a spring obeying Hooke's law. The second term, the integration term, provides a restoring force that grows with time, and thus ensures that the static position error is zero. If there is a constant torque loading, the motor will still be able to achieve zero position error.

The third term, the derivative term, provides a force proportional to the rate of change of position error. It acts just like viscous damping in a damped spring and mass system (like a shock absorber in an automobile). The sampling interval associated with the derivative term is user-selectable; this capability enables the LM628 to control a wider range of inertial loads (system mechanical time constants) by providing a better approximation of the continuous derivative. In general, longer sampling intervals are useful for low-velocity operations.

In operation, the filter algorithm receives a 16-bit error signal from the loop summing-junction. The error signal is saturated at 16 bits to ensure predictable behavior. In addition to being multiplied by filter coefficient kp, the error signal is added to an accumulation of previous errors (to form the integral signal) and, at a rate determined by the chosen *derivative* sampling interval, the previous error is subtracted from it (to form the derivative signal). All filter multiplications are 16-bit operations; only the bottom 16 bits of the product are used.

The integral signal is maintained to 24 bits, but only the top 16 bits are used. This scaling technique results in a more usable (less sensitive) range of coefficient ki values. The 16 bits are right-shifted eight positions and multiplied by filter coefficient ki to form the term which contributes to the motor control output. The absolute magnitude of this product is compared to coefficient il, and the lesser, appropriately signed magnitude then contributes to the motor control signal.

The derivative signal is multiplied by coefficient kd each *derivative* sampling interval. This product contributes to the motor control output *every* sample interval, independent of the user-chosen *derivative* sampling interval.

The kp, limited ki, and kd product terms are summed to form a 16-bit quantity. Depending on the output mode (wordsize), either the top 8 or top 12 bits become the motor control output signal.

LM628 READING AND WRITING OPERATIONS

The host processor writes commands to the LM628 via the host I/O port when Port Select ($\overline{PS}$) input (Pin 16) is logic low. The desired command code is applied to the parallel port line and the Write ($\overline{WR}$) input (Pin 15) is strobed. The command byte is latched into the LM628 on the rising edge of the $\overline{WR}$ input. When writing command bytes it is necessary to first read the status byte and check the state of a flag called the "busy bit" (Bit 0). If the busy bit is logic high, no command write may take place. The busy bit is never high longer than 100 μs, and typically falls within 15 μs to 25 μs.

The host processor reads the LM628 status byte in a similar manner: by strobing the Read ($\overline{RD}$) input (Pin 13) when $\overline{PS}$ (Pin 16) is low; status information remains valid as long as $\overline{RD}$ is low.

Writing and reading data to/from the LM628 (as opposed to writing commands and reading status) are done with $\overline{PS}$ (Pin 16) logic high. These writes and reads are always an integral number (from one to seven) of two-byte words, with the first byte of each word being the more significant. Each byte requires a write ($\overline{WR}$) or read ($\overline{RD}$) strobe. When transferring data words (byte-pairs), it is necessary to first read the status byte and check the state of the busy bit. When the

Theory of Operation (Continued)

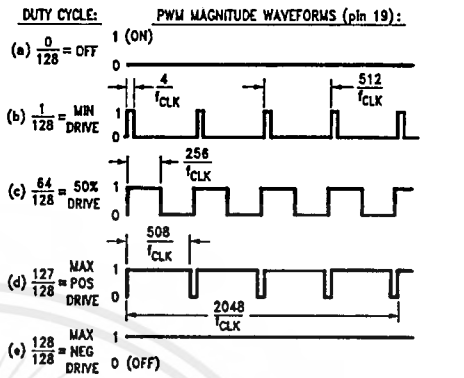
busy bit is logic low, the user may then sequentially transfer both bytes comprising a data word, but the busy bit must again be checked and found to be low before attempting to transfer the next byte pair (when transferring multiple words). Data transfers are accomplished via LM628-internal interrupts (which are not nested); the busy bit informs the host processor when the LM628 may not be interrupted for data transfer (or a command byte). If a command is written when the busy bit is high, the command will be ignored.

The busy bit goes high immediately after writing a command byte, or reading or writing a second byte of data (See *Figures 5* thru *7*).

MOTOR OUTPUTS

The LM628 DAC output port can be configured to provide either a latched eight-bit parallel output or a multiplexed 12-bit output. The 8-bit output can be directly connected to a flow-through (non-input-latching) D/A converter; the 12-bit output can be easily demultiplexed using an external 6-bit latch and an input-latching 12-bit D/A converter. The DAC output data is offset-binary coded; the 8-bit code for zero is 80 hex and the 12-bit code for zero is 800 hex. Values less than these cause a negative torque to be applied to the motor and, conversely, larger values cause positive motor torque. The LM628, when configured for 12-bit output, provides signals which control the demultiplexing process. See *Figure 8* for details.

The LM629 provides 8-bit, sign and magnitude PWM output signals for directly driving switch-mode motor-drive amplifiers. *Figure 11* shows the format of the PWM magnitude output signal.



TL/H/9219–13

Note: Sign output (pin 18) not shown

FIGURE 11. PWM Output Signal Format

TABLE II. LM628 User Command Set

Command	Type	Description	Hex	Data Bytes	Note
RESET	Initialize	Reset LM628	00	0	1
PORT8	Initialize	Select 8-Bit Output	05	0	2
PORT12	Initialize	Select 12-Bit Output	06	0	2
DFH	Initialize	Define Home	02	0	1
SIP	Interrupt	Set Index Position	03	0	1
LPEI	Interrupt	Interrupt on Error	1B	2	1
LPES	Interrupt	Stop on Error	1A	2	1
SBPA	Interrupt	Set Breakpoint, Absolute	20	4	1
SBPR	Interrupt	Set Breakpoint, Relative	21	4	1
MSKI	Interrupt	Mask Interrupts	1C	2	1
RSTI	Interrupt	Reset Interrupts	1D	2	1
LFIL	Filter	Load Filter Parameters	1E	2 to 10	1
UDF	Filter	Update Filter	04	0	1
LTRJ	Trajectory	Load Trajectory	1F	2 to 14	1
STT	Trajectory	Start Motion	01	0	3
RDSTAT	Report	Read Status Byte	None	1	1, 4
RDSIGS	Report	Read Signals Register	0C	2	1
RDIP	Report	Read Index Position	09	4	1
RDDP	Report	Read Desired Position	08	4	1
RDRP	Report	Read Real Position	0A	4	1
RDDV	Report	Read Desired Velocity	07	4	1
RDRV	Report	Read Real Velocity	0B	2	1
RDSUM	Report	Read Integration Sum	0D	2	1

Note 1: Commands may be executed "On the Fly" during motion.

Note 2: Commands not applicable to execution during motion.

Note 3: Command may be executed during motion if acceleration parameter was not changed.

Note 4: Command needs no code because the command port status-byte read is totally supported by hardware.

User Command Set

GENERAL

The following paragraphs describe the user command set of the LM628. Some of the commands can be issued alone and some require a supporting data structure. As examples, the command STT (STarT motion) does not require additional data; command LFIL (Load FILter parameters) requires additional data (derivative-term sampling interval and/or filter parameters).

Commands are categorized by function: initialization, interrupt control, filter control, trajectory control, and data reporting. The commands are listed in Table II and described in the following paragraphs. Along with each command name is its command-byte code, the number of accompanying data bytes that are to be written (or read), and a comment as to whether the command is executable during motion.

Initialization Commands

The following four LM628 user commands are used primarily to initialize the system for use.

RESET COMMAND: RESET the LM628

Command Code:	00 Hex
Data Bytes:	None
Executable During Motion:	Yes

This command (and the hardware reset input, Pin 27) results in setting the following data items to zero: filter coefficients and their input buffers, trajectory parameters and their input buffers, and the motor control output. A zero motor control output is a half-scale, offset-binary code: (80 hex for the 8-bit output mode; 800 hex for 12-bit mode). During reset, the DAC port outputs 800 hex to "zero" a 12-bit DAC and reverts to 80 hex to "zero" an 8-bit DAC. The command also clears five of the six interrupt masks (only the SBPA/SBPR interrupt is masked), sets the output port size to 8 bits, and defines the current absolute position as home. Reset, which may be executed at any time, will be completed in less than 1.5 ms. Also see commands PORT8 and PORT12.

PORT8 COMMAND: Set Output PORT Size to 8 Bits

Command Code:	05 Hex
Data Bytes:	None
Executable During Motion:	Not Applicable

The default output port size of the LM628 is 8 bits; so the PORT8 command need not be executed when using an 8-bit DAC. This command must not be executed when using a 12-bit converter; it will result in erratic, unpredictable motor behavior. The 8-bit output port size is the required selection when using the LM629, the PWM-output version of the LM628.

PORT12 COMMAND: Set Output PORT Size to 12 Bits

Command Code:	06 Hex
Data Bytes:	None
Executable During Motion:	Not Applicable

When a 12-bit DAC is used, command PORT12 should be issued very early in the initialization process. Because use of this command is determined by system hardware, there is only one foreseen reason to execute it later: if the RESET command is issued (because an 8-bit output would then be selected as the default) command PORT12 should be immediately executed. This command must not be issued when using an 8-bit converter or the LM629, the PWM-output version of the LM628.

DFH COMMAND: DeFine Home

Command Code:	02 Hex
Data Bytes:	None
Executable During Motion:	Yes

This command declares the current position as "home", or absolute position 0 (Zero). If DFH is executed during motion it will not affect the stopping position of the on-going move unless command STT is also executed.

Interrupt Control Commands

The following seven LM628 user commands are associated with conditions which can be used to interrupt the host computer. In order for any of the potential interrupt conditions to actually interrupt the host via Pin 17, the corresponding bit in the interrupt mask data associated with command MSKI must have been set to logic high (the non-masked state).

The identity of all interrupts is made known to the host via reading and parsing the status byte. Even if all interrupts are masked off via command MSKI, the state of each condition is still reflected in the status byte. This feature facilitates polling the LM628 for status information, as opposed to interrupt driven operation.

SIP COMMAND: Set Index Position

Command Code:	03 Hex
Data Bytes:	None
Executable During Motion:	Yes

After this command is executed, the absolute position which corresponds to the occurrence of the next index pulse input will be recorded in the index register, and bit 3 of the status byte will be set to logic high. The position is recorded when both encoder-phase inputs and the index pulse input are logic low. This register can then be read by the user (see description for command RDIP) to facilitate aligning the definition of home position (see description of command DFH) with an index pulse. The user can also arrange to have the LM628 interrupt the host to signify that an index pulse has occurred. See the descriptions for commands MSKI and RSTI.

LPEI COMMAND: Load Position Error for Interrupt

Command Code:	1B Hex
Data Bytes:	Two
Data Range:	0000 to 7FFF Hex
Executable During Motion:	Yes

An excessive position error (the output of the loop summing junction) can indicate a serious system problem; e.g., a stalled rotor. Instruction LPEI allows the user to input a threshold for position error detection. Error detection occurs when the absolute magnitude of the position error exceeds the threshold, which results in bit 5 of the status byte being set to logic high. If it is desired to also stop (turn off) the motor upon detecting excessive position error, see command LPES, below. The first byte of threshold data written with command LPEI is the more significant. The user can have the LM628 interrupt the host to signify that an excessive position error has occurred. See the descriptions for commands MSKI and RSTI.

Interrupt Control Commands (Continued)

LPES COMMAND: Load Position Error for Stopping

Command Code:	1A Hex
Data Bytes:	Two
Data Range:	0000 to 7FFF Hex
Executable During Motion:	Yes

Instruction LPES is essentially the same as command LPEI above, but adds the feature of turning off the motor upon detecting excessive position error. The motor drive is not actually switched off, it is set to half-scale, the offset-binary code for zero. As with command LPEI, bit 5 of the status byte is also set to logic high. The first byte of threshold data written with command LPES is the more significant. The user can have the LM628 interrupt the host to signify that an excessive position error has occurred. See the descriptions for commands MSKI and RSTI.

SBPA COMMAND:

Command Code:	20 Hex
Data Bytes:	Four
Data Range:	C0000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

This command enables the user to set a breakpoint in terms of absolute position. Bit 6 of the status byte is set to logic high when the breakpoint position is reached. This condition is useful for signaling trajectory and/or filter parameter updates. The user can also arrange to have the LM628 interrupt the host to signify that a breakpoint position has been reached. See the descriptions for commands MSKI and RSTI.

SBPR COMMAND:

Command Code:	21 Hex
Data Bytes:	Four
Data Range:	See Text
Executable During Motion:	Yes

This command enables the user to set a breakpoint in terms of relative position. As with command SBPA, bit 6 of the status byte is set to logic high when the breakpoint position (relative to the current commanded target position) is reached. The relative breakpoint input value must be such that when this value is added to the target position the result remains within the absolute position range of the system (C0000000 to 3FFFFFFF hex). This condition is useful for signaling trajectory and/or filter parameter updates. The user can also arrange to have the LM628 interrupt the host to signify that a breakpoint position has been reached. See the descriptions for commands MSKI and RSTI.

MSKI COMMAND: MaSK Interrupts

Command Code:	1C Hex
Data Bytes:	Two
Data Range:	See Text
Executable During Motion:	Yes

The MSKI command lets the user determine which potential interrupt condition(s) will interrupt the host. Bits 1 through 6 of the status byte are indicators of the six conditions which are candidates for host interrupt(s). When interrupted, the host then reads the status byte to learn which condition(s) occurred. Note that the MSKI command is immediately followed by two data bytes. Bits 1 through 6 of the second (less significant) byte written determine the masked/unmasked status of each potential interrupt. Any zero(s) in this 6-bit field will mask the corresponding interrupt(s); any one(s) enable the interrupt(s). Other bits comprising the two bytes have no effect. The mask controls only the host interrupt process; reading the status byte will still reflect the actual conditions independent of the mask byte. See Table III.

TABLE III. Mask and Reset Bit Allocations for Interrupts

Bit Position	Function
Bits 15 thru 7	Not Used
Bit 6	Breakpoint Interrupt
Bit 5	Position-Error Interrupt
Bit 4	Wrap-Around Interrupt
Bit 3	Index-Pulse Interrupt
Bit 2	Trajectory-Complete Interrupt
Bit 1	Command-Error Interrupt
Bit 0	Not Used

RSTI COMMAND: ReSeT Interrupts

Command Code:	1D Hex
Data Bytes:	Two
Data Range:	See Text
Executable During Motion:	Yes

When one of the potential interrupt conditions of Table III occurs, command RSTI is used to reset the corresponding interrupt flag bit in the status byte. The host may reset one or all flag bits. Resetting them one at a time allows the host to service them one at a time according to a priority programmed by the user. As in the MSKI command, bits 1 through 6 of the second (less significant) byte correspond to the potential interrupt conditions shown in Table III. Also see description of RDSTAT command. Any zero(s) in this 6-bit field reset the corresponding interrupt(s). The remaining bits have no effect.

Filter Control Commands

The following two LM628 user commands are used for setting the derivative-term sampling interval, for adjusting the filter parameters as required to tune the system, and to control the timing of these system changes.

LFIL COMMAND: Load FILter Parameters

Command Code:	1E Hex
Data Bytes:	Two to Ten
Data Ranges . . .	
Filter Control Word:	See Text
Filter Coefficients:	0000 to 7FFF Hex (Pos Only)
Integration Limit:	0000 to 7FFF Hex (Pos Only)
Executable During Motion:	Yes

The filter parameters (coefficients) which are written to the LM628 to control loop compensation are: kp, ki, kd, and il (integration limit). The integration limit (il) constrains the contribution of the integration term

$$\left[ki * \sum_{N=0}^{n} e(n) \right]$$

(see Eq. 1) to values equal to or less than a user-defined maximum value; this capability minimizes integral or reset "wind-up" (an overshooting effect of the integral action). The positive-only input value is compared to the absolute

Filter Control Commands (Continued)

magnitude of the integration term; when the magnitude of integration term value exceeds il, the il value (with appropriate sign) is substituted for the integration term value.

The derivative-term sampling interval is also programmable via this command. After writing the command code, the first two data bytes that are written specify the derivative-term sampling interval and which of the four filter parameters is/are to be written via any forthcoming data bytes. The first byte written is the more significant. Thus the two data bytes constitute a filter control word that informs the LM628 as to the nature and number of any following data bytes. See Table IV.

TABLE IV. Filter Control word Bit Allocation

Bit Position	Function
Bit 15	Derivative Sampling Interval Bit 7
Bit 14	Derivative Sampling Interval Bit 6
Bit 13	Derivative Sampling Interval Bit 5
Bit 12	Derivative Sampling Interval Bit 4
Bit 11	Derivative Sampling Interval Bit 3
Bit 10	Derivative Sampling Interval Bit 2
Bit 9	Derivative Sampling Interval Bit 1
Bit 8	Derivative Sampling Interval Bit 0
Bit 7	Not Used
Bit 6	Not Used
Bit 5	Not Used
Bit 4	Not Used
Bit 3	Loading kp Data
Bit 2	Loading ki Data
Bit 1	Loading kd Data
Bit 0	Loading il Data

Bits 8 through 15 select the derivative-term sampling interval. See Table V. The user must locally save and restore these bits during successive writes of the filter control word.

Bits 4 through 7 of the filter control word are not used.

Bits 0 to 3 inform the LM628 as to whether any or all of the filter parameters are about to be written. The user may choose to update any or all (or none) of the filter parameters. Those chosen for updating are so indicated by logic one(s) in the corresponding bit position(s) of the filter control word.

The data bytes specified by and immediately following the filter control word are written in pairs to comprise 16-bit words. The order of sending the data words to the LM628 corresponds to the descending order shown in the above description of the filter control word; i.e., beginning with kp, then ki, kd and il. The first byte of each word is the more-significant byte. Prior to writing a word (byte pair) it is necessary to check the busy bit in the status byte for readiness. The required data is written to the primary buffers of a double-buffered scheme by the above described operations; it is not transferred to the secondary (working) registers until the UDF command is executed. This fact can be used advantageously; the user can input numerous data ahead of their actual use. This simple pipeline effect can relieve potential host computer data communications bottlenecks, and facilitates easier synchronization of multiple-axis controls.

UDF COMMAND: UpDate Filter

Command Code:	04 Hex
Data Bytes:	None
Executable During Motion:	Yes

The UDF command is used to update the filter parameters, the specifics of which have been programmed via the LFIL command. Any or all parameters (derivative-term sampling interval, kp, ki, kd, and/or il) may be changed by the appropriate command(s), but command UDF must be executed to affect the change in filter tuning. Filter updating is synchronized with the calculations to eliminate erratic or spurious behavior.

Trajectory Control Commands

The following two LM628 user commands are used for setting the trajectory control parameters (position, velocity, acceleration), mode of operation (position or velocity), and direction (velocity mode only) as required to describe a desired motion or to select the mode of a manually directed stop, and to control the timing of these system changes.

LTRJ COMMAND: Load TRaJectory Parameters

Command Code:	1F Hex
Data Bytes:	Two to Fourteen
Data Ranges . . .	
Trajectory Control Word:	See Text
Position:	C0000000 to 3FFFFFFF Hex
Velocity:	00000000 to 3FFFFFFF Hex (Pos Only)
Acceleration:	00000000 to 3FFFFFFF Hex (Pos Only)
Executable During Motion:	Conditionally, See Text

TABLE V. Derivative-Term Sampling Interval Selection Codes

	Bit Position								Selected Derivative Sampling Interval
	15	14	13	12	11	10	9	8	
	0	0	0	0	0	0	0	0	256 μs
	0	0	0	0	0	0	0	1	512 μs
	0	0	0	0	0	0	1	0	768 μs
	0	0	0	0	0	0	1	1	1024 μs, etc . . .
thru	1	1	1	1	1	1	1	1	65,536 μs

Note: Sampling intervals shown are when using an 8.0 MHz clock. The 256 corresponds to 2048/8 MHz; sample intervals must be scaled for other clock frequencies.

Trajectory Control Commands (Continued)

The trajectory control parameters which are written to the LM628 to control motion are: acceleration, velocity, and position. In addition, indications as to whether these three parameters are to be considered as absolute or relative inputs, selection of velocity mode and direction, and manual stopping mode selection and execution are programmable via this command. After writing the command code, the first two data bytes that are written specify which parameter(s) is/are being changed. The first byte written is the more significant. Thus the two data bytes constitute a trajectory control word that informs the LM628 as to the nature and number of any following data bytes. See Table VI.

TABLE VI. Trajectory Control Word Bit Allocation

Bit Position	Function
Bit 15	Not Used
Bit 14	Not Used
Bit 13	Not Used
Bit 12	Forward Direction (Velocity Mode Only)
Bit 11	Velocity Mode
Bit 10	Stop Smoothly (Decelerate as Programmed)
Bit 9	Stop Abruptly (Maximum Deceleration)
Bit 8	Turn Off Motor (Output Zero Drive)
Bit 7	Not Used
Bit 6	Not Used
Bit 5	Acceleration Will Be Loaded
Bit 4	Acceleration Data Is Relative
Bit 3	Velocity Will Be Loaded
Bit 2	Velocity Data Is Relative
Bit 1	Position Will Be Loaded
Bit 0	Position Data Is Relative

Bit 12 determines the motor direction when in the velocity mode. A logic one indicates forward direction. This bit has no effect when in position mode.

Bit 11 determines whether the LM628 operates in velocity mode (Bit 11 logic one) or position mode (Bit 11 logic zero).

Bits 8 through 10 are used to select the method of *manually stopping* the motor. These bits are *not* provided for one to merely specify the desired *mode* of stopping, in position mode operations, normal stopping is always smooth and occurs automatically at the end of the specified trajectory. Under exceptional circumstances it may be desired to manually intervene with the trajectory generation process to affect a premature stop. In velocity mode operations, however, the normal means of stopping *is* via bits 8 through 10 (usually bit 10). Bit 8 is set to logic one to stop the motor by turning off motor drive output (outputting the appropriate offset-binary code to apply zero drive to the motor); bit 9 is set to one to stop the motor abruptly (at maximum available acceleration, by setting the target position equal to the current position); and bit 10 is set to one to stop the motor smoothly by using the current user-programmed acceleration value. Bits 8 through 10 are to be used *exclusively;* only one bit should be a logic one at any time.

Bits 0 through 5 inform the LM628 as to whether any or all of the trajectory controlling parameters are about to be written, and whether the data should be interpreted as absolute or relative. The user may choose to update any or all (or none) of the trajectory parameters. Those chosen for updating are so indicated by logic one(s) in the corresponding bit position(s). Any parameter may be changed while the motor is in motion; however, if acceleration is changed then the next STT command must not be issued until the LM628 has completed the current move or has been manually stopped.

The data bytes specified by and immediately following the trajectory control word are written in pairs which comprise 16-bit words. Each data item (parameter) requires two 16-bit words; the word and byte order is most-to-least significant. The order of sending the parameters to the LM628 corresponds to the descending order shown in the above description of the trajectory control word; i.e., beginning with acceleration, then velocity, and finally position.

Acceleration and velocity are 32 bits, positive only, but range only from 0 (00000000 hex) to $[2^{30}]-1$ (3FFFFFFF hex). The bottom 16 bits of both acceleration and velocity are scaled as fractional data; therefore, the least-significant integer data bit for these parameters is bit 16 (where the bits are numbered 0 through 31). To determine the coding for a given velocity, for example, one multiplies the desired velocity (in counts per sample interval) times 65,536 and converts the result to binary. The units of acceleration are counts per sample per sample. The value loaded for acceleration must not exceed the value loaded for velocity. Position is a signed, 32-bit integer, but ranges only from $-[2^{30}]$ (C0000000 hex) to $[2^{30}]-1$ (3FFFFFFF Hex).

The required data is written to the primary buffers of a double-buffered scheme by the above described operations; it is not transferred to the secondary (working) registers until the STT command is executed. This fact can be used advantageously; the user can input numerous data ahead of their actual use. This simple pipeline effect can relieve potential host computer data communications bottlenecks, and facilitates easier synchronization of multiple-axis controls.

STT COMMAND: STarT Motion Control

Command Code:	01 Hex
Data Bytes:	None
Executable During Motion:	Yes, If acceleration has not been changed

The STT command is used to execute the desired trajectory, the specifics of which have been programmed via the LTRJ command. Synchronization of multi-axis control (to within one sample interval) can be arranged by loading the required trajectory parameters for each (and every) axis and then simultaneously issuing a single STT command to all axes. This command may be executed at any time, unless the acceleration value has been changed and a trajectory has not been completed or the motor has not been manually stopped. If STT is issued during motion and acceleration has been changed, a command error interrupt will be generated and the command will be ignored.

Data Reporting Commands

The following seven LM628 user commands are used to obtain data from various registers in the LM628. Status, position, and velocity information are reported. With the exception of RDSTAT, the data is read from the LM628 data port after first writing the corresponding command to the command port.

Data Reporting Commands (Continued)

RDSTAT COMMAND: ReaD STATus Byte

Command Code: None
Byte Read: One
Data Range: See Text
Executable During Motion: Yes

The RDSTAT command is really not a command, but is listed with the other commands because it is used very frequently to control communications with the host computer. There is no identification code; it is directly supported by the hardware and may be executed at any time. The single-byte status read is selected by placing $\overline{CS}$, $\overline{PS}$ and $\overline{RD}$ at logic zero. See Table VII.

TABLE VII. Status Byte Bit Allocation

Bit Position	Function
Bit 7	Motor Off
Bit 6	Breakpoint Reached [Interrupt]
Bit 5	Excessive Position Error [Interrupt]
Bit 4	Wraparound Occurred [Interrupt]
Bit 3	Index Pulse Observed [Interrupt]
Bit 2	Trajectory Complete [Interrupt]
Bit 1	Command Error [Interrupt]
Bit 0	Busy Bit

Bit 7, the motor-off flag, is set to logic one when the motor drive output is off (at the half-scale, offset-binary code for zero). The motor is turned off by any of the following conditions: power-up reset, command RESET, excessive position error (if command LPES had been executed), or when command LTRJ is used to manually stop the motor via turning the motor off. Note that when bit 7 is set in conjunction with command LTRJ for producing a manual, motor-off stop, the actual setting of bit 7 does not occur until command STT is issued to affect the stop. Bit 7 is cleared by command STT, except as described in the previous sentence.

Bit 6, the breakpoint-reached interrupt flag, is set to logic one when the position breakpoint loaded via command SBPA or SBPR has been exceeded. The flag is functional independent of the host interrupt mask status. Bit 6 is cleared via command RSTI.

Bit 5, the excessive-position-error interrupt flag, is set to logic one when a position-error interrupt condition exists. This occurs when the error threshold loaded via command LPEI or LPES has been exceeded. The flag is functional independent of the host interrupt mask status. Bit 5 is cleared via command RSTI.

Bit 4, the wraparound interrupt flag, is set to logic one when a numerical "wraparound" has occurred. To "wraparound" means to exceed the position address space of the LM628, which could occur during velocity mode operation. If a wraparound has occurred, then position information will be in error and this interrupt helps the user to ensure position data integrity. The flag is functional independent of the host interrupt mask status. Bit 4 is cleared via command RSTI.

Bit 3, the index-pulse acquired interrupt flag, is set to logic one when an index pulse has occurred (if command SIP had been executed) and indicates that the index position register has been updated. The flag is functional independent of the host interrupt mask status. Bit 3 is cleared by command RSTI.

Bit 2, the trajectory complete interrupt flag, is set to logic one when the trajectory programmed by the LTRJ command and initiated by the STT command has been completed. Because of overshoot or a limiting condition (such as commanding the velocity to be higher than the motor can achieve), the motor may not yet be at the final commanded position. This bit is the logical OR of bits 7 and 10 of the Signals Register, see command RDSIGS below. The flag functions independently of the host interrupt mask status. Bit 2 is cleared via command RSTI.

Bit 1, the command-error interrupt flag, is set to logic one when the user attempts to read data when a write was appropriate (or vice versa). The flag is functional independent of the host interrupt mask status. Bit 1 is cleared via command RSTI.

Bit 0, the busy flag, is frequently tested by the user (via the host computer program) to determine the busy/ready status prior to writing and reading any data. Such writes and reads may be executed only when bit 0 is logic zero (not busy). Any command or data writes when the busy bit is high will be ignored. Any data reads when the busy bit is high will read the current contents of the I/O port buffers, not the data expected by the host. Such reads or writes (with the busy bit high) will not generate a command-error interrupt.

RDSIGS COMMAND: ReaD SIGnalS Register

Command Code: 0C Hex
Bytes Read: Two
Data Range: See Text
Executable During Motion: Yes

The LM628 internal "signals" register may be read using this command. The first byte read is the more significant. The less significant byte of this register (with the exception of bit 0) duplicates the status byte. See Table VIII.

TABLE VIII. Signals Register Bit Allocation

Bit Position	Function
Bit 15	Host Interrupt
Bit 14	Acceleration Loaded (But Not Updated)
Bit 13	UDF Executed (But Filter Not yet Updated)
Bit 12	Forward Direction
Bit 11	Velocity Mode
Bit 10	On Target
Bit 9	Turn Off upon Excessive Position Error
Bit 8	Eight-Bit Output Mode
Bit 7	Motor Off
Bit 6	Breakpoint Reached [Interrupt]
Bit 5	Excessive Position Error [Interrupt]
Bit 4	Wraparound Occurred [Interrupt]
Bit 3	Index Pulse Acquired [Interrupt]
Bit 2	Trajectory Complete [Interrupt]
Bit 1	Command Error [Interrupt]
Bit 0	Acquire Next Index (SIP Executed)

Bit 15, the host interrupt flag, is set to logic one when the host interrupt output (Pin 17) is logic one. Pin 17 is set to logic one when any of the six host interrupt conditions occur (if the corresponding interrupt has not been masked). Bit 15 (and Pin 17) are cleared via command RSTI.

Bit 14, the acceleration-loaded flag, is set to logic one when acceleration data is written to the LM628. Bit 14 is cleared by the STT command.

Data Reporting Commands (Continued)

Bit 13, the UDF-executed flag, is set to logic one when the UDF command is executed. Because bit 13 is cleared at the end of the sampling interval in which it has been set, this signal is very short-lived and probably not very profitable for monitoring.

Bit 12, the forward direction flag, is meaningful only when the LM628 is in velocity mode. The bit is set to logic one to indicate that the desired direction of motion is "forward"; zero indicates "reverse" direction. Bit 12 is set and cleared via command LTRJ. The actual setting and clearing of bit 12 does not occur until command STT is executed.

Bit 11, the velocity mode flag, is set to logic one to indicate that the user has selected (via command LTRJ) velocity mode. Bit 11 is cleared when position mode is selected (via command LTRJ). The actual setting and clearing of bit 11 does not occur until command STT is executed.

Bit 10, the on-target flag, is set to logic one when the trajectory generator has completed its functions for the last-issued STT command. Bit 10 is cleared by the next STT command.

Bit 9, the turn-off on-error flag, is set to logic one when command LPES is executed. Bit 9 is cleared by command LPEI.

Bit 8, the 8-bit output flag, is set to logic one when the LM628 is reset, or when command PORT8 is executed. Bit 8 is cleared by command PORT12.

Bits 0 through 7 replicate the status byte (see Table VII), with the exception of bit 0. Bit 0, the acquire next index flag, is set to logic one when command SIP is executed; it then remains set until the next index pulse occurs.

RDIP COMMAND: ReaD Index Position

Command Code:	09 Hex
Bytes Read:	Four
Data Range:	C0000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

This command reads the position recorded in the index register. Reading the index register can be part of a system error checking scheme. Whenever the SIP command is executed, the new index position minus the old index position, divided by the incremental encoder resolution (encoder lines times four), should always be an integral number. The RDIP command facilitates acquiring these data for host-based calculations. The command can also be used to identify/verify home or some other special position. The bytes are read in most-to-least significant order.

RDDP COMMAND: ReaD Desired Position

Command Code:	08 Hex
Bytes Read:	Four
Data Range:	C0000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

This command reads the instantaneous desired (current *temporal*) position output of the profile generator. This is the "setpoint" input to the position-loop summing junction. The bytes are read in most-to-least significant order.

RDRP COMMAND: ReaD Real Position

Command Code:	0A Hex
Bytes Read:	Four
Data Range:	C0000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

This command reads the current actual position of the motor. This is the feedback input to the loop summing junction. The bytes are read in most-to-least significant order.

RDDV COMMAND: ReaD Desired Velocity

Command Code:	07 Hex
Bytes Read:	Four
Data Range:	C0000001 to 3FFFFFFF
Executable During Motion:	Yes

This command reads the integer and fractional portions of the instantaneous desired (current *temporal*) velocity, as used to generate the desired position profile. The bytes are read in most-to-least significant order. The value read is properly scaled for numerical comparison with the user-supplied (commanded) velocity; however, because the two least-significant bytes represent *fractional* velocity, only the two most-significant bytes are appropriate for comparison with the data obtained via command RDRV (see below). Also note that, although the velocity *input* data is constrained to positive numbers (see command LTRJ), the data returned by command RDDV represents a *signed* quantity where negative numbers represent operation in the reverse direction.

RDRV COMMAND: ReaD Real Velocity

Command Code:	0B Hex
Bytes Read:	Two
Data Range:	C000 to 3FFF Hex, See Text
Executable During Motion:	Yes

This command reads the *integer* portion of the instantaneous actual velocity of the motor. The internally maintained fractional portion of velocity is not reported because the reported data is derived by reading the incremental encoder, which produces only integer data. For comparison with the result obtained by executing command RDDV (or the user-supplied input value), the value returned by command RDRV must be multiplied by 2^{16} (shifted left 16 bit positions). Also, as with command RDDV above, data returned by command RDRV is a *signed* quantity, with negative values representing reverse-direction motion.

RDSUM COMMAND: ReaD Integration-Term SUMmation Value

Command Code:	0D Hex
Bytes Read:	Two
Data Range:	00000 Hex to ± the Current Value of the Integration Limit
Executable During Motion:	Yes

This command reads the value to which the integration term has accumulated. The ability to read this value may be helpful in initially or adaptively tuning the system.

Typical Applications

Programming LM628 Host Handshaking (Interrupts)

A few words regarding the LM628 host handshaking will be helpful to the system programmer. As indicated in various portions of the above text, the LM628 handshakes with the host computer in two ways: via the host interrupt output (Pin 17), or via polling the status byte for "interrupt" conditions. When the hardwired interrupt is used, the status byte is also read and parsed to determine which of six possible conditions caused the interrupt.

Typical Applications (Continued)

When using the hardwired interrupt it is very important that the host interrupt service routine does not interfere with a command sequence which might have been in progress when the interrupt occurred. If the host interrupt service routine were to issue a command to the LM628 while it is in the middle of an ongoing command sequence, the ongoing command will be aborted (which could be detrimental to the application).

Two approaches exist for avoiding this problem. If one is using hardwired interrupts, they should be disabled at the host prior to issuing any LM628 command sequence, and re-enabled after each command sequence. The second approach is to avoid hardwired interrupts and poll the LM628 status byte for "interrupt" status. The status byte always reflects the interrupt-condition status, independent of whether or not the interrupts have been masked.

Typical Host Computer/Processor Interface

The LM628 is interfaced with the host computer/processor via an 8-bit parallel bus. *Figure 12* shows such an interface and a minimum system configuration.

As shown in *Figure 12*, the LM628 interfaces with the host data, address and control lines. The address lines are decoded to generate the LM628 $\overline{CS}$ input; the host address LSB directly drives the LM628 $\overline{PS}$ input. *Figure 12* also shows an 8-bit DAC and an LM12 Power Op Amp interfaced to the LM628.

LM628 and High Performance Controller (HPC) Interface

Figure 13 shows the LM628 interfaced to a National HPC High Performance Controller. The delay and logic associated with the $\overline{WR}$ line is used to effectively increase the write-data hold time of the HPC (as seen at the LM628) by causing the $\overline{WR}$ pulse to rise early. Note that the HPC CK2 output provides the clock for the LM628. The 74LS245 is used to decrease the read-data hold time, which is necessary when interfacing to fast host busses.

Interfacing a 12-Bit DAC

Figure 14 illustrates use of a 12-bit DAC with the LM628. The 74LS378 hex gated-D flip-flop and an inverter demultiplex the 12-bit output. DAC offset must be adjusted to minimize DAC linearity and monotonicity errors. Two methods exist for making this adjustment. If the DAC1210 has been socketed, remove it and temporarily connect a 15 kΩ resistor between Pins 11 and 13 of the DAC socket (Pins 2 and 6 of the LF356) and adjust the 25 kΩ potentiometer for 0V at Pin 6 of the LF356.

If the DAC is not removable, the second method of adjustment requires that the DAC1210 inputs be presented an all-zeros code. This can be arranged by commanding the appropriate move via the LM628, but with no feedback from the system encoder. When the all-zeros code is present, adjust the pot for 0V at Pin 6 of the LF356.

A Monolithic Linear Drive Using LM12 Power Op Amp

Figure 15 shows a motor-drive amplifier built using the LM12 Power Operational Amplifier. This circuit is very simple and can deliver up to 8A at 30V (using the LM12L/LM12CL). Resistors R1 and R2 should be chosen to set the gain to provide maximum output voltage consistent with maximum input voltage. This example provides a gain of 2.2, which allows for amplifier output saturation at ±22V with a ±10V input, assuming power supply voltages of ±30V. The amplifier gain should not be higher than necessary because the system is non-linear when saturated, and because gain should be controlled by the LM628. The LM12 can also be configured as a current driver, see 1987 Linear Databook, Vol. 1, p. 2–280.

Typical PWM Motor Drive Interfaces

Figure 16 shows an LM18298 dual full-bridge driver interfaced to the LM629 PWM outputs to provide a switch-mode power amplifier for driving small brush/commutator motors. *Figure 17* shows an LM621 brushless motor commutator interfaced to the LM629 PWM outputs and a discrete device switch-mode power amplifier for driving brushless DC motors.

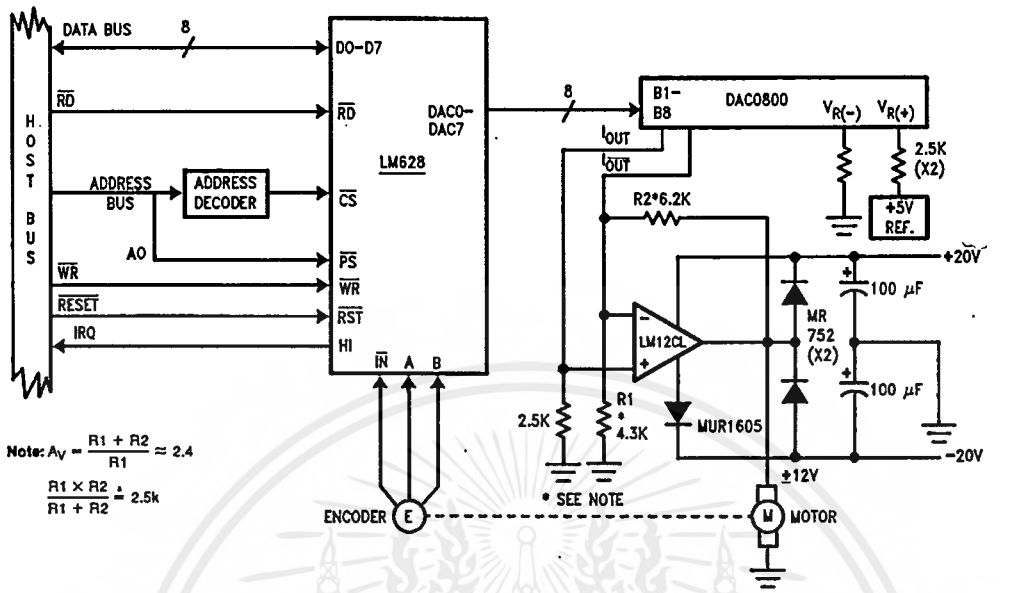
Incremental Encoder Interface

The incremental (position feedback) encoder interface consists of three lines: Phase A (Pin 2), Phase B (Pin 3), and Index (Pin 1). The index pulse output is not available on some encoders. The LM628 will work with both encoder types, but commands SIP and RDIP will not be meaningful without an index pulse (or alternative input for this input . . . be sure to tie Pin 1 high if not used).

Some consideration is merited relative to use in high Gaussian-noise environments. If noise is added to the encoder inputs (either or both inputs) and is such that it is not sustained until the next encoder transition, the LM628 decoder logic will reject it. Noise that mimics quadrature counts or persists through encoder transitions must be eliminated by appropriate EMI design.

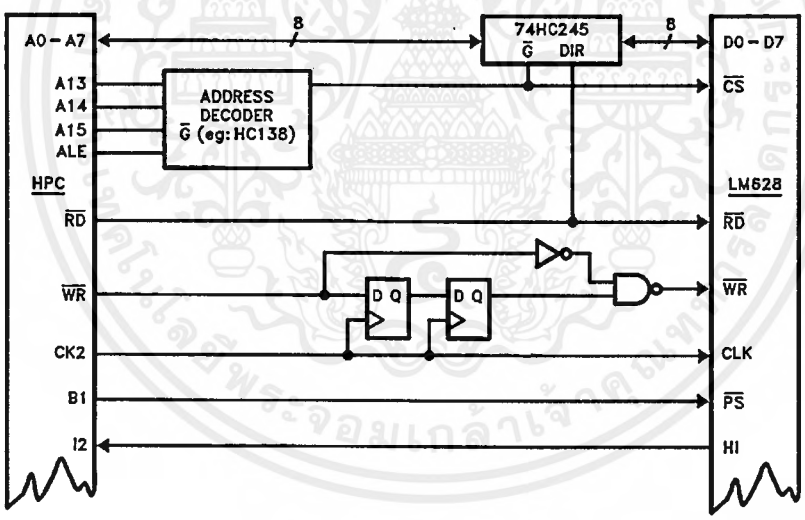
Simple digital "filtering" schemes merely reduce susceptibility to noise (there will always be noise pulses longer than the filter can eliminate). Further, any noise filtering scheme reduces decoder bandwidth. In the LM628 it was decided (since simple filtering does not eliminate the noise problem) to not include a noise filter in favor of offering maximum possible decoder bandwidth. Attempting to drive encoder signals too long a distance with simple TTL lines can also be a source of "noise" in the form of signal degradation (poor risetime and/or ringing). This can also cause a system to lose positional integrity. Probably the most effective countermeasure to noise induction can be had by using balanced-line drivers and receivers on the encoder inputs. *Figure 18* shows circuitry using the DS26LS31 and DS26LS32.

Typical Applications (Continued)



TL/H/9219–14

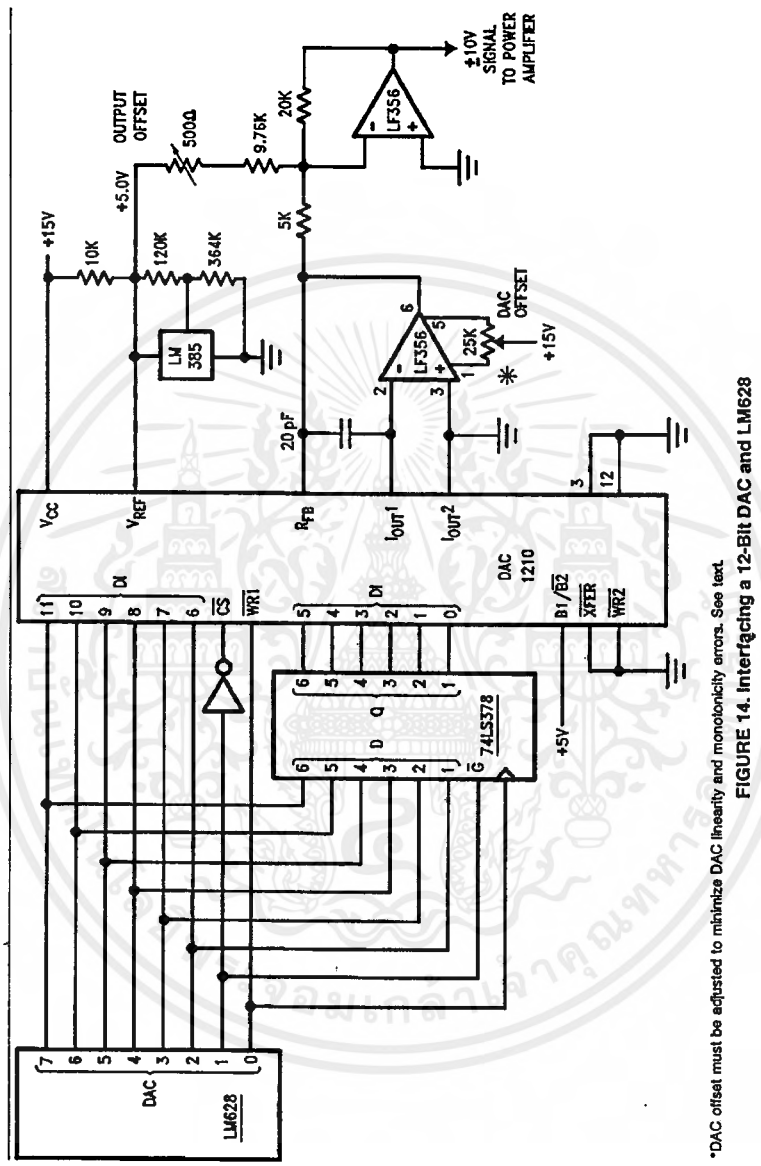
FIGURE 12. Host Interface and Minimum System Configuration



TL/H/9219–15

FIGURE 13. LM628 and HPC Interface

Typical Applications (Continued)

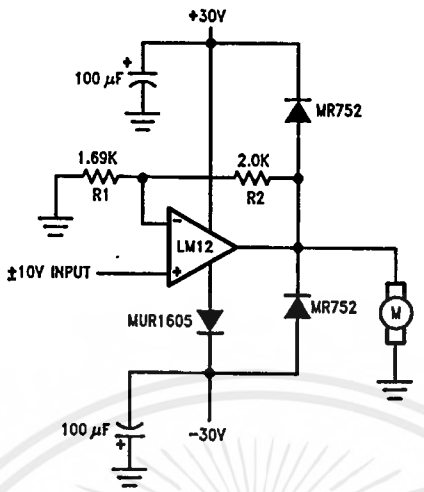


TL/H/9219–16

*DAC offset must be adjusted to minimize DAC linearity and monotonicity errors. See text.

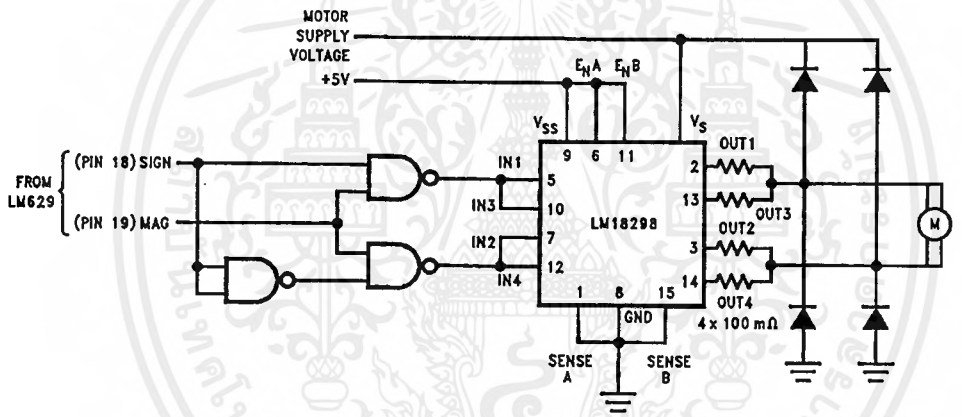
FIGURE 14. Interfacing a 12-Bit DAC and LM628

Typical Applications (Continued)



TL/H/9219–17

FIGURE 15. Driving a Motor with the LM12 Power Op Amp



TL/H/9219–18

FIGURE 16. PWM Drive for Brush/Commutator Motors

Typical Applications (Continued)

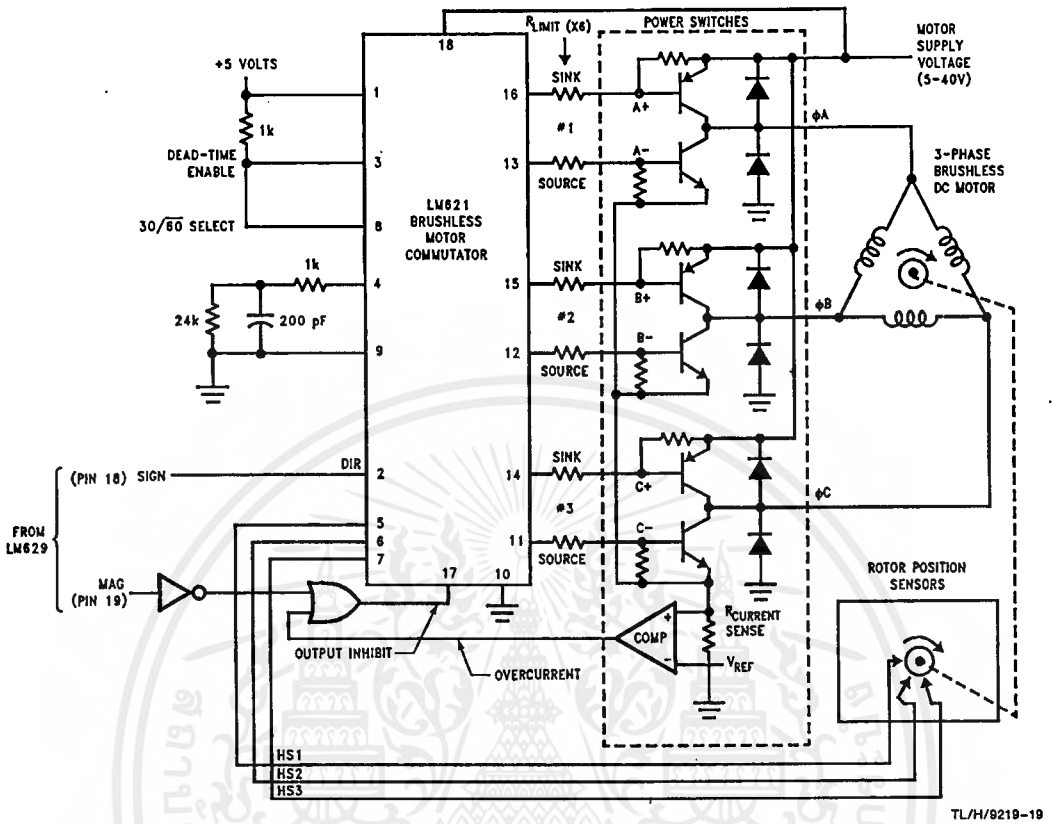


FIGURE 17. PWM Drive for Brushless Motors

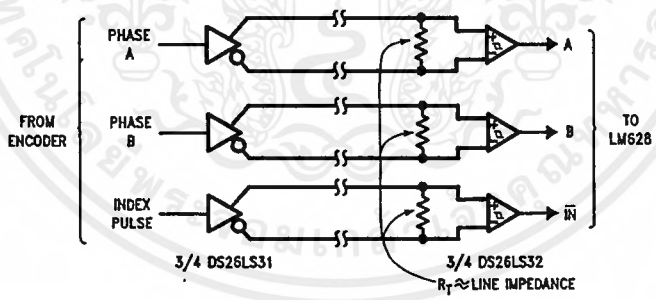


FIGURE 18. Typical Balanced-Line Encoder Input Circuit