DANIEL DE PAULA NESTROVSKY

MARCEL HENRIQUE VIEIRA GONÇALVES

Nota final
7,6 (sete e seis)
[illegible]
22/01/04

MÁQUINA DE SHAPING:
MÓDULO DE CONTROLE

Trabalho apresentado a Escola Politécnica da Universidade de São Paulo para a conclusão do curso de graduação em Engenharia Mecatrônica.

São Paulo
2003

DANIEL DE PAULA NESTROVSKY
MARCEL HENRIQUE VIEIRA GONÇALVES

MÁQUINA DE SHAPING: MÓDULO DE CONTROLE

Trabalho apresentado a Escola Politécnica da Universidade de São Paulo para a conclusão do curso de graduação em Engenharia Mecatrônica.

Área de concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Julio Cezar Adamowski

São Paulo
2003

Aos nossos familiares e amigos, que são as pessoas mais importantes em nossas vidas.

RESUMO

O presente trabalho discute o sistema de controle de uma máquina para usinagem de prancha de surf e propõem os atuadores, sensores de retroação e o hardware (driver para os atuadores e CPU) para que o controle seja possível. No decorrer deste texto são apresentados a descrição de um sistema e da malha controle, teoria sobre interpolação linear, a geração de perfil trapezoidal de velocidade e a apresentação do LM629, um CI da National que implementa um controlador PID e gera o perfil trapezoidal de velocidade. Depois é a apresentada a seleção dos motores, encoders, detalhes para a construção hardware, o software para teste e um método experimental para a sintonia do PID. Por fim discute-se a possibilidade do uso do hardware desenvolvido para o controle efetivo de uma máquina de usinagem.

ABSTRACT

This report discuss the control system for a surf shape machining machine and suggest the motors, sensors for feedback and the hardware (driver and CPU) to do this control. The text shows the description of a control system and a close-loop control system, theory about linear interpolation, trapezoidal velocity profile and a brief description of the LM629, a National IC that implement a digital PID filter and the trapezoidal profile. Then the selection of the motors, the encoders, details for the hardware construction, the test software and an experimental method to tuning the PID filter is showed. Finally, the possibility to use the developed hardware for the control of the machine is discussed.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

Este trabalho versa sobre uma máquina CNC dedicada a usinagem de pranchas, que são em geral feitas de um material polimérico, poliuretano rígido, e utilizadas na prática de Surf.

Uma versão dessa máquina já existe e foi desenvolvida pelo Professor orientador deste projeto em meados dos anos 90, com o intuito de atender às necessidades de um "Hobista". Com o decorrer dos anos o negócio que a princípio era em pequena escala se tornou rentável e volumoso, de forma que a necessidade de uma máquina mais rápida e mais precisa se tornou imprescindível para a sustentação do fabricante no mercado que se tornou competitivo e exigente.

Esta necessidade criou a motivação de participar deste desafio de reprojetar a máquina, tornando-a mais rápida, mais precisa e claro, se possível, de mesmo custo. Devido à complexidade do projeto serão atacadas somente duas frentes: projeto do sistema de controle e estudo da usinagem do material, sendo os temas desenvolvidos por duas equipes. Cabe a esta equipe o tema de controle. O projeto das demais partes envolvidas na melhora será abordado no futuro, estando fora do escopo deste trabalho.

No decorrer do texto é apresentada descrição de um sistema e de uma malha de controle generalista e uma introdução à interpolação linear e a ao perfil de velocidade Trapezoidal bem como o LM629, CI da National que implementa um controlador PID digital e gera perfil de velocidade trapezoidal. Então é descrito o hardware (CPU e drivers) que possibilita os uso deste CI para o controle da máquina, além da escolha dos motores e sensores e o software desenvolvido para demonstrar a eficácia do hardware para a função e teste do sistema. Ao final do texto discute-se a possibilidade do uso do material desenvolvido para o efetivo controle da máquina.

2. REVISÃO DA LITERATURA

2.1. Descrição do Sistema de Controle

Para o controle dos motores de corrente contínua será necessário mudar a malha de controle da máquina de aberta (adequada para o uso de motores de passo) para uma malha fechada. Isso propiciará maior precisão e velocidade.

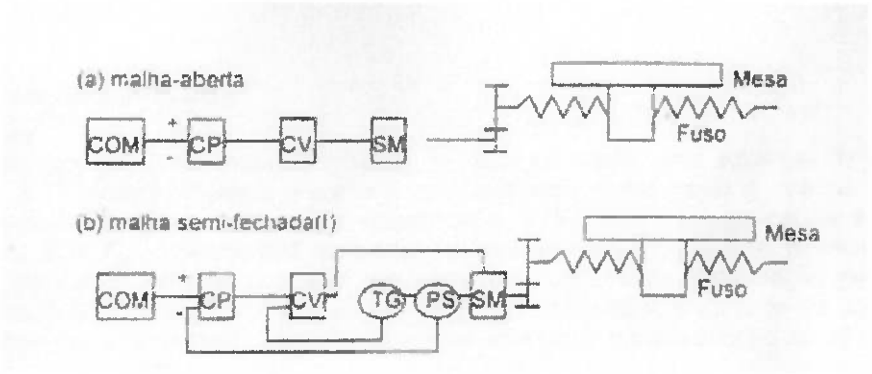


Fig. 2-1 - Esquema da malha de controle atual (a) e a que será implementada (b).

A malha adotada será do tipo com a retro alimentação sendo feita através de encoder (ou resolver) acoplado ao eixo dos motores.

2.2. Descrição da Malha de Controle

A malha de controle usada é ilustrada a baixo:

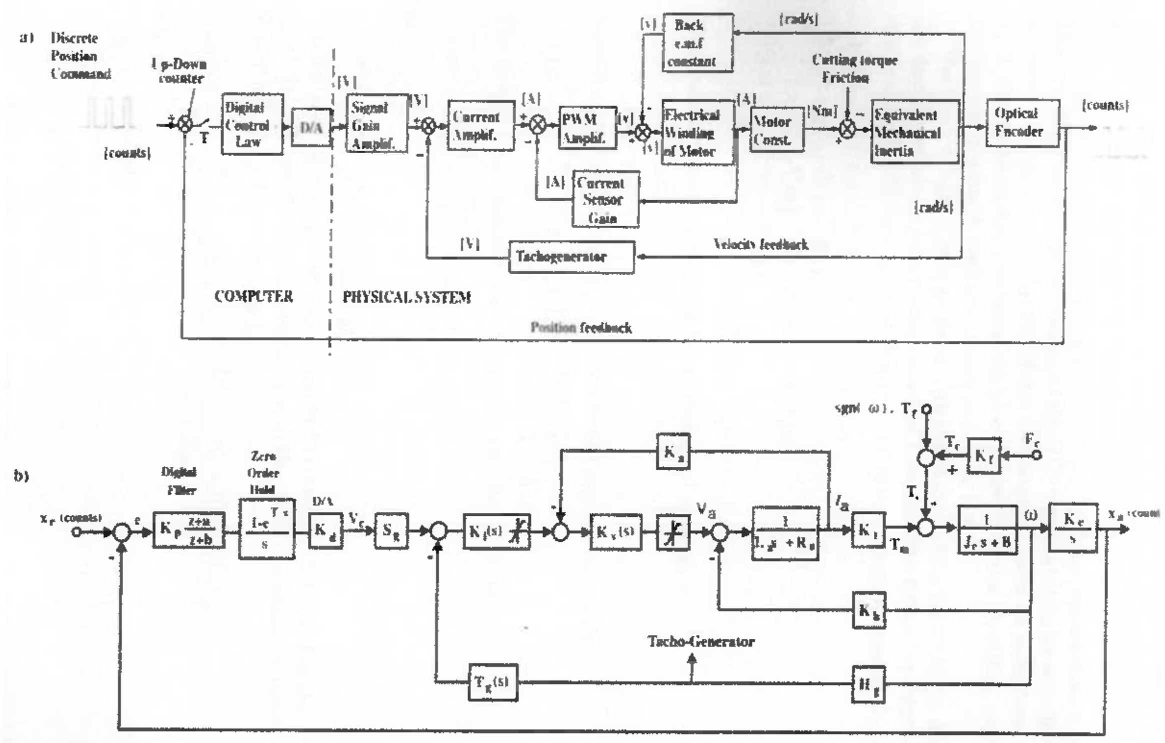


Fig. 2-2 - Diagrama de controle para o sistema de avanço. (a) Partes físicas. (b) Funções de transferência

Nota-se que o sistema se divide em duas partes: uma parte física e outra implementada em um computador.

Deduz-se que o modelo para este sistema[1] é representado por uma função de transferência com a seguinte forma:

$$G_v(s) = \frac{K_1}{s^2 + K_2 s + K_3} \tag{1}$$

O contador pode ser modelado como:

$$\frac{X_a(s)}{\omega(s)} = \frac{K_e}{s} \tag{2}$$

Multiplicando (1) por (2) obtém-se a equação da planta a ser controlada ($G_c(s)$).

O sinal de controle V_c do controle digital que será implementado é aplicado a planta através de um conversor D/A mantedor a cada T segundos (tempo de amostragem) com ganho K_d, e o equivalente a planta em tempo discreto é dado por:

$$G_c(z) = K_d\left(1 - z^{-1}\right)Z\left[\frac{G_c(s)}{s}\right] \tag{3}$$

Que depois de executar a transformada fica:

$$G_c(z) = \frac{K_d K_1 K_e}{K_3} \frac{b_2 z^2 + b_1 z + b_0}{(z-1)\left(z^2 + a_1 z + a_0\right)} \tag{4}$$

E os parâmetros são:

$$\begin{aligned}
b_2 &= T - \frac{1}{\omega_d} e^{-\xi\omega_n T} \sin(\omega_d T) \\
&\quad - \frac{K_2}{K_3}\left\{1 - e^{-\xi\omega_n T}\left[\frac{\xi\omega_n}{\omega d} \sin(\omega_d T) + \cos(\omega_d)\right]\right\}, \\
b_1 &= 2e^{-\xi\omega_n T}\left[\frac{\sin(\omega_d T)}{\omega_d} - T\cos(\omega_d T)\right] \\
&\quad + \frac{K_2}{K_1}\left(1 - e^{-2\xi\omega_n T}\right) - 2e^{-\xi\omega_n T}\sin(\omega_d T)\frac{K_2}{K_3}\frac{\xi\omega_n}{\omega_d}, \\
b_0 &= Te^{-2\xi\omega_n T} - \frac{1}{\omega_d} e^{-\xi\omega_n T} \sin(\omega_d T) \\
&\quad + \frac{K_2}{K_1}\left\{e^{-2\xi\omega_n T} + e^{-\xi\omega_n T}\left[\frac{\xi\omega_n}{\omega_d} \sin(\omega_d T) - \cos(\omega_d T)\right]\right\}, \\
a_1 &= -2e^{-\xi\omega_n T}\cos(\omega_d T), \\
a_0 &= e^{-2\xi\omega_n T}
\end{aligned}$$

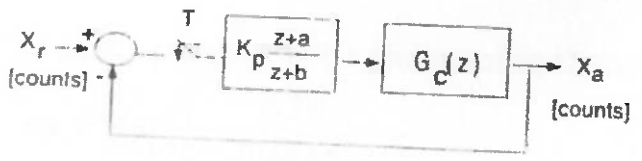


Fig. 2-3 - Diagrama de blocos simplificado

O diagrama de bloco correspondente ao controle de posição é o mostrado acima e a função de transferência em z correspondente a malha fechada é:

$$G_{cl}(z) = \frac{D(z)G_c(z)}{1 + D(z)G_c(z)} \tag{5}$$

Em que $D(z)$ um filtro digital, parte computacional, e tem função de transferência típica[1]:

$$D(z) = K_p \frac{z+a}{z+b} \tag{6}$$

Onde K_p é o ganho proporcional, a é o zero do filtro e b o pólo, e serão escolhidos de forma a garantir uma resposta transitória desejada.

Além disso, deve-se minimizar o erro em regime permanente para alcançar precisão em contornos usando vários eixos. Em regime permanente um contorno com velocidade de avanço f_c constante é modelado como uma entrada em rampa:

$$X_r(z) = \frac{f_c T z}{(z-1)^2} \tag{7}$$

Sendo o erro em malha fechada é dado por:

$$E(z) = \frac{1}{1 + D(z)G_c(z)} X_r(z) \tag{8}$$

Aplica-se o teorema do valor final chegando-se a ao erro de velocidade:

$$e_{ss} = \lim_{z \to 1} \frac{f_c T z}{(z-1)D(z)G_c(z)} \Rightarrow e_{ss} = \frac{f_c K_3 (1+b)}{K_1 K_e K_p K_d (1+a)} \tag{9}$$

Tem-se desta forma o erro de velocidade constante e fica evidente de quanto maior o ganho da em malha aberta menor será o erro de seguimento. Porém o ganho em malha fechada é limitado pela inércia do sistema e os limites do sistema de acionamento e amplificadores.

Mostra-se que para uma entrada em degrau o erro vai a zero.

2.3. Interpolador Linear em Tempo Real

Esta máquina deverá ser capaz de seguir trajetórias complexas interpoladas através de seguimentos de retas. Para tanto será apresentado aqui um algoritmo baseado na integração digital da velocidade para dois eixos que será expandido para os três eixos durante a implementação.

O objetivo deste algoritmo é fazer o centro da ferramenta sair de um ponto P_s e chegar à P_e percorrendo uma trajetória reta.

No instante t a posição dos eixos será

$$\begin{aligned} x(t) &= x_s + \int_0^t f_x(t)dt, \\ y(t) &= y_s + \int_0^t f_y(t)dt, \end{aligned} \tag{10}$$

em que f são as velocidades dos eixos e o período de integração T_i varia com o tempo durante o a aceleração e a desaceleração como será mostrado na próxima parte. Este algoritmo será executado N vezes nos intervalos T_i e a equação (10) em tempo discreto fica

$$\begin{aligned} x(k) &= x(k-1) + f_x(k)T_i(k), \\ y(k) &= y(k-1) + f_y(k)T_i(k) \end{aligned} \tag{11}$$

e

$$f_x(k) = \frac{\Delta x}{T_i(k)}, \qquad f_y(k) = \frac{\Delta y}{T_i(k)}. \tag{12}$$

A variação de T_i resulta na variação da velocidade de avanço dos eixos, entretanto o acréscimo de deslocamento são constantes dados por

$$\Delta x = \frac{x_e - x_s}{N}, \qquad \Delta y = \frac{y_e - y_s}{N}. \tag{13}$$

Substituindo as equações (12) e (13) em (11) chega-se a

$$x(k) = x(k-1) + \Delta x, \quad y(k) = y(k-1) + \Delta y. \tag{14}$$

Os incrementos são constantes e são calculados no início da rotina de interpolação.

O esboço de uma implementação em tempo real é dado abaixo

Tmin	Tempo mínimo entre interpolações
f	Velocidade de avanço
xs, ys	Posição inicial
xe, ye	Posição final
deltax	Distância em x
deltay	Distância em y
sign(x)	Direção de x (1: positivo ou -1:negativo)
sign(y)	Direção de y (1 ou -1)
N	Número de iterações
dx, dy	tamanho dos passos
xrem yrem	passos que faltam em x e y

(Todos valores são inteiros com exceção de f e Tmim)

O cálculo inicial é:

```
deltax = abs( xe-xs )
deltay = abs( ye-ys )
sign(x) = sign(xe-xs )
sign(y) = sign(ye-ys )
deltau = f*Tmin
N = sqrt( deltax^2 + deltay^2 )/deltau
```

```
dx = deltax/N
dy = deltay/N
xrem = deltax - dx*N
yrem = deltay - dy*N
line (xs, dx, xrem, sign(x), N) //Envia para controle de x
line (ys, dy, yrem, sign(y), N) //Envia para controle de y
```

Após a iniciação dos eixos, cada um pode ser calculado todo período T_i com o seguinte código:

```
function line (xs, dx, xrem, sign(x), N)
{
  x[0] = xs
  xerror = 0
  for (i = 1; i<=N; i++)
  {
    x[i] = x[i-1] + sign(x)*dx
    xerror = xerror + xrem
    if ( xerror >= N)
    {
      x[i] = x[i] + sign(x)
      xerror = xerror - N
    }
  }
}
```

As posições geradas são enviadas ao controlador a cada intervalo de interpolação.

2.4. Perfil de Velocidade Trapezoidal

A aceleração e desaceleração são definidas dentro do programa do CNC. As unidades são convertidas para *counts*, o menor deslocamento que o sistema pode detectar. E o caminho da ferramenta é dividido em N pequenos seguimentos na direção dos eixos durante o período de interpolação T_i.

O menor tempo de interpolação deve ser igual ao período de amostragem T ou um múltiplo do mesmo. O avanço f e o $T_{\min}$ são definidos no software de controle e o passo de interpolação, como adiantado no algoritmo de interpolação linear é

$$\Delta u = fT_{\min} . \qquad (15)$$

Quando o avanço muda durante o processo, Δu é mantido constante mas o tempo de interpolação T_i é atualizado como

$$T_i = \frac{\Delta u}{f} . \qquad (16)$$

Usando período de amostragem variável muitos eixos podem ser sincronizados a partir de um cálculo unidimensional. A interpolação de velocidade e posição ficam desacopladas. Admitindo um deslocamento de L em uma direção arbitrária, o interpolador é executado N vezes com intervalo de T_i em entre eles:

$$N = \frac{L}{\Delta u}. \tag{17}$$

O número total de iterações é dividido em vários estágios dependendo do perfil de velocidade usado.

Para um perfil trapezoidal, que será usado pelo interpolador desta máquina, há três estágios distintos: a aceleração, velocidade constante e desaceleração como mostrado na figura.

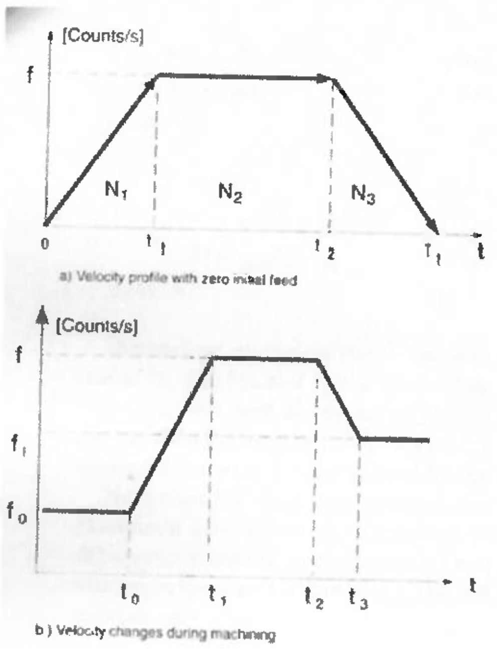


Fig. 2-4 - Perfil trapezoidal para velocidade

Se o avanço inicial é e a aceleração A é constante a distância percorrida durante esse período é

$$l_1 = \int_0^{t_1} A t dt = \frac{A t_1^2}{2} \tag{18}$$

Como $t_1 = f/A$, o número de intervalos de interpolação durante é

$$N_1 = \frac{l_1}{\Delta u} = \frac{f^2}{2A\Delta u} \tag{19}$$

Para a desaceleração D é semelhante

$$N_2 = \frac{l_1}{\Delta u} = \frac{f^2}{2D\Delta u} \tag{20}$$

Em uma aplicação como esta que está sendo desenvolvida neste trabalho não é interessante fazer a máquina parar antes de iniciar um novo seguimento. Então para acelerar de um avanço f_0 para um novo avanço f o deslocamento fica

$$l_1 = \int_{t_0}^{t_1} [f_0 + A(t - t_0)]dt = \int_0^{\tau_a} [f_0 + A\tau]d\tau = f\tau_a + \frac{A\tau_a^2}{2} \tag{21}$$

Em que $\tau_a = t_1 - t_0 = (f - f_0)/A$ então

$$l_1 = \frac{f^2 - f_0^2}{2A} \tag{22}$$

e

$$N_1 = \frac{f^2 - f_0^2}{2A\Delta u} \tag{23}$$

Para desacelerar de f para f_l o processo é semelhante e obtém-se

$$N_3 = \frac{f^2 - f_l^2}{2D\Delta u} \tag{24}$$

Os números de iterações são inteiros arredondados.

O intervalo de interpolação T_i deve mudar a cada intervalo durante a fases de aceleração para manter o passo de interpolação Δu constante. Desta forma

$$\Delta u = \int_{t_{k-1}}^{t_k} Atdt = \frac{A}{2}(t_k^2 - t_{k-1}^2) = \frac{A}{2}(t_k - t_{k-1})(t_k + t_{k-1})$$

Substituindo $T_i(k) = (t_k - t_{k-1})$ e $t_k = f(k)/A$`, $t_{k-1} = f(k-1)/A$ temos o período de interpolação para cada intervalo

$$T_i(k) = \frac{2\Delta u}{f(k) + f(k-1)} \tag{25}$$

O que lava ao seguinte pseudo-algoritmo

```
for (k = 1; K <= N1; k++)
{
    $f(k) = \sqrt{f_0^2 + 2kA\Delta u}$
    $T_i(k) = \frac{2\Delta u}{f(k) + f(k-1)}$
}

for (k = 1; K <= N2; k++)
{
    $T_i(k) = \frac{\Delta u}{f}$
}

for (k = 1; K <= N3; k++)
{
    $f(k) = \sqrt{f_0^2 - 2kD\Delta u}$
    $T_i(k) = \frac{2\Delta u}{f(k) + f(k-1)}$
}
```

Note que as fases são codificadas em funções separadas e são executadas quando necessário.

Note também que avanço e acelerações são regulados pelo vetor deslocamento, o que garante sincronismo e velocidade correta em todos atuadores ativos, contribuindo para o movimento vetorial no espaço. Tomando um movimento em dois eixos, o vetor deslocamento é

$$\vec{\Delta u} = \Delta x\vec{i} + \Delta y\vec{j} \tag{26}$$

Dividindo ambos os lados por T_i tem-se

$$\vec{f} = f_x\vec{i} + f_y\vec{j} \tag{27}$$

em que a amplitude do avanço é $f = \sqrt{f_x^2 + f_y^2}$ e f_x, f_y são as velocidades nos atuadores x e y. Então, uma vez definido Δu, o período de interpolação T_i e o números de iterações N_1, N_2 e N_3 calculados, as velocidades e o incremento de posição para os atuadores x e y estão automaticamente definidos pelo algoritmo, como foi mostrado no algoritmo de interpolação exposto anteriormente.

2.5. O LM629

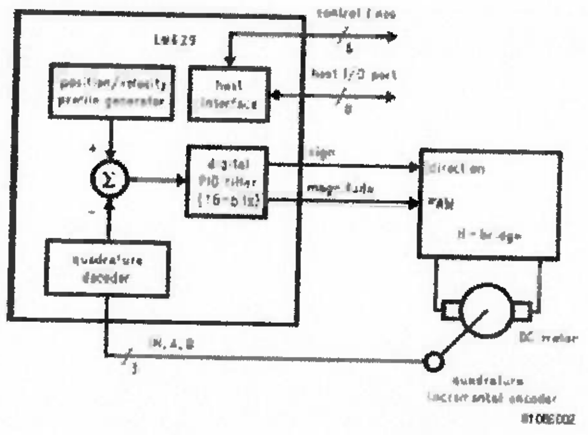


Fig. 2-5 - Diagrama de blocos para controle de motor CC usando LM629

O LM629 da National Semiconductor é um processador dedicado para controle de movimento e pode ser usado com uma variedade de motores CC que possua um sinal incremental em quadratura para realimentação (encoder). Ele implementa um controlador PID digital e executa as tarefas em tempo real necessárias para o controle do movimento. O software de controle da CPU desenvolvida faz a interface com o CI através de um conjunto de comandos.

As características deste CI são:

- Registradores de posição, velocidade e aceleração de 32 bits;
- Coeficientes dos filtros do PID com 16 bits;
- Período de amostragem derivativo programável;
- Saída sinal-magnitude PWM de 8 bits;
- Velocidade, posição alvo e os filtros do PID podem ser alterados durante a execução do movimento;

Este equipamento pode ser operado de duas maneiras: modo posição e modo velocidade. No modo posição define-se a posição alvo, a velocidade máxima e a aceleração e controlador se encarrega em produzir um perfil de velocidade trapezoidal, como o descrito anteriormente. No caso do modo velocidade o motor é acelerado até a velocidade alvo com a aceleração desejada e mantém a velocidade até receber um comando de parada, quando desacelera até parar.

A possibilidade de mudar a velocidade durante o movimento permite a criação de trajetórias como, por exemplo, na figura abaixo. Os pontos de mudança de velocidade podem ser ajusta através de *breakpoints*, que são posições onde o controlador interrompe o *host* avisando que chegou ao ponto desejado.

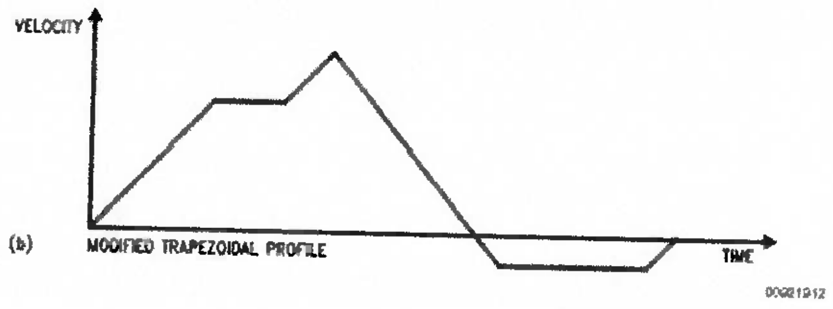


Fig. 2-6- Exemplo de perfil de velocidade

3. MATERIAIS E MÉTODOS

Nesta parte do texto serão descritos cada componente que compõe o sistema e detalhes referentes a critério de seleção e/ou concepção.

3.1. Atuadores e Sensores

3.1.1. Motores

Dimensionamento do motor do trilho:

- Massa dos blocos das guias de medição de força de usinagem: 5kg
- Massa da mesa de fixação em alumínio: 20kg
- Massa total do sistema: 5 + 20 = 25kg
- Aceleração envolvida: 1m/s^2
- Força de corte máxima: 40N
- Força total envolvida: 65N
- Torque envolvido: 65*(28E-03) = 1,82N.m, onde 28E-03m é o raio primitivo da polia envolvida.
- Para uma rotação de 1800rpm = 30rps do motor e uma redução de 1:5 => 6rps, a potência envolvida é de P = 1,82*6 = 10,92W
- Servo motor selecionado: MT-3363 com as seguintes características:
  - Peso 5,9kg
  - Rotação 4000rpm
  - Torque 1,3N.m
  - Tensão 120V
  - Corrente 4.76A
  - Fabricante BALDOR

Dimensionamento do motor da guia:

- Massa dos blocos das guias de medição de força de usinagem:15kg
- Massa do motor: 5kg
- Massa ferramenta: 2kg
- Massa total do sistema: 22kg
- Massa do fuso: 1,25kg
- Passo do fuso: 5mm

- Diâmetro do fuso: 25mm
- Coeficiente de atrito: 0,1
- Coeficiente de escorregamento: 0,005
- Força de avanço: 100N
- Força na direção do deslocamento: 0N

Torque estático:

Test = (passo*coef.atrito(25*a + Força na direção deslocamento))/2*3,14 = 1,99E-03 N.m

Torque de Rolamento:

Trol = passo*diâmetro fuso/2*força avanço = 25E-03 N.m

Torque de avanço:

Tavan = passo/2*3,14*força de avanço = 0,31N.m

Torque total:

Ttot = 1,99E-03 + 25E-03 + 0,31 = 0,34 N.m

- Para uma rotação de 2400rpm = 40rps do motor, a potência envolvida é de
P = 0,34*40 = 13,5W
- Servo motor selecionado: MT-3353 com as seguintes características:
  - Peso 4Kg
  - Rotação 4000rpm
  - Torque 0,7N.m
  - Tensão 120V
  - Corrente 2,68A
  - Fabricante BALDOR

Os motores foram selecionados a partir dos valores de torque, e considera-se que várias aproximações de massa foram feitas.

Para a seleção dos motores foi considerada a movimentação da estrutura a abaixo.

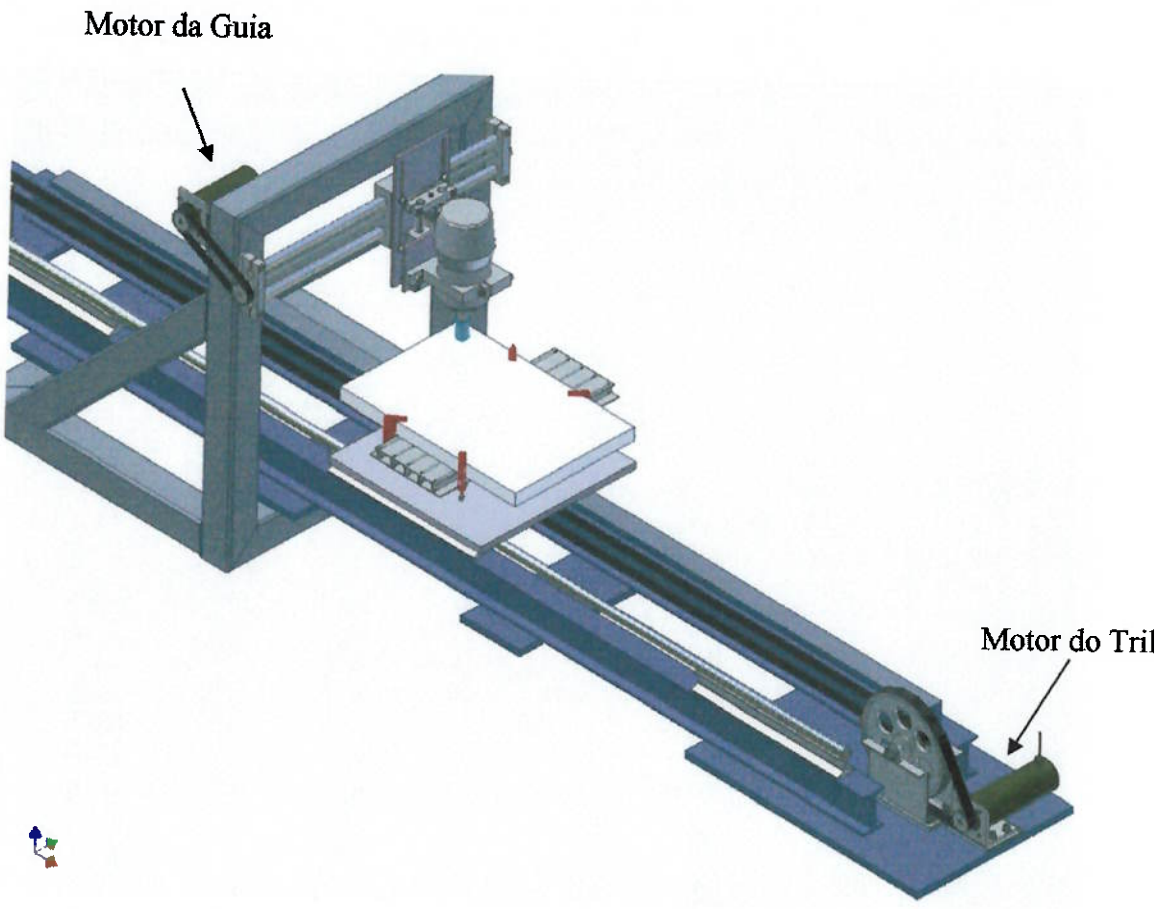


Fig. 3-1 - Desenho artístico da máquina de usinagem

3.1.2. Encoders

A realimentação de posição do LM629 deve ser incremental e em quadratura podendo haver, ou não, um sinal de index. Portanto o gerador de pulsos (encoder) selecionado, o RI 42-0/1000 AR.41DA da Veeder-Root, atende este requisito.

Este equipamento possui as seguintes características:

- 1000 linhas de resolução;
- saída em quadratura com sinal de index;
- saída push-pull 5V, +30mA.

3.2. Hardware

O hardware para controle é composto de dois drivers de potência e uma placa CPU com um microcontrolador PIC que funciona como host de dois LM629 e faz interface com um PC através de uma porta serial RS232 e um terminal de telnet.

3.2.1. Fonte: Alimentação dos Motores

A fonte utilizada é composta de um transformador de potência com duas saídas de 65VAC 5A e uma ponte retificadora de potência disposta em um dissipador.

3.2.2. Driver

O driver consiste em um amplificador de potência que recebe o sinal de controle do controlador, amplifica e entrega potência ao atuador.

O driver desenvolvido possui, como componentes essenciais, quatro MOSFETs IRF840 e dois Gate drives IR2104 formando uma ponte H conforme o esquemático abaixo.

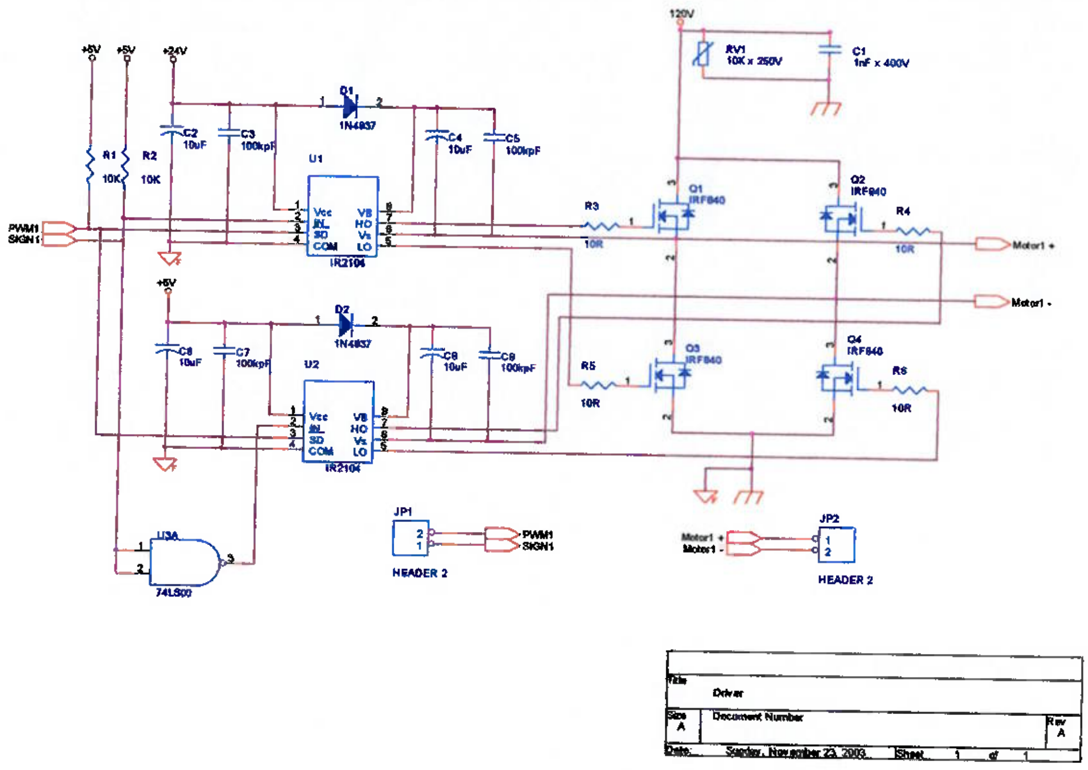


Fig. 3-2 - Esquema elétrico Driver

3.2.3. Placa CPU

A placa foi desenvolvida com um microcontrolador PIC16F876A rodando com um cristal de 20Mhz. Possui um diodo-led para sinalização e um botão de reset de hardware. A placa possui uma comunicação serial do tipo RS232 para comunicação com o computador. A comunicação serial é feita através de um cabo serial ligada à placa CPU e um circuito de conversão de sinais TTL – RS232

O microcontrolador funciona como host dos dois CIs dedicados (LM629) que estão dispostos na mesma placa. A comunicação com os CIs é feita através de um BUS de dados compartilhado entre os dois componentes. Para a seleção do CI desejado para passar os comandos, utiliza-se o sinal de ChipEnable que habilita o CI desejado para a recepção de comandos.

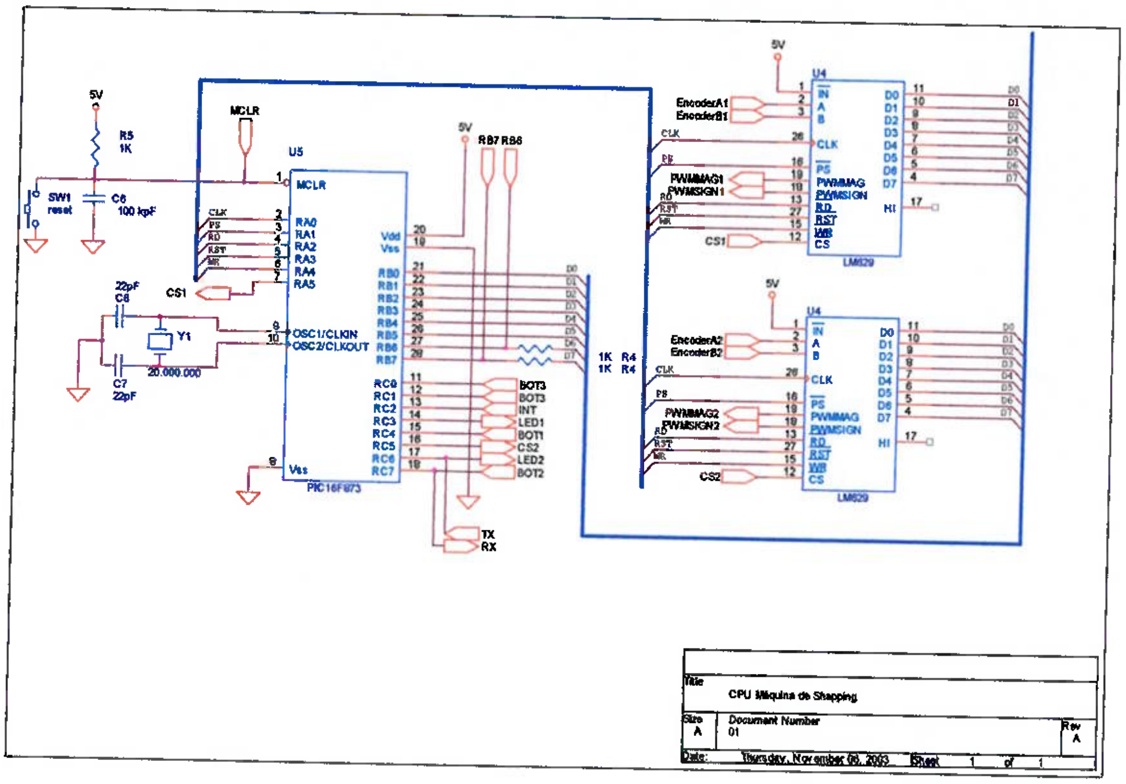


Fig. 3-3 - Esquema elétrico placa CPU (página 1/2)

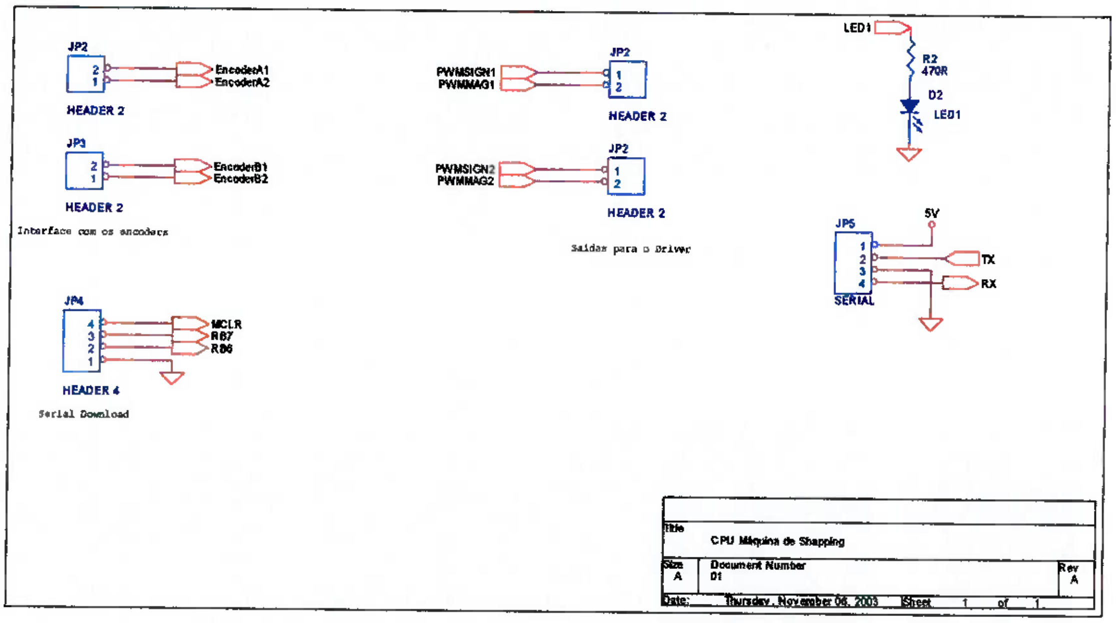


Fig. 3-4 - Esquema elétrico para CPU (página 2/2)

3.3. Software

É necessário um software para o teste do hardware e para demonstrar a capacidade do sistema controlar a máquina e realizar interpolação linear. O software implementa os comandos básicos para o funcionamento do LM629 e possui três módulos com diferentes funções, um para controle manual (modo manual), um para demonstrar a capacidade de se implementar interpolação linear (modo automático) e outro para a sintonia dos coeficientes do filtro PID (modo tuning). Será apresentada uma descrição funcional de cada módulo e um fluxograma.

3.3.1. Modo Manual

Este modo possibilita movimentar os motores da máquina com velocidade constante pré-definida tanto na direção positiva como negativa dos eixos e definir o zero absoluto para ambos motores.

Para usar este modo deve-se selecionar a opção 1 do menu principal, a partir deste momento usas as teclas a, s, d e w para acionar os motores, a tecla h para definir o zero absoluto (home) e x para sair deste modo.

Para mover o eixo x na direção positiva, deve-se pressionar a tecla d, o motor começa a se movimentar, então para pará-lo pressiona-se a tecla d novamente. Os demais sentidos e direções seguem o mesmo procedimento. A tabela abaixo relaciona os comandos e ações.

Tab. 3-1 - Comandos para Modo Manual

Tecla	Ação
D	Eixo X positivo
A	Eixo X negativo
W	Eixo Y positivo
S	Eixo Y negativo

3.3.2. Modo Automático

Esta é a opção 2 do menu principal. Neste modo o usuário define o próximo ponto em coordenadas relativas (x e y em mm) e a velocidade de avanço (f em mm/s), a CPU calcula os parâmetros, envia para os LM629 e sincroniza o início da operação, então pede novo ponto para o usuário e espera o término do movimento anterior para iniciar o novo ciclo.

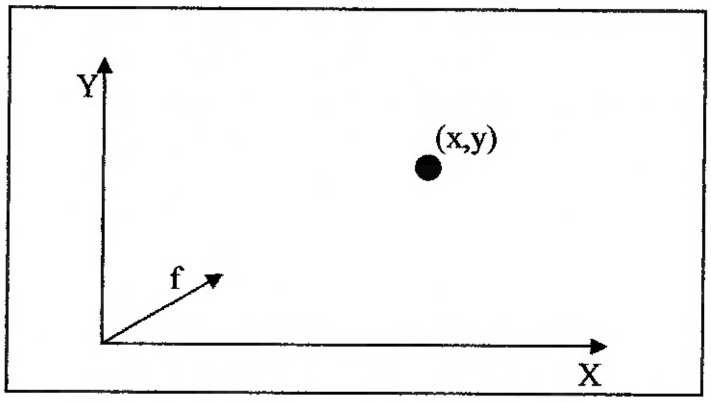


Fig. 3-5 - Dados para se obter uma trajetória linear

Para obter uma reta, os eixos devem acelerar sincronizados, obter velocidades máximas ao mesmo tempo e desacelerar e parar juntos, para tanto deve-se obter perfis de velocidade como o da figura abaixo.

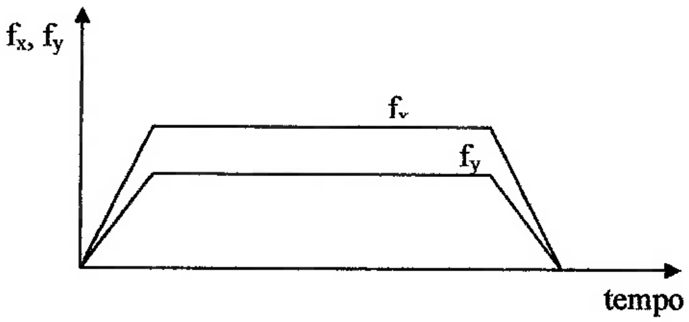


Fig. 3-6 - Perfis de velocidades desejados

A integral ao longo do tempo de f_y (posição y) deve ser em cada instante m vezes a integral de f_x, ou seja:

$$y = \int_0^t f_y dy = m\int_0^t f_x dx = mx \tag{28}$$

onde f_i é a velocidade de cada eixo e

$$m = \left|\frac{y}{x}\right|. \tag{29}$$

Para que isso ocorra deve-se ter a seguinte relações entre velocidades e acelerações:

$f_y = mf_x$ e $a_y = ma_x$. (30) (31)

Partindo deste princípio a função que calcula os parâmetros a serem enviados ao LM629 verifica qual distância a ser percorrida em módulo é maior e projeta o avanço f nesta direção, supondo x a distância de maior módulo temos:

$$f_x = \frac{x}{\sqrt{x^2 + y^2}} f \tag{32}$$

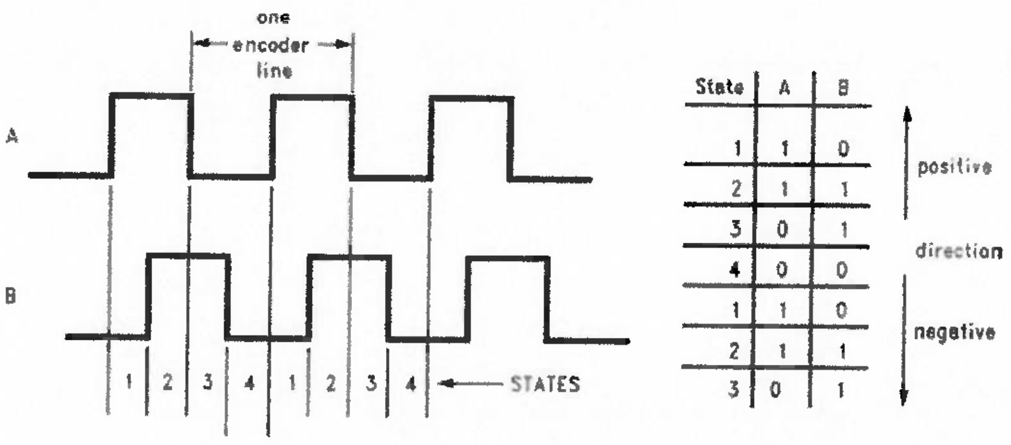
e obtém-se f_y da equação (30).

A aceleração a_x será a aceleração máxima da máquina e a_y é obtido de (31).

O procedimento é análogo para o caso de y maior que x.

Para que estes parâmetros possam ser enviados ao LM629 os valores de acelerações, velocidades e distâncias devem ser convertidos para as unidades que o CI usa, isto é, *counts* para distância e *samples* (amostras) para tempo. Para o encoder acoplado diretamente ao eixo do motor deve-se possuir as seguintes informações sobre o sistema para efetuar estas conversões:

- Quantas revoluções do eixo do motor são necessárias para o deslocamento de 1mm da mesa (rev2mm),
- Qual é a resolução do encoder (número de ranhuras, N), que deve ser multiplicado por quatro (observe a Fig 3.7), pois o LM629 recebe o sinal em quadratura, o que permite contar 4N posições em uma revolução do motor (rev2counts),
- Quantos segundos por *sample* é a taxa de amostragem do seu controlador, no caso do LM629 este número é 2048 dividido pelo cloak imposto ao CI (sec2sample).



State	A	B
1	1	0
2	1	1
3	0	1
4	0	0
1	1	0
2	1	1
3	0	1

Fig. 3-7 - Sinal em quadratura do encoder e tabela de decodificação

Com estes dados em mão as conversões são realizadas da seguinte maneira:

$$x_c = x \times mm2rev \times rev2counts \tag{33}$$

$$f_{x_c} = f_x \times mm2rev \times rev2counts \times \sec 2sample \tag{34}$$

$$a_{x_c} = a_x \times mm2rev \times rev2counts \times \sec 2sample \times \sec 2sample \tag{35}$$

O índice c indica que são os valores convertidos e para o eixo y deve-se realizar o mesmo procedimento.

No LM629 o valor de posição é um inteiro com sinal de 32bits, portanto deve-se arredondar o valor de x_c antes de enviá-lo. Velocidade e aceleração são inteiros de 32bits, porém os 16bits mais significativos correspondem a parte inteira e os 16bits menos significativos a parte fracionada, portanto estes parâmetros devem ser multiplicados por 65536 e então arredondados.

Para finalizar os parâmetros são carregados nos respectivos LM629 e então é enviado o comando STT (start) a cada CI para iniciar a execução do movimento. Cada controlador executa as tarefas em tempo real e garante que o perfil de velocidade seja o mais próximo possível do programado, desta maneira os eixos x e y estão desecoplados.

3.3.3. Modo Tuning

A opção 3 do menu principal proporciona uma interface com o usuário para se encontrar os valores dos parâmetros do filtro PID de cada LM629. Os detalhes de como se proceder estão na próxima parte (Setup Mínimo - Parâmetros PID).

3.3.4. Fluxograma do software

Aqui serão apresentados os fluxogramas do software, o primeiro é do funcionamento geral e os outros são do funcionamento de cada módulo. A intenção aqui é somente mostrar o fluxo de execução do software de teste do hardware.

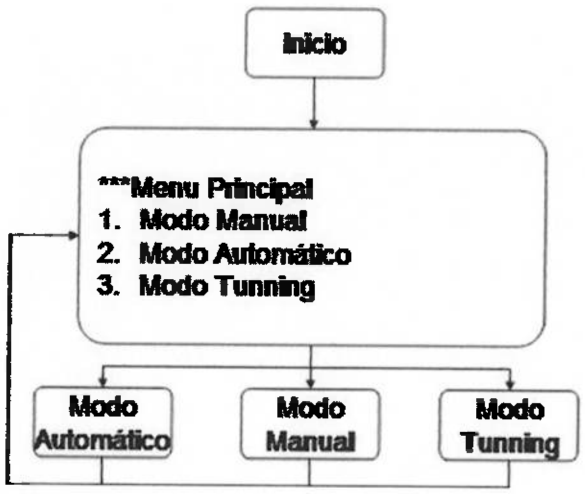


Fig. 3-8 - Fluxograma principal

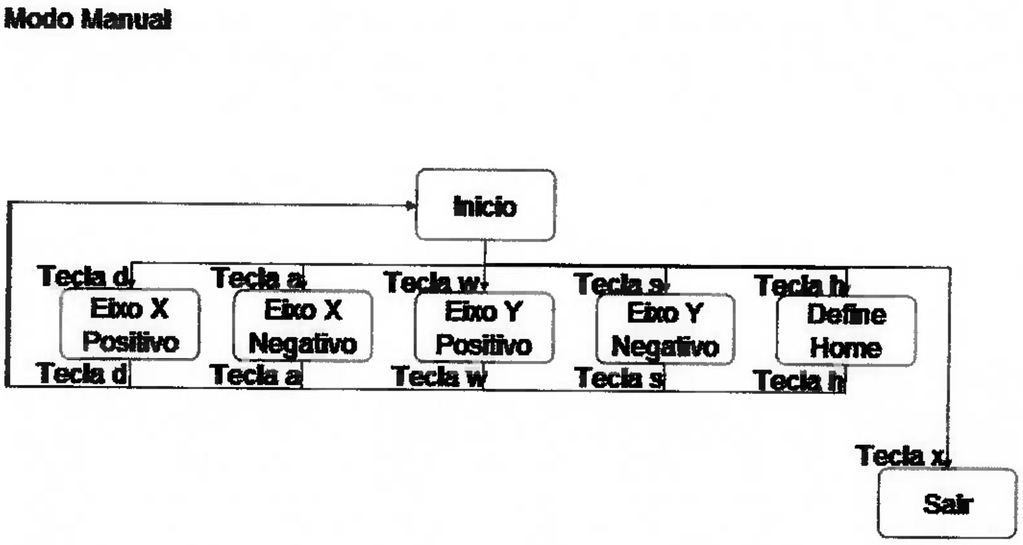


Fig. 3-9 - Fluxograma do modo manual

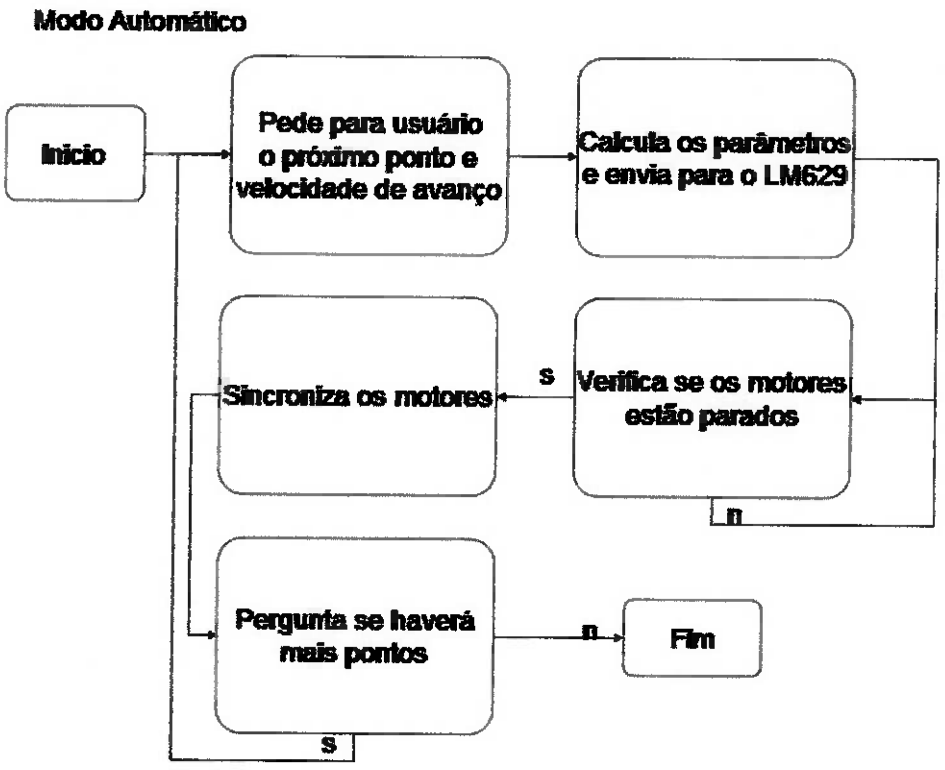

Modo Automático
Inicio
Pede para usuário o próximo ponto e velocidade de avanço
Calcula os parâmetros e envia para o LM629
Verifica se os motores estão parados
s
Sincroniza os motores
n
Pergunta se haverá mais pontos
n
Fim
s

Fig. 3-10 - Fluxograma modo automático

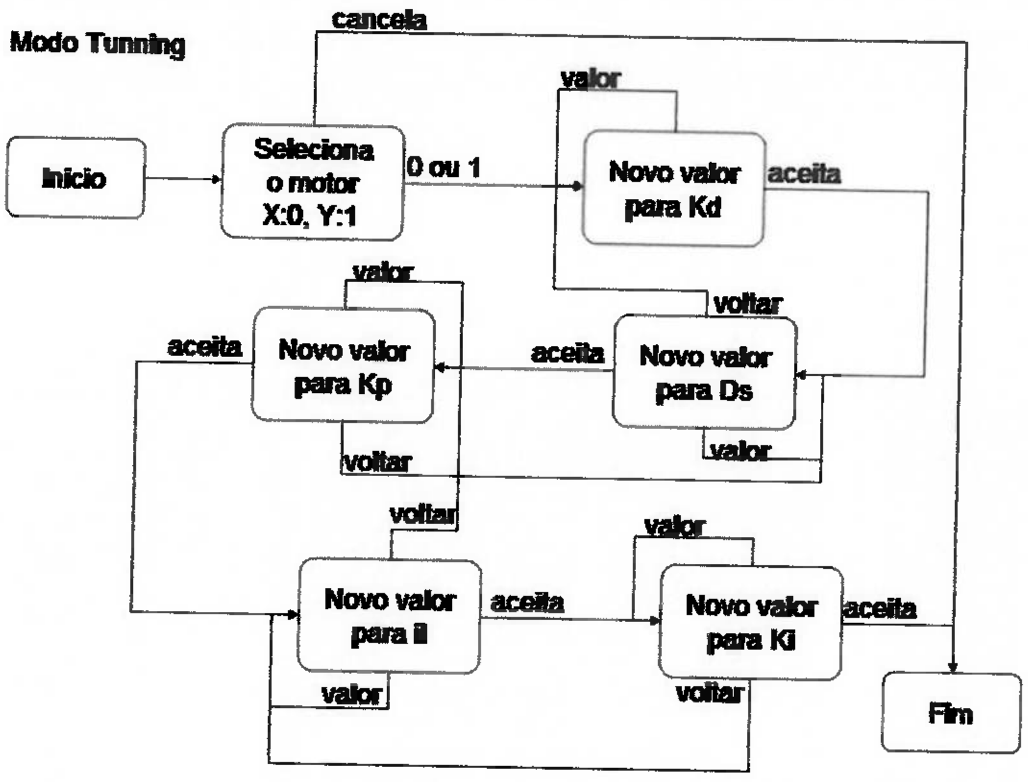


Fig. 3-11 - Fluxograma modo tuning

3.4. Setup Mínimo

3.4.1. Parâmetros do PID

Para que o LM629 funcione de maneira adequada, deve-se carregar os parâmetros adequados para o controlador, de forma a se obter uma resposta com o sobre-sinal (M_P), tempo de subida (t_r) e tempo de acomodação adequado (t_s). Nesta parte será descrito um método experimental, normalmente usado quando não se conhece afundo a dinâmica do sistema, para encontrar os parâmetros do PID de forma a se obter um sistema com amortecimento crítico, isto é, o menor tempo de subida sem sobre-sinal.

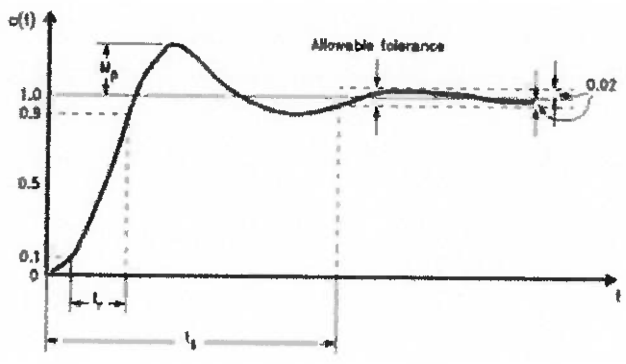


Fig. 3-12 - Resposta em degrau

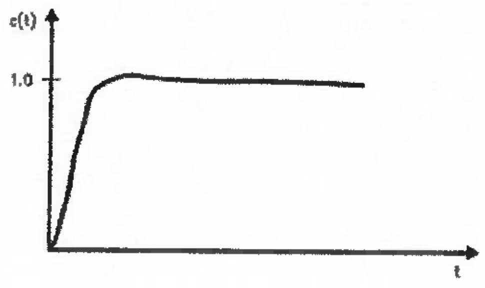


Fig. 3-13 – Resposta em degrau de um sistema com amortecimento crítico

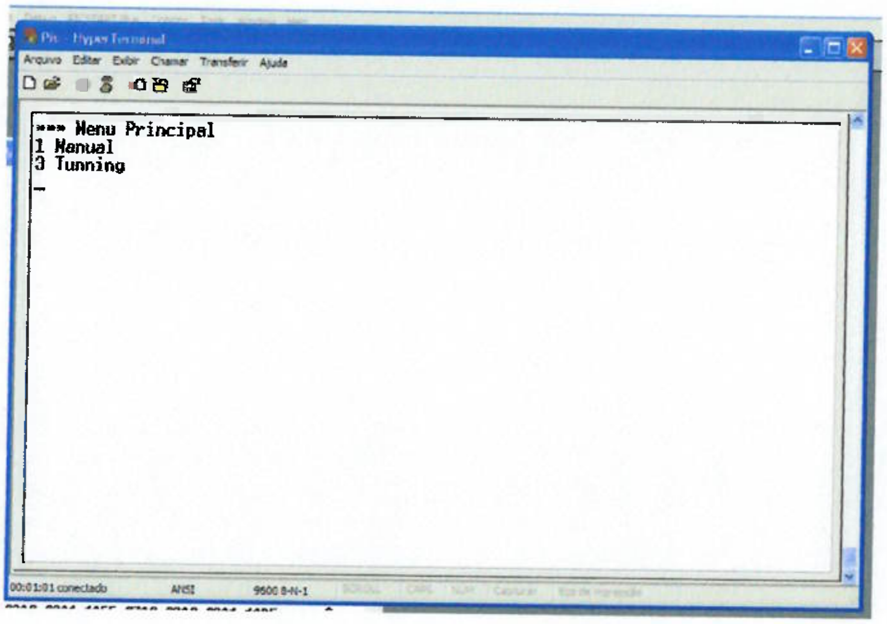
Os parâmetros a serem definidos para o filtro digital do LM629 são K_p (ganho proporcional), K_i (ganho integral), K_d (ganho derivativo), T_d (período de amostragem derivativo) e i_l (limite integral), e o modo "Tuning" do software foi desenvolvido para que estes números pudessem ser encontrados de forma conveniente.

Este método consiste de vários passos que devem ser seguidos.

1. Preparar o sistema:

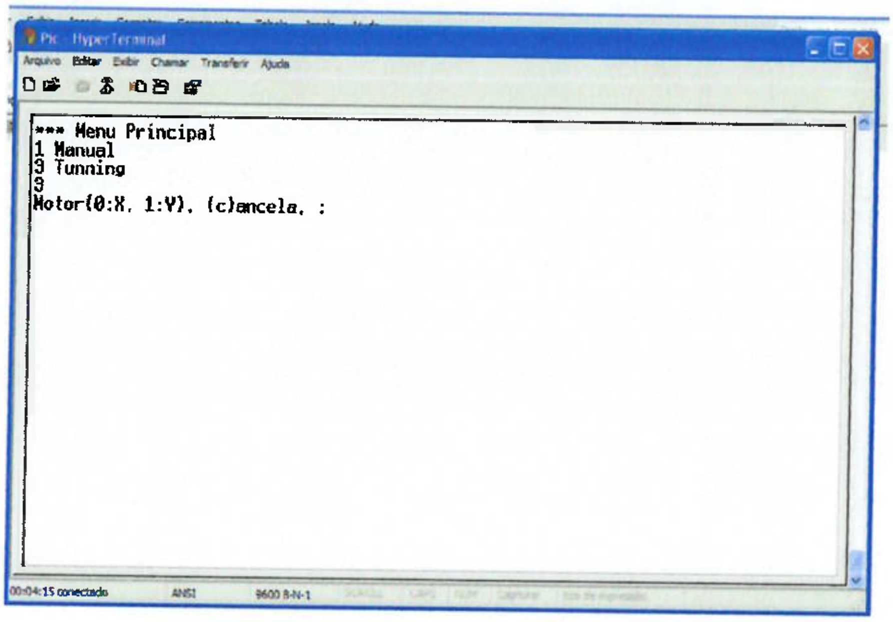
   Deve-se realizar um reset por hardware do LM629 que controla o motor, carregar K_p=K_i=i_l=0, K_d=2 e d_s=1, onde T_d = 256μs x d_s, enviar um comando para que o motor mantenha a posição atual. Qualquer movimento do eixo do motor será um erro, porém com K_p e K_i setados para zero o loop de controle não pode corrigir.

   Usando o modo "Tuning" este procedimento é executado quando se seleciona o motor para sintonizar.



```
*** Menu Principal
1 Manual
3 Tunning
_
```

Fig. 3-14 - Tela para seleção do modo Tuning



```
*** Menu Principal
1 Manual
3 Tunning
3
Motor(0:X, 1:Y), (c)ancela, :
```

Fig. 3-15 – Escolha do motor a ser sintonizado

2. Determinação do ganho derivativo

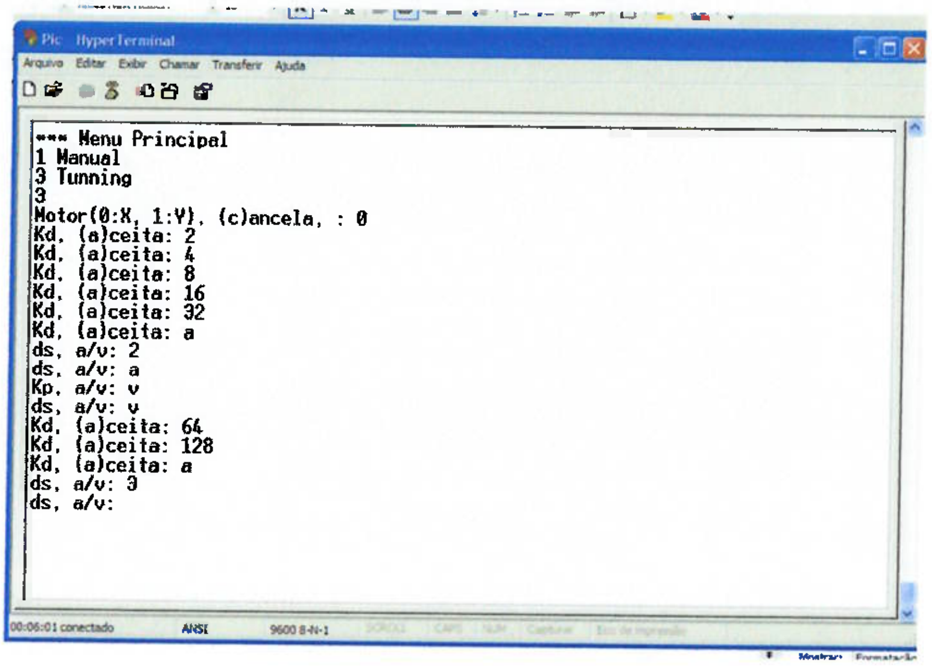
   O ganho derivativo amortece o sistema para diminuir o sobre-sinal e tempo de acomodação. A constante de proporcionalidade é d_s x K_d.

   Estes coeficientes são determinados por um processo iterativo. K_d é aumentado sistematicamente até o sistema começar a oscilar em alta freqüência. Então se soma um a d_s. Este procedimento é repetido até se chegar ao valor apropriado para o sistema.

Em geral k_d e d_s devem ser selecionados para que o produto deles dê o maior valor possível.

Uma maneira sugerida pelo AN-693 (Apêndice B) da National é iniciar K_d em 2 e dobrar seu valor a cada passo. Ao aumentar K_d ao tentar girar o eixo dôo motor, sente-se que a resistência aumenta. Por K_d proporcionar uma força proporcional a velocidade, quanto mais rápido se gira o eixo maior a resistência.

Após a seleção do motor para sintonizar, será requisitado um valor para K_d ou pode-se a aceitar o valor atual (dois na primeira vez). Aceitando o valor de K_d é requerido um valor para d_s, aceitar (a) o valor atual ou voltar (v) para ajustar K_d conforme o método. Para passar para a próxima parte, deve-se aceitar o valor de K_d e aceitar o valor de d_s.



```
*** Menu Principal
1 Manual
3 Tunning
3
Motor(0:X, 1:Y), (c)ancela, : 0
Kd, (a)ceita: 2
Kd, (a)ceita: 4
Kd, (a)ceita: 8
Kd, (a)ceita: 16
Kd, (a)ceita: 32
Kd, (a)ceita: a
ds, a/v: 2
ds, a/v: a
Kp, a/v: v
ds, a/v: v
Kd, (a)ceita: 64
Kd, (a)ceita: 128
Kd, (a)ceita: a
ds, a/v: 3
ds, a/v:
```

Fig. 3-16 – Processo de sintonia do Kd e ds de forma iterativa

3. Determinação do ganho proporcional

   A parte proporcional diminui o erro devido à inércia e a carga. Esta força é proporcional ao erro de posição e a constante de proporcionalidade é K_p. Para determinar K_p deve-se aumentá-lo e verificar o amortecimento do sistema, até se verificar um amortecimento crítico.

   O amortecimento é verificado manualmente. Girando o eixo do motor com a mão, percebe-se que a constate de "mola" aumenta conforme o K_p fica maior. Quando se gira o eixo, o controlador percebe o erro de posição e tenta retornar o eixo para a posição inicial. Se K_p é muito pequeno, este retorno é muito lento e o sistema está superamortecido. Se K_p muito grande, o retorno é rápido, haverá um sobre-sinal e o sistema demorarão a estabilizar. O valor de K_p deve ser o maior possível de forma a não causar sobre-sinal.

O AN-693 (Apêndice B) sugere iniciar K_p com dois e dobrá-lo a cada iteração até a situação ideal.

Pelo software é possível mudar K_p até que o valor ideal seja encontrado.

4. Determinação do ganho integral

   A parte integral prove uma força que pode eliminar o erro de seguimento enquanto o eixo esta rodando e a deflexão de carregamento estático enquanto o eixo está parado.

   A constante de proporcionalidade é K_i.

   Altos valores de K_i faz com que haja rápida compensação do torque, porém aumenta o sobre-sinal e o tempo para o sistema estabilizar. O valor de K_i deve ser o menor que dê uma solução de compromisso entre sobre-sinal, tempo de acomodação e o tempo para cancelar o efeito de um torque estático.

   A força corretiva proporcionado pela parte integral aumenta linearmente com o tempo. I_l limita esta força, prevenindo para que ela não aumente até infinito.

   Em muitos sistemas I_l pode ser configurado para seu valor máximo sem prejuízos. Se I_l for zero a parte integral não terá efeito.

   O último passo do modo "Tuning" é configurar K_i e I_l.

4. RESULTADOS

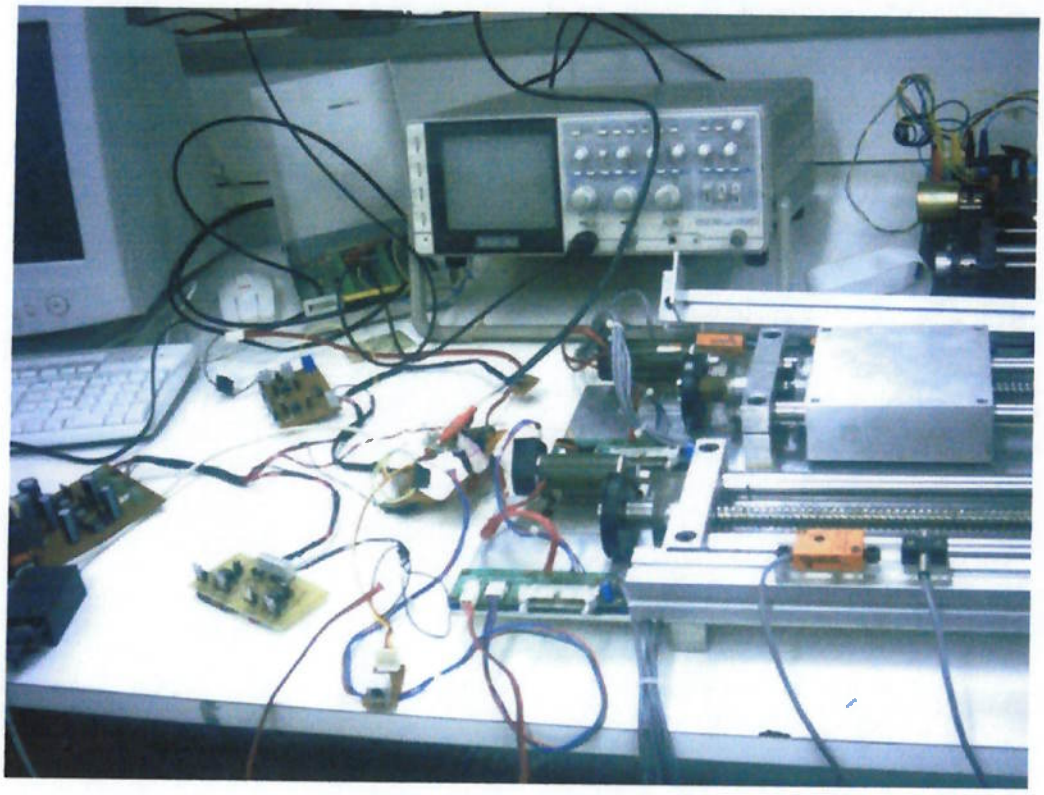


Fig. 4-1- Protótipo funcional

Para a análise dos resultados, construiu-se um protótipo capaz de demonstrar a capacidade e a viabilidade do projeto apresentado.

O protótipo constitui-se de um hardware eletrônico composto de 2 drivers de potência para acionamento dos motores, uma CPU de controle para monitoramento do LM629, duas guias compostas de motores de corrente contínua e dois encoders com leitura em quadratura. Para tanto fez-se as ligações elétricas e mecânicas necessárias afim de observar o comportamento dos motores no sistema mecânico.

Observou-se que o protótipo construído é capaz de comandar sistemas de posicionamento com velocidades variáveis com precisão razoável, visto que esta depende muito dos encoders utilizados e das folgas mecânicas do sistema de transmissão. Para esta aplicação observou-se que a precisão variava entre 1 e 2 milímetros, utilizando um encoder de resolução de 500 linhas e um sistema mecânico um tanto quanto desalinhado.

Do hardware construído, notou-se que a fonte utilizada apresentava uma queda de rendimento quando os motores eram acionados ao mesmo tempo em velocidades altas. Isto fez com que a demonstração ficasse prejudicada quando altas velocidades eram empregadas aos motores. Para tento basta utilizar uma fonte com potência maior.

Quanto aos drivers apresentados, notou-se que foram muito eficientes e não apresentaram muitos problemas com dissipação de calor nos Mosfets, visto que o acionamento em PWM tende a ser ideal.

4.1. Hardware

Nesta seção serão apresentadas as placas confeccionadas.

4.1.1. Driver

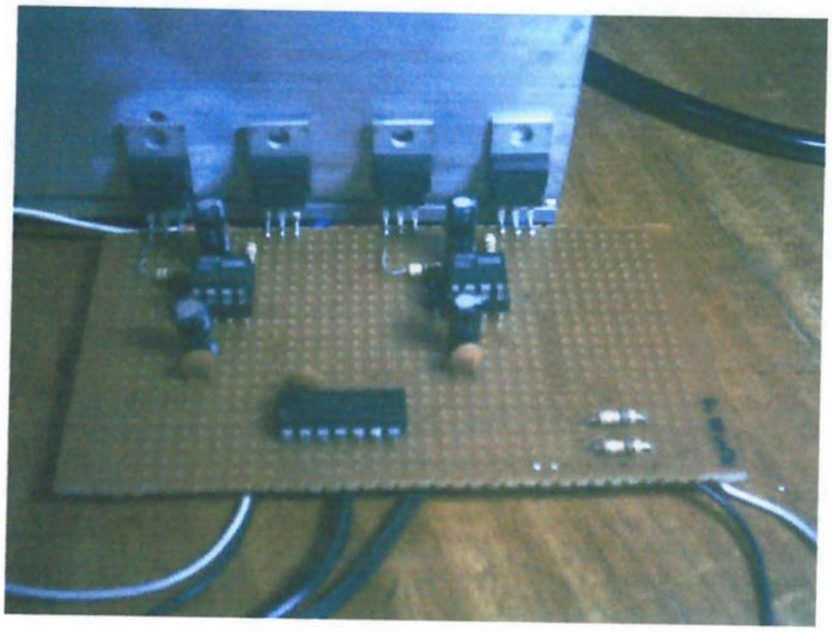


Fig. 4-2 - Driver para acionamento do motor

O Driver construído visou o funcionamento de motores de tensão de operação de 120VCC e corrente de 8A, porém pode ser utilizado para o acionamento de qualquer tipo de motor de corrente contínua desde que a tensão mínima de operação do motor seja de 24 VCC e a corrente não exceda 8A.Utilizou-se um dissipador que fosse suficiente para dissipar a potência de 4 MOSFETs de 50W cada, ou seja, 200W aproximadamente, pois no pior caso quando os dois motores estiverem funcionando quatro transistores estarão conduzindo.

4.1.2. Placa CPU

A placa CPU construída foi desenvolvida em uma placa padrão visando a versatilidade e facilidade da construção. Os componentes foram selecionados conforme as necessidades e requisitos implícitos de outras partes do hardware. A placa atende aos requisitos e necessidades do trabalho.

4.2. Software

O software foi escrito em linguagem C para ser compilado usando o compilador PCM 3.147, que compila o código para ser executado em microcontroladores PIC de 14bits da MicroChip, e usa alguns arquivos de include proprietários deste compilador. O código-fonte comentado está disponível em anexo.

O software apresenta o fluxo proposto e os comandos enviados para o LM629 foram depurados usando a porta serial e enviam os dados da forma correta.

O teste com hardware não pode ser feito até a presente data devido a atrasos no processo de importação dos LM629.

5. DISCUSSÃO

Os resultados apresentados pelo protótipo montado mostram-se muito satisfatórios se levarmos em consideração o fato de que várias adaptações tiveram de ser feitas, assim como várias dificuldades mostraram-se relevantes no bom desempenho do sistema.

Observa-se que o tuning dos atuadores, ou seja, a definição dos parâmetros do controlador PID, é muito difícil de ser feita, pois os parâmetros são muito influenciadores no desempenho do movimento e na sua precisão.

O parâmetro Kp (parte proporcional do PID) que é responsável pela rigidez do movimento dos atuadores é um dos parâmetros mais "lineares" e de mais fácil calibração e é responsável para contornar problemas de rigidez do sistema.

O parâmetro Kd (parte derivativa do PID) mostrou-se mais difícil de ser calibrado, pois uma pequena mudança deste fator, torna o movimento dos atuadores muito melhor ou muito pior. Observa-se que conforme este fator é mudado, pode-se observar um movimento cheio de "soquinhos" dos atuadores. O parâmetro ds é um fator relacionado com o tempo derivativo do intervalo de amostragem. A combinação dos parâmetros Kd e ds, influenciam na estabilização estática do atuador, ou seja, influencia no sobresinal do sistema de posicionamento. Quando este fator está bem ajustado, o motor fica parado estavelmente, senão este fica oscilando em alta freqüência nos dois sentidos sem nunca estabilizar sua posição.

O parâmetro Ki (parte integrativa do PID) não foi estudada neste protótipo pois os testes foram feitos sem carga.

A calibração em geral é difícil de ser feita, porém pode melhorar muito o desempenho de um sistema mecânico (atuador + transmissão) se corretamente efetuada.

6. CONCLUSÕES

O hardware de controle apresentado neste trabalho é suficiente para se implementar o controle da máquina proposta possibilitando a geração de trajetória linear da ferramenta. Trajetórias mais sofisticadas são possíveis usando o recurso de *breakpoints*, que avisa quando o motor chegou a determinada posição, disponível no LM629 e fazendo o pré-processamento dos pontos a serem interpolados de forma a se encontrar os *breakpoints* onde devem ocorrer as mudanças de velocidade dos motores de forma a se obter os perfis de velocidade que gerem a trajetória desejada. Para isso o microcontrolador embarcado seria usado como intermediário entre o LM629 um PC (ou uma CPU dedicada para esta finalidade) e o microcontrolador seria responsável em receber os *breakpoints* e velocidades calculados pelo PC via canal serial e gerenciar todas as peculiaridades do LM629, como estouro de contadores, limites de erros integrais entre outros. A parte do hardware que se designa aos atuadores (drivers de potência) atende aos requisitos presentes, porém em uma aplicação real deveria ser mais robusto, ou seja, possuir sensores de corrente para evitar super-aquecimento no caso de algum motor travar, possuir um cooler mais adequado para dissipação da energia entre outras características para melhorar o desempenho e evitar falhas. A fonte utilizada também poderia ser melhorada, visto que a tensão utilizada no motor atualmente não é o valor exato de operação do motor.

Portanto conclui-se que o hardware apresentado é suficiente para demonstrar a funcionalidade dos recursos explorados neste trabalho, porém deve ser melhorado para uso comercial.

7. ANEXO – CÓDIGO FONTE DO SOFTWARE DE TESTE DO HARDWARE

```
/***********Software de controle*********/
/* Arquivo principal
   27/10/2003
   Marcel Henrique Vieira Gonçalves
   Daniel de Paula Nestrovsky
   Versão 0101
   Compilador PCW                        */

/** Para funcionar no modo com interpolação deve-se descomentar a próxima linha
  * Modo interpolação: Filtros do PID pré-definidos
  *                    Interpolação ponto a ponto com entrada pelo usuário
  *                    através do terminal
  * Modo calibração:   Para encontrar os coeficentes dos filtros
  */
#define interpolacao

#include "C:\Workdir\Documentos\TF\shaping0101.h"
  #include <float.h>
  #include <math.h>
  #include <stdio.h>
  #include <string.h>
  #include <stdlib.h>

#ZERO_RAM

/*Variaveis Locais*/
UCHAR EstPrinc;
UCHAR EstModoManual;
UCHAR EstModoAuto;
UCHAR EstModoTunning;

UCHAR UsrStr[8];     //String para receber os parâmetros via comunicação serial
UCHAR contled;
int1 EstLed;

/*Contantes*/
const unsigned char ACEITA[] = ", a/v: ";

/*Prototipos Locais*/
void  main(void);
UCHAR RDSTAT ( void );
void  Inicializacao  (void);
void  WriteCmd (UCHAR aCmd);
void  WriteWord (UCHAR *aData);
void  ReadWord (UCHAR *aData);
void  SendCmd (UCHAR aCmd, UCHAR *aData, UCHAR aDataSize);
void  GetParam (UCHAR aCmd, UCHAR *aData, UCHAR aDataSize);
void  LM629HdwReset(int1 aMotor);
void  SelectMotor (int1 aMotor);
void  SendTrajParam (int32 aPos, int32 aVel, int32 aAcel);
void  SendVelParam (int1 aDir, int1 aParar, int32 aVel, int32 aAcel);
void  Start(void);
void  CalculateTrajParam (float aXnext, float aYnext, float aF);
void  LoadFilterParam (UCHAR aTd, int16 aKp, int16 aKi, int16 aKd, int16 aIl);
void  UpdateFilters(void);

#int_RTCC
RTCC_isr()        //1.6ms
  {
  ContLed++;
  if (ContLed == 100)
    {
    EstLed = ~EstLed;
    output_bit(LED, EstLed);
    }
  //Relogios();
  }

#int_TIMER1
TIMER1_isr()      //10.3ms
  {

  }

#int_TIMER2
TIMER2_isr()      //100us
  {
  }


void main(void)
  {
  unsigned int16 Kd, Kp, Ki, il;
  UCHAR Td;

  float ponto[3];

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_2);
  setup_spi(FALSE);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_32);
  setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);
  setup_timer_2(T2_DIV_BY_4,126,1);
  enable_interrupts(INT_RTCC);
  enable_interrupts(INT_TIMER1);
  enable_interrupts(INT_TIMER2);
  enable_interrupts(global);

  Inicializacao();
  delay_ms(1);
  //disable_interrupts(INT_RTCC);
  disable_interrupts(INT_TIMER1);
  disable_interrupts(INT_TIMER2);

  while(1)
    {
    switch (EstPrinc)
      {
      case Idle: //Liguei E estou aguardando o inicio do recebimento da
                 //trajetoria
        puts("*** Menu Principal");
        puts("1 Manual");
#ifdef Interpolacao
        puts("2 Automatico");
#endif
        puts("3 Tunning");
        gets( UsrStr );
        EstPrinc = atoi( UsrStr );
      break;

      case ModoManual:
        switch (EstModoManual)
          {
          case Mover:
            {
            unsigned char tecla;
            tecla = getc();
            if (tecla == 'd')
              {// Eixo X na direção positiva com velocidade constante
              SelectMotor (MOTORX);   //Seleciona LM629 que controla o eixo X
              SendVelParam (1, 0, VEL_MANUAL, ACEL_MANUAL);  //Envia comando
                                                  // para andar com velocidade
                                                  // constante
              Start();    //Envia comando para iniciar o movimento
              do    //Espera a tecla d ser precionada novamente
                {
                tecla = 'x';
                tecla = getc();
                }
              while (tecla != 'd');
              SelectMotor (MOTORX);
              SendVelParam (1, 1, VEL_MANUAL, ACEL_MANUAL);   //Envia comando
                                                  // que desliga o motor
              Start();
              }
            else if (tecla == 'a')
              {// X negativo
              SelectMotor (MOTORX);
              SendVelParam (0, 0, VEL_MANUAL, ACEL_MANUAL);
              Start();
              do
                {
                tecla = 'x';
                tecla = getc();
                }
              while (tecla != 'a');
              SelectMotor (MOTORX);
              SendVelParam (0, 1, VEL_MANUAL, ACEL_MANUAL);
              Start();
              }

            else if (tecla == 'w')
              {// Y positivo
              SelectMotor (MOTORY);
              SendVelParam (1, 0, VEL_MANUAL, ACEL_MANUAL);
              Start();
              do
                {
                tecla = 'x';
                tecla = getc();
                }
              while (tecla != 'w');
              SelectMotor (MOTORY);
              SendVelParam (1, 1, VEL_MANUAL, ACEL_MANUAL);
              Start();
              }
            else if (tecla == 's')
              {// Y negativo
              SelectMotor (MOTORY);
              SendVelParam (0, 0, VEL_MANUAL, ACEL_MANUAL);
              Start();
              do
                {
                tecla = 'x';
                tecla = getc();
                }
              while (tecla != 's');
              SelectMotor (MOTORY);
              SendVelParam (0, 1, VEL_MANUAL, ACEL_MANUAL);
              Start();
              }
```

```
             else if (tecla == 'h')
               {
               // Guarda a posição atual como HOME: (0, 0) absoluto
               SelectMotor (MOTORX);
               SendCmd ( CMDDfh, NULL, CMDDfhSize );
               SelectMotor (MOTORY);
               SendCmd ( CMDDfh, NULL, CMDDfhSize );
               }
             else if (tecla == 'x')
               EstModoManual = Sair;
             }
           break;

           case Sair:
             EstPrinc = Idle;
             EstModoManual = Mover;
           break;
           }
       break;

#ifdef Interpolacao
       case ModoAuto:
         switch ( EstModoAuto )
           {
           case ProximoPonto:
             {
             // Recebe próximo ponto e velocidade de avanço
             int i;
             for (i=0 ; i<2 ; i++)
               {
               printf ("\n%c: ", (i==0) ? 'x' : 'y' );
               gets( UsrStr );

               ponto[i] = atof( UsrStr );

               }
             printf ( "\nVel: " );
             gets( UsrStr );
             if ( isdigit( UsrStr[0] ) )
             ponto[2] = atof( UsrStr );

             EstModoAuto = CalcTrajParam;
             }
           break;

           case CalcTrajParam:
             CalculateTrajParam (ponto[0], ponto[1], ponto[2]);
             EstModoAuto = SincronizaEixos;
           break;

           case SincronizaEixos:
             // Sincroniza os eixos
             SelectMotor(MOTORX);
             Start();
             SelectMotor(MOTORY);
             Start();
             EstModoAuto = FimProg;
           break;

           case FimProg:
             // Pergunta ao usuário se deseja executar mais uma trajetória
             printf ("\nMais?: ");
             gets(UsrStr);
             if ( UsrStr[0] == 's' )
               EstModoAuto = ProximoPonto;
             else if ( UsrStr[0] == 'n' )
               {
               EstPrinc = Idle;
               EstModoAuto = ProximoPonto;
               }
           break;
           }
       break;
#endif
//#ifndef Interpolacao
       case ModoTunning:
         switch ( EstModoTunning )
           {
           case SelMotor:
             printf ( "\nMotor(0:X, 1:Y), (c)ancela, : " );
             gets ( UsrStr );
             if ( isdigit( UsrStr[0] ) )
               {
               int Motor;
               Motor = atoi( UsrStr );
               if ( Motor <= MOTORY )
                 {
                 Kd = 2; Td = 0; Kp = 0; Ki = 0; il = 0;
                 SelectMotor( bit_test( Motor, 0 ) );
                 LM629HdwReset( bit_test( Motor, 0 ) );
                 LoadFilterParam (Td, Kp, Ki, Kd, il);
                 UpdateFilters();
                 SendTrajParam (0, 0, 0);
                 Start();
                 EstModoTunning = AcertaKd;
                 }
               }
             else if ( UsrStr[0] == 'c' )
               {
               EstPrinc = Idle;
```

```
            }
          break;

        case AcertaKd:
          printf ( "\nKd, (a)ceita: " );
          gets ( UsrStr );
          if ( isdigit( UsrStr[0] ) )
            {
            Kd = atol( UsrStr );
            LoadFilterParam (Td, Kp, Ki, Kd, il);
            UpdateFilters();
            }
          else if ( UsrStr[0] == 'a' )
            {
            EstModoTunning = AcertaDs;
            }
        break;

        case AcertaDs:
          printf ( "\nds%s", ACEITA);
          gets ( UsrStr );
          if ( isdigit( UsrStr[0] ) )
            {
            Td = atoi( UsrStr );
            LoadFilterParam (Td, Kp, Ki, Kd, il);
            UpdateFilters();
            }
          else if ( UsrStr[0] == 'a' )
            {
            EstModoTunning = AcertaKp;
            }
          else if ( UsrStr[0] == 'v' )
            {
            EstModoTunning = AcertaKd;
            }
        break;

        case AcertaKp:
          printf ( "\nKp%s", ACEITA );
          gets ( UsrStr );
          if ( isdigit( UsrStr[0] ) )
            {
            Kp = atol( UsrStr );
            LoadFilterParam (Td, Kp, Ki, Kd, il);
            UpdateFilters();
            }
          else if ( UsrStr[0] == 'a' )
            {
            EstModoTunning = AcertaKi;
            }
          else if ( UsrStr[0] == 'v' )
            {
            EstModoTunning = AcertaDs;
            }
        break;

        case AcertaKi:
          printf ( "\nKi%s",ACEITA );
          gets ( UsrStr );
          if ( isdigit( UsrStr[0] ) )
            {
            Ki = atol( UsrStr );
            LoadFilterParam (Td, Kp, Ki, Kd, il);
            UpdateFilters();
            }
          else if ( UsrStr[0] == 'a' )
            {
            EstModoTunning = Acertail;
            }
          else if ( UsrStr[0] == 'v' )
            {
            EstModoTunning = AcertaKp;
            }
        break;

        case Acertail:
          printf ( "\nil%s", ACEITA );
          gets ( UsrStr );
          if ( isdigit( UsrStr[0] ) )
            {
            il = atol( UsrStr );
            LoadFilterParam (Td, Kp, Ki, Kd, il);
            UpdateFilters();
            }
          else if ( UsrStr[0] == 'a' )
            {
            EstPrinc = Idle;
            EstModoTunning = SelMotor;
            }
          else if ( UsrStr[0] == 'v' )
            {
            EstModoTunning = AcertaKi;
            }
          break;
        }
      break;
//#endif
      default:
        EstPrinc = Idle;
      break;
```

```
        }
      }
    }

void  Inicializacao  (void)
   {
   port_b_pullups (false);

   // Estados iniciais da máquina de estados
   EstPrinc = Idle;
   EstModoTunning = SelMotor;
   EstModoAuto = ProximoPonto;
   EstModoManual = Mover;

#ifdef Interpolacao
   // Executa o reset dos LM629
   LM629HdwReset(MOTORX);
   LM629HdwReset(MOTORY);

   // Carrega os filtros do PID para cada LM629
   SelectMotor(MOTORX);
   LoadFilterParam (0, 1000, 200, 2000, 1000);  //(Td, Kp, Ki, Kd, il)
   UpdateFilters();
   SelectMotor(MOTORY);
   LoadFilterParam (0, 1000, 200, 2000, 1000);
   UpdateFilters();
#endif

   }

// Estas funções seguem o timing definido no DataSheet

// Lê o byte de Status do LM629
UCHAR RDSTAT ( void )
   {
   UCHAR aData;
   output_low(bPS);
   output_low(bRD);
   delay_us(1);
   aData = GetData;
   output_high(bRD);
   printf ("R%x", aData);
   return (aData);
   }

// Envia um byte com o comando para o LM629
void  WriteCmd (UCHAR aCmd)
   {
   CheckBusyBit;
   output_low(bPS);
   output_low(bWR);
   SetData(aCmd);
   delay_us(1);
   output_high(bWR);
   }

// Envia uma palavra de 16bits para o LM629
// Os parâmetros dos comandos do LM629 são enviados em palavras de 16bits
void  WriteWord (UCHAR *aData)
   {
   CheckBusyBit;
   output_high(bPS);
   output_low(bWR);
   SetData(*aData);
   printf(" %x ", *aData);
   delay_us(1);
   output_high(bWR);
   output_low(bWR);
   SetData(*(aData+1));
   printf(" %x ", *(aData+1));
   delay_us(1);
   output_high(bWR);
   }


// Recebe uma palavra de 16bits do LM629
// O LM629 reporta dados em palavras de 16bits
// Deve-se passar um poonteiro para um UCHAR[2]
void ReadWord (UCHAR *aData)
   {
   CheckBusyBit;
   output_high(bPS);
   output_low(bRD);
   delay_us(1);
   *aData = GetData;
   output_high(bRD);
   output_low(bRD);
   delay_us(1);
   *(aData+1) = aData;
   output_high(bRD);
   CheckBusyBit;
   }

// Envia um comando de controle e seus parâmetros para o LM629
void SendCmd (UCHAR aCmd, UCHAR *aData, UCHAR aDataSize)
   {
   UCHAR i;
   // Envia o comando
   WriteCmd(aCmd);
   printf("\n%x ", aCmd);
   // Envia os parâmetros
   for (i=0; i<aDataSize; i+=2)
     {
     WriteWord( (aData+i) );
     }
   }


 // Envia o comando de Data Report e recebe palavras de controle do LM629
 // O ponteiro aData deve ser grande suficiente para receber os dados
 void GetParam (UCHAR aCmd, UCHAR *aData, UCHAR aDataSize)
   {
   UCHAR i;
   // Envia o comando de report
   WriteCmd(aCmd);
   // Recebe os dados
   for (i=0; i<aData; i+=2)
     {
     ReadWord(aData+i);
     }
   }

 // Reseta o LM629 por hadware e suas interrupções
 // Executa a rotina de reset proposta no AN-693
 void LM629HdwReset(int1 aMotor)
   {
   UCHAR sts;
   UCHAR data[2];
   SelectMotor (aMotor);

   // Reset por hardware
   do
     {

     do
       {

       //if (aMotor == MOTORX)
         output_low(bRSTX);
       //else
       //  output_low(bRSTY);

       delay_ms(5);

       //if (aMotor == MOTORX)
         output_high(bRSTX);
       //else
       //  output_high(bRSTY);
       sts = RDStat();

       delay_ms(5);    // Desve-se esperar pelo menos 1.5ms
       sts = RDStat();

       }
     while ( (sts != 0xc4) && (sts != 0x84) );

     // Reset das interrupções
     data[0] = 0x00;
     data[1] = 0x00;

     SendCmd (CMDRsti, &data[0], CMDRstiSize);
     CheckBusyBit;
     sts = RDStat();
     printf("%x\n", sts);
     }
   while ( (sts != 0xc0) && (sts != 0x80) );

   }


 // Seleciona o LM629 a ser usado fazendo o intertravamento entre eles
 // Ativa o chip select de um e desativa o do outro, para que os componetes
 //possam usar o mesmo bus de dados
 void SelectMotor (int1 aMotor)
   {
   if (aMotor == MOTORX)
     {
     output_high(bCSY);
     output_low (bCSX);
     printf("x");
     }
   else
     {
     output_high(bCSX);
     output_low(bCSY);
     printf("y");
     }
   }

 // Envia os parâmetros de trajetória para o modo posição
 // Os bytes mais significativos devem ser enviados antes dos LSB
 void  SendTrajParam (int32 aPos, int32 aVel, int32 aAcel)
   {
   UCHAR traj[14];
   traj[0]=0x00;   // Position Mode
   traj[1]=0x2B;   // Aceleração e Velocidade absolutas e Posição relativa
   traj[2]=*(&aAcel+3);
   traj[3]=*(&aAcel+2);
   traj[4]=*(&aAcel+1);
   traj[5]=*(&aAcel);
   traj[6]=*(&aVel+3);
   traj[7]=*(&aVel+2);
   traj[8]=*(&aVel+1);
   traj[9]=*(&aVel);
   traj[10]=*(&aPos+3);
   traj[11]=*(&aPos+2);
   traj[12]=*(&aPos+1);
   traj[13]=*(&aPos);

   SendCmd (CMDLtrj, &traj[0], 14);
   }

// Envia parâmetros de trajetória para o modo velocidade
void  SendVelParam (int1 aDir, int1 aParar, int32 aVel, int32 aAcel)
   {
  UCHAR traj[10];

  traj[0]=0x08;     // Velocity mode
  if (aParar == 1) // aParar == 1 comando para parar
    {
    bit_set(traj[0],0);
    traj[1]=0x00;

    SendCmd (CMDLtrj, &traj[0], 2);
    return;
    }

  if (aDir == 1)  // aDir == 1 sentido positivo
    bit_set(traj[0],4);

  traj[1]=0x28;   // Aceleração e Velocidade absolutas
  traj[2]=*(&aAcel+3);
  traj[3]=*(&aAcel+2);
  traj[4]=*(&aAcel+1);
  traj[5]=*(&aAcel);
  traj[6]=*(&aVel+3);
  traj[7]=*(&aVel+2);
  traj[8]=*(&aVel+1);
  traj[9]=*(&aVel);

  SendCmd (CMDLtrj, &traj[0], 10);
  }

// Recebe o proximo ponto (em coordenadas relativas e mm),
// a velocidade de avanço (em mm/s), calcula os parametros
// da trajetoria e envia para os motores.
void CalculateTrajParam (float aXnext, Float aYnext, float aF)
  {
  int32 POS, VEL, ACEL;
  float aFx, aFy;        //Velocidades de avanço
  float aAcelx, aAcely; //Acelerações

  // Cálculo dos parâmetros
  if ( (fabs(aXnext) > fabs(aYnext)) )
    {
    // Projeção de aF no eixo X
    aFx = ((fabs(aXnext))/sqrt((aXnext*aXnext)+(aYnext*aYnext)))*(aF);

    // Verifica se não excede a velocidade máxima
    if (aFx > VEL_MAX) aFx = VEL_MAX;

    // Ajusta a velocidade de y em relação a x
    aFy = fabs(((aYnext)/(aXnext))*aFx);

    // Impõe a máxima aceleração ao eixo x
    aAcelx = ACEL_MAX;
    // Ajusta a aceleração de y em relação a x
    aAcely = fabs(((aYnext)/(aXnext))*aAcelx);
    }
  else
    {
    // Procedimento análogo ao outro caso
    aFy = ((fabs(aYnext))/sqrt((aXnext*aXnext)+(aYnext*aYnext)))*(aF);
    if (aFy>VEL_MAX) aFy = VEL_MAX;
    aFx = fabs(((aXnext)/(aYnext))*aFy);

    aAcely = ACEL_MAX;
    aAcelx = fabs(((aXnext)/(aYnext))*aAcely);
    }

  // Conversão dos parâmetros de mm para counts
  aXnext = aXnext * mm2revX * rev2counts;
  aFx    = aFx    * mm2revX * rev2counts;
  aAcelx = aAcelx * mm2revX * rev2counts;

  aYnext = aYnext * mm2revY * rev2counts;
  aFy    = aFy    * mm2revY * rev2counts;
  aAcely = aAcely * mm2revY * rev2counts;

  // Conversão da unidade de tempo, de s para samples e s*s para sample*sample
  aFx    = aFx    * sec2sample;
  aAcelx = aAcelx * sec2sample * sec2sample;

  aFy    = aFy    * sec2sample;
  aAcely = aAcely * sec2sample * sec2sample;

  //Escalonando a velocidade e a aceleração
  aFx    = aFx    * SCALE_FACTOR;
  aAcelx = aAcelx * SCALE_FACTOR;
  aFy    = aFy    * SCALE_FACTOR;
  aAcely = aAcely * SCALE_FACTOR;
  printf("%f", aFx);

  // Verifica se o valor da aceleração não excede o valor da velocidade
  // No LM629 esta condição deve ser satisfeita
  if (aAcelx > aFx)
    {
    aAcelx = aFx;
    aAcely = aFy;
    }

  // Envia parâmetros para cada LM629
  SelectMotor ( MOTORX );
  POS = aXnext;
  VEL = aFx;
  ACEL = aAcelx;
  SendTrajParam ( POS, VEL, ACEL );
  SelectMotor ( MOTORY );
  POS = aYnext;
  VEL = aFy;
  ACEL = aAcely;
  SendTrajParam ( POS, VEL, ACEL );
  }

// Carrega os filtros do PID no LM629
void  LoadFilterParam (UCHAR aTd, int16 aKp, int16 aKi, int16 aKd, int16 aIl)
  {
  UCHAR filters[10];
  filters[0]=aTd;
  filters[1]=0x0F;   // Kp, Ki, Kd, il seram carregados
  filters[2]=*(&aKp+1);
  filters[3]=*(&aKp);
  filters[4]=*(&aKi+1);
  filters[5]=*(&aKi);
  filters[6]=*(&aKd+1);
  filters[7]=*(&aKd);
  filters[8]=*(&aIl+1);
  filters[9]=*(&aIl);

  SendCmd (CMDLfil, &filters[0], 10);
  }

// Inicia a execução da trajetória
// Após carregar os parâmetros, este comando deve ser executado
// para iniciar o movimento
void  Start(void)
  {
  SendCmd ( CMDStt, NULL, CMDSttSize );
  }

// Atualiza os coeficiêntes do filtro PID
// Após carregar os filtros, este comando deve ser executado
// para atualizar os coeficientes no LM629
void  UpdateFilters(void)
  {
  SendCmd ( CMDUdf, NULL, CMDUdfSize );
  }
```

```
/***********Software de controle*********/
/* Arquivo de Include
   27/10/2003
   Marcel Henrique Vieira Gonçalves
   Daniel de Paula Nestrowsky
   Versão 0101                          */

#include <16F876a.h>

#use delay(clock=20000000)
#use rs232(baud=9600,xmit=pin_c6,rcv=pin_c7)
#fuses HS,NOPUT,NOWDT,NOLVP,NOPROTECT,NOBROWNOUT

#define UCHAR  unsigned char

#define MOTORX      0
#define MOTORY      1

#define LED     PIN_C3

#define bCSX    PIN_A5
#define bCSY    PIN_C2
#define bRSTX   PIN_C0
#define bRSTY   PIN_C1

//#define bINTER  PIN_C2
#define bPS     PIN_A1
#define bRD     PIN_A2
#define bWR     PIN_A3
//#define Bot1    PIN_C1
//#define Bot2    PIN_C4

#define BDATA   PORTB
#define GetData      input_b()
#define SetData(x)   output_b(x)
#define RDBusy (RDStat()) & 0x01
#define RDTrajComplete (RDStat()) & 0x04
#define CheckBusyBit while(RDBusy)  //Verifica o busy bit

#define bDATA0  PIN_B0
#define bDATA1  PIN_B1
#define bDATA2  PIN_B2
#define bDATA3  PIN_B3
#define bDATA4  PIN_B4
#define bDATA5  PIN_B5
#define bDATA6  PIN_B6
#define bDATA7  PIN_B7

#define BotApertado 0

//Máquina de Estados Principal
#define Idle                0
#define ModoManual          1
  // Máquina de estado ModoManual
  #define Mover               1
  #define Sair                2
#define ModoAuto            2
  // Máquina de estado ModoAuto
  #define ProximoPonto        0
  #define CalcTrajParam       1
  #define TrajPronta          2
  #define CarregaTrajParam    3
  #define SincronizaEixos     4
  #define FimProg             5
#define ModoTunning         3
  // Máquina de estados ModoTunning
  #define SelMotor            0
  #define AcertaKd            1
  #define AcertaDs            2
  #define AcertaKp            3
  #define AcertaKi            4
  #define Acertail            5

//Comandos para o LM629
//Descrição de cada comando se encotra no Data Sheet
#define CMDReset      0x00
#define CMDResetSize 0

#define CMDDfh        0x02
#define CMDDfhSize    0

#define CMDSp         0x03
#define CMDSoSize     0

#define CMDLpei       0x1d
#define CMDLpeiSize   2

#define CMDLpes       0x1a
#define CMDLpesSize   2

#define CMDSbpa       0x20
#define CMDSbpaSize   4

#define CMDSbpr       0x21
#define CMDSbprSize   4

#define CMDMski       0x1c
#define CMDMskiSize   2

#define CMDRsti       0x1d
```

```
#define CMDRstiSize  2

#define CMDLfil       0x1e


#define CMDUdf        0x04
#define CMDUdfSize    0

#define CMDLtrj       0x1f


#define CMDStt        0x01
#define CMDSttSize     0

#define CMDRdsigs      0x0c
#define CMDRdsigsSize 2

#define CMDRdip        0x09
#define CMDRdipSize    4

#define CMDRddp        0x08
#define CMDRddpSize    4

#define CMDRdrp        0x0a
#define CMDRdrpSize    4

#define CMDRddv        0x07
#define CMDRddvSize    4

#define CMDRdrv        0x0b
#define CMDRdrvSize    2

#define CMDSum         0x0d
#define CMDSumSize     2


// Constantes de conversão
// Encoder acoplado no eixo do motor
#define mm2revX        0.883392//0.02841037  // (rev/mm)
#define mm2revY        0.883392//.2  // (rev/mm)

#define rev2counts     2000 // (counts/rev)  Encoder de 500 linhas acoplado ao eixo do motor
#define sec2sample     .000939//LM_FREQ = 2,18MHz(2048/LM_FREQ) // (sec/sample)
#define SCALE_FACTOR   65536

//Const de definição do projeto
#define ACEL_MAX       30// (mm/s^2)
#define VEL_MAX        70 // (mm/s)

#define VEL_MANUAL     307691  //counts/sample
#define ACEL_MANUAL    1150  //counts/sample/sample
```

8. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Altintas, Yusuf, *Manufacturing Automation,* 1ª Ed, Cambridge University Press, 2000

[2] Baldor Eletric Company (http://www.baldor.com/products/dc_motors.asp)

[3] Beer, Ferdinand P. e Johnston E. Russel, Jr, *Mecânica Vetorial para Engenheiros Cinemática e Dinâmica,* 5ª Ed, Editora Makron Books, 1994

[4] Ogata Katsuhiko, *Engenharia de Controle Moderno,* 3ª Ed, Editora LTC, 1998

[5] Ogata Katsuhiko, *Discrete-time Control System,* Prentice Hall, 1987

APÊNDICE A – DATA SHEET LM628/LM629

National Semiconductor

January 2003

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)

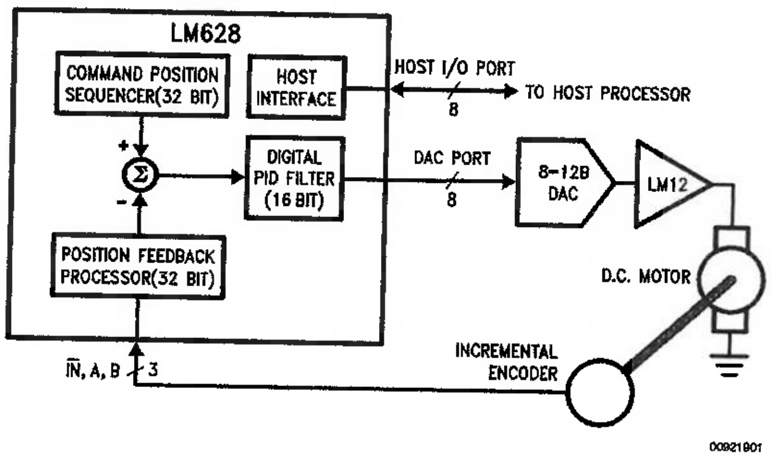
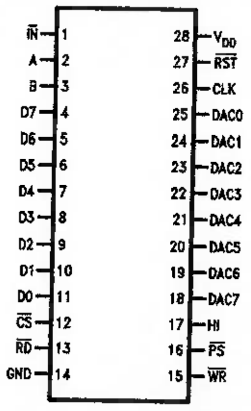


FIGURE 1. Block Diagram

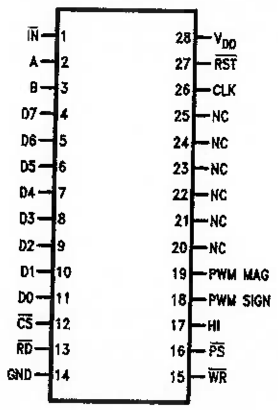
Connection Diagrams

LM628N



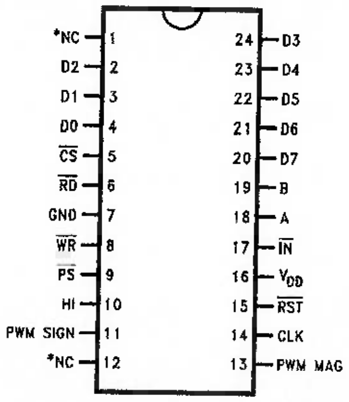
00921902

LM629N



00921903

LM629M



*Do not connect.

00921921

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

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 ($T_A \le 85°C$, (Note 2)	605 mW
ESD Tolerance (C_{ZAP} = 120 pF, R_{ZAP} = 1.5k)	1000V

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	$0 \le V_{IN} \le V_{DD}$	−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	$0 \le V_{OUT} \le V_{DD}$	−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.

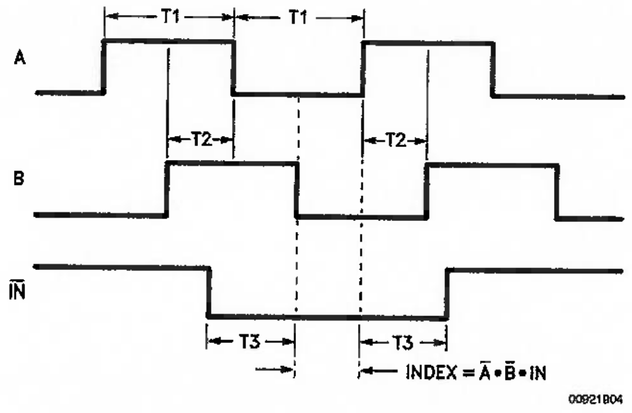


FIGURE 2. Quadrature Encoder Input Timing

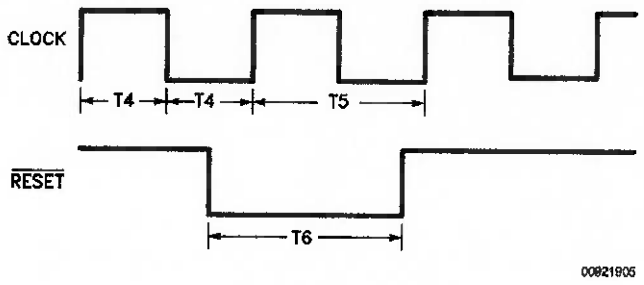


FIGURE 3. Clock and Reset Timing

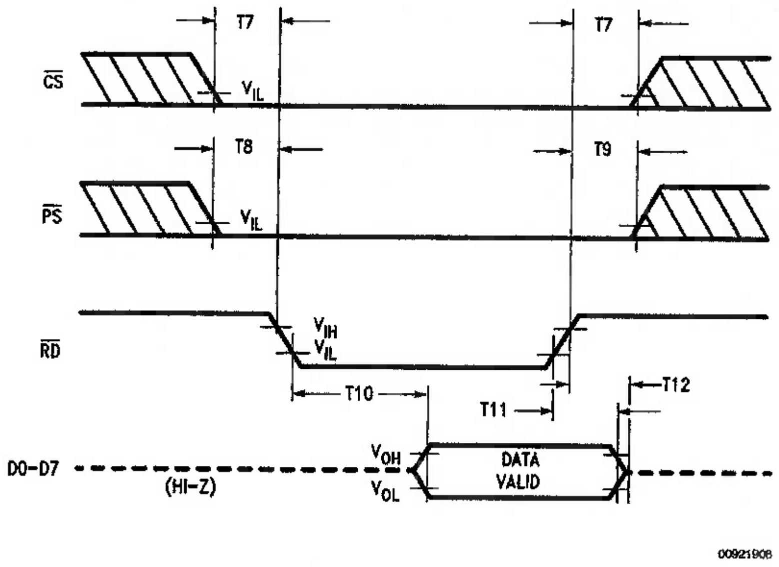


FIGURE 4. Status Byte Read Timing

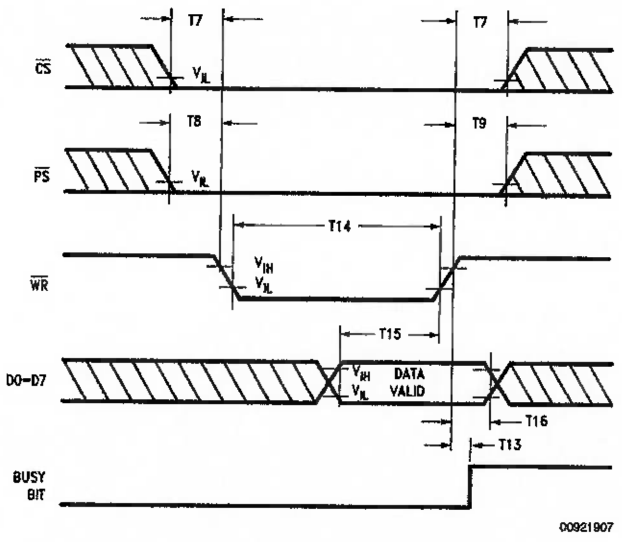


FIGURE 5. Command Byte Write Timing

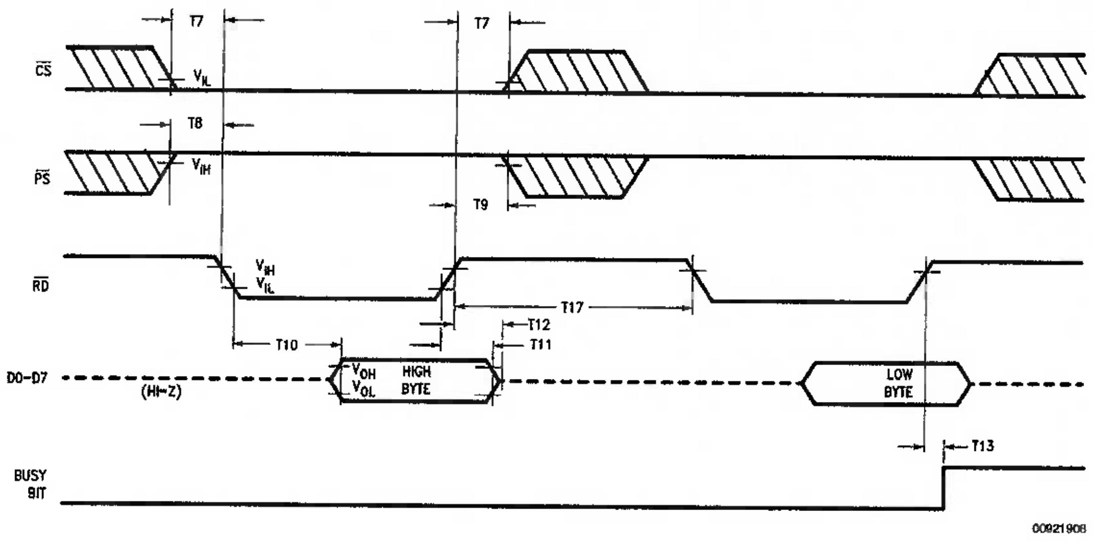


FIGURE 6. Data Word Read Timing

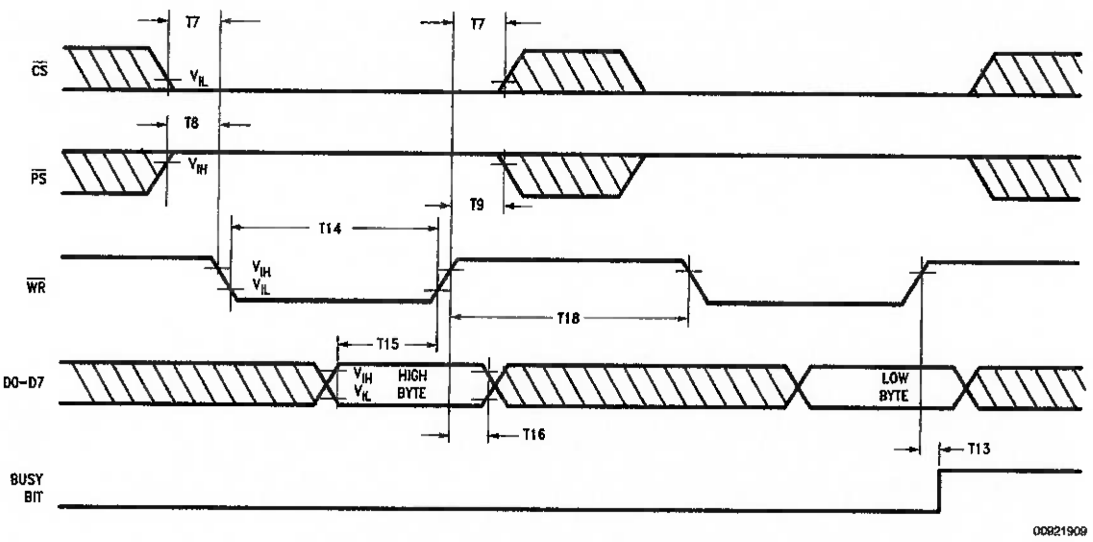


FIGURE 7. Data Word Write Timing

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"

Pinout Description (Continued)

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).

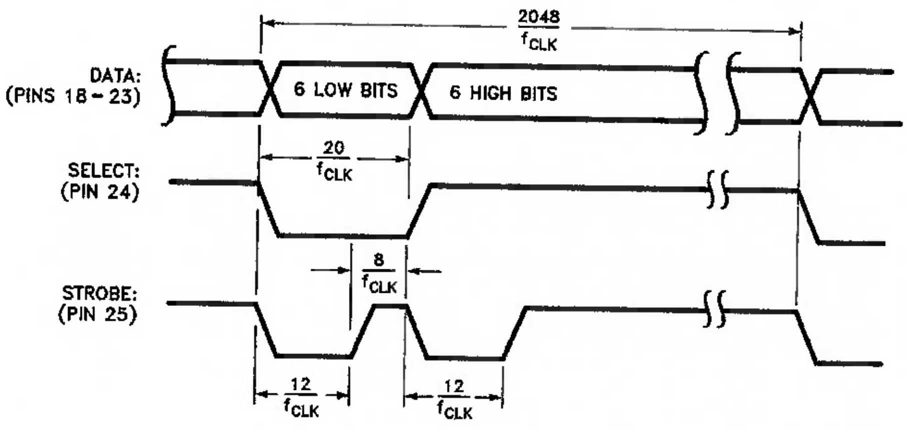


FIGURE 8. 12-Bit Multiplexed Output Timing

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 1* 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.

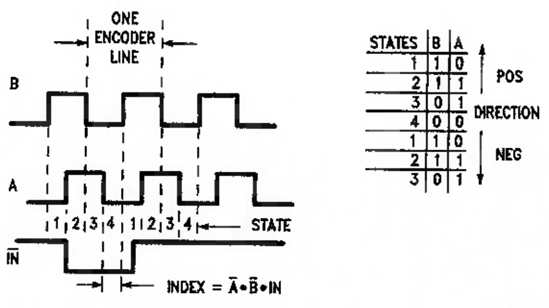
TABLE 1. System Specifications Summary

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)

Theory of Operation (Continued)

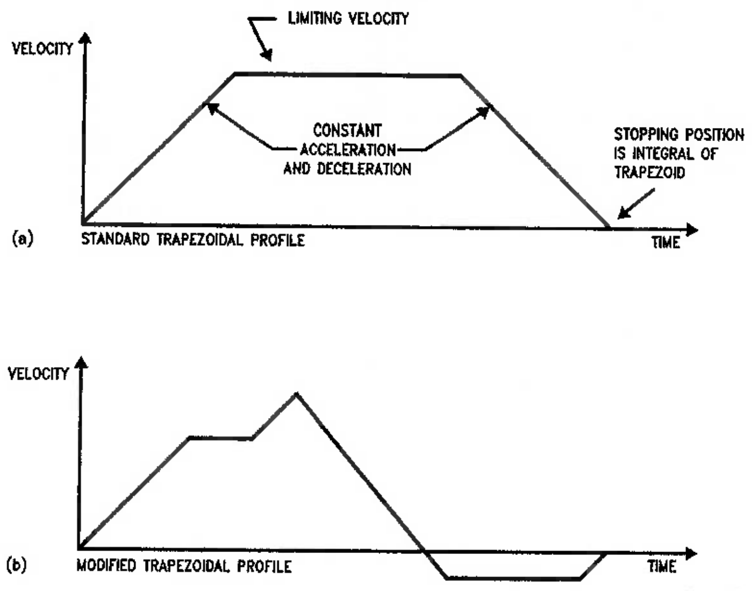
TABLE 1. System Specifications Summary (Continued)

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: $2048/f_{CLK}$



00921911

FIGURE 9. Quadrature Encoder Signals



00921912

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,

Theory of Operation (Continued)

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:

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:

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

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.

Theory of Operation (Continued)

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 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 *Figure 5* thru *Figure 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 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.

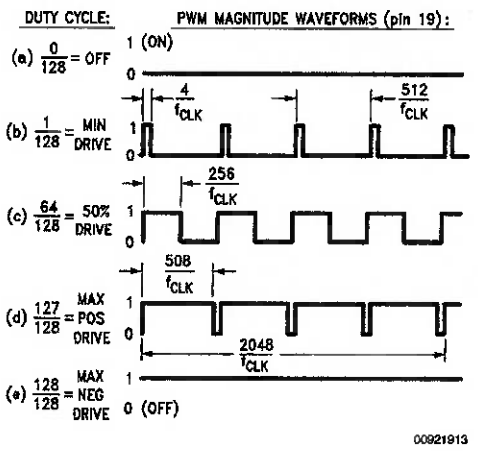


FIGURE 11. PWM Output Signal Format (Sign output (pin 18) not shown)

TABLE 2. 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

Theory of Operation (Continued)

TABLE 2. LM628 User Command Set (Continued)

Command	Type	Description	Hex	Data Bytes	Note
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 4: Commands may be executed "On the Fly" during motion.

Note 5: Commands not applicable to execution during motion.

Note 6: Command may be executed during motion if acceleration parameter was not changed.

Note 7: 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 2* 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.

Initialization Commands (Continued)

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.

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

Interrupt Control Commands
(Continued)

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 3.*

TABLE 3. 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 3* 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 3.* 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 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 4.*

TABLE 4. 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 5.* 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;

Filter Control Commands (Continued)

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 5. 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 8: 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.

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 6*.

TABLE 6. 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

Trajectory Control Commands

(Continued)

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.

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 7.*

TABLE 7. 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

Data Reporting Commands
(Continued)

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 8*.

TABLE 8. 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.

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), 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,

Data Reporting Commands

(Continued)

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.

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

Typical Applications (Continued)

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.

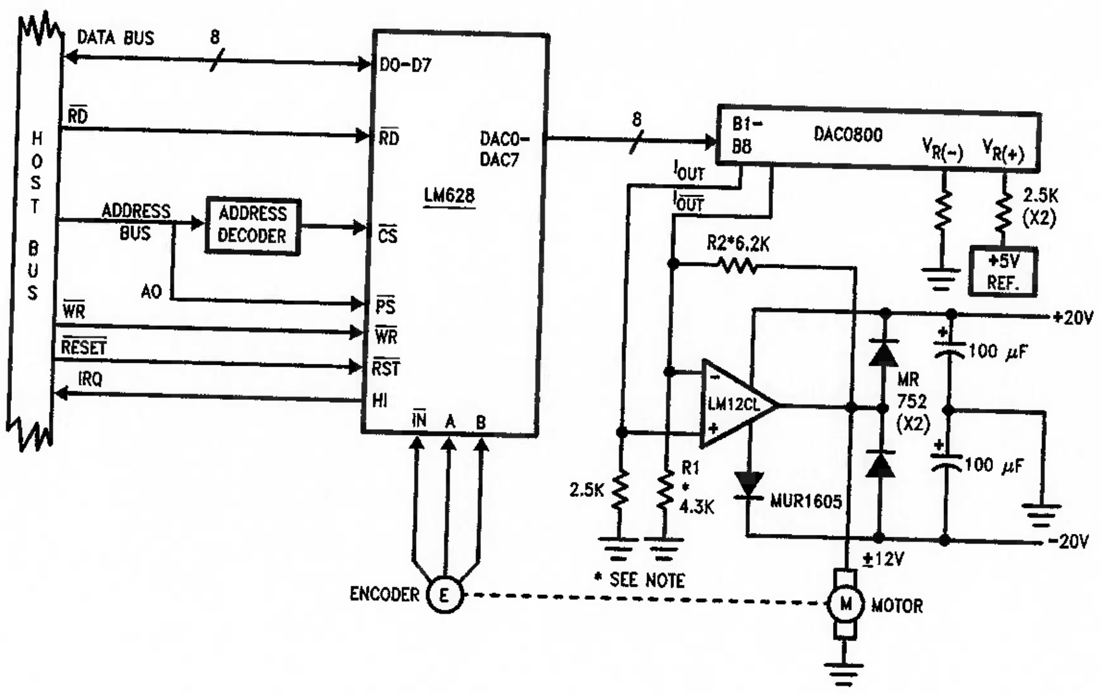
Incremental Encoder Interface

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 17* shows circuitry using the DS26LS31 and DS26LS32.

Typical Applications (Continued)



Note:

$$A_V = \frac{R1 + R2}{R1} \approx 2.4$$

$$\frac{R1 \times R2}{R1 + R2} \triangleq 2.5k$$

FIGURE 12. Host Interface and Minimum System Configuration

Typical Applications (Continued)

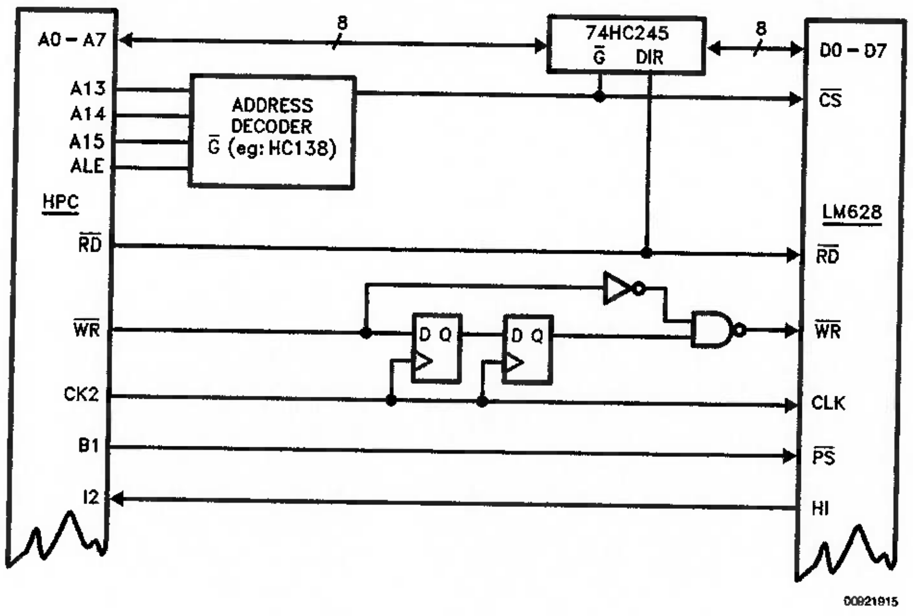
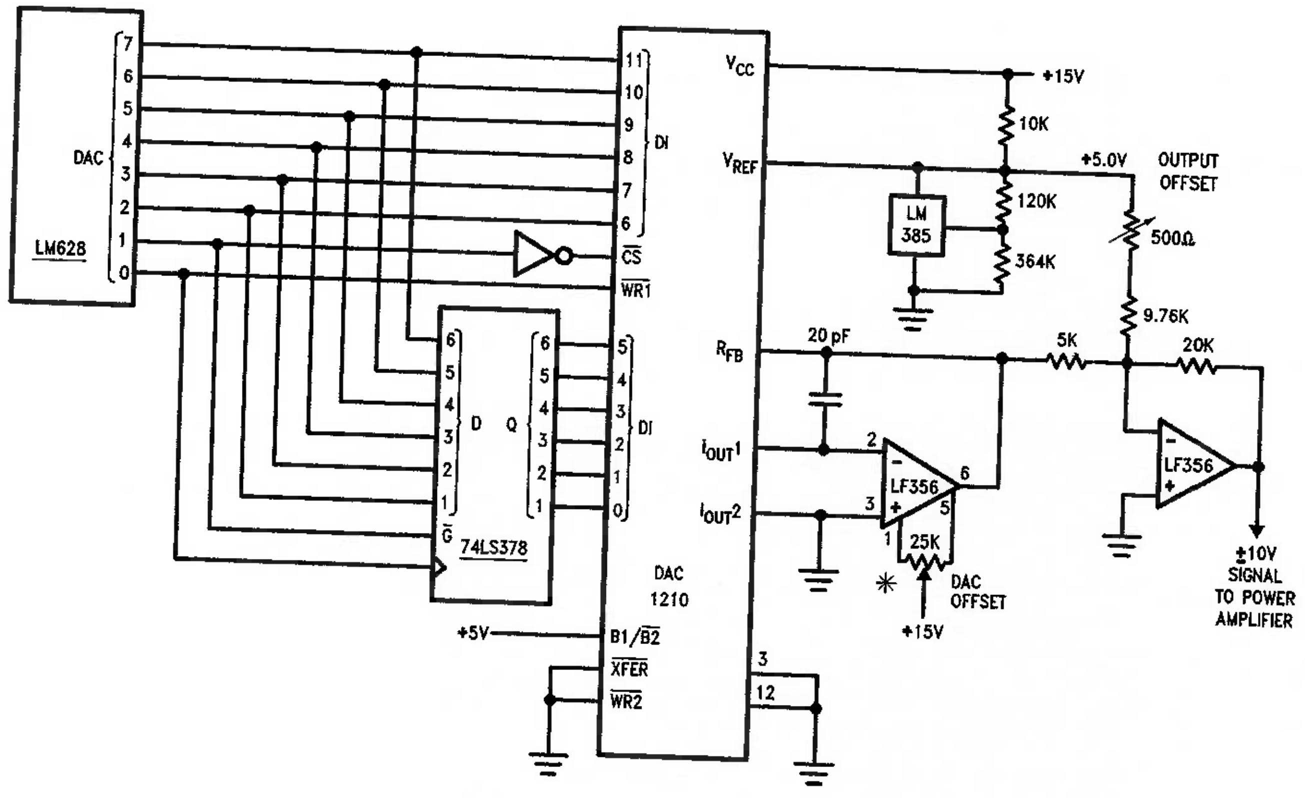


FIGURE 13. LM628 and HPC Interface

Typical Applications (Continued)



*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)

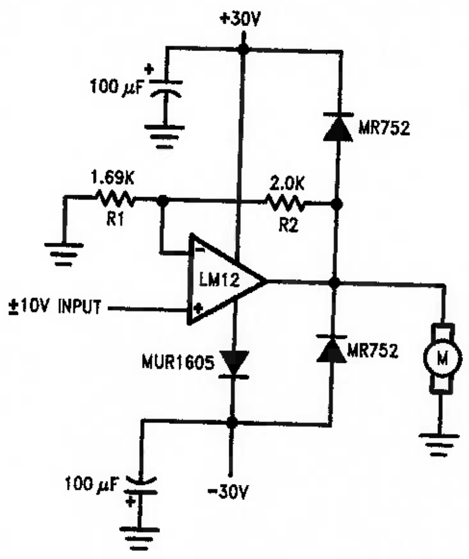


FIGURE 15. Driving a Motor with the LM12 Power Op Amp

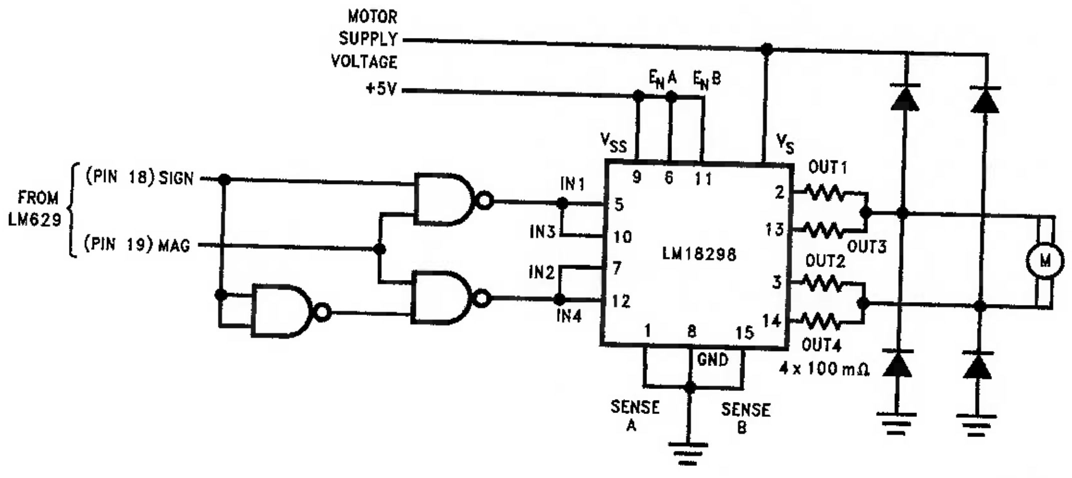


FIGURE 16. PWM Drive for Brush/Commutator Motors

Typical Applications (Continued)

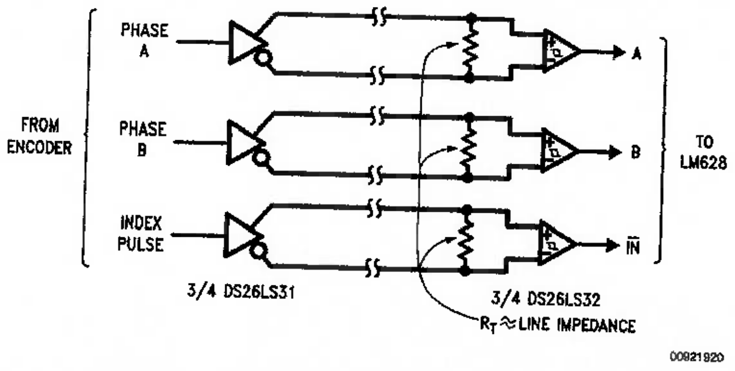
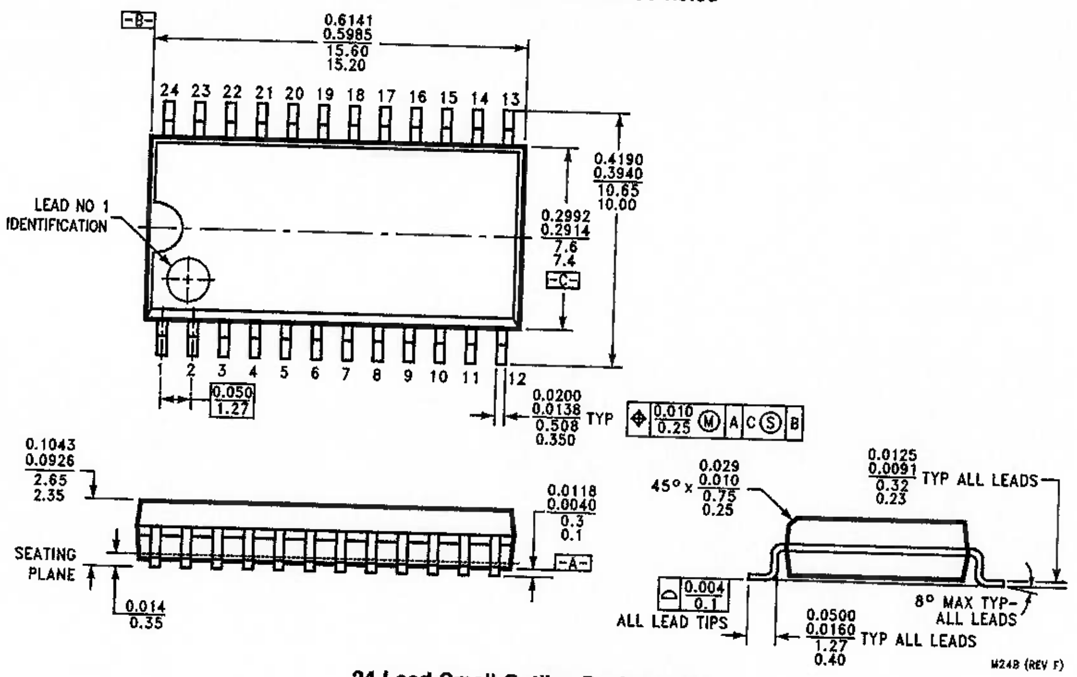
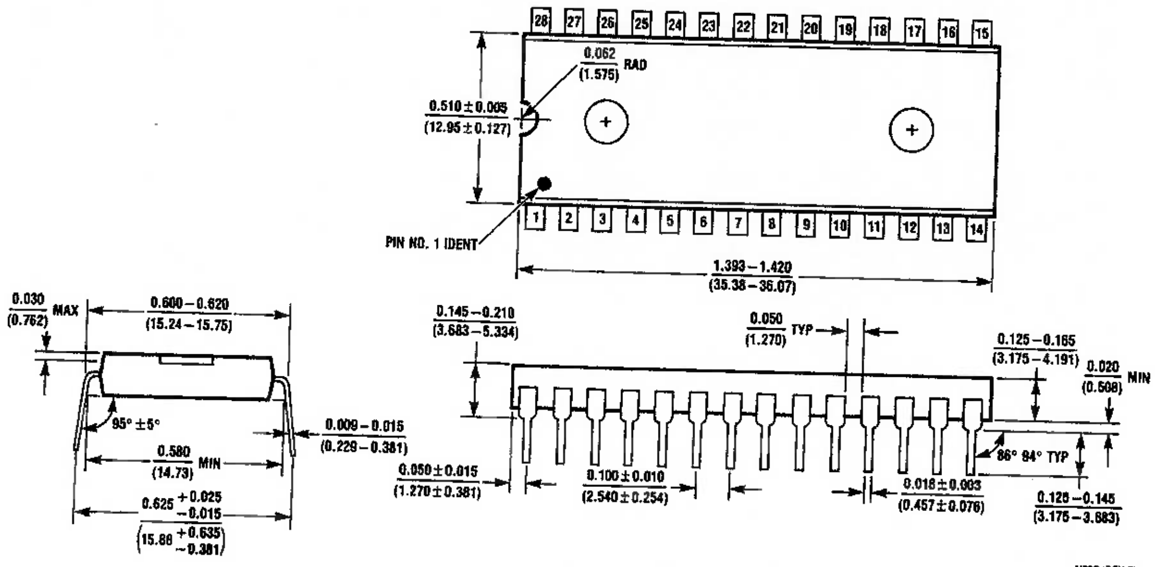


FIGURE 17. Typical Balanced-Line Encoder Input Circuit

Physical Dimensions inches (millimeters) unless otherwise noted



24-Lead Small Outline Package (M)
Order Number LM629M-6 or LM629M-8
NS Package Number M24B



28 Lead Molded Dual-In-Line Package (N)
Order Number LM628N-6, LM628N-8, LM629N-6 or LM629N-8
NS Package Number N28B

Notes

National Semiconductor
Americas Customer Support Center
Email: new.feedback@nsc.com
Tel: 1-800-272-9959

www.national.com

National Semiconductor
Europe Customer Support Center
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor
Asia Pacific Customer Support Center
Fax: 65-6250 4466
Email: ap.support@nsc.com
Tel: 65-6254 4466

National Semiconductor
Japan Customer Support Center
Fax: 81-3-5639-7507
Email: nsj.crc@jksmtp.nsc.com
Tel: 81-3-5639-7560

APÊNDICE B – GUIA DE PROGRAMAÇÃO LM629

LM628 Programming Guide

National Semiconductor
Application Note 693
Steven Hunt
January 1999

Introduction

The LM628/LM629 are dedicated motion control processors. Both devices control DC and brushless DC servo motors, as well as, other servomechanisms that provide a quadrature incremental feedback signal. Block diagrams of typical LM628/LM629-based motor control systems are shown in *Figures 1, 2.*

As indicated in the figures, the LM628/LM629 are bus peripherals; both devices must be programmed by a host processor. This application note is intended to present a concrete starting point for programmers of these precision motion controllers. It focuses on the development of short programs that test overall system functionality and lay the groundwork for more complex programs. It also presents a method for tuning the loop-compensation PID filter. (Note 1)

Reference System

Figure 15 is a detailed schematic of a closed-loop motor control system. All programs presented in this paper were developed using this system. For application of the programs in other LM628-based systems, changes in basic programming structure are not required, but modification of filter coefficients and trajectory parameters may be required.

I. Program Modules

Breaking programs for the LM628 into sets of functional blocks simplifies the programming process; each block executes a specific task. This section contains examples of the principal building blocks (modules) of programs for the LM628.

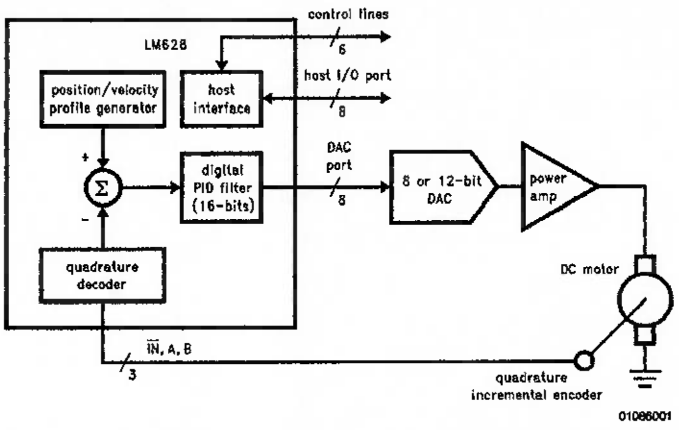


FIGURE 1. LM628-Based Motor Control System

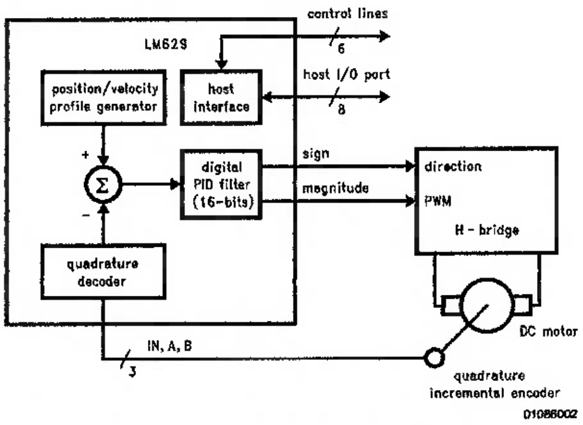


FIGURE 2. LM629-Based Motor Control System

Note 1: For the remainder of this paper, all statements about the LM628 also apply to the LM629 unless otherwise noted.

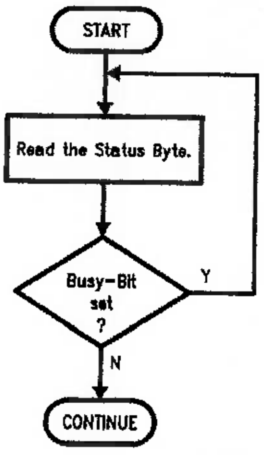
I. Program Modules (Continued)

BUSY-BIT CHECK MODULE

The first module required for successful programming of the LM628 is a busy-bit check module.

The busy-bit, bit zero of the status byte, is set immediately after the host writes a command byte, or reads or writes the second byte of a data word. See *Figure 5*. While the busy-bit is set, the LM628 will ignore any commands or attempts to transfer data.

A busy-bit check module that polls the Status Byte and waits until the busy-bit is reset will ensure successful host/LM628 communications. ***It must be inserted after a command write, or a read or write of the second byte of a data word.*** *Figure 3* represents such a busy-bit check module. This module will be used throughout subsequent modules and programs.



01086003

FIGURE 3. Busy-bit Check Module

Reading the Status Byte is accomplished by executing a RDSTAT command. RDSTAT is directly supported by LM628 hardware and is executed by pulling $\overline{CS}$, $\overline{PS}$, and $\overline{RD}$ logic low.

INITIALIZATION MODULE

In general, an initialization module contains a reset command and other initialization, interrupt control, and data reporting commands.

The example initialization module, detailed in *Table 1*, contains a hardware reset block and a PORT 12 command.

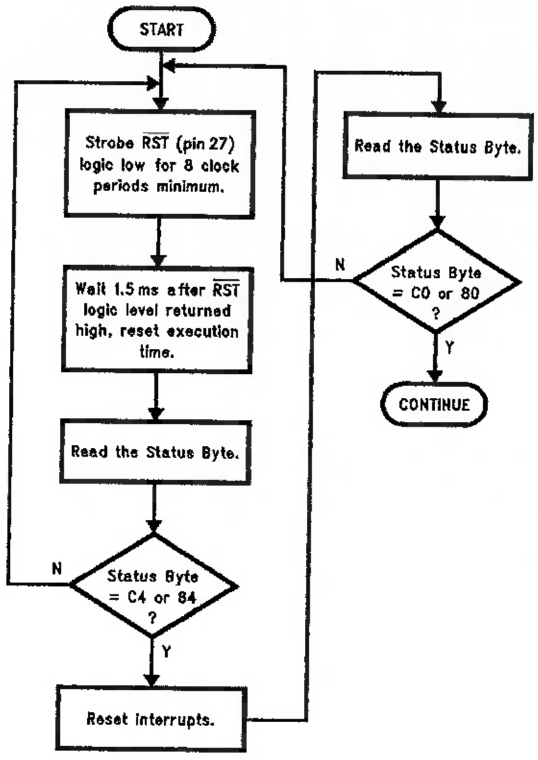
Hardware Reset Block

Immediately following power-up, a hardware reset ***must be*** executed. Hardware reset is initiated by strobing $\overline{RST}$ (pin 27) logic low for a ***minimum of eight LM628 clock periods.***

The reset routine begins after $\overline{RST}$ is returned to logic high. During the reset execution time, **1.5 ms** maximum, the LM628 will ignore any commands or attempts to transfer data.

A hardware reset forces the LM628 into the state described in what follows.

1. The derivative sampling coefficient, d_S, is set to one, and all other filter coefficients and filter coefficient input buffers are set to zero. With d_S set to one, the derivative sampling interval is set to $2048/f_{CLK}$.
2. All trajectory parameters and trajectory parameters input buffers are set to zero.
3. The current absolute position of the shaft is set to zero ("home").
4. The breakpoint interrupt is masked (disabled), and the remaining five interrupts are unmasked (enabled).
5. The position error threshold is set to its maximum value, 7FFF hex.
6. The DAC output port is set for an 8-bit DAC interface.


START
Strobe $\overline{RST}$ (pin 27) logic low for 8 clock periods minimum.
Wait 1.5 ms after $\overline{RST}$ logic level returned high, reset execution time.
Read the Status Byte.
N
Status Byte = C4 or 84 ?
Y
Reset Interrupts.
Read the Status Byte.
N
Status Byte = C0 or 80 ?
Y
CONTINUE

01086005

FIGURE 4. Hardware Reset Block

Figure 4 illustrates a hardware reset block that includes an LM628 functionality test. This test *should be* completed immediately following all hardware resets.

I. Program Modules (Continued)

Reset Interrupts

TABLE 1. Initialization Module (with Hardware Reset)

Port	Bytes	Command	Comments
	(Note 5)	hardware reset	Strobe $\overline{RST}$, pin 27, logic low for eight clock periods minimum.
		wait	The maximum time to complete hardware reset tasks is 1.5 ms. During this reset execution time, the LM628 will ignore any commands or attempts to transfer data.
c (Note 2)	xx (Note 3)	RDSTAT	This command reads the status byte. It is directly supported by LM628 hardware and can be executed at any time by pulling $\overline{CS}$, $\overline{PS}$, and $\overline{RD}$ logic low. Status information remains valid as long as $\overline{RD}$ is logic low.
		decision	If the status byte is C4 hex or 84 hex, continue. Otherwise loop back to hardware reset.
c	1D	RSTI	This command resets *only* the interrupts indicated by zeros in bits one through six of the next data word. It also resets bit fifteen of the Signals Register and the host interrupt output pin (pin 17).
			Busy-bit Check Module
d	xx	HB (Note 4)	don't care
d	00	LB	Zeros in bits one through six indicate *all* interrupts will be reset.
			Busy-bit Check Module
c	xx	RDSTAT	This command reads the status byte.
		decision	If the status byte is C0 hex or 80 hex, continue. Otherwise loop back to hardware reset.
c	06	PORT12	The reset default size of the DAC port is eight bits. This command initializes the DAC port for a 12-bit DAC. It should not be issued in systems with an 8-bit DAC.
			Busy-bit Check Module

Note 2: The 8-bit host I/O port is a dual-mode port; it operates in command or data mode. The logic level at $\overline{PS}$ (pin 16) selects the mode. Port c represents the LM628 command port-commands are written to the command port and the Status Byte is read from the command port. A logic level of "0" at $\overline{PS}$ selects the command port. Port d represents the LM628 data port—data is both written to and read from the data port. A logic level of "1" at $\overline{PS}$ selects the data port.

Note 3: x - don't care

Note 4: HB - high byte, LB - low byte

Note 5: All values represented in hex.

An RSTI command sequence allows the user to reset the interrupt flag bits, bits one through six of the status byte. See *Figure 5.* It contains an RSTI command and one data word. The RSTI command initiates resetting the interrupt flag bits. Command RSTI also resets the host interrupt output pin (pin 17).

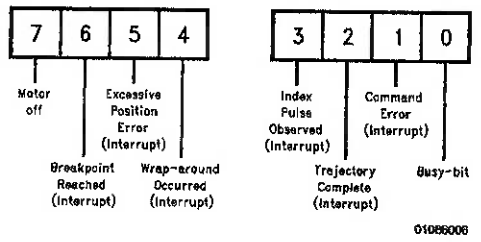

7
6
5
4
3
2
1
0
Motor off
Excessive Position Error (Interrupt)
Breakpoint Reached (Interrupt)
Wrap-around Occurred (Interrupt)
Index Pulse Observed (Interrupt)
Command Error (Interrupt)
Trajectory Complete (Interrupt)
Busy-bit
01086006

FIGURE 5. Status Byte Bit Allocation

Immediately following the RSTI command, a single data word is written. The first byte is not used. Logical zeros in bits one through six of the second byte reset the corresponding interrupts. See *Figure 6.* Any combination of the interrupt flag bits can be reset within a single RSTI command sequence. This feature allows interrupts to be serviced according to a user-programmed priority.

I. Program Modules (Continued)

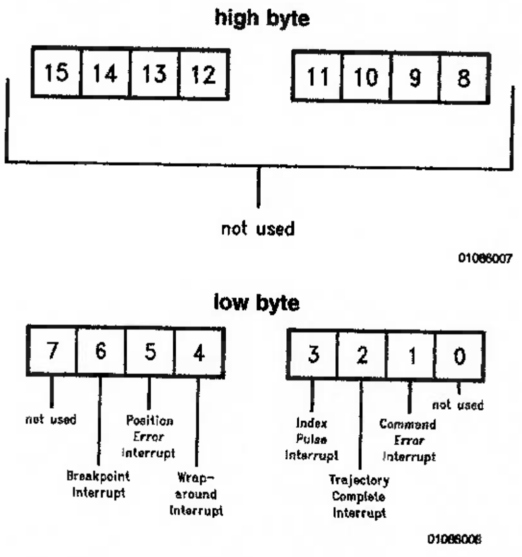


FIGURE 6. Interrupt Mask/Reset Bit Allocations

In the case of the example module, the second byte of the RSTI data word, 00 hex, resets *all* interrupt flag bits. See *Table 1*.

DAC Port Size

During both hardware and software resets, the DAC output port defaults to 8-bit mode. If an LM628 control loop utilizes a 12-bit DAC, command PORT12 should be issued immediately following the hardware reset block and all subsequent resets. Failure to issue command PORT12 will result in erratic, unpredictable motor behavior.

If the control loop utilizes an 8-bit DAC, command PORT12 must not be executed; this too will result in erratic, unpredictable motor behavior.

An LM629 will ignore command PORT8 (as it provides an 8-bit sign/magnitude PWM output). Command PORT12 *should not* be issued in LM629-based systems.

Software Reset Considerations

After the initial hardware reset, resets can be accomplished with either a hardware reset or command RESET (software reset). Software and hardware resets execute the same tasks (Note 6) and require the same execution time, 1.5 ms maximum. During software reset execution, the LM628 will ignore any commands or attempts to transfer data.

The hardware reset module includes an LM628 functionality test. This test is ***not*** required after a software reset.

Table 2 details an initialization module that uses a software reset.

Note 6: In the case of a software reset, the position error threshold remains at its pre-reset value.

TABLE 2. Initialization Module (with Software Reset)

Port	Bytes	Command	Comments
c	00	RESET	See Initialization Module text.
		wait	The maximum time to complete RESET tasks is 1.5 ms.
c	06	PORT12	The RESET default size of the DAC port is eight bits. This command initializes the DAC port for a 12-bit DAC. It should not be issued in a system with an 8-bit DAC.
Busy-bit Check Module			
c	1D	RSTI	This command resets *only* the interrupts indicated by zeros in bits one through six of the next data word. It also resets bit fifteen of the Signals Register and (pin 17) the host interrupt output pin.
Busy-bit Check Module			
d	xx	HB	Don't care
d	00	LB	Zeros in bits one through six indicate *all* interrupts will be reset.
Busy-bit Check Module			

Comments

Figure 7 illustrates, in simplified block diagram form, the LM628. The profile generator provides the control loop input, desired shaft position. The quadrature decoder provides the control loop feedback signal, actual shaft position. At the first summing junction, actual position is subtracted from desired position to generate the control loop error signal, position error. This error signal is filtered by the PID filter to provide the motor drive signal.

After executing the example initialization module, the following observations are made. With the integration limit term (i_L) and the filter gain coefficients (k_p, k_i, and k_d) initialized to zero, the filter gain is zero. Moreover, after a reset, desired shaft position tracks actual shaft position. Under these conditions, the motor drive signal is zero. The control system can not affect shaft position. The shaft should be stationary and "free wheeling". If there is significant drive amplifier offset, the shaft may rotate slowly, but with minimal torque capability.

Note: Regardless of the free wheeling state of the shaft, the LM628 continuously tracks shaft absolute position.

I. Program Modules (Continued)

FILTER PROGRAMMING MODULE

The example filter programming module is shown in *Table 4*.

Load Filter Parameters (Coefficients)

An LFIL (Load FILter) command sequence includes command LFIL, a filter control word, and a variable number of data words.

The LFIL command initiates loading filter coefficients into input buffers.

The two data bytes, written immediately after LFIL, comprise the filter control word. The first byte programs the derivative sampling coefficient, d_s (i.e. selects the derivative sampling interval). The second byte indicates, with logical ones in respective bit positions, which of the remaining four filter coefficients will be loaded. See *Figure 8,Table 3*. Any combination of the four coefficients can be loaded within a single LFIL command sequence.

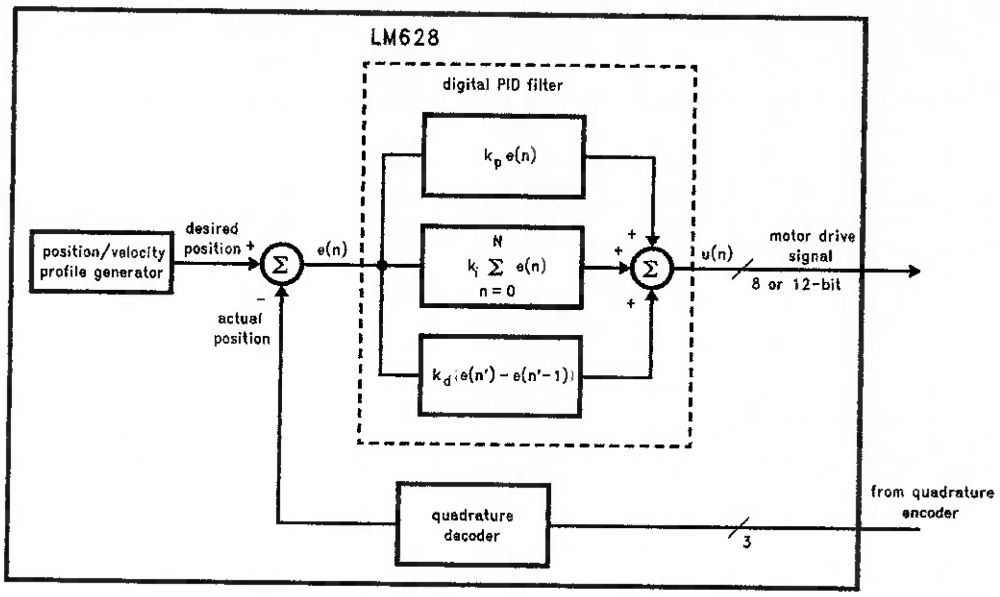
Immediately following the filter control word, the filter coefficients are written. Each coefficient is written as a pair of data bytes, a data word. Because any combination of the four coefficients can be loaded within a single LFIL command sequence, the number of data words following the filter control word can vary in the range from zero to four.

In the case of the example module, the first byte of the filter control word, 00 hex, programs a derivative sampling coefficient of one. The second byte, x8 hex, indicates only the proportional gain coefficient will be loaded.

Immediately following the filter control word, the proportional gain coefficent is written. In this example, k_p is set to ten with the data word 000A hex. The other three filter coefficients remain at zero, their reset value.

Update Filter

The update filter command, UDF, transfers new filter coefficients from input buffers to working registers. Until UDF is executed, the new filter coefficients do not affect the transfer characteristic of the filter.



01088009

FIGURE 7. LM628—Simplified Block Diagram Form

I. Program Modules (Continued)

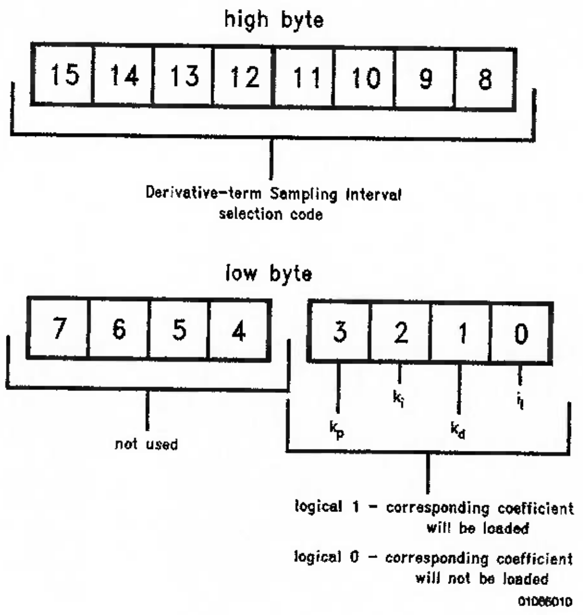

high byte
15 14 13 12 11 10 9 8
Derivative-term Sampling Interval selection code
low byte
7 6 5 4
3 2 1 0
not used
k_p
k_i
k_d
i_l
logical 1 – corresponding coefficient will be loaded
logical 0 – corresponding coefficient will not be loaded
01086010

FIGURE 8. Filter Control Word Bit Allocation

TABLE 3. Derivative—Term Sampling Interval Selection Codes

Filter Control Word Bit Position								d_s	Selected Derivative-Term Sampling Interval—T_d
15	14	13	12	11	10	9	8		
0	0	0	0	0	0	0	0	1	T_s
0	0	0	0	0	0	0	1	2	$2T_s$
0	0	0	0	0	0	1	0	3	$3T_s$
0	0	0	0	0	0	1	1	4	$4T_s$
				•				•	•
				•				•	•
				•				•	•
1	1	1	1	1	1	1	1	256	$256T_s$

$T_s = (2048) \times \left(\frac{1}{f_{CLK}}\right)$ System Sample Period

$T_d = d_s \times T_s$ Derivative-term Sampling Interval

I. Program Modules (Continued)

TABLE 4. Filter Programming Module

Port	Bytes	Command	Comments
c	1E	LFIL	This command initiates loading the filter coefficients input buffers.
			Busy-bit Check Module
d d	00 x8	HB LB	These two bytes are the filter control word. A 00 hex HB sets the derivative sampling interval to 2048/f_{CLK} by setting d_s to one. A x8 hex LB indicates only k_p will be loaded. The other filter parameters will remain at zero, their reset default value.
			Busy-bit Check Module
d d	00 0A	HB LB	These two bytes set k_p to ten.
			Busy-bit Check Module
c	04	UDF	This command transfers new filter coefficients from input buffers to working registers. Until UDF is executed, coefficients loaded via the LFIL command do not affect the filter transfer characteristic.
			Busy-bit Check Module

Comments

After executing both the example initialization and example filter programming modules, the following observations are made. Filter gain is nonzero, but desired shaft position continues to track actual shaft position. Under these conditions, the motor drive signal remains at zero. The shaft should be stationary and "free wheeling". If there is significant drive amplifier offset, the shaft may rotate slowly, but with minimal torque capability.

Initially, k_p should be set below twenty, d_s should be set to one, and k_i, k_d, and i_l should remain at zero. These values will not provide optimum system performance, but they will be sufficient to test system functionality. See Tuning the PID Filter.

TRAJECTORY PROGRAMMING MODULE

Table 5 details the example trajectory programming module.

Load Trajectory Parameters

An LTRJ (Load TRaJectory) command sequence includes command LTRJ, a trajectory control word, and a variable number of data words.

The LTRJ command initiates loading trajectory parameters into input buffers.

The two data bytes, written immediately after LTRJ, comprise the trajectory control word. The first byte programs, with logical ones in respective bit positions, the trajectory mode (velocity or position), velocity mode direction, and stopping mode. See Stop Module. The second byte indicates, with logical ones in respective bit positions, which of the three trajectory parameters will be loaded. It also indicates whether the parameters are absolute or relative. See *Figure 9.* Any combination of the three parameters can be loaded within a single LTRJ command sequence.

Immediately following the trajectory control word, the trajectory parameters are written. Each parameter is written as a pair of data words (four data bytes). Because any combination of the three parameters can be loaded within a single LTRJ command sequence, the number of data words following the trajectory control word can vary in the range from zero to six.

In the case of the example module, the first byte of the trajectory control word, 00 hex, programs the LM628 to operate in position mode. The second byte, 0A hex, indicates velocity and position will be loaded and both parameters are absolute. Four data words, two for each parameter loaded, follow the trajectory control word.

Start Motion Control

The start motion control command, STT (STarT), transfers new trajectory parameters from input buffers to working registers and begins execution of the new trajectory. Until STT is executed, the new trajectory parameters do not affect shaft motion.

Note: At this point no actual trajectory parameters are loaded. Calculation of trajectory parameters and execution of example moves is left for a later section.

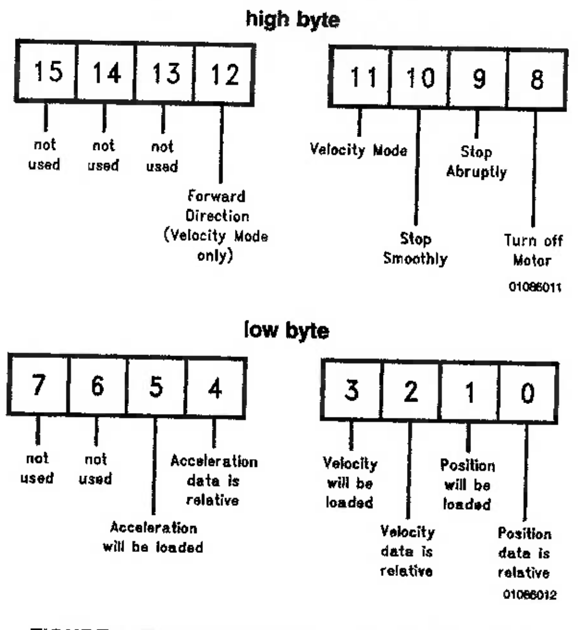


FIGURE 9. Trajectory Control Word Bit Allocation

I. Program Modules (Continued)

TABLE 5. Trajectory Programming Module

Port	Bytes	Command	Comments
c	1F	LTRJ	This command initiates loading the trajectory parameters input buffers.
Busy-bit Check Module			
d d	00 0A	HB LB	These two bytes are the trajectory control word. A 0A hex LB indicates velocity and position will be loaded and both parameters are absolute.
Busy-bit Check Module			
d d	xx xx	HB LB	Velocity is loaded in two data words. These two bytes are the high data word.
Busy-bit Check Module			
d d	xx xx	HB LB	velocity data word (low)
Busy-bit Check Module			
d d	xx xx	HB LB	Position is loaded in two data words. These two bytes are the high data word.
Busy-bit Check Module			
d d	xx xx	HB LB	position data word (low)
Busy-bit Check Module			
c	01	STT	STT must be issued to execute the desired trajectory.
Busy-bit Check Module			

STOP MODULE

This module demonstrates the programming flow required to stop shaft motion.

While the LM628 operates in position mode, normal stopping is always smooth and occurs automatically at the end of a specified trajectory (i.e. no stop module is required). Under exceptional conditions, however, a stop module can be used to affect a premature stop.

While the LM628 operates in velocity mode, stopping is always accomplished via a stop module.

The example stop module, shown in *Table 5*, utilizes an LTRJ command sequence and an STT command.

Load Trajectory Parameters

Bits eight through ten of the trajectory control word select the stopping mode. See *Figure 9*.

In the case of the example module, the first byte of the trajectory control word, x1 hex, selects motor-off as the desired stopping mode. This mode stops shaft motion by setting the motor drive signal to zero (the appropriate offset-binary code to apply zero drive to the motor).

Setting bit nine of the trajectory control word selects stop abruptly as the desired stopping mode. This mode stops shaft motion (at maximum deceleration) by setting the target position equal to the current position.

Setting bit ten of the trajectory control word selects stop smoothly as the desired stopping mode. This mode stops shaft motion by decelerating at the current user-programmed acceleration rate.

Note: Bits eight through ten of the trajectory control word must be used exclusively; only one of them should be logic one at any time.

Start Motion Control

The start motion control command, STT, must be executed to stop shaft motion.

Comments

After shaft motion is stopped with either an "abrupt" or a "smooth" stop module, the control system will attempt to hold the shaft at its current position. If forced away from this desired resting position and released, the shaft will move back to the desired position. Unless new trajectory parameters are loaded, execution of another STT command will restart the specified move.

After shaft motion is stopped with a "motor-off" stop module, desired shaft position tracks actual shaft position. Consequently, the motor drive signal remains at zero and the control system can not affect shaft position; the shaft should be stationary and free wheeling. If there is significant drive amplifier offset, the shaft may rotate slowly, but with minimal torque capability. Unless new trajectory parameters are loaded, execution of another STT command will restart the specified move.

I. Program Modules (Continued)

TABLE 6. Stop Module (Motor-Off)

Port	Bytes	Command	Comments
c	1F	LTRJ	This command initiates loading the trajectory parameters input buffers.
Busy-bit Check Module			
d	x1	HB	These two bytes are the trajectory control word. A x1 hex HB selects motor-off as the
d	00	LB	desired stopping mode. A 00 hex LB indicates no trajectory parameters will be loaded.
Busy-bit Check Module			
c	01	STT	The start motion control command, STT, must be executed to stop shaft motion.
Busy-bit Check Module			

II. Programs

This section focuses on the development of four brief LM628 programs.

LOOP PHASING PROGRAM

Following initial power-up, the correct polarity of the motor drive signal must be determined. If the polarity is incorrect (loop inversion), the drive signal will push the shaft away from its desired position rather than towards it. This results in "motor runaway", a condition characterized by the motor running continuously at high speed.

The loop phasing program, detailed in *Table 7*, contains both the example initialization and filter programming modules. It also contains an LTRJ command sequence and an STT command.

Note: Execution of this simple program is only required the *first* time a new system is used.

Load Trajectory Parameters

An LTRJ (Load TRaJectory) command sequence includes command LTRJ, a trajectory control word, and a variable number of data words.

In the case of the Loop Phasing Program, the first byte of the trajectory control word, 00 hex, programs the LM628 to operate in position mode. The second byte, 00 hex, indicates no trajectory parameters will be loaded (i.e. in this program, zero data words follow the trajectory control word). The three trajectory parameters will remain at zero, their reset value.

Start Motion Control

The start motion control command, STT (STarT), transfers new trajectory parameters from input buffers to working registers and begins execution of the new trajectory. Until STT is executed, the new trajectory parameters do not affect shaft motion.

TABLE 7. Loop Phasing Program

Port	Bytes	Command	Comments
Initialization Module			
Filter Programming Module			
c	1F	LTRJ	This command initiates loading the trajectory parameters input buffers.
Busy-bit Check Module			
d	00	HB	These two bytes are the trajectory control word. A 00 hex LB indicates no trajectory
d	00	LB	parameters will be loaded.
Busy-bit Check Module			
c	01	STT	STT must be issued to execute the desired trajectory.

Comments

Execution of command STT results in execution of the desired trajectory. With the acceleration set at zero, the profile generator generates a desired shaft position that is both constant and equal to the current absolute position. See *Figure 7*. Under these conditions, the control system will attempt to hold the shaft at its current absolute postion. The shaft will feel lightly "spring loaded". If forced (CAREFULLY) away from its desired position and released, the shaft will spring back to the desired position.

If the polarity of the motor drive signal is incorrect (loop inversion), motor runaway will occur immediately after execution of command STT, or after the shaft is forced (CAREFULLY) from its resting position.

Loop inversion can be corrected with one of three methods: interchanging the shaft position encoder signals (channel A and channel B), interchanging the motor power leads, or inverting the motor command signal before application to the motor drive amplifier. For LM629 based systems, loop inversion can be corrected by interchanging the motor power leads, interchanging the shaft position encoder signals, or logically inverting the PWM sign signal.

SIMPLE ABSOLUTE POSITION MOVE

The Simple Absolute Position Move Program, detailed in *Table 8*, utilizes both the initialization and filter programming modules, as well as, an LTRJ command sequence and an STT command.

Factors that influenced the development of this program included the following: the program must demonstrate simple trajectory parameters calculations, the program must demonstrate the programming flow required to load and execute an absolute position move, and correct completion of the move must be verifiable through simple observation.

Move: The shaft will accelerate at 0.1 rev/sec^2 until it reaches a maximum velocity of 0.2 rev/sec, and then decelerate to a stop exactly two revolutions from the starting position. See *Figure 10*.

II. Programs (Continued)

Note: Absolute position is position measured relative to zero (home). An absolute position move is a move that ends at a specified absolute position. For example, independent of the current absolute position of the shaft, if an absolute position of 30,000 counts is specified, upon completion of the move the absolute position of the shaft will be 30,000 counts (i.e. 30,000 counts relative to zero). The example program calls for a position move of two revolutions. Because the starting absolute position is 0 counts, the move is accomplished by specifying an absolute position of 8000 counts. See *Table 8*.

The Quadrature Incremental Encoder

As a supplement to the trajectory parameters calculations, a brief discussion is provided here to differentiate between encoder ***lines*** and encoder ***counts***.

A quadrature incremental shaft encoder encodes shaft rotation as electrical pulses. *Figure 11* details the signals generated by a 3-channel quadrature incremental encoder. The LM628 decodes (or "counts") a quadrature incremental signal to determine the absolute position of the shaft.

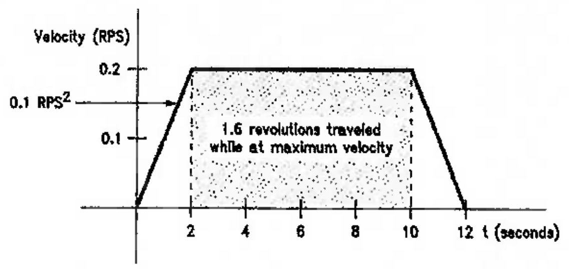
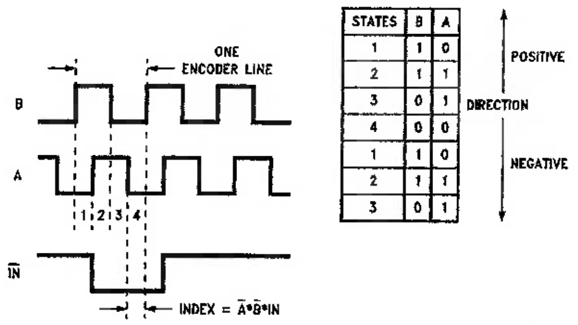


FIGURE 10. Velocity Profile for Simple Absolute Position Move Program



STATES	B	A
1	1	0
2	1	1
3	0	1
4	0	0
1	1	0
2	1	1
3	0	1

FIGURE 11. 3-Channel Quadrature Encoder Signals

The resolution of a quadrature incremental encoder is usually specified as a number of ***lines***. This number indicates the number of cycles of the output signals for each complete shaft revolution. For example, an N-line encoder generates N cycles of its output signals during each complete shaft revolution.

By definition, two signals that are in quadrature are 90° out of phase. When considered together, channels A and B (*Figure 11*) traverse four distinct digital states during each full cycle of either channel. Each state transition represents one ***count*** of shaft motion. The leading channel indicates the direction of shaft rotation.

Each line, therefore, represents one cycle of the output signals, and each cycle represents four encoder counts.

$$\left(N\frac{\text{CYCLES}}{\text{REVOLUTION}}\right)\times\left(4\frac{\text{COUNTS}}{\text{CYCLE}}\right)=4N\frac{\text{COUNTS}}{\text{REVOLUTION}}$$

The reference system uses a one thousand line encoder.

$$\left(1000\frac{\text{CYCLES}}{\text{REVOLUTION}}\right)\times\left(4\frac{\text{COUNTS}}{\text{CYCLE}}\right)=4000\frac{\text{COUNTS}}{\text{REVOLUTION}}$$

Sample Period

Sampling of actual shaft position occurs at a fixed frequency, the reciprocal of which is the system sample period. The system sample period is the unit of time upon which shaft acceleration and velocity are based.

$$T_S = (2048)\times\left(\frac{1}{f_{\text{CLOCK}}}\right)\ \text{System Sample Period}$$

The reference system uses an 8 MHz clock. The sample period of the reference system follows directly from the definition.

$$T_S = (2048)\times\left(\frac{1}{8\times10^6\ \text{Hz}}\right) = 256\times10^{-6}\frac{\text{SECONDS}}{\text{SAMPLE}}$$

Trajectory Parameters Calculations

The shaft will accelerate at 0.1 rev/sec^2 until it reaches a maximum velocity of 0.2 rev/sec, and then decelerate to a stop exactly two revolutions from the starting position.

Trajectory parameters calculations for this move are detailed in *Figure 12*.

Comments

After completing the move, the control system will attempt to hold the shaft at its current absolute position. The shaft will feel lightly "spring loaded". If forced away from its desired resting position and released, the shaft will move back to the desired position.

II. Programs (Continued)

$$A = \left(4000\,\frac{\text{COUNTS}}{\text{REVOLUTION}}\right) \times \left(256 \times 10^{-6}\,\frac{\text{SECONDS}}{\text{SAMPLE}}\right)^2 \times \left(0.1\,\frac{\text{REVOLUTIONS}}{\text{SECOND}^2}\right) = 2.62 \times 10^{-5}\,\frac{\text{COUNTS}}{\text{SAMPLE}^2}$$

$$A = \left(2.62 \times 10^{-5}\,\frac{\text{COUNTS}}{\text{SAMPLE}^2}\right) \times (65{,}536) = 1.718\,\frac{\text{COUNTS}}{\text{SAMPLE}^2} \qquad \text{Acceleration Scaled}$$

$$A = 2\,\frac{\text{COUNTS}}{\text{SAMPLE}^2} \qquad \text{Acceleration Rounded}$$

$$A = 00\ 00\ 00\ 02 \text{ hex}\,\frac{\text{COUNTS}}{\text{SAMPLE}^2}$$

$$V = \left(4000\,\frac{\text{COUNTS}}{\text{REVOLUTION}}\right) \times \left(256 \times 10^{-6}\,\frac{\text{SECONDS}}{\text{SAMPLE}}\right) \times \left(0.2\,\frac{\text{REVOLUTIONS}}{\text{SECOND}}\right) = 0.2048\,\frac{\text{COUNTS}}{\text{SAMPLE}}$$

$$V = \left(0.2048\,\frac{\text{COUNTS}}{\text{SAMPLE}}\right) \times (65{,}536) = 13{,}421.77\,\frac{\text{COUNTS}}{\text{SAMPLE}} \qquad \text{Velocity Scaled}$$

$$V = 13{,}422\,\frac{\text{COUNTS}}{\text{SAMPLE}} \qquad \text{Velocity Rounded}$$

$$V = 00\ 00\ 34\ 6E \text{ hex}\,\frac{\text{COUNTS}}{\text{SAMPLE}}$$

$$P = \left(4000\,\frac{\text{COUNTS}}{\text{REVOLUTION}}\right) \times (2.0 \text{ REVOLUTIONS}) = 8000 \text{ COUNTS}$$

$$P = 00\ 00\ 1F\ 40 \text{ hex COUNTS}$$

01066029

FIGURE 12. Calculations of Trajectory Parameters for Simple Absolute Position Move

II. Programs (Continued)

TABLE 8. Simple Absolute Position Move Program

Port	Bytes	Command	Comments
			Initialization Module
			Filter Programming Module
c	1F	LTRJ	This command initiates loading the trajectory parameters input buffers
			Busy-bit Check Module
d d	00 2A	HB LB	These two bytes are the trajectory control word. A 2A hex LB indicates acceleration, velocity, and position will be loaded and all three parameters are absolute.
d d	00 00	HB LB	Acceleration is loaded in two data words. These two bytes are the high data word. In this case, the acceleration is 0.1 rev/sec^2.
			Busy-bit Check Module
d d	00 02	HB LB	acceleration data word (low)
d d	00 00	HB LB	velocity is loaded in two data words. These two bytes are the high data word. In this case, the velocity is 0.2 rev/sec.
			Busy-bit Check Module
d d	34 6E	HB LB	velocity data word (low)
			Busy-bit Check Module
d d	00 00	HB LB	Position is loaded in two data words. These two bytes are the high data word. In this case, the position loaded is eight thousand counts. This results in a move of two revolutions in the forward direction.
			Busy-bit Check Module
d d	1F 40	HB LB	position data word (low)
			Busy-bit Check Module
c	01	STT	STT must be issued to execute the desired trajectory.

SIMPLE RELATIVE POSITION MOVE

This program demonstrates the programming flow required to load and execute a relative position move. See *Table 9.*

Move: Independent of the current resting position of the shaft, the shaft will complete thirty revolutions in the reverse direction. Total time to complete the move is fifteen seconds. Total time for acceleration and deceleration is five seconds.

Note: Target position is the final requested position. If the shaft is stationary, and motion has not been stopped with a "motor-off" stop module, the current absolute position of the shaft is the target position. If motion has been stopped with a "motor-off" stop module, or a position move has begun, the absolute position that corresponds to the endpoint of the current trajectory is the target position. Relative position is position measured relative to the current target position of the shaft. A relative position move is a move that ends the specified "relative" number of counts away from the current target position of the shaft. For example, if the current target position of the shaft is 10 counts, and a relative position of 30,000 counts is specified, upon completion of the move the absolute position of the shaft will be 30,010 counts (i.e. 30,000 counts relative to 10 counts).

Load Trajectory Parameters

The first byte of the trajectory control word, 00 hex, programs position mode operation. The second byte, 2B hex, indicates all three trajectory parameters will be loaded. It also indicates both acceleration and velocity will be absolute values while position will be a relative value.

Trajectory Parameters Calculations

Independent of the current resting position of the shaft, the shaft will complete thirty revolutions in the reverse direction. Total time to complete the move is fifteen seconds. Total time for acceleration and deceleration is five seconds.

The reference system utilizes a one thousand line encoder. The number of counts for each complete shaft revolution and the total counts for this position move are determined.

$$\left(1000\,\frac{\text{CYCLES}}{\text{REVOLUTION}}\right)\times\left(4\,\frac{\text{COUNTS}}{\text{CYCLE}}\right)=4000\,\frac{\text{COUNTS}}{\text{REVOLUTION}}$$

$$\left(4000\,\frac{\text{COUNTS}}{\text{REVOLUTION}}\right)\times(30\ \text{REVOLUTIONS})=120{,}000\ \text{COUNTS}$$

With respect to time, two-thirds of the move is made at maximum velocity and one-third is made at a velocity equal to one-half the maximum velocity (Note 7). Therefore, total counts traveled during acceleration and deceleration periods is one-fifth the total counts traveled. See *Figure 13.*

II. Programs (Continued)

$$\frac{120{,}000\text{ COUNTS}}{5} = 24{,}000\text{ COUNTS} \quad \text{total counts traveled during acceleration and deceleration}$$

$$\frac{24{,}000\text{ COUNTS}}{2} = 12{,}000\text{ COUNTS} \quad \text{counts traveled during acceleration}$$

The reference system uses an 8 MHz clock. The sample period of the reference system is determined.

$$T_S = (2048) \times \left(\frac{1}{8 \times 10^6\text{Hz}}\right) = 256 \times 10^{-6}\,\frac{\text{SECONDS}}{\text{SAMPLE}}$$

The number of samples during acceleration (and deceleration) is determined.

$$\frac{2.5\text{ SECONDS}}{256 \times 10^{-6}\,\frac{\text{SECONDS}}{\text{SAMPLE}}} = 9766\text{ SAMPLES} \quad \text{number of samples during acceleration}$$

Using the number of counts traveled during acceleration and the number of samples during acceleration, acceleration is determined.

$$s = \frac{at^2}{2} \quad \text{distance traveled during time t at acceleration a}$$

$$a = \frac{2s}{t^2} = \frac{(2) \times (12{,}000\text{ COUNTS})}{(9766\text{ SAMPLES})^2} = 0.000252\,\frac{\text{COUNTS}}{\text{SAMPLE}^2}$$

Total counts traveled while at maximum velocity is four-fifths the total counts traveled.

$$\frac{(4) \times (120{,}000\text{ COUNTS})}{5} = 96{,}000\text{ COUNTS}$$

Note 7: Average velocity during acceleration and deceleration periods is one-half the maximum velocity.

TABLE 9. Simple Relative Position Move Program

Port	Bytes	Command	Comments
Initialization Module			
Filter Programming Module			
c	1F	LTRJ	This command initiates loading the trajectory parameters input buffers.
Busy-bit Check Module			
d	00	HB	These two bytes are the trajectory control word. A 2B hex LB indicates all three parameters will be loaded and both acceleration and velocity will be absolute values while position will be a relative value.
d	2B	LB	
Busy-bit Check Module			
d	00	HB	Acceleration is loaded in two data words. These two bytes are the high data word. In this case, the acceleration is 17 counts/sample^2.
d	00	LB	
Busy-bit Check Module			
d	00	HB	acceleration data word (low)
d	11	LB	
Busy-bit Check Module			
d	00	HB	Velocity is loaded in two data words. These two bytes are the high data word. In this case, velocity is 161,087 counts/sample.
d	02	LB	
Busy-bit Check Module			
d	75	HB	velocity data word (low)
d	3F	LB	
Busy-bit Check Module			
d	FF	HB	Position is loaded in two data words. These two bytes are the high data word. In this case, the position loaded is −120,000 counts. This results in a move of thirty revolutions in the reverse direction.
d	FE	LB	
Busy-bit Check Module			
d	2B	HB	position data word (low)
d	40	LB	
Busy-bit Check Module			
c	01	STT	STT must be issued to execute the desired trajectory.

II. Programs (Continued)

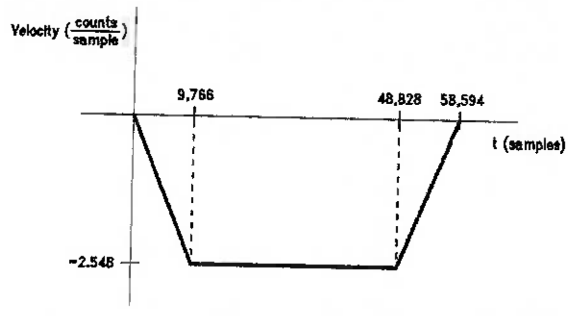


FIGURE 13. Velocity Profile for Simple Relative Position Move Program

The number of samples while at maximum velocity is determined.

$$\frac{10\ \text{SECONDS}}{256 \times 10^{-6}\,\dfrac{\text{SECONDS}}{\text{SAMPLE}}} = 39{,}062\ \text{SAMPLES}\ \begin{smallmatrix}\text{number of samples while at}\\ \text{maximum velocity}\end{smallmatrix}$$

Using the total counts traveled while at maximum velocity and the number of samples while at maximum velocity, velocity is determined.

$$\frac{96{,}000\ \text{COUNTS}}{39{,}062\ \text{SAMPLES}} = 2.458\,\frac{\text{COUNTS}}{\text{SAMPLE}}$$

Both acceleration and velocity values are scaled.

$$\left(0.000252\,\frac{\text{COUNTS}}{\text{SAMPLE}^2}\right) \times (65{,}536) = 16.515\,\frac{\text{COUNTS}}{\text{SAMPLE}^2}$$

$$\left(2.458\,\frac{\text{COUNTS}}{\text{SAMPLE}}\right) \times (65{,}536) = 161{,}087.488\,\frac{\text{COUNTS}}{\text{SAMPLE}}$$

Acceleration and velocity are rounded to the nearest integer and all three trajectory parameters are converted to hexadecimal.

$$A = 17 = 00\ 00\ 00\ 11\ \text{hex}\,\frac{\text{COUNTS}}{\text{SAMPLE}^2}$$

$$V = 161{,}087 = 00\ 02\ 75\ 3F\ \text{hex}\,\frac{\text{COUNTS}}{\text{SAMPLE}}$$

$$P = -120{,}000 = \text{FF FE 2B 40 hex COUNTS}$$

BASIC VELOCITY MODE MOVE WITH BREAKPOINTS

This program demonstrates basic velocity mode programming and the (typical) programming flow required to set both absolute and relative breakpoints. See *Table 10.*

Move: The shaft will accelerate at 1.0 rev/sec^2 until it reaches a maximum velocity of 2.0 rev/sec. After completing twenty forward direction revolutions (including revolutions during acceleration), the shaft will accelerate at 1.0 rev/sec^2 until it reaches a maximum velocity of 4.0 rev/sec. After completing twenty forward direction revolutions (including revolutions during acceleration), the shaft will decelerate (at 1.0 rev/sec^2) to a stop. See *Figure 14.*

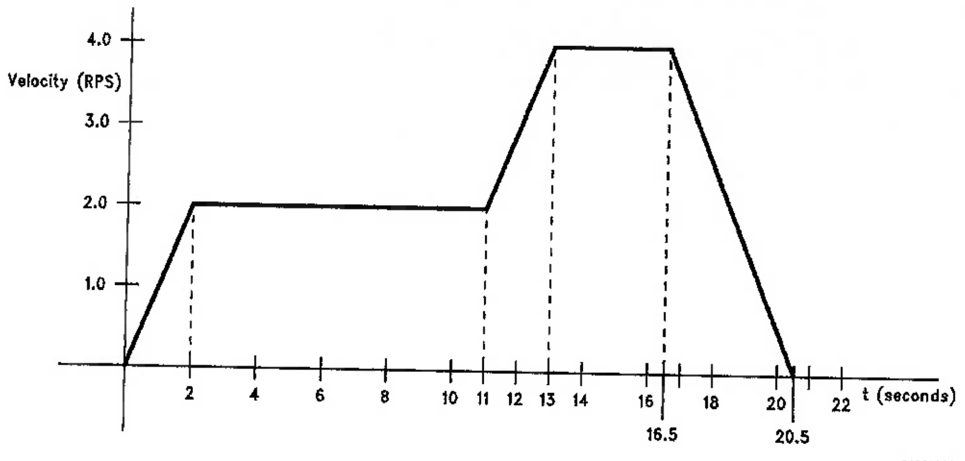


FIGURE 14. Velocity Profile for Basic Velocity Mode with Breakpoints Program

Mask Interrupts

An MSKI command sequence allows the user to determine which interrupt conditions result in host interrupts; interrupting the host via the host interrupt output (pin 17). It contains an MSKI command and one data word.

The MSKI command initiates interrupt masking.

Immediately following the MSKI command, a single data word is written. The first byte is not used. Bits one through six of the second byte determine the masked/unmasked status of each interrupt. See *Figure 6.* Any zeros in this 6-bit field mask (disable) the corresponding interrupts while any ones unmask (enable) the corresponding interrupts.

II. Programs (Continued)

In the case of the examlple program, the second byte of the MSKI data word, 40 hex, enables the breakpoint interrupt. All other interrupts are disabled (masked).

When interrupted, the host processor can read the Status Byte to determine which interrupt condition(s) occurred. See *Figure 5*.

Note: Command MSKI controls only the host interrupt process. Bits one through six of the Status Byte reflect actual conditions independent of the masked/unmasked status of individual interrupts. This feature allows interrupts to be serviced with a polling scheme.

Set Breakpoints (Absolute and Relative)

An SBPA command sequence enables the user to set breakpoints in terms of absolute shaft position. An SBPR command sequence enables setting breakpoints relative to the current target position. When a breakpoint position is reached, bit six of the status byte, the breakpoint interrupt flag, is set to logic high. If this interrupt is enabled (unmasked), the host will be interrupted via the host interrupt output (pin 17).

An SBPA (or SBPR) command initiates loading/setting a breakpoint. The two data words, written immediately following the SBPA (or SBPR) command, represent the breakpoint position.

The example program contains a relative breakpoint set at 80,000 counts relative to position zero (the current target position). This represents a move of twenty forward direction revolutions. When this position is reached, the LM628 interrupts the host processor, and the host executes a sequence of commands that increases the maximum velocity, resets the breakpoint interrupt flag, and loads an absolute breakpoint.

The example program contains an absolute breakpoint set at 160,000 counts. When this absolute position is reached, the LM628 interrupts the host processor, and the host executes a Smooth Stop Module.

Breakpoint positions for this example program are determined.

$$\left(4000\,\frac{\text{COUNTS}}{\text{REVOLUTION}}\right) \times (20\ \text{REVOLUTIONS})$$

$$= 80{,}000\ \text{COUNTS}\quad \begin{matrix}\text{relative}\\\text{breakpoint}\end{matrix}$$

$$\left(4000\,\frac{\text{COUNTS}}{\text{REVOLUTION}}\right) \times (40\ \text{REVOLUTIONS})$$

$$= 160{,}000\ \text{COUNTS}\quad \begin{matrix}\text{absolute}\\\text{breakpoint}\end{matrix}$$

Load Trajectory Parameters

This example program contains two LTRJ command sequences. The trajectory control word of the first LTRJ command sequence, 1828 hex, programs forward direction velocity mode, and indicates an absolute acceleration and an absolute velocity will be loaded. The trajectory control word of the second LTRJ command sequence, 180C hex, programs forward direction velocity mode, and indicates a relative velocity will be loaded. See *Figure 9.*

Trajectory parameters calculations follow the same format as those detailed for the simple absolute position move. See *Figure 12.*

TABLE 10. Basic Velocity Mode Move with Breakpoints Program

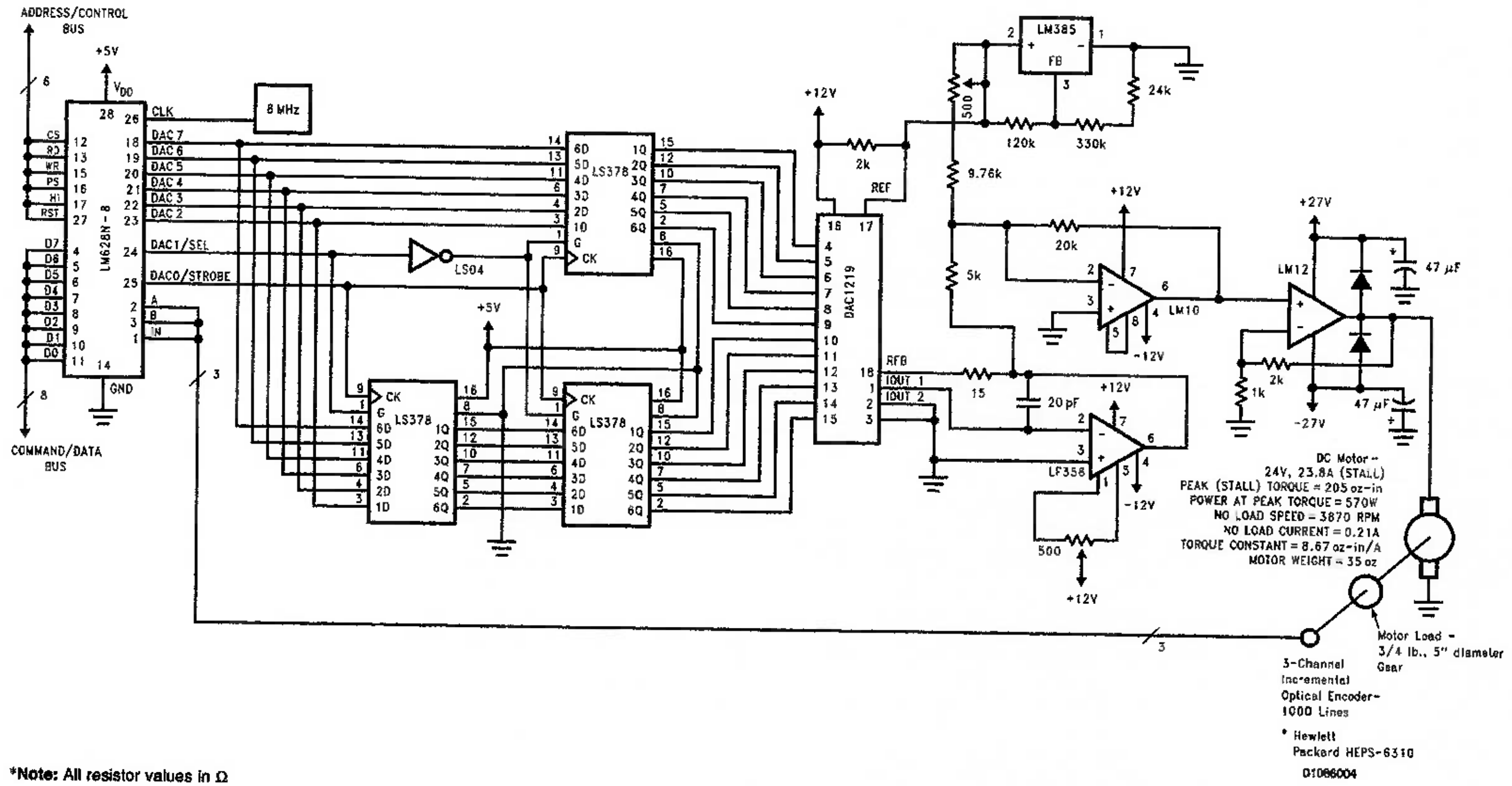
Port	Bytes	Command	Comments
			Initialization Module
			Filter Programming Module
c	1C	MSKI	Mask interrupts.
			Busy-bit Check Module
d	xx	HB	don't care
d	40	LB	A 40 hex LB enables (unmasks) the breakpoint interrupt. All other interrupts are disabled (masked).
			Busy-bit Check Module
c	21	SPBR	This command initiates loading a relative breakpoint.
			Busy-bit Check Module
d	00	HB	A breakpoint is loaded in two data words. These two bytes are the high data word. In this case, the breakpoint is 80,000 counts relative to the current commanded target position (zero).
d	01	LB	
			Busy-bit Check Module
d	38	HB	breakpoint data word (low)
d	80	LB	
			Busy-bit Check Module
c	1F	LTRJ	Load trajectory.
			Busy-bit Check Module
d	18	HB	These two bytes are the trajectory control word. A 18 hex HB programs forward direction velocity mode operation. A 28 hex LB indicates acceleration and velocity will be loaded and both values are absolute.
d	28	LB	

II. Programs (Continued)

TABLE 10. Basic Velocity Mode Move with Breakpoints Program (Continued)

Port	Bytes	Command	Comments
			Busy-bit Check Module
d d	00 00	HB LB	Acceleration is loaded in two data words. These two bytes are the high data word. In this case, the acceleration is 1.0 rev/sec^2.
			Busy-bit Check Module
d d	00 11	HB LB	acceleration data word (low)
			Busy-bit Check Module
d d	00 02	HB LB	Velocity is loaded in two data words. These two bytes are the high data word. In this case, velocity is 2.0 rev/s.
			Busy-bit Check Module
d d	0C 4A	HB LB	velocity data word (low)
			Busy-bit Check Module
c	01	STT	Start motion control.
			Busy-bit Check Module
c	1F	LTRJ	This command initiates loading the trajectory parameters input buffers.
			Busy-bit Check Module
d d	18 0C	HB LB	These two bytes are the trajectory control word. A 18 hex HB programs forward direction velocity mode operation. A 0C hex LB indicates only velocity will be loaded and it will be a relative value.
			Busy-bit Check Module
d d	00 02	HB LB	Velocity is loaded in two data words. These two bytes are the high data word. In this case, velocity is 2.0 rev/s. Because this is a relative value, the current velocity will be increased by 2.0 rev/s. The resultant velocity will be 4.0 rev/s.
			Busy-bit Check Module
d d	0C 4A	HB LB	velocity data word (low)
		wait	This wait represents the host processor waiting for an LM628 breakpoint interrupt.
c	01	STT	Start motion control.
			Busy-bit Check Module
c	1D	RSTI	Reset interrupts.
			Busy-bit Check Module
d d	xx 00	HB LB	don't care Zeros in bits one through six reset **all** interrupts.
			Busy-bit Check Module
c	20	SPBA	This command initiates loading an absolute breakpoint.
			Busy-bit Check Module
d d	00 02	HB LB	A breakpoint is loaded in two data words. These two bytes are the high data word. In this case, the breakpoint is 160,000 counts absolute.
			Busy-bit Check Module
d d	71 00	HB LB	breakpoint data word (low)
		wait	This wait represents the host processor waiting for an LM628 breakpoint interrupt.
			"Smooth" Stop Module

II. Programs (Continued)



*Note: All resistor values in Ω

FIGURE 15. Reference System

III. Tuning the PID Filter

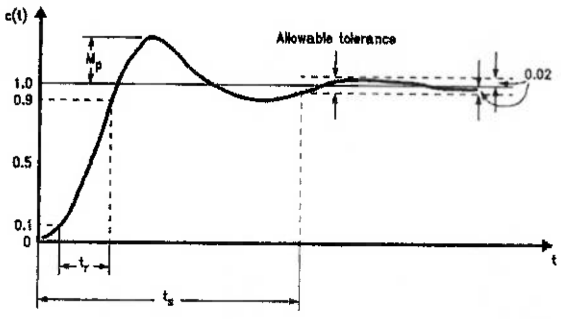
BACKGROUND

The transient response of a control system reveals important information about the "quality" of control, and because a step input is easy to generate and sufficiently drastic, the transient response of a control system is often characterized by the response to a step input, the system step response.

In turn, the step response of a control system can be characterized by three attributes: maximum overshoot, rise time, and settling time. These step response attributes are defined in what follows and detailed graphically in *Figure 16.*

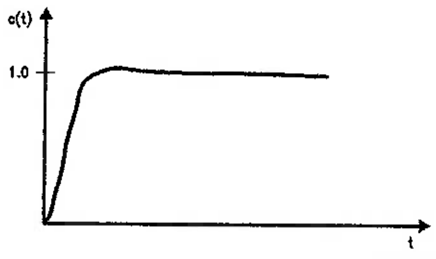
1. The maximum overshoot, Mp, is the maximum peak value of the response curve measured from unity. The amount of maximum overshoot directly indicates the relative stability of the system.
2. The rise time, t_r, is the time required for the response to rise from ten to ninety percent of the final value.
3. The settling time, t_s, is the time required for the response to reach and stay within two percent of the final value.

A critically damped control system provides optimum performance. The step response of a critically damped control system exhibits the minimum possible rise time that maintains zero overshoot and zero ringing (damped oscillations). *Figure 17* illustrates the step response of a critically damped control system.



01086017

FIGURE 16. Unit Step Response Curve Showing Transient Response Attributes



01086018

FIGURE 17. Unit Step Response of a Critically Damped System

INTRODUCTION

The LM628 is a digital PID controller. The loop-compensation filter of a PID controller is usually tuned experimentally, especially if the system dynamics are not well known or defined.

The ultimate goal of tuning the PID filter is to critically damp the motor control system—provide optimum tracking and settling time.

As shown in *Figure 7,* the response of the PID filter is the sum of three terms, a proportional term, an integral term, and a derivative term. Five variables shape this response. These five variables include the three gain coefficients (k_p, k_i, and k_d), the integration limit coefficient (i_l), and the derivative sampling coefficient (d_s). *Tuning the filter equates to determining values for these variable coefficients, values that critically damp the control system .*

Filter coefficients are best determined with a two-step experimental approach. In the first step, the values of k_p, k_i, and k_d (along with i_l and d_s) are systematically varied until reasonably good response characteristics are obtained. Manual and visual methods are used to evaluate the effect of each coefficient on system behavior. In the second step, an oscilloscope trace of the system step response provides detailed information on system damping, and the filter coefficients, determined in step one, are modified to critically damp the system.

Note: In step one, adjustments to filter coefficient values are inherently coarse, while in step two, adjustments are inherently fine. Due to this coarse/fine nature, steps one and two complement each other, and the two-step approach is presented as the "best" tuning method. The PID filter can be tuned with either step one or step two alone.

STEP ONE—MANUAL VISUAL METHOD

Introduction

In the first step, the values of k_p, k_i, and k_d (along with i_l and d_s) are systematically varied until reasonably good response characteristics are obtained. Manual and visual methods are used to evaluate the effect of each coefficient on system behavior.

Note: The next four numbered sections are ordered steps to tuning the PID filter.

1. Prepare the System

The initialization section of the filter tuning program is executed to prepare the system for filter tuning. See *Table 11.* This section initializes the system, presets the filter parameters (k_p, k_i, $il = 0$, $k_d = 2$, $d_s = 1$), and commands the control loop to hold the shaft at the current position.

After executing the initialization section of the filter tuning program, both desired and actual shaft positions equal zero; the shaft should be stationary. Any displacement of the shaft constitutes a position error, but with both k_p and k_i set to zero, the control loop can not correct this error.

2. Determine the Derivative Gain Coefficient

The filter derivative term provides damping to eliminate oscillation and minimize overshoot and ringing, stabilize the system. Damping is provided as a force proportional to the rate of change of position error, and the constant of proportionality is $k_d \times d_s$. See *Figure 18.*

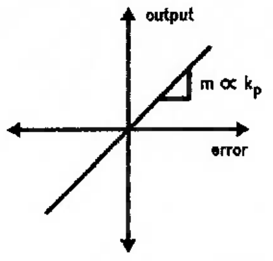
Coefficients k_d and d_s are determined with an iterative process. Coefficient k_d is systematically increased until the shaft begins high frequency oscillations. Coefficient d_s is then increased by one. The entire process is repeated until d_s reaches a value appropriate for the system.

III. Tuning the PID Filter (Continued)

The system sample period sets the time interval between updates of position error. The derivative sampling interval is an integer multiple of the system sample period. See *Table 3*. It sets the time interval between successive position error samples used in the derivative term, and, therefore, directly affects system damping. The derivative sampling interval should be five to ten times smaller than the system mechanical time constant — this means many systems will require low d_s. In general, however, k_d and d_s should be set to give the largest $k_d \times d_s$ product that maintains acceptably low motor vibrations.

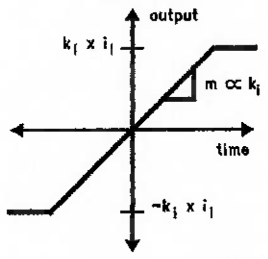
Note: Starting k_d at two and doubling it is a good method of increasing k_d. Manually turning the shaft reveals that with each increase of k_d, the resistance of the shaft to turning increases. The shaft feels increasingly sluggish and, because k_d provides a force proportional to the rate of change of position error, the faster the shaft is turned the more sluggish it feels. For the reference system, the final values of k_d and d_s are 4000 and 4 respectively.

Proportional Term



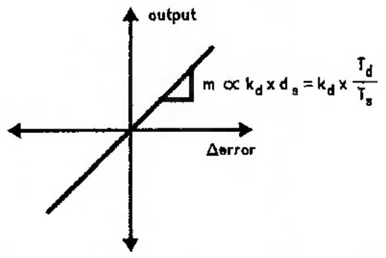
01086019

Integral Term



01086020

Derivative Term



01086021

FIGURE 18. Proportional, Integral, and Derivative (PID) Force Components

TABLE 11. Initialization Section— Filter Tuning Program

Port	Bytes	Command	Comments
c	00	RESET	See Initialization Module Text
		wait	The maximum time to complete RESET tasks is 1.5 ms.
c	06	PORT12	The RESET default size of the DAC port is eight bits. This command initializes the DAC port for a 12-bit DAC. It should not be issued in systems with an 8-bit DAC.
			Busy-bit Check Module
c	1D	RSTI	This command resets **only** the interrupts indicated by zeros in bits one through six of the next data word. It also resets bit fifteen of the Signals Register and the host interrupt pin (pin 17).
			Busy-bit Check Module
d	xx	HB	don't care
d	00	LB	Zeros in bits one through six indicate **all** interrupts will be reset.
			Busy-bit Check Module
c	1C	MSKI	This command masks the interrupts indicated by zeros in bits one through six of the next data word.
			Busy-bit Check Module
d	xx	HB	don't care

III. Tuning the PID Filter (Continued)

TABLE 11. Initialization Section— Filter Tuning Program (Continued)

Port	Bytes	Command	Comments
d	04	LB	A 04 hex LB enables (unmasks) the trajectory complete interrupt. All other interrupts are disabled (masked). See *Figure 6.*
Busy-bit Check Module			
c	1E	LFIL	This command initiates loading the filter coefficients input buffers.
Busy-bit Check Module			
d d	00 x2	HB LB	These two bytes are the filter control word. A 00 hex HB sets the derivative sampling interval to $2048/f_{CLK}$ by setting d_s to one. A x2 hex LB indicates only k_d will be loaded. The other filter parameters will remain at zero, their reset default value.
Busy-bit Check Module			
d d	00 02	HB LB	These two bytes set k_d to two.
Busy-bit Check Module			
c	04	UDF	This command transfers new filter coefficients from input buffers to working registers. Until UDF is executed, coefficients loaded via the LFIL command do not affect the filter transfer characteristic.
Busy-bit Check Module			
c	1F	LTRJ	This command initiates loading the trajectory parameters input buffers.
Busy-bit Check Module			
d d	00 00	HB LB	These two bytes are the trajectory control word. A 00 hex LB indicates no trajectory parameters will be loaded.
Busy-bit Check Module			
c	01	STT	STT must be issued to execute the desired trajectory.

3. Determine the Proportional Gain Coefficient

Inertial loading causes following (or tracking) error, position error associated with a moving shaft. External disturbances and torque loading cause displacement error, position error associated with a stationary shaft. The filter proportional term provides a restoring force to minimize these position errors. The restoring force is proportional to the position error and increases linearly as the position error increases. See *Figure 18.* The proportional gain coefficient, k_p, is the constant of proportionality.

Coefficient k_p is determined with an iterative process—the value of k_p is increased, and the system damping is evaluated. This is repeated until the system is critically damped.

System damping is evaluated manually. Manually turning the shaft reveals each increase of k_p increases the shaft "stiffness". The shaft feels spring loaded, and if forced away from its desired holding position and released, the shaft "springs" back. If k_p is too low, the system is over damped, and the shaft recovers too slowly. If k_p is too large, the system is under damped, and the shaft recovers too quickly. This causes overshoot, ringing, and possibly oscillation. The proportional gain coefficient, k_p, is increased to the largest value that does not cause excessive overshoot or ringing. At this point the system is critically damped, and therefore provides optimum tracking and settling time.

Note: Starting k_p at two and doubling it at each iteration is a good method of increasing k_p. The final value of k_p for the reference system is 40.

4. Determine the Integral Gain Coefficient

The filter proportional term minimizes the errors due to inertial and torque loading. The integral term, however, provides a corrective force that can eliminate following error while the shaft is spinning and the deflection effects of a static torque load while the shaft is stationary. This corrective force is proportional to the position error and increases linearly with time. See *Figure 18.* The integral gain coefficient, k_i, is the constant of proportionality.

High values of k_i provide quick torque compensation, but increase overshoot and ringing. In general, k_i should be set to the smallest value that provides the appropriate compromise between three system characteristics: overshoot, settling time, and time to cancel the effects of a static torque load. In systems without significant static torque loading, a k_i of zero may be appropriate.

The corrective force provided by the integral term increases linearly with time. The integration limit coefficient, i_l, acts as a clamping value on this force to prevent integral wind-up, a backlash effect. As noted in *Figure 18,* i_l limits the summation of error (over time), not the product of k_i and this summation. In many systems i_l can be set to its maximum value, 7FFF hex, without any adverse effects. The integral term has no effect if i_l is set to zero.

For the test system, the final values of k_i and i_l are 5 and 1000 respectively.

STEP TWO—STEP RESPONSE METHOD

Introduction

The step response of a control system reveals important information about the "quality" of control—specifically, detailed information on system damping.

In the second step to tuning the PID filter, an oscilloscope trace of the control system step response is used to accurately evaluate system damping, and the filter coefficients, determined in step one, are fine tuned to critically damp the system.

III. Tuning the PID Filter (Continued)

Software Considerations

The step generation section of the filter tuning program provides the control loop with a repetitive small-signal step input. This is accomplished by repeatedly executing a small position move with high maximum velocity and high acceleration. See *Figure 19* and *Table 12*.

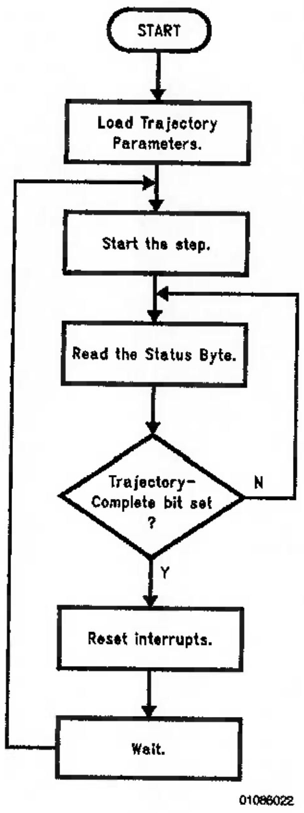


FIGURE 19. Step Generation Section of Filter Tuning Program

III. Tuning the PID Filter (Continued)

TABLE 12. Step Generation Section—Filter Tuning Program

Port	Bytes	Command	Comments
c	1F	LTRJ	This command initiates loading the trajectory parameters input buffers.
			Busy-bit Check Module
d d	00 2B	HB LB	These two bytes are the trajectory control word. A 2B hex LB indicates acceleration, velocity, and position will be loaded and both acceleration and velocity are absolute while position is relative.
			Busy-bit Check Module
d d	00 04	HB LB	Acceleration is loaded in two data words. These two bytes are the high data word.
			Busy-bit Check Module
d d	93 E0	HB LB	acceleration data word (low)
			Busy-bit Check Module
d d	00 07	HB LB	Velocity is loaded in two data words. These two bytes are the high data word.
			Busy-bit Check Module
d d	A1 20	HB LB	velocity data word (low)
			Busy-bit Check Module
d d	00 00	HB LB	Position is loaded in two data words. These two bytes are the high data word.
			Busy-bit Check Module
d d	00 C8	HB LB	position data word (low)
			Busy-bit Check Module
c	01	STT	STT must be issued to execute the desired trajectory.
			Busy-bit Check Module
c	xx	RDSTAT	This command reads the Status Byte. It is directly supported by LM628 hardware and can be executed at any time by pulling $\overline{CS}$, $\overline{PS}$, and $\overline{RD}$ logic low. Status information remains valid as long as $\overline{RD}$ is logic low.
		decision	If the Trajectory Complete interrupt bit is set, continue. Otherwise loop back to RDSTAT.
c	1D	RSTI	This command resets **only** the interrupts indicated by zeros in bits one through six of the next data word. It also resets bit fifteen of the Signals Register and the host interrupt pin (pin 17).
d d	xx 00	HB LB	don't care Zeros in bits one through six indicate **all** interrupts will be reset.
		wait	This wait block inserts a delay between repetitions of the step input. The delay is application specific, but a good range of values for the delay is 5 ms to 5000 ms.
		loop	Loop back to STT.

Hardware Considerations

For a motor control system, an oscilloscope trace of the system step response is a graph of the real position of the shaft versus time after a small and instantaneous change in desired position.

For an LM628-based system, no extra hardware is needed to view the system step response. During a step, the voltage across the motor represents the system step response, and an oscilloscope is used to generate a graph of this response (voltage).

For an LM629-based system, extra hardware is needed to view the system step response. *Figure 20* illustrates a circuit for this purpose. During a step, the voltage output of this circuit represents the system step response, and an oscilloscope is used to generate a graph of this response.

The oscilloscope trigger signal, a rectangular pulse train, is taken from the host interrupt output pin (pin 17) of the LM628/LM629. This signal is generated by the combination of a trajectory complete interrupt and a reset interrupts (RSTI) command. See *Figure 19*.

Note: The circuit of *Figure 20* can be used to view the step response of an LM628-based system.

III. Tuning the PID Filter (Continued)

Observations

What follows are example oscilloscope traces of the step response of the reference system.

Note 8: All traces were generated using the circuit of *Figure 20*.

Note 9: All traces were generated using the following "step" trajectory parameters: relative position, 200 counts; absolute velocity, 500,000 counts/sample; acceleration, 300,000 counts/sample/sample. These values generated a good small-signal step input for the reference system; other systems will require different trajectory parameters. In general, step trajectory parameters consist of a small relative position, a high velocity, and a high acceleration.

The position parameter must be relative. Otherwise, a define home command (DFH) must be added to the main loop of the step generation section—filter tuning program. See *Figure 19*.

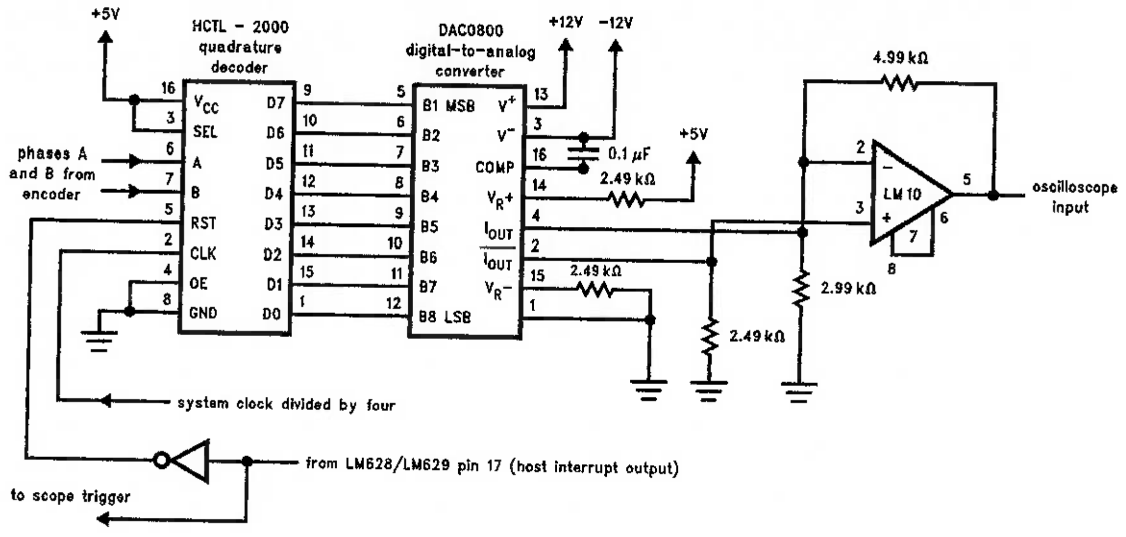
The circuit for viewing the system step response uses an 8-bit analog-to-digital converter. See *Figure 20*. To prevent converter overflow, the step position parameter must not be set higher than 200 counts.

Note 10: The circuit of *Figure 20* produces an "inverted" step response graph. The oscilloscope input was inverted to produce a positive-going (more familiar) step response graph.

Figure 21 represents the step response of an under damped control system; this response exhibits excessive overshoot and long settling time. The filter parameters used to generate this response were as follows: k_p, 35; k_i, 5; k_d, 600; d_s, 4; i_l, 1000. *Figure 21* indicates the need to increase k_d, the derivative gain coefficient.

Figure 22 represents the step response of an over damped control system; this response exhibits excessive rise time which indicates a sluggish system. The filter parameters used to generate this response were as follows: k_p, 35; k_i, 5; k_d, 10,000; d_s, 7; i_l, 1000. *Figure 22* indicates the need to decrease k_d and d_s.

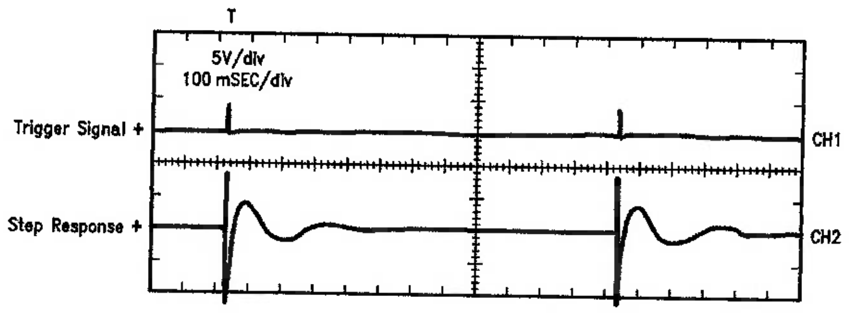
Figure 23 represents the step response of a critically damped control system; this response exhibits virtually zero overshoot and short rise time. The filter parameters used to generate this response were as follows: k_p, 40; k_i, 5; k_d, 4000; d_s, 4; i_l, 1000.



+5V
HCTL - 2000 quadrature decoder
VCC
SEL
A
B
RST
CLK
OE
GND
D7
D6
D5
D4
D3
D2
D1
D0
phases A and B from encoder
DAC0800 digital-to-analog converter
B1 MSB
B2
B3
B4
B5
B6
B7
B8 LSB
V+
V-
COMP
VR+
IOUT
IOUT
VR-
+12V
-12V
0.1 µF
2.49 kΩ
4.99 kΩ
2.99 kΩ
LM10
oscilloscope input
system clock divided by four
from LM628/LM629 pin 17 (host interrupt output)
to scope trigger

01086023

FIGURE 20. Circuit for Viewing the System Step Response with an Oscilloscope



01086024

FIGURE 21. The Step Response of an Under Damped Control System

III. Tuning the PID Filter (Continued)

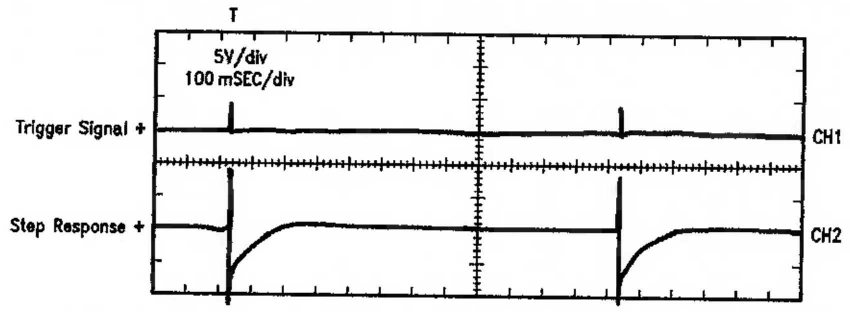


FIGURE 22. The Step Response of an Over Damped Control System

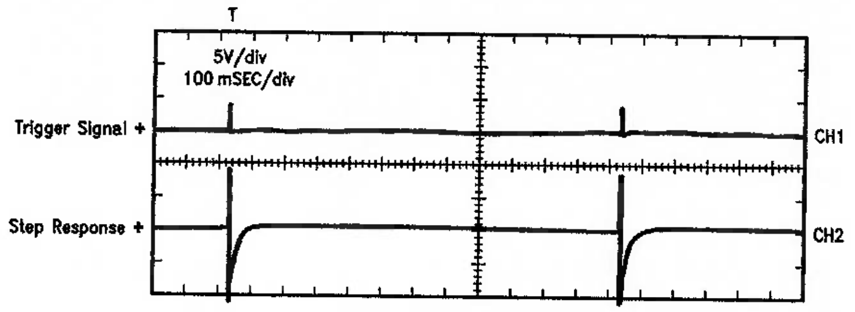


FIGURE 23. The Step Response of a Critically Damped Control System

National Semiconductor Corporation
Americas
Email: support@nsc.com

www.national.com

National Semiconductor Europe
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: ap.support@nsc.com

National Semiconductor Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507

APÊNDICE C – GUIA DE USO LM629

LM628/629 User Guide

National Semiconductor
Application Note 706
October 1993

1.0 Introduction

1.1 APPLICATION NOTE OBJECTIVE

This application note is intended to explain and complement the information in the data sheet and also address the common user questions. While no initial familiarity with the LM628/629 is assumed, it will be useful to have the LM628/629 data sheet close by to consult for detailed descriptions of the user command set, timing diagrams, bit assignments, pin assignments, etc.

After the following brief description of the LM628/629, Section 2.0 gives a fairly full description of the device's operation, probably more than is necessary to get going with the device. This section ends with an outline of how to tune the control system by adjusting the PID filter coefficients.

Section 3 "User Command Set" discusses the use of the LM628/629 commands. For a detailed description of each command the user should refer to the data sheet.

Section 4 "Helpful User Ideas" starts with a short description of the actions necessary to get going, then proceeds to talk about some performance enhancements and follows on with a discussion of a couple of operating constraints of the device.

Section 5 "Theory" is a short foray into theory which relates the PID coefficients that would be calculated from a continuous domain control loop analysis to those of the discrete domain including the scaling factors inherent to the LM628/629. No attempt is made to discuss control system theory as such, readers should consult the ample references available, some suggestions are made at the end of this application note. Section 5 concludes with an example trajectory calculation, reviving those perhaps forgotten ideas about acceleration, velocity, distance and time.

Section 6 "Questions and Answers", is in question and answer format and is born out of and dedicated to the many interesting discussions with customers that have taken place.

1.2 BRIEF DESCRIPTION OF LM628/629

LM628/629 is a microcontroller peripheral that incorporates in one device all the functions of a sample-data motion control system controller. Using the LM628/629 makes the potentially complex task of designing a fast and precise motion control system much easier. Additional features, such as trajectory profile generation, on the "fly" update of loop compensation and trajectory, and status reporting, are included. Both position and velocity motion control systems can be implemented with the LM628/629.

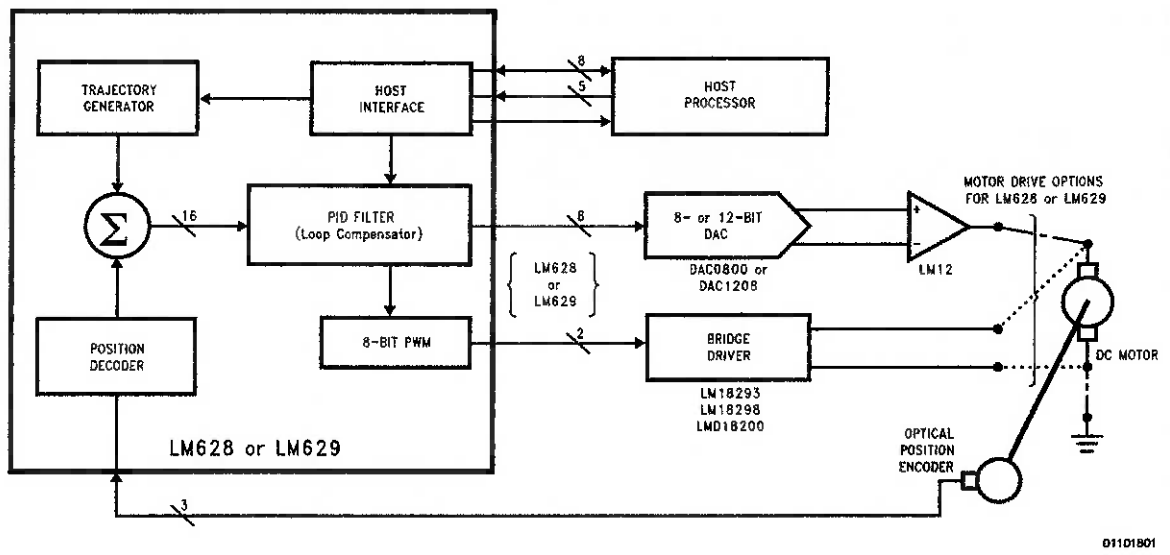


FIGURE 1. LM628 and LM629 Typical System Block Diagram

LM628/629 is itself a purpose designed microcontroller that implements a position decoder, a summing junction, a digital PID loop compensation filter, and a trajectory profile generator, *Figure 1*. Output format is the only difference between LM628 and LM629. A parallel port is used to drive an 8- or 12-bit digital-to-analog converter from the LM628 while the LM629 provides a 7-bit plus sign PWM signal with sign and magnitude outputs. Interface to the host microcontroller is via an 8-bit bi-directional data port and six control lines which includes host interrupt and hardware reset. Maximum sampling rates of either 2.9 kHz or 3.9 kHz are available by

1.0 Introduction (Continued)

choosing the LM6268/9 device options that have 6 MHz or 8 MHz maximum clock frequencies (device -6 or -8 suffixes).

In operation, to start a movement, a host microcontroller downloads acceleration, velocity and target position values to the LM628/629 trajectory generator. At each sample interval these values are used to calculate new demand or "set point" positions which are fed into the summing junction. Actual position of the motor is determined from the output signals of an optical incremental encoder. Decoded by the LM628/629's position decoder, actual position is fed to the other input of the summing junction and subtracted from the demand position to form the error signal input for the control loop compensator. The compensator is in the form of a "three term" PID filter (proportional, integral, derivative), this is implemented by a digital filter. The coefficients for the PID digital filter are most easily determined by tuning the control system to give the required response from the load in terms of accuracy, response time and overshoot. Having characterized a load these coefficient values are downloaded from the host before commencing a move. For a load that varies during a movement more coefficients can be downloaded and used to update the PID filter at the moment the load changes. All trajectory parameters except acceleration can also be updated while a movement is in progress.

2.0 Device Description

2.1 HARDWARE ARCHITECTURE

Four major functional blocks make up the LM628/629 in addition to the host and output interfaces. These are the Trajectory Profile Generator, Loop Compensating PID Filter, Summing Junction and Motor Position Decoder (*Figure 1*).

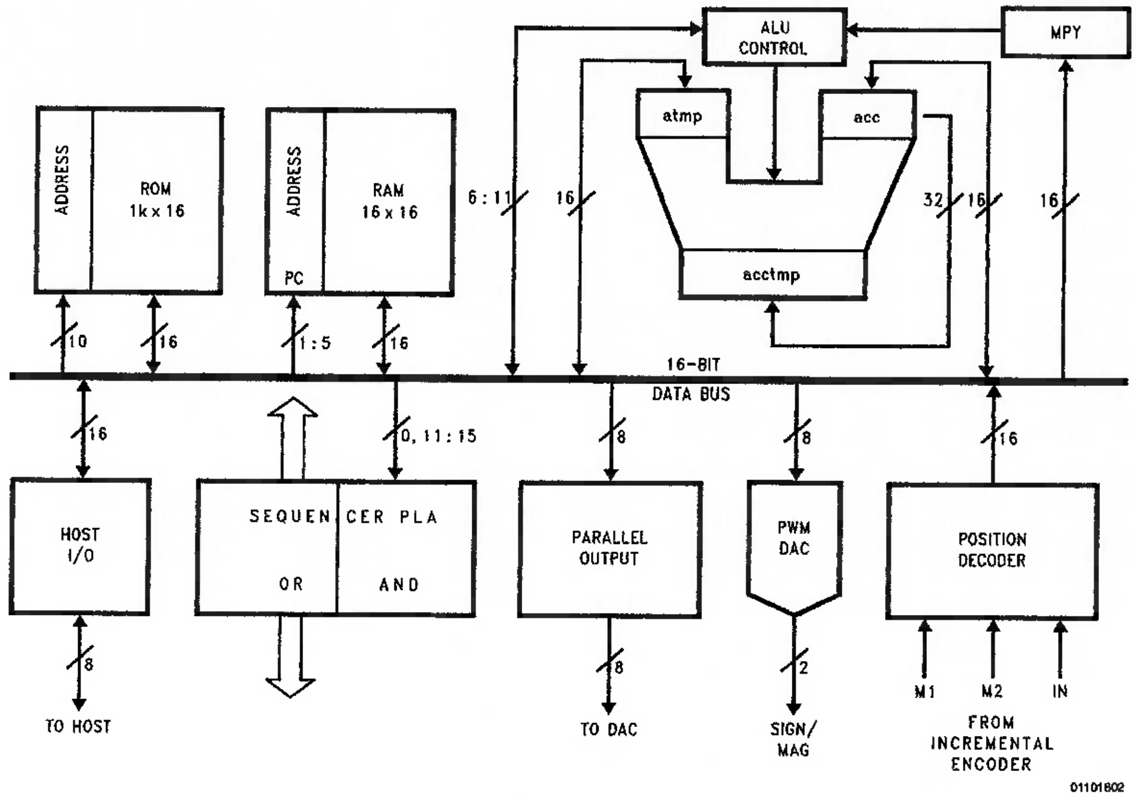


FIGURE 2. Hardware Architecture of LM628/629

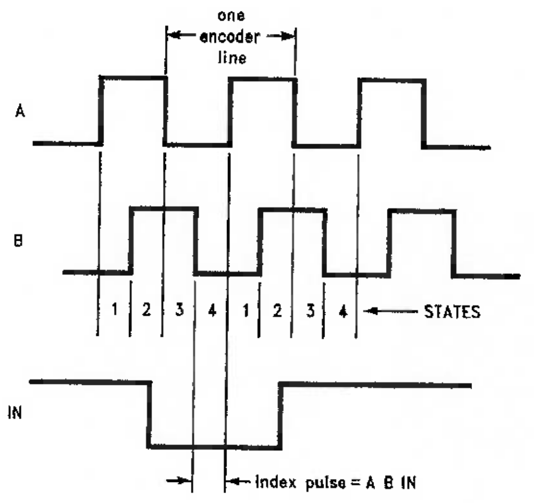
Details of how LM628/629 is implemented by a purpose designed microcontroller are shown in *Figure 2*. The control algorithm is stored in a 1k x 16-bit ROM and uses 16-bit wide instructions. A PLA decodes these instructions and provides data transfer timing signals for the single 16-bit data and instruction bus. User variable filter and trajectory profile parameters are stored as 32-bit double words in RAM. To provide sufficient dynamic range a 32-bit position register is used and for consistency. 32 bits are also used for velocity and acceleration values. A 32-bit ALU is used to support the 16 x 16-bit multiplications of the error and PID digital filter coefficients.

2.2 MOTOR POSITION DECODER

LM628/629 provides an interface for an optical position shaft encoder, decoding the two quadrature output signals to provide position and direction information, *Figure 3*. Optionally a third index position output signal can be used to capture position once per revolution. Each of the four states of the quadrature position signal are decoded by the LM628/629 giving a 4 times increase in position resolution over the number of encoder lines. An "N" line encoder will be decoded as "4N" position counts by LM628/629.

2.0 Device Description (Continued)

Position decoder block diagram, *Figure 4*, shows three lines coming from the shaft encoder, M1, M2 and Index. From these the decoder PLA determines if the motor has moved forward, backward or stayed still and then drives a 16-bit up-down counter that keeps track of actual motor position. Once per revolution when all three lines including the index line are simultaneously low, *Figure 3*, the current position count is captured in an index latch.



State	A	B
1	1	0
2	1	1
3	0	1
4	0	0
1	1	0
2	1	1
3	0	1

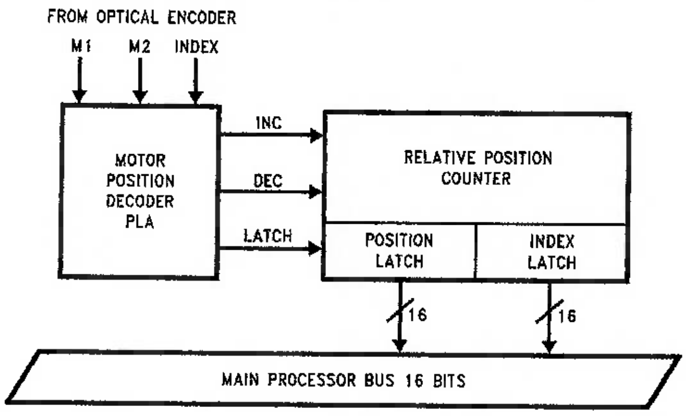
positive

direction

negative

01101803

FIGURE 3. Quadrature Encoder Output Signals and Direction Decode Table


FROM OPTICAL ENCODER
M1
M2
INDEX
MOTOR POSITION DECODER PLA
INC
DEC
LATCH
RELATIVE POSITION COUNTER
POSITION LATCH
INDEX LATCH
16
16
MAIN PROCESSOR BUS 16 BITS

01101804

FIGURE 4. LM628/629 Motor Position Decoder

The 16-bit up-down counter is used to capture the difference in position from one sample to the next. A position latch attached to the up-down counter is strobed at the same time in every sample period by a sync pulse that is generated in hardware. The position latch is read soon after the sync pulse and is added to the 32-bit position register in RAM that holds the actual current position. This is the value that is subtracted in the summing junction every sample interval from the new desired position calculated by the trajectory generator to form the error input to the PID filter.

Maximum encoder state capture rate is determined by the minimum number of clock cycles it takes to decode each encoder state, see *Figure 3*, this minimum number is 8 clock cycles, capture of the index pulse is also achieved during these 8 clock cycles. This gives a more than adequate 1 MHz maximum encoder state capture rate with the 8 MHz f_{CLK} devices (750 kHz for the 6 MHz f_{CLK} devices). For example, with the 1 MHz capture rate, a motor using a 500 line encoder will be moving at 30,000 rpm.

There is some limited signal conditioning at the decoder input to remove problems that would occur due to the asynchronous position encoder input being sampled on signal edges by the synchronous LM628/629. But there is no noise filtering as such on the encoder lines so it is important that they are kept clean and away from noise sources.

2.0 Device Description (Continued)

2.3 TRAJECTORY PROFILE GENERATOR

Desired position inputs to the summing junction, *Figure 1*, within the LM628/629 are provided by an internal independent trajectory profile generator. The trajectory profile generator takes information from the host and computes for each sample interval a new current desired position. The information required from the host is, operating mode, either position or velocity, target acceleration, target velocity and target position in position mode.

2.4 DEFINITIONS RELATING TO PROFILE GENERATION

The units of position and time, used by the LM628/629, are counts (4 x N encoder lines) and samples (sample intervals = 2048/f_{CLK}) respectively. Velocity is therefore calculated in counts/sample and acceleration in counts/sample/sample.

Definitions of "target", "desired" and "actual" within the profile generation activity as they apply to velocity, acceleration and position are as follows. Final requested values are called "target", such as target position. The values computed by the profile generator each sample interval on the way to the target value are called "desired". Real values from the position encoder are called "actual".

For example, the current actual position of the motor will typically be a few counts away from the current desired position because a new value for desired position is calculated every sample interval during profile generation. The difference between the current desired position and current actual position relies on the ability of the control loop to keep the motor on track. In the extreme example of a locked rotor there could be a large difference between the current actual and desired positions.

Current desired velocity refers to a fixed velocity at any point on a on-going trajectory profile. While the profile demands acceleration, from zero to the target velocity, the velocity will incrementally increase at each sample interval.

Current actual velocity is determined by taking the difference in the actual position at the current and the previous sample intervals. At velocities of many counts per sample this is reasonably accurate, at low velocities, especially below one count per sample, it is very inaccurate.

3.0 Profile Generation

Trajectory profiles are plotted in terms of velocity versus time, *Figure 5*, and are velocity profiles by reason that a new desired position is calculated every sample interval. For constant velocity these desired position increments will be the same every sample interval, for acceleration and deceleration the desired position increments will respectively increase and decrease per sample interval. Target position is the integral of the velocity profile.

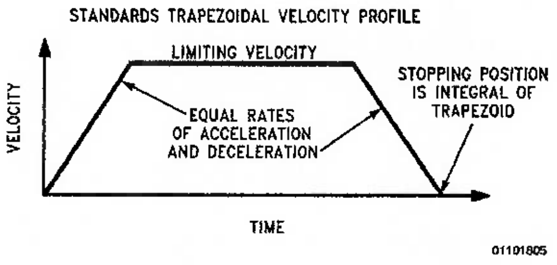


FIGURE 5. Typical Trajectory Velocity Profile

When performing a move the LM628/629 uses the information as specified by the host and accelerates until the target velocity is reached. While doing this it takes note of the number of counts taken to reach the target velocity. This number of counts is subtracted from the target position to determine where deceleration should commence to ensure the motor stops at the target position. LM628/629 deceleration rates are equal to the acceleration rates. In some cases, depending on the relative target values of velocity, acceleration and position, the target velocity will not be reached and deceleration will commence immediately from acceleration.

3.1 TRAJECTORY RESOLUTION

The resolution the motor sees for position is one integral count. The algorithm used to calculate the trajectory adds the velocity to the current desired position once per sample period and produces the next desired position point. In order to allow very low velocities it is necessary to have velocities of fractional counts per sample. The LM628/629 in addition to the 32-bit position range keeps track of 16 bits of fractional position. The need for fractional velocity counts can be illustrated by the following example using a 500 line (2000 count) encoder and an 8 MHz clock LM628/629 giving a 256 µs sample interval. If the smallest resolution is 1 count per sample then the minimum velocity would be 2 revolutions per second or 120 rpm. (1/2000 revs/count x 1/256 µs counts/second). Many applications require velocities and steps in velocity less than this amount. This is provided by the fractional counts of acceleration and velocity.

3.2 POSITION, VELOCITY AND ACCELERATION RESOLUTION

Every sample cycle, while the profile demands acceleration, the acceleration register is added to the velocity register which in turn is added to the position register. When the demand for increasing acceleration stops, only velocity is added to the position register. Only integer values are output from the position register to the summing junction and so fractional position counts must accumulate over many sample intervals before an integer count is added and the position register changed. *Figure 6* shows the position, velocity and acceleration registers.

The position dynamic range is derived from the 32 bits of the integer position register, *Figure 6*. The MSB is used for the direction sign in the conventional manner, the next bit 30 is used to signify when a position overflow called "wraparound" has occurred. If the wraparound bit is set (or reset when

3.0 Profile Generation (Continued)

going in a negative direction) while in operation the status byte bit 4 is set and optionally can be used to interrupt the host. The remaining 30 bits provide the available dynamic range of position in either the positive or negative direction (±1,073,741,824 counts).

Velocity has a resolution of $1/2^{16}$ counts/sample and acceleration has a resolution of $1/2^{16}$ counts/sample/sample as mentioned above. The dynamic range is 30 bits in both cases. The loss of one bit is due to velocity and acceleration being unsigned and another bit is used to detect wrap-around. This leaves 14 bits or 16,383 integral counts and 16 bits for fractional counts.

3.3 VELOCITY MODE

LM628 supports a velocity mode where the motor is commanded to continue at a specified velocity, until it is told to stop (LTRJ bits 9 or 10). The average velocity will be as specified but the instantaneous velocity will vary. Velocities of fractional counts per sample will exhibit the poorest instantaneous velocity. Velocity mode is a subset of position mode where the position is continually updated and moved ahead of the motor without a specified stop position. Care should be exercised in the case where a rotor becomes locked while in velocity mode as the profile generator will continue to advance the position. When the rotor becomes free high velocities will be attained to catch-up with the current desired position.

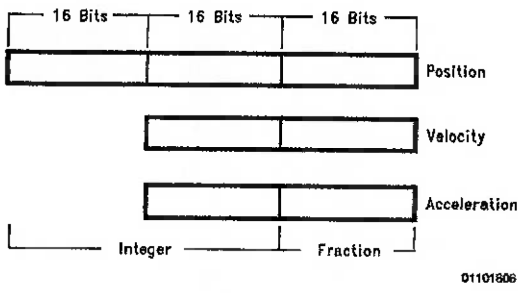


FIGURE 6. Position, Velocity and Acceleration Registers

3.4 MOTOR OUTPUT PORT

LM628 output port is configured to 8 bits after reset. The 8-bit output is updated once per sample interval and held until it is updated during the next sample interval. This allows use of a DAC without a latch. For 12-bit operation the PORT12 command should be issued immediately after reset. The output is multiplexed in two 6-bit words using pins 18 through 23. Pin 24 is low for the least significant word and high for the most significant. The rising edge of the active low strobe from pin 25 should be used to strobe the output into an external latch, see *Figure 7*. The DAC output is offset binary code, the zero codes are hex'80' for 8 bits and hex'800' for 12 bits.

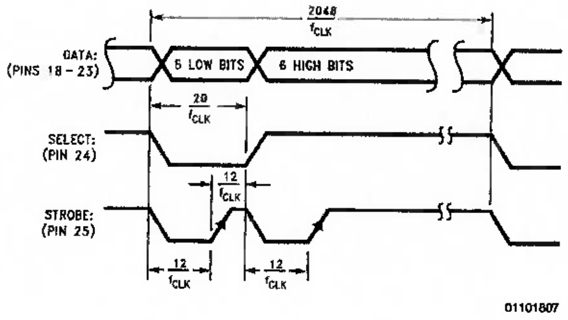
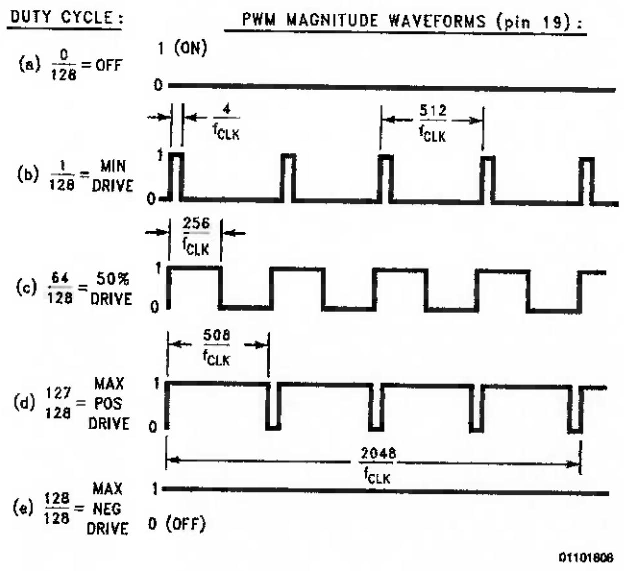


FIGURE 7. LM628 12-Bit DAC Output Multiplexed Timing

The choice of output resolution is dependant on the user's application. There is a fundamental trade-off between sampling rate and DAC output resolution, the LM628 8-bit output at a 256 µs sampling interval will most often provide as good results as a slower, e.g. microcontroller, implementation which has a 4 ms typical sampling interval and uses a 12-bit output. The LM628 also gives the choice of a 12-bit DAC output at a 256 µs sampling interval for high precision applications.

LM629 PWM sign and magnitude signals are output from pins 18 and 19 respectively. The sign output is used to control motor direction. The PWM magnitude output has a resolution of 8 bits from maximum negative drive to maximum positive drive. The magnitude output has an off condition, with the output at logic low, which is useful for turning a motor off when using a bridge motor drive circuit. The minimum duty cycle is 1/128 increasing to a maximum of 127/128 in the positive direction and a maximum of 128/128 in the negative direcition, i.e., a continuous output. There are four PWM periods in one LM629 sample interval. With an 8 MHz clock this increases the PWM output rate to 15.6 kHz from the LM629 maximum 3.9 kHz sample rate, see *Figure 8* for further timing information.

3.0 Profile Generation (Continued)



Note: Sign output (pin 18) not shown.

FIGURE 8. LM629 PWM Output Signal Format

3.5 HOST INTERFACE

LM628/629 has three internal registers: status, high, and low bytes, *Figure 9*, which are used to communicate with the host microcontroller. These are controlled by the $\overline{RD}$, $\overline{WR}$, and $\overline{PS}$ lines and by use of the busy bit of the status byte. The status byte is read by bringing $\overline{RD}$ and $\overline{PS}$ low, bit 0 is the busy bit. Commands are written by bringing $\overline{WR}$ and $\overline{PS}$ low. When $\overline{PS}$ is high, $\overline{WR}$brought low writes data into LM628/629 and similarly, $\overline{RD}$ is brought low to read data from LM628/629. Data transfer is a two-byte operation written in most to least significant byte order. The above description assumes that $\overline{CS}$ is low.

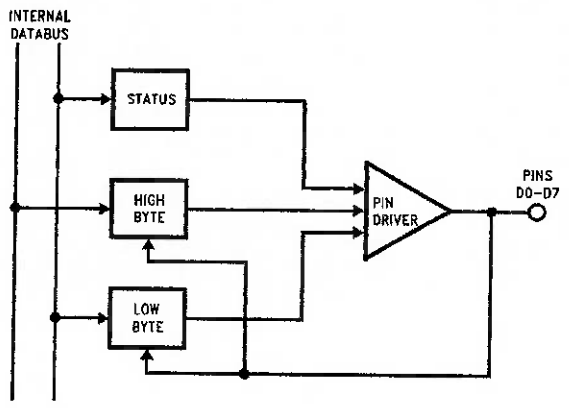


FIGURE 9. Host Interface Internal I/O Registers

3.6 HARDWARE BUSY BIT OPERATION

Before and between all command byte and data byte pair transfers, the busy bit must be read and checked to be at logic low. If the busy bit is set and commands are issued they will be ignored and if data is read it will be the current contents of the I/O buffer and not the expected data. The busy bit is set after the rising edge of the write signal for commands and the second rising edge of the respective read or write signal for two byte data transfers, *Figure 10.* The busy bit remains high for approximately 15 µs.

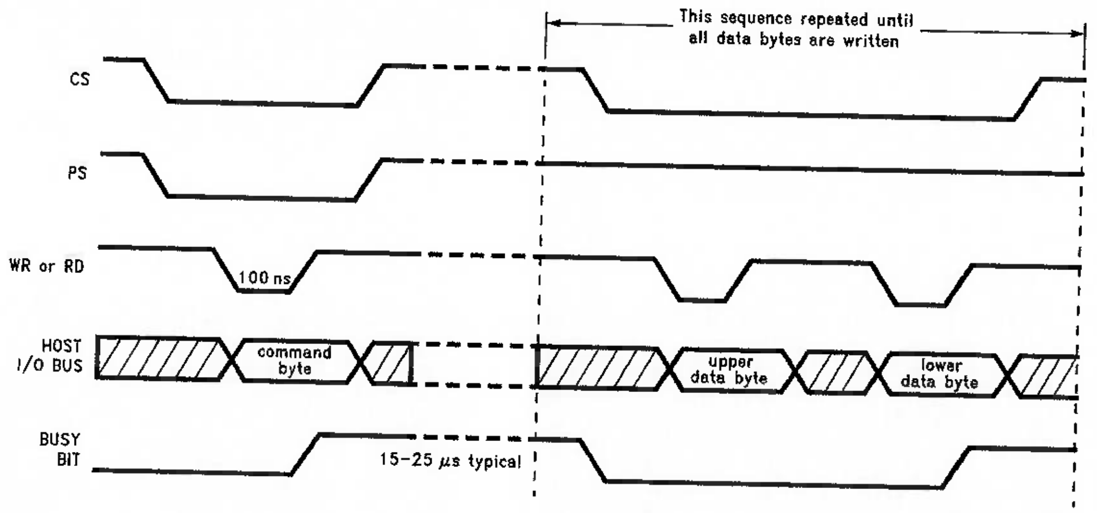


FIGURE 10. Busy Bit Operation during Command and Data Write Sequence

The busy bit reset to logic low indicates that high and low byte registers shown in *Figure 9* have been either loaded or read by the LM628/629 internal microcode. To service the command or data transfer this microcode which performs the trajectory and filter calculations is interrupted, except in critical areas, and the on-going calculation is suspended. The microcode was designed this way to achieve minimum latency when communicating with the host. However, if this communication becomes too frequent and on-going calculations are interrupted too often corruption will occur. In a 256 µs sample interval, the filter calculation takes 50 µs, outputting a sample 10 µs and trajectory calculation 90 µs. If the LM628 behaves in a manner that is unexpected the host communication rate should be checked in relation to these timings.

3.7 FILTER INITIAL VALUES AND TUNING

When connecting up a system for the first time there may be a possibility that the loop phasing is incorrect. As this may cause violent oscillation it is advisable to initially use a very low value of proportional gain, say $k_p = 1$ (with k_d, k_i and il all set to zero), which will provide a weak level of drive to the motor. (The Start command, STT, is sent to LM628/629 to close the control loop and energize the motor.) If the system does oscillate with this low value of k_p then the motor connections should be reversed.

Having determined that the loop phasing is correct k_p can be increased to a value of about 20 to see that the control system basically works. This value of k_p should hold the motor shaft reasonably stiffly, returning the motor to the set position, which will be zero until trajectory values have been input and a position move performed. If oscillation or unacceptable ringing occurs with a k_p value of 20 reduce this until it stops. Low values of acceleration and velocity can now be input, of around 100, and a position move commanded to say 1000 counts. All values suggested here are decimal. For details of loading trajectory and filter parameters see Section 3.0, reference (5) and the data sheet.

It is useful at this stage to try different values of acceleration and velocity to get a feel for the system limitations. These can be determined by using the reporting commands of desired and actual position and velocity, to see if the error between desired and actual positions of the motor are constant and not increasing without bound. See Section 3.6 and the data sheet for information about the reporting commands. Clearly it will be difficult to tune for best system response if the motor and its load cannot achieve the demanded values of acceleration and velocity. When correct operation is confirmed and limiting values understood, filter tuning can commence.

Due to the basic difficulty of accurately modeling a control system, with the added problem of variations that can occur in mechanical components over time and temperature, it is always necessary at some stage to perform tuning empirically. Determining the PID filter coefficients by tuning is the preferred method with LM628/629 because of the inherent flexibility in changing the filter coefficients provided by this programmable device.

Before tuning a control system the effect of each of the PID filter coefficients should be understood. The following is a very brief review, for a detailed understanding reference (2) should be consulted. The proportional coefficient, k_p, provides adjustment of the control system loop proportional gain, as this is increased the output steady state error is reduced. The error between the required and actual position is effectively divided by the loop gain. However there is a natural limitation on how far k_p can be increased on its own to reduce output position error because a reduction in phase margin is also a consequence of increasing k_p. This is first encountered as ringing about the final position in response to a step change input and then instability in the form of oscillation as the phase margin becomes zero. To improve stability, k_d, the derivative coefficient, provides a damping effect by providing a term proportional to velocity in an-

3.0 Profile Generation (Continued)

tiphase to the ringing, or viewed in another way, adds some leading phase shift into the loop and increases the phase margin.

In the tuning process the coefficients k_p and k_d are iteratively increased to their optimum values constrained by the system constants and are trade-offs between response time, stability and final position error. When k_p and k_d have been determined the integral coefficient, k_i, can be introduced to remove steady state errors at the load. The steady state errors removed are the velocity lag that occurs with a constant velocity output and the position error due to a constant static torque. A value of integration limit, il, has to be input with k_i, otherwise k_i will have no effect. The integral coefficient k_i adds another variable to the system to allow further optimization, very high values of k_i will decrease the phase margin and hence stability, see Section 5 and reference (2) for more details. Reference (5) gives more details of PID filter tuning and how to load filter parameters.

Figure 11 illustrates how a relatively slow response with overshoot can be compensated by adjustment of the PID filter coefficients to give a faster critically damped response.

4.0 User Command Set

4.1 OVERVIEW

The following types of User Commands are available:

Initialization

Filter control commands

Trajectory control commands

Interrupt control commands

Data reporting commands

User commands are single bytes and have a varying number of accompanying data bytes ranging from zero to fourteen depending upon the command. Both filter and trajectory control commands use a double buffered scheme to input data. These commands load primary registers with multiple words of data which are only transferred into secondary working registers when the host issues a respective single byte user command. This allows data to be input before its actual use which can eliminate any potential communication bottlenecks and allow synchronized operation of multiple axes.

4.2 HOST-LM628/629 COMMUNICATION—THE BUSY BIT

Communication flow between the LM628/629 and its host is controlled by using a busy bit, bit 0, in the Status Byte. The busy bit must be checked to be at logic 0 by the host before commands and data are issued or data is read. This includes between data byte pairs for commands with multiple words of data.

4.3 LOADING THE TRAPEZOIDAL VELOCITY PROFILE GENERATOR

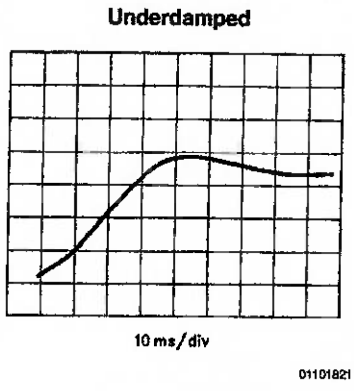
To initiate a motor move, trajectory generator values have to be input to the LM628/629 using the Load Trajectory Parameters, LTRJ, command. The command is followed by a trajectory control word which details the information to be loaded in subsequent data words. *Table 1* gives the bit allocations, a bit is set to logic 1 to give the function shown.

TABLE 1. Trajectory Control Word Bit Allocations

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

Bits 0 to 5 determine whether any, all or none of the position, velocity or acceleration values are loaded and whether they are absolute values or values relative to those previously loaded. All trajectory values are 32-bit values, position values are both positive and negative. Velocity and acceleration are 16-bit integers with 16-bit fractions whose absolute value is always positive. When entering relative values ensure that the absolute value remains positive. The manual stop commands bits 8, 9 and 10 are intended to allow an unprogrammed stop in position mode, while a position move is in progress, perhaps by the demand of some external event, and to provide a method to stop in velocity mode. They do not specify how the motor will stop in position mode at the end of a normal position move. In position mode a programmed move will automatically stop with a deceleration rate equal to the acceleration rate at the target position. Setting a stop bit along with other trajectory parameters at the beginning of a move will result in no movement! Bits 8, 9 and 10 should only be set one at a time, bit 8 turns the motor off by outputting zero drive to the motor, bit 9 stops the motor at maximum deceleration by setting the target position equal to the current position and bit 10 stops the motor using the current user-programmed acceleration value. Bit 11 is set for operating in velocity mode and bit 12 is set for forward direction in velocity mode.

4.0 User Command Set (Continued)



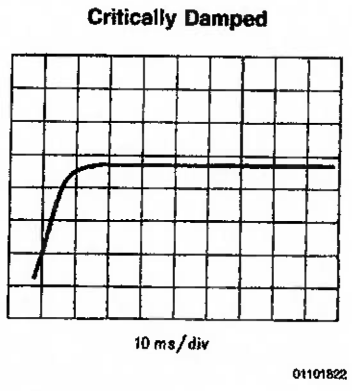


FIGURE 11. Position vs Time for 100 Count Step Input

Following immediately after the trajectory control word should be two 16-bit data words for each parameter specified to be loaded. These should be in the descending order of the trajectory control word bits, that is acceleration, velocity and position. They are written to the LM628/629 as two pairs of data bytes in most to least significant byte order. The busy bit should be checked between the command byte and the data byte pair forming the trajectory control word and the individual data byte pairs of the data. The Start command, STT, transfers the loaded trajectory data into the working registers of the double buffered scheme to initiate movement of the motor. This buffering allows any parameter, except acceleration, to be updated while the motor is moving by loading data with the LTRJ command and to be later executed by using the STT command.

New values of acceleration can be loaded with LTRJ while the motor is moving, but cannot be executed by the STT command until the trajectory has completed or the drive to the motor is turned off by using bit 8 of the trajectory control word. If acceleration has been changed and STT is issued while the drive to the motor is still present, a command error interrupt will be generated and the command ignored. Separate pairs of LTRJ and STT commands should be issued to first turn the motor off and then update acceleration. System operation when changing acceleration while the motor is moving, but with the drive removed, is discussed in Section 4.5.1.

4.4 LOADING PID FILTER COEFFICIENTS

PID filter coefficients are loaded using the Load Filter Parameters, LFIL, command and are the proportional coefficient k_p, derivative coefficient k_d and integral coefficient k_i. Associated with k_i, an integration limit, il, has to be loaded. This constrains the magnitude of the integration term of the PID filter to the il value, see Section 4.4.2. Associated with the derivative coefficient, a derivative sample rate can be chosen from $2048/f_{CLK}$ to $(2048 \times 256)/f_{CLK}$ in steps of $2048/f_{CLK}$, see Section 4.4.1.

The first pair of data bytes following the LFIL command byte form the filter control word. The most significant byte sets the derivative sample rate, the fastest rate, $2048/f_{CLK}$, being hex'00' the slowest rate $(2048 \times 256)/f_{CLK}$ being hex'FF'. The lower four bits of the least significant byte tell the LM628/629 which of the coefficients is going to be loaded, bit 3 is k_p, bit 2 is k_i, bit 1 is k_d and bit 0 is il. Each filter coefficient and the integration limit can range in value from hex'0000' to '7FFF', positive only. If all coefficient values are loaded then ten bytes of data, including the filter control word, will follow the LFIL command. Again the busy bit has to be checked between the command byte and filter control word and between data byte pairs. Use of new filter coefficient values by the LM628/629 is initiated by issuing the single byte Update Filter command, UDF.

When controlled movement of the motor has been achieved, by programming the filter and trajectory, attention turns to incorporating the LM628/629 into a system. Interrupt Control Commands and Data Reporting Commands enable the host microcontroller to keep track of LM628/629 activity.

4.5 INTERRUPT CONTROL COMMANDS

There are five commands that can be used to interrupt the host microcontroller when a predefined condition occurs and two commands that control interrupt operation. When the LM628/629 is programmed to interrupt its host, the event which caused this interrupt can be determined from bits 1 to 6 of the Status Byte (additionally bit 0 is the busy bit and bit 7 indicates that the motor is off). All the Interrupt Control commands are executable during motion.

The Mask Interrupts command, MSKI, is used to tell LM628/629 which of bits 1 to 6 will interrupt the host through use of interrupt mask data associated with the command. The data is in the form of a data byte pair, bits 1–6 of the least significant byte being set to logic 1 when an interrupt source is enabled. The Reset Interrupts command, $\overline{\text{RSTI}}$, resets interrupt bits in the Status Byte by sending a data byte pair, the least significant byte having logic 0 in bit positions 1 to 6 if they are to be reset.

Executing the Set Index Position command, SIP, causes bit 3 of the status byte to be set when the absolute position of the next index pulse is recorded in the index register. This can be read with the command, Read Index Position, RDIP.

Executing either Load Position Error for Interrupt, LPEI, or Load Position Error for Stopping, LPES, commands, sets bit 5 of the Status Byte when a position error exceeding a specified limit occurs. An excessive position error can indicate a serious system problem and these two commands give the option when this occurs of either interrupting the

4.0 User Command Set (Continued)

host or stopping the motor and interrupting the host. The excessive position is specified following each command by a data byte pair in most to least significant byte order.

Executing either Set Break Point Absolute, SBPA, or Set Break Point Relative, SBPR, commands, sets bit 6 of the status byte when either the specified, absolute or relative, breakpoint respectively is reached. The data for SBPA can be the full position range (hex'C0000000' to '3FFFFFFF') and is sent in two data byte pairs in most to least significant byte order. The data for the Set Breakpoint Relative command is also of two data byte pairs, but its value should be such that when added to the target position it remains within the absolute position range. These commands can be used to signal the moment to update the on-going trajectory or filter coefficients. This is achieved by transferring data from the primary registers, previously loaded using LTRJ or LFIL, to working registers, using the STT or UDF commands.

Interrupt bits 1, 2 and 4 of the Status Byte are not set by executing interrupt commands but by events occurring during LM628/629 operation as follows. Bit 1 is the command error interrupt, bit 2 is the trajectory complete interrupt and bit 4 is the wraparound interrupt. These bits are also masked and reset by the MSKI and RSTI commands respectively. The Status Byte still indicates the condition of interrupt bits 1–6 when they are masked from interrupting the host, allowing them to be incorporated in a polling scheme.

4.6 DATA REPORTING COMMANDS

Read Status Byte, RDSTAT, supported by a hardware register accessed via $\overline{CS}$, $\overline{RD}$ and $\overline{PS}$ control, is the most frequently used method of determining LM628/629 status. This is primarily to read the busy bit 0 while communicating commands and data as described in Section 3.2.

There are seven other user commands which can read data from LM628/629 data registers.

The Read Signals Register command, RDSIGS, returns a 16-bit data word to the host. The least-significant byte repeats the RDSTAT byte except for bit 0 which indicates that a SIP command has been executed but that an index pulse has not occurred. The most significant byte has 6 bits that indicate set-up conditions (bits 8, 9, 11, 12, 13 and 14). The other two bits of the RDSIGS data word indicate that the trajectory generator has completed its function, bit 10, and that the host interrupt output (Pin 17) has been set to logic 1, bit 15. Full details of the bit assignments of this command can be found in the data sheet.

The Read Index Position, RDIP, command reads the position recorded in the 32 bits of the index register in four data bytes. This command, with the SIP command, can be used to acquire a home position or successive values. These could be used, for example, for gross error checking.

Both on-going 32-bit position inputs to the summing junction can be read. Read desired position, RDDP, reads the current desired position the demand or "set point input" from the trajectory generator and Read Real Position, RDRP, reads the current actual position of the motor.

Read Desired Velocity, RDDV, reads the current desired velocity used to calculate the desired position profile by the trajectory generator. It is a 32-bit value containing integer and fractional velocity information. Read Real Velocity, RDRV, reads the instantaneous actual velocity and is a 16-bit integer value.

Read Integration-Term Summation Value, RDSUM, reads the accumulated value of the integration term. This is a 16-bit value ranging from zero to the current, il, integration limit value.

4.7 SOFTWARE EXAMPLE

The following example shows the flow of microcontroller commands needed to get the LM628/629 to control a simple motor move. As it is non-specific to any microcontroller pseudo commands $\overline{WR}$,XXXXH and RD,XXXXH with hex immediate data will be used to indicate read and write operations respectively by the host to and from the LM628/629. Decisions use IF..THEN..ELSE. BUSY is a user routine to check the busy bit in the Status Byte, WAIT is a user routine to wait 1.5 ms after hardware reset.

```
LABEL    MNEMONIC           :REMARK
Initialization:
         WAIT               :Routine to wait 1.5 ms after reset.
         RDSTAT             :Check correct RESET operation by reading the
                            :Status Byte. This should be either hex'84' or 'C4'
         IF Status byte not equal hex'84' or 'C4' THEN repeat
         hardware RESET
                            :Make decision concerning validity of RESET
```

Optionally the Reset can be further checked for correct operation as follows. It is useful to include this to reset all interrupt bits in the Status Byte before further action:

```
           MSKI             :Mask interrupts
           BUSY             :Check busy bit 0 routine
           WR, 0000H        :Host writes two zero bytes of data to
                            :LM628/629. This mask disables all interrupts.
           BUSY             :Check busy bit
           RSTI             :Reset Interrupts command
           BUSY             :Check busy bit
           WR, 0000H        :Host writes two zero bytes of data to LM628/629
           RDSTAT           :Status byte should read either hex'80' or 'C0'
           IF Status Byte not equal hex'80' or 'C0' THEN repeat
           hardware RESET
                            :
           IF Status Byte equal to hex'C0' THEN continue ELSE PORT
                            :
           BUSY             :Check busy bit
```

4.0 User Command Set (Continued)

```
                RSTI            :Reset Interrupts
                BUSY            :Check busy bit
                WR, 0000H       :Reset all interrupt bits
      Set Output Port Size for a 12-bit DAC.
      PORT      BUSY            :Check busy bit
                PORT12          :Sets LM628 output port to 12-bits
                                 (Only for systems with 12-bit DAC)
      Load Filter Parameters
               BUSY             :Check busy bit
               LFIL             :Load Filter Parameters command
               BUSY             :Check busy bit
               WR, 0008H        :Filter Control Word
                                :    Bits 8 to 15 (MSB) set the derivative
                                :sample rate.
                                :    Bit 3     Loading kp data
                                :    Bit 2     Loading ki data
                                :    Bit 1     Loading kd data
                                :    Bit 0     Loading il data
                                :Choose to load kp only at maximum
                                :derivative sample rate then Filter Control
                                :Word = 0008H
               BUSY             :Check busy bit
               WR, 0032H        :Choose kp = 50, load data byte pair MS
                                :byte first
      Update Filter
               BUSY             :Check busy bit
               UDF              :
      Load Trajectory Parameters
               BUSY             :Check busy bit
               LTRJ             :Load trajectory parameters command.
               BUSY             :Check busy bit
               WR, 002AH        :Load trajectory control word:
                                :    See Table I
                                :Choose Position mode, and load absolute
                                :acceleration, velocity and position. Then
                                :trajectory control word = 002AH. This means
                                :6 pairs of data bytes should follow.
               BUSY             :Check busy bit
               WR, XXXXH        :Load Acceleration integer word MS byte first
               BUSY             :Check busy bit
               WR, XXXXH        :Load Acceleration fractional word MS byte first
               BUSY             :Check busy bit
               WR, XXXXH        :Load Velocity integer word MS byte first
               BUSY             :Check busy bit
               WR, XXXXH        :Load Velocity fractional word MS byte first
               BUSY             :Check busy bit

               WR, XXXXH        :Load Position MS byte pair first
               BUSY             :Check busy bit
               WR, XXXXH        :Load position LS byte pair
      Start Motion
               BUSY             :Check busy bit
               STT              :Start command
      Check for Trajectory complete.
               RDSTAT           :Check Status Byte bit 2 for trajectory
                                :complete
      Busy bit check routine
      BUSY     RDSTAT           :Read status byte
               If bit 0 is set THEN BUSY ELSE RETURN
               END
```

*Consult reference (5) for more information on programming the LM628/629.

4.0 User Command Set (Continued)

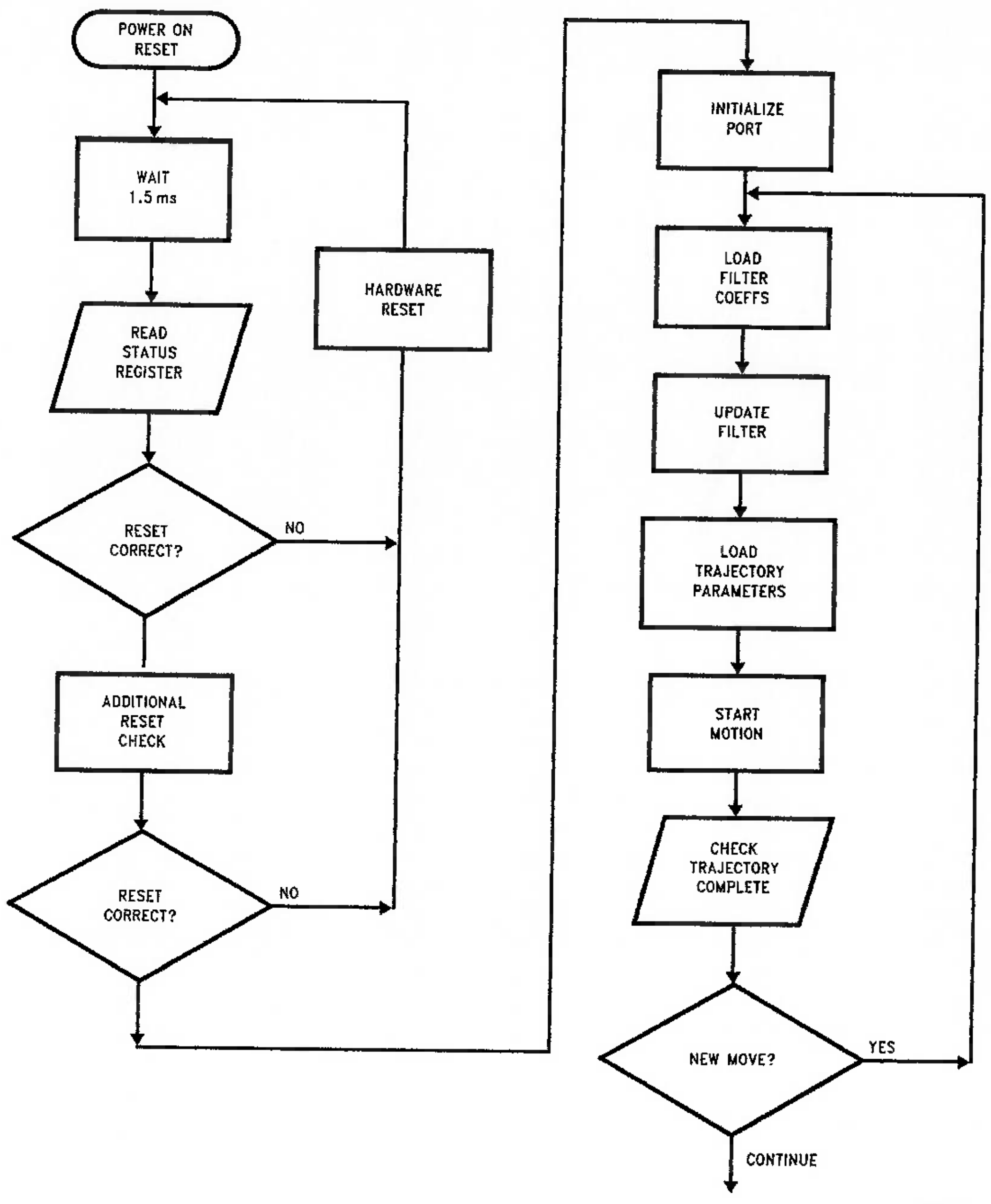


FIGURE 12. Basic Software Flow

5.0 Helpful User Ideas

5.1 GETTING STARTED

This section outlines the actions that are necessary to implement a simple motion control system using LM628/629. More details on how LM628/629 works and the use of the User Command Set are given in the sections "2.0 DEVICE DESCRIPTION" and "3.0 USER COMMAND SET".

5.2 HARDWARE

The following hardware connections need to be made:

5.2.1 Host Microcontroller Interface

Interface to the host microcontroller is via an 8-bit command/data port which is controlled by four lines. These are the conventional chip select $\overline{CS}$, read $\overline{RD}$, write $\overline{WR}$ and a line called Port Select $\overline{PS}$, see *Figure 13*. $\overline{PS}$ is used to select

5.0 Helpful User Ideas (Continued)

user Command or Data transfer between the LM628/629 and the host. In the special case of the Status Byte (RDSTAT) bringing $\overline{PS}$, $\overline{CS}$ and $\overline{RD}$ low together allows access to this hardware register at any time. An optional interrupt line, HI, from the LM628/629 to the host can be used. A microcontroller output line is necessary to control the LM628/629 hardware reset action.

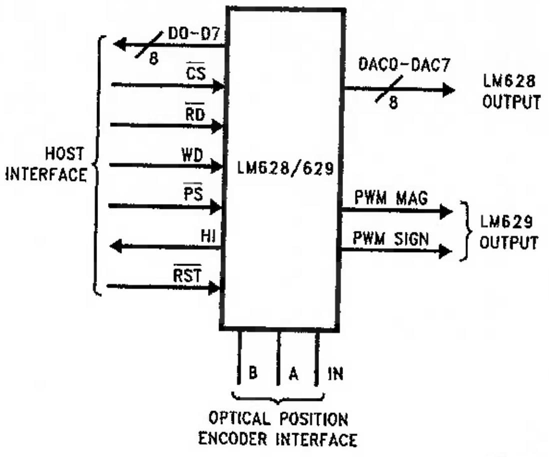
5.2.2 Position Encoder Interface

The two optical incremental position encoder outputs feed into the LM628/629 quadrature decoder TTL inputs A and B. The leading phase of the quadrature encoder output defines the forward direction of the motor and should be connected to input A. Optionally an index pulse may be used from the position encoder. This is connected to the $\overline{IN}$ input, which should be tied high if not used, see *Figure 13.*

5.2.3 Output Interface

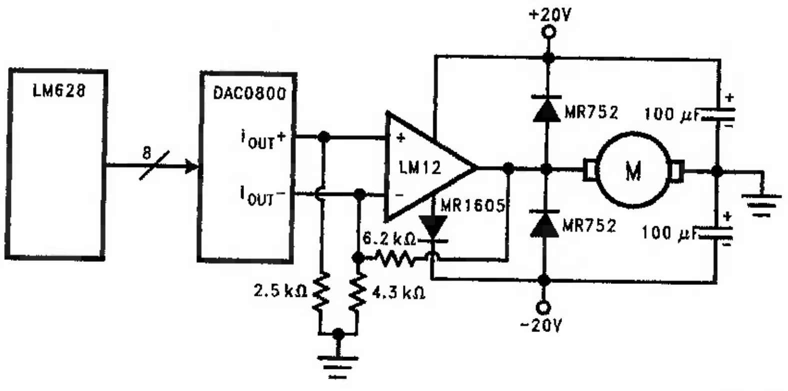
LM628 has a parallel output of either 8 or 12 bits, the latter is output as two multiplexed 6-bit words. *Figure 14* illustrates how a motor might be driven using a LM12 power linear amplifier from the output of 8-bit DAC0800.

LM629 has a sign and magnitude PWM output, *Figure 13,* of 7-bit resolution plus sign. *Figure 15* shows how the LM629 sign and magnitude outputs can be used to control the outputs of an LM18293 quad half-H driver. The half-H drivers are used in pairs, by using 100 mΩ current sharing resistors, and form a full-H bridge driver of 2A output. The sign bit is used to steer the PWM LM629 magnitude output to either side of the H-bridge lower output transistors while holding the upper transistors on the opposite side of the H-bridge continuously on.



01101813

FIGURE 13. LM628 and LM629 Host, Output and Position Encoder Interfaces



01101814

FIGURE 14. LM628 Example of Linear Motor Drive Using LM12

5.0 Helpful User Ideas (Continued)

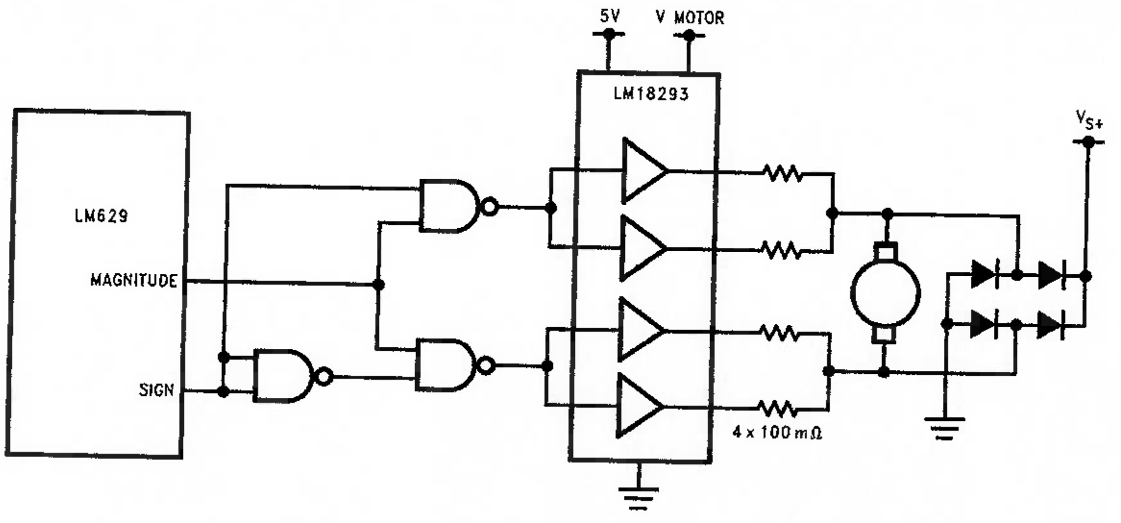


FIGURE 15. LM629 H-Bridge Motor Drive Example Using LM18293

5.3 SOFTWARE

Making LM628/629 perform a motion control function requires that the host microcontroller, after initializing LM628/629, loads coefficients for the PID filter and then loads trajectory information. The interrupt and data reporting commands can then be used by the host to keep track of LM628/629 actions. For detailed descriptions see the LM628/629 data sheet and Section 3.

5.4 INITIALIZATION

There is only one initialization operation that must be performed; a check that hardware reset has operated correctly. If required, the size of the LM628 output port should be configured. Other operations which might be part of user's system initialization are discussed under Interrupt and Data Reporting commands, Sections 3.5 and 3.6.

5.4.1 Initializing LM628 Output Port

Reset sets the LM628 output port size to 8 bits. If a 12-bit DAC is being used, then the output port size is set by the use of the PORT12 command.

5.4.2 Hardware RESET Check

The hardware reset is activated by a logic low pulse at pin 27, $\overline{\text{RST}}$, from the host of greater than 8 clock cycles. To ensure that this reset has operated correctly the Status Byte should be checked immediately after the reset pin goes high, it should read hex'00'. If the reset is successful this will change to hex'84' or 'C4' within 1.5 ms. If not, the hardware reset and check should be repeated. A further check can be used to make certain that a reset has been successful by using the Reset Interrupts command, RSTI. Before sending the RSTI, issue the Mask Interrupts command, MSKI, and mask data that disables all interrupts, this mask is sent as two bytes of data equaling hex'0000'. Then issue the RSTI command plus mask data that resets all interrupts, this equals hex'0000' and is again sent as two bytes. Do not forget to check the busy bit between the command byte and data byte pairs. When the chip has reset properly the status byte will change from hex'84' or 'C4' to hex'80' or 'C0'.

5.4.3 Interrupt Commands

Optionally the commands which cause the LM628/629 to take action on a predefined condition (e.g., SIP, LPEI, LPES, SBPA and SBPR) can be included in the initialization, these are discussed under Interrupt Commands.

5.5 PERFORMANCE REFINEMENTS

5.5.1 Derivative Sample Rate

The derivative sample interval is controllable to improve the stability of low velocity, high inertia loads. At low speeds, when fractional counts for velocity are used, the integer position counts, desired and actual, only change after several sample intervals of the LM628/629 ($2048/f_{CLK}$). This means that for sample intervals between integer count changes the error voltage will not change for successive samples. As the derivative term, k_d, multiplies the difference betweeen the previous and current error values, if the derivative sample interval is the same as the sample interval, several consecutive sample intervals will have zero derivative term and hence no damping contribution. Lengthening the derivative sample interval ensures a more constant derivate term and hence improved stability. Derivative sample interval is loaded with the filter coefficient values as the most significant byte of the LFIL control word everytime the command is used, the host therefore needs to store the current value for re-loading at times of filter coefficient change.

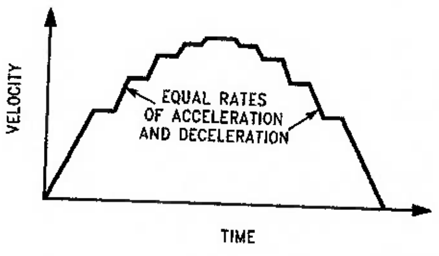
5.5.2 Integral Windup

Along with the integral filter coefficient, k_i, an integration limit, il, has to be input into LM628/629 which allows the user to set the maximum value of the integration term of equation (3), Section 5.2.2. This term is then able to accumulate up to

5.0 Helpful User Ideas (Continued)

the value of the integration limit and any further increase due to error of the same sign is ignored. Setting the integration limit enables the user to prevent an effect called "Integral Windup". For example, if an LM628/629 attempts to accelerate a motor at a faster rate than it can achieve, a very large integral term will result. When the LM628/629 tries to stop the motor at the target position the large accumulated integral term will dominate the filter and cause the motor to badly overshoot, and thus integral windup has occurred.

5.5.3 Profiles Other Than Trapezoidal



01101816

FIGURE 16. Generating a Non-Trapezoidal Profile

If it is required to have a velocity profile other than trapezoidal, this can be accomplished by breaking the profile into small pieces each of which is part of a small trapezoid. A piecewise linear approximation to the required profile can then be achieved by changing the maximum velocity before the trapezoid has had time to complete, see *Figure 16*.

5.5.4 Synchronizing Axes

For controlling tightly coupled coordinated motion between multiple-axes, synchronization is required. The best possible synchronization that can be achieved between multiple LM628/629 is within one sample interval, (2048/f_{CLK}, 256 µs for an 8 MHz clock, 341 µs for a 6 MHz clock). This is achieved by using the pipeline feature of the LM628/629 where all controlled axes are loaded individually with trajectory values using the LTRJ command and then simultaneously given the start command STT. PID filter coefficients can be updated in a similar manner using LFIL and UDF commands.

5.6 OPERATING CONSTRAINTS

5.6.1 Updating Acceleration on the Fly

Whereas velocity and target position can be updated while the motor is moving, on the "fly", the algorithm described in Section 2.5 prevents this for acceleration. To change acceleration while the motor is moving in mid-trajectory the motor off command has to be issued by setting LTRJ command bit 8. Then the new acceleration can be loaded, again using the LTRJ command. When the start command STT is issued the motor will be energized and the trajectory generator will start generating a new profile from the actual position when the STT command was issued. In doing this the trajectory generator will assume that the motor starts from a stationary position in the normal way. If the motor has sufficient inertia and is still moving when the STT command is issued then the control loop will attempt to bring the motor on to the new profile, possibly with a large error value being input to the PID filter and a consequential saturated output until the motor velocity matches the profile. This is a classic case of overload in a feedback system. It will operate in an open loop manner until the error input gets within controllable bounds and then the feedback loop will close. Performance in this situation is unpredictable and application specific. LM628/629 was not intentionally designed to operate in this way.

5.6.2 Command Update Rate

If an LM628/629 is updated too frequently by the host it will not keep up with the commands given. The LM628/629 aborts the current trajectory calculation when it receives a new STT command, resulting in the output staying at the value of the previous sample. For this reason it is recommended that trajectory is not updated at a greater rate than once every 10 ms.

6.0 Theory

6.1 PID FILTER

6.1.1 PID Filter in the Continuous Domain

The LM628/629 uses a PID filter as the loop compensator, the expression for the PID filter in the continuous domain is:

$$H(s) = K_p + K_i/s + K_d s \quad (1)$$

Where K_p = proportional coefficient
K_i = integral coefficient
K_d = derivative coefficient

6.0 Theory (Continued)

6.1.2 PID Filter Bode Plots

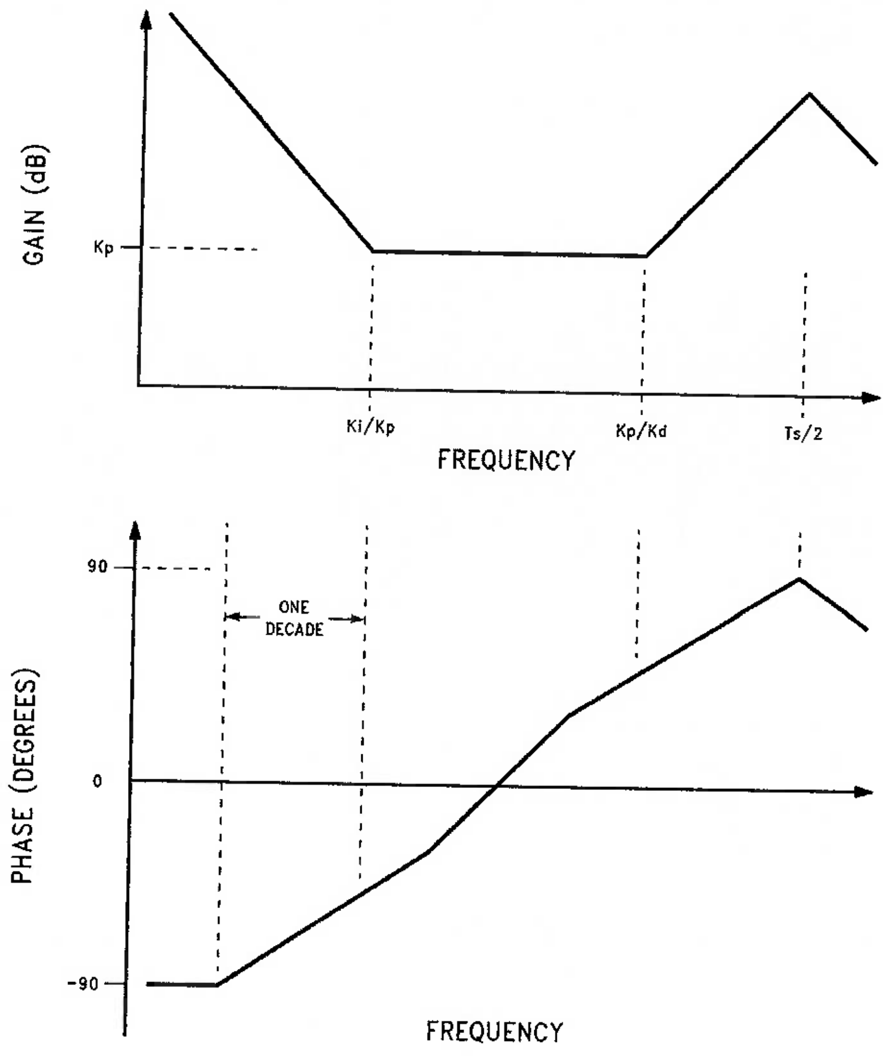


FIGURE 17. Bode Plots of PID Transfer Function

The Bode plots for this function (shown in *Figure 17*) show the effect of the individual terms of *Equation (1)*. The proportional term, K_p provides adjustment of proportional gain. The derivative term K_d increases the system bandwidth but more importantly adds leading phase shift to the control loop at high frequencies. This improves stability by counteracting the lagging phase shift introduced by other control loop components such as the motor. The integral term, K_i, provides a high DC gain which reduces static errors, but introduces a lagging phase shift at low frequencies. The relative magnitudes of K_d, K_i and loop proportional gain have to be adjusted to achieve optimum performance without introducing instability.

6.2 PID FILTER COEFFICIENT SCALING FACTORS FOR LM628/629

While the easiest way to determine the PID filter coefficient k_p, k_d, and k_i values is to use tuning as described in Section 2.11, some users may want to use a more theoretical approach to at least find initial starting values before fine tuning. As very often this analysis is performed in the continuous (s) domain and transformed into the discrete digital domain for implementation, the relationship between the continuous domain coefficients and the values input into LM628/629 is of interest.

6.0 Theory (Continued)

6.2.1 PID Filter Difference Equation

In the discrete domain, *Equation (1)* becomes the difference equation:

$$u(n) = K_p e(n) + K_i T \sum_{n=0}^{N} e(n) + K_d/T_s[e(n) - e(n-1)] \quad (2)$$

Where:

T is the sample interval $2048/f_{CLK}$

T_s is the derivative sample interval $(2048/f_{CLK} \times (1..255)$

6.2.2 Difference Equation with LM628/629 Coefficients

In terms of LM628/629 coefficients, *Equation (2)* becomes:

$$u(n) = k_p e(n) + k_i \sum_{n=0}^{N} e(n) + k_d[e(n') - e(n' - 0)] \quad (3)$$

Where:

k_p, k_i and k_d are the discrete-time LM628/629 coefficients

e(n) is the position error at sample time n

n' indicates sampling at the derivative sampling rate.

The error signal e(n) [or e(n')] is a 16-bit number from the output of the summing junction and is the input to the PID filter. The 15-bit filter coefficients are respectively multiplied by the 16-bit error terms as shown in *Equation (3)*A to produce 32-bit products.

6.2.3 LM628/629 PID Filter Output

The proportional coefficient k_p is multiplied by the error signal directly. The error signal is continually summed at the sample rate to previously accumulated errors to form the integral signal and is maintained to 24 bits. To achieve a more usable range from this term, only the most significant 16 bits are used and multiplied by the integral coefficient, k_i. The absolute value of this product is compared with the integration limit, il, and the smallest value, appropriately signed, is used. To form the derivative signal, the previous error is subtracted from the current error over the derivative sampling interval. This is multiplied by the derivative coefficient k_d and the product contributes every sample interval to the output independently of the user chosen derivative sample interval.

The least significant 16 bits of the 32-bit products from the three terms are added together to produce the resulting u(n) of *Equation (3)* each sample interval. From the PID filter 16-bit result, either the most significant 8 or 12 bits are output, depending on the output word size being used. A consequence of this and the use of the 16 MSB's of the integral signal is a scaling of the filter coefficients in relation to the continuous domain coefficients.

6.2.4 Scaling for k_p and k_d

Figure 18 gives details of the multiplication and output for k_p and k_d. Taking the output from the MS byte of the LS 16 bits of the 32-bit result register causes an effective 8-bit right-shift or division of 256 associated with k_p and k_d as follows:

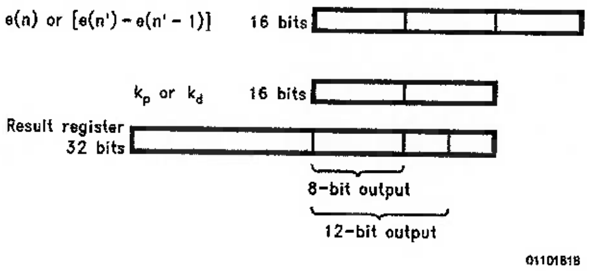


FIGURE 18. Scaling of k_p and k_d

Result $= k_p \times e(n)/256 = K_p \times e(n)$ ∴ $k_p = 256 \times K_p$.

Similarly for k_d:

Result $= (k_d \times [e(n') - e(n'-1)])/256$

$= K_d/T_s \times e(n)$ ∴ $k_d = 256 \times K_d/T_s$

Where T_s is the derivative sampling rate.

6.2.5 Scaling for k_i

Figure 19 shows the multiplication and output for the integral term k_i. The use of a 24-bit register for the error terms summation gives further scaling:

Result $= k_i/256 \times > e(n)/256$

$= K_i \times T$ ∴ $k_i = 65536\ K_i \times T$.

Where T is the sampling interval $2048/f_{CLK}$.

For a 12-bit output the factors are:

$k_p = 16 \times K_p$, $k_d = 16 \times K_d/T_s$ and $k_i = 4096\ K_i \times T$.

If the 32-bit result register overflows into the most significant 16-bits as a result of a calculation, then all the lower bits are set high to give a predictable saturated output.

6.3 AN EXAMPLE OF A TRAJECTORY CALCULATION

Problem: Determine the trajectory parameters for a motor move of 500 revolutions in 1 minute with 15 seconds of acceleration and deceleration respectively. Assume the optical incremental encoder used has 500 lines.

The LM628/629 quadrature decoder gives four counts for each encoder line giving 2000 counts per revolution in this example. The total number of counts for this position move is 2000 x 500 = 1,000,000 counts.

By definition, average velocity during the acceleration and deceleration periods, from and to zero, is half the maximum

6.0 Theory (Continued)

velocity. In this example, half the total time to make the move (30 seconds) is taken by acceleration and deceleration. Thus in terms of time, half the move is made at maximum velocity and half the move at an average velocity of half this maximum. Therefore, the combined distance traveled during acceleration and deceleration is half that during maximum velocity or ⅓ of the total, or 333,333 counts. Acceleration and deceleration takes 166,667 counts respectively.

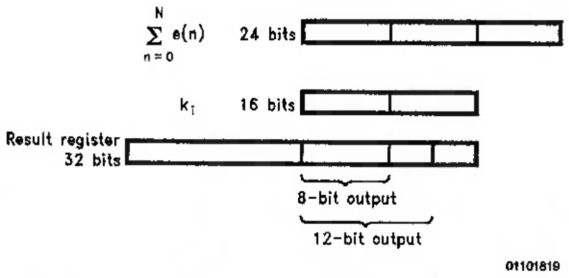


FIGURE 19. Scaling for k_I

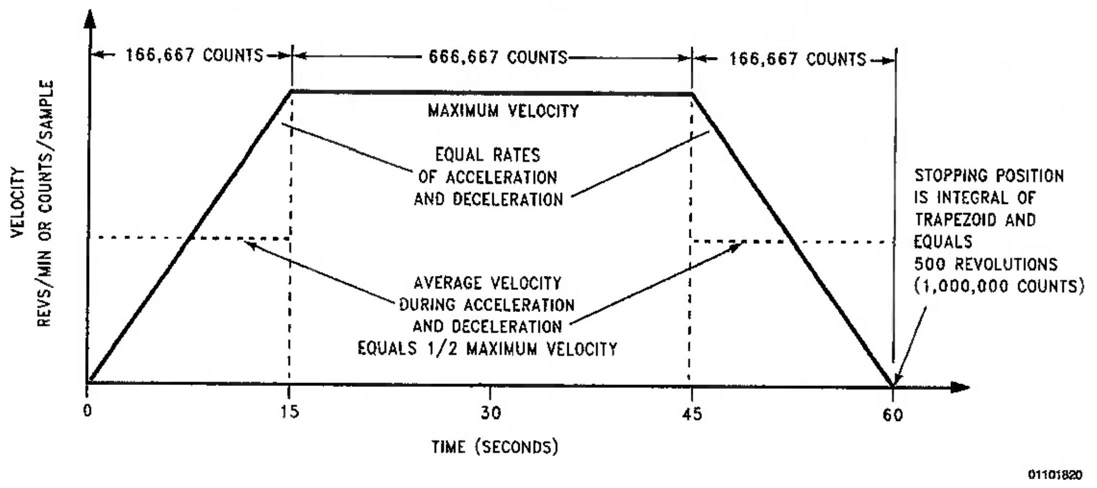


FIGURE 20. Trajectory Calculation Example Profile

The time interval used by the LM628/629 is the sample interval which is 256 µs for a f_{CLK} of 8 MHz.

The number of sample periods in 15 seconds = 15s/ 256 µs = 58,600 samples

Remembering that distance s = $at^2/2$ is traveled due to acceleration 'a' and time 't'.

Therefore acceleration a = $2S/t^2$

= 2 x 166,667/58,600

= 97.1 x 10^{-6} counts/sample ^{2}

Acceleration and velocity values are entered into LM628/629 as a 32-bit integer double-word but represents a 16-bit integer plus 16-bit fractional value. To achieve this acceleration and velocity decimal values are scaled by 65536 and any remaining fractions discarded. This value is then converted to hex to enter into LM628 in four bytes.

Scaled acceleration a = 97.1 x 10^{-6}x 65536

= 6.36 decimal = 00000006 hex.

The maximum velocity can be calculated in two ways, either by the distance in counts traveled at maximum velocity divided by the number of samples or by the acceleration multiplied by the number of samples over acceleration duration, as follows:

Velocity = 666,667/117,200 = 97.1 x 10^{-6} x 58,600

= 5.69 counts/sample

Scaled by 65536 becomes 372,899.8 decimal = 0005B0A3 hex.

Inputting these values for acceleration and velocity with the target position of 1,000,000 decimal, 000F4240 hex will achieve the desired velocity profile.

7.0 Questions and Answers

7.1 THE TWO MOST POPULAR QUESTIONS

Why doesn't the motor move, I've loaded filter parameters, trajectory parameters and issued Update Filter, UDF, and Start, STT, commands?

Answer: The most like cause is that a stop bit (one of bits 8, 9 or 10 of the trajectory control word) has been set in error,

7.0 Questions and Answers
(Continued)

supposedly to cause a stop in position mode. This is unnecessary, in position mode the trajectory stops automatically at the target position, see Section 3.3.

Can acceleration be changed on the fly?

Answer: No, not directly and a command error interrupt will be generated when STT is issued if acceleration has been changed. Acceleration can be changed if the motor is turned off first using bit 8 of the Load Trajectory Parameter, LTRJ, trajectory control word, see Section 4.6.1.

7.2 MORE ON ACCELERATION CHANGE

What happens at restart if acceleration is changed with the motor drive off and the motor is still moving?

Answer: The trajectory generation starting position is the actual position when the STT command is issued, but assumes that the motor is stationary. If the motor is moving the control loop will attempt to bring the motor back onto an accelerating profile, producing a large error value and less than predictable results. The LM628/629 was not designed with the intention to allow acceleration changes with moving motors.

Is there any way to change acceleration?

Answer: Acceleration change can be simulated by making many small changes of maximum velocity. For instance if a small velocity change is loaded, using LTRJ and STT commands, issuing these repeatedly at predetermined time intervals will cause the maximum velocity to increment producing a piecewise linear acceleration profile. The actual acceleration between velocity increments remains the same.

7.3 MORE ON STOP COMMANDS

What happens if the on-going trajectory is stopped by setting LTRJ control word bits 9 or 10, stop abruptly or stop smoothly, and then restarted by issuing Start, STT?

Answer: While stopped the motor position will be held by the control loop at the position determined as a result of issuing the stop command. Issuing STT will cause the motor to restart the trajectory toward the original target position with normal controlled acceleration.

What happens if the on-going trajectory is stopped by setting LTRJ control word bit 8, motor-off?

Answer: The LM628's DAC output is set to mid-scale, this puts zero volts on the motor which will still have a dynamic braking effect due to the commutation diodes. The LM629's PWM output sets the magnitude output to zero with a similar effect. If the motor freewheels or is moved the desired and actual positions will be the same. This can be verified using the RDDP and RDRP commands. When Start, STT, is issued the loop will be closed again and the motor will move toward the original trajectory from the actual current position.

If the motor is off, how can the control loop be closed and the motor energized?

Answer: Simply by issuing the Start, STT command. If any previous trajectory has completed then the motor will be held in the current position. If a trajectory was in progress when the motor-off command was issued then the motor will restart and move to the target position in position mode, or resume movement in velocity mode.

7.4 MORE ON DEFINE HOME

What happens if the Define Home command, DFH, is issued while a current trajectory is in progress?

Answer: The position where the DFH command is issued is reset to zero, but the motor still stops at the original position commanded, i.e., the position where DFH is issued is substracted from the original target position.

Does issuing Define Home, DFH, zero both the trajectory and position register?

Answer: Yes, use Read Real Position, RDRP, and Read Desired Position, RDDP to verify.

7.5 MORE ON VELOCITY

Why is a command error interrupt generated when inputting negative values of relative velocity?

Answer: Because the negative relative velocity would cause a negative absolute velocity which is not allowed. Negative absolute values of velocity imply movement in the negative direction which can be achieved by inputting a negative position value or in velocity mode by not setting bit 12. Similarly negative values of acceleration imply deceleration which occurs automatically at the acceleration rate when the LM628/629 stops the motor in position mode or if making a transition from a higher to a lower value of velocity.

What happens in velocity (or position) mode when the position range is exceeded?

Answer: The position range extends from maximum negative position hex'C0000000' to maximum positive position hex'3FFFFFFF' using a 32-bit double word. Bit 31 is the direction bit, logic 0 indicates forward direction, bit 30 is the wraparound bit used to control position over-range in velocity (or position) mode.

When the position increases past hex'3FFFFFFF' the wraparound bit 30 is set, which also sets the wraparound bit in the Status byte bit 4. This can be polled by the host or optionally used to interrupt the host as defined by the MSKI commands. Essentially the host has to manage wraparound by noting its occurrence and resetting the Status byte wraparound bit using the RSTI command. When the wraparound bit 30 is set in the position register so is the direction bit. This means one count past maximum positive position hex'3FFFFFFF' moves the position register onto the maximum negative position hex'C0000000'. Continued increase in positive direction causes the position register to count up to zero and back to positive values of position and on toward another wraparound.

Similarly when traveling in a negative direction, using two's complement arithmetic, position counts range from hex'FFFFFFFF' (−1 decimal) to the maximum negative position of hex'C0000000'. One more negative count causes the position register to change to hex'3FFFFFFF', the maximum positive position. This time the wraparound bit 30 is reset, causing the wraparound bit 4 of the status byte to be set. Also the direction bit 31 is reset to zero. Further counts in the negative direction cause the position register to count down to zero as would be expected. With management there is no reason why absolute position should be lost, even when changing between velocity and position modes.

7.0 Questions and Answers

(Continued)

7.6 MORE ON USE OF COMMANDS

If filter parameter and trajectory commands are pipelined for synchronization of axes, can the Update Filter, UDF, and Start, STT, commands be issued consecutively?

Answer: Yes.

Can commands be issued between another command and its data?

Answer: No.

What is the response time of the set breakpoint commands, SBPA and SBPR?

Answer: There is an uncertainty of one sample interval in the setting of the breakpoint bit 6 in the Status Byte in response to these commands.

What happens when the Set Index Position, SIP, command is issued?

Answer: On the next occurrence of all three inputs from the position encoder being low the corresponding position is loaded into the index register. This can be read with the Read Index Position command, RDIP. Bit 0 of the Read Signals register, shows when an SIP command has been issued but the index position has not yet been acquired. RDSIGS command accesses the Read Signals Register.

What happens if the motor is not able to keep up with the specified trajectory acceleration and velocity values?

Answer: A large, saturated, position error will be generated, and the control loop will be non-linear. The acceleration and velocity values should be set within the capability of the motor. Read Desired and Real Position commands, RDDP and RDRP can be used to determine the size of the error. The Load Position Error commands, for either host Interrupt or motor Stopping, LPEI and LPES, can be used to monitor the error size for controlled action where safety is a factor.

When is the command error bit 1 in the Status Byte set?

Answer:

1. When an acceleration change is attempted when the motor is moving and the drive on.
2. When loading a relative velocity would cause a negative absolute velocity.
3. Incorrect reading and writing operations generally.

What does the trajectory complete bit 2 in the Status Byte indicate?

Answer: That the trajectory loaded by LTRJ and initiated by STT has completed. The motor may or may not be at this position. Bit 2 is also set when the motor stop commands are executed and completed.

What do the specified minimum and maximum values of velocity mean in reality?

Answer: Assume a 500 line encoder = 1/2000 revs/count is used.

The maximum LM628/629 velocity is 16383 counts/sample and for a 8 MHz clock the LM628/629 sample rate is 3.9k samples/second, multiplying these values gives 32k revs/second or 1.92M rpm.

The maximum encoder rate is 1M counts/second multiplied by 1/2000 revs/count gives 500 revs/second or 30k rpm. The encoder capture rate therefore sets the maximum velocity limit.

The minimum LM628/629 velocity is 1/65536 counts/sample (one fractional count), multiplying this value by the sample rate and encoder revs/count gives 30 x 10^{-6} revs/second or 1.8 x 10^{-3} rpm.

The LM628 provides no limitation to practical values of velocity.

How long will it take to get to position wraparound in velocity mode traveling at 5000 rpm with a 500 line encoder?

Answer: 107 minutes.

8.0 References and Further Reading

1. LM628/LM629 Precision Motion Controller. Data sheet March 1989.
2. Automatic Control Systems. Benjamin C. Kuo. Fifth edition Prentice-Hall 1987.
3. DC Motors, Speed Controls, Servo Systems. Robbins & Myers/Electro Craft.
4. PID Algorithms and their Computer Implementation. D.W. Clarke. Institute of Measurement and Control, Trans. v. 6 No. 6 Oct/Dec 1984 86/178.
5. LM628 Programming Guide. Steven Hunt. National Semiconductor Application Note AN-693.

Notes

National Semiconductor Corporation
Americas
Email: support@nsc.com

www.national.com

National Semiconductor Europe
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: ap.support@nsc.com

National Semiconductor Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507

APÊNDICE D – DATASHEET IR2104 – GATE DRIVE

International
IOR Rectifier

Data Sheet No. PD60046-Q

IR2104(S)

HALF-BRIDGE DRIVER

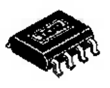
Features

- Floating channel designed for bootstrap operation
  Fully operational to +600V
  Tolerant to negative transient voltage
  dV/dt immune
- Gate drive supply range from 10 to 20V
- Undervoltage lockout
- 3.3V, 5V and 15V input logic compatible
- Cross-conduction prevention logic
- Internally set deadtime
- High side output in phase with input
- Shut down input turns off both channels
- Matched propagation delay for both channels

Product Summary

V_{OFFSET}	600V max.
I_O+/-	130 mA / 270 mA
V_{OUT}	10 - 20V
$t_{on/off}$ (typ.)	680 & 150 ns
Deadtime (typ.)	520 ns

Packages

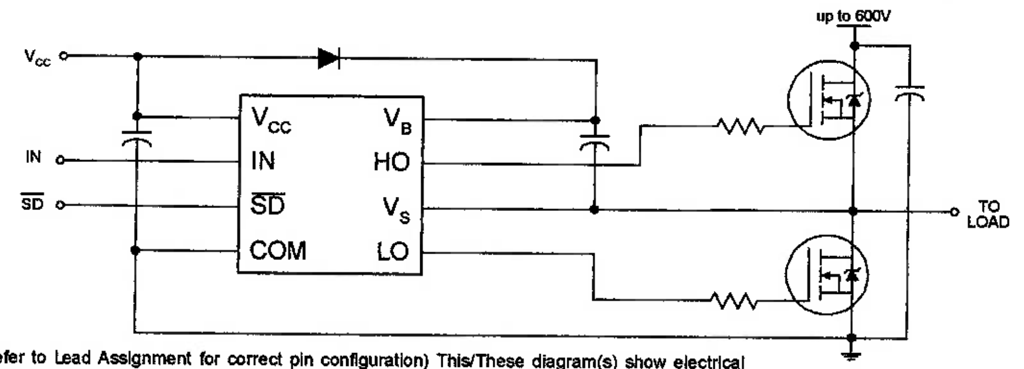

8 Lead SOIC
IR2104S

8 Lead PDIP
IR2104

Description

The IR2104(S) are high voltage, high speed power MOSFET and IGBT drivers with dependent high and low side referenced output channels. Proprietary HVIC and latch immune CMOS technologies enable ruggedized monolithic construction. The logic input is compatible with standard CMOS or LSTTL output, down to 3.3V logic. The output drivers feature a high pulse current buffer stage designed for minimum driver cross-conduction. The floating channel can be used to drive an N-channel power MOSFET or IGBT in the high side configuration which operates from 10 to 600 volts.

Typical Connection



(Refer to Lead Assignment for correct pin configuration) This/These diagram(s) show electrical connections only. Please refer to our Application Notes and DesignTips for proper circuit board layout.

Absolute Maximum Ratings

Absolute maximum ratings indicate sustained limits beyond which damage to the device may occur. All voltage parameters are absolute voltages referenced to COM. The thermal resistance and power dissipation ratings are measured under board mounted and still air conditions.

Symbol	Definition		Min.	Max.	Units
V_B	High side floating absolute voltage		-0.3	625	V
V_S	High side floating supply offset voltage		V_B - 25	V_B + 0.3	
V_{HO}	High side floating output voltage		V_S - 0.3	V_B + 0.3	
V_{CC}	Low side and logic fixed supply voltage		-0.3	25	
V_{LO}	Low side output voltage		-0.3	V_{CC} + 0.3	
V_{IN}	Logic input voltage (IN & $\overline{SD}$)		-0.3	V_{CC} + 0.3	
dV_s/dt	Allowable offset supply voltage transient		—	50	V/ns
P_D	Package power dissipation @ $T_A \le +25°C$	(8 lead PDIP)	—	1.0	W
		(8 lead SOIC)	—	0.625	
Rth_{JA}	Thermal resistance, junction to ambient	(8 lead PDIP)	—	125	°C/W
		(8 lead SOIC)	—	200	
T_J	Junction temperature		—	150	°C
T_S	Storage temperature		-55	150	
T_L	Lead temperature (soldering, 10 seconds)		—	300	

Recommended Operating Conditions

The Input/Output logic timing diagram is shown in Figure 1. For proper operation the device should be used within the recommended conditions. The V_S offset rating is tested with all supplies biased at 15V differential.

Symbol	Definition	Min.	Max.	Units
V_B	High side floating supply absolute voltage	V_S + 10	V_S + 20	V
V_S	High side floating supply offset voltage	Note 1	600	
V_{HO}	High side floating output voltage	V_S	V_B	
V_{CC}	Low side and logic fixed supply voltage	10	20	
V_{LO}	Low side output voltage	0	V_{CC}	
V_{IN}	Logic input voltage (IN & $\overline{SD}$)	0	V_{CC}	
T_A	Ambient temperature	-40	125	°C

Note 1: Logic operational for V_S of -5 to +600V. Logic state held for V_S of -5V to $-V_{BS}$. (Please refer to the Design Tip DT97-3 for more details).

Dynamic Electrical Characteristics

V_{BIAS} (V_{CC}, V_{BS}) = 15V, C_L = 1000 pF and T_A = 25°C unless otherwise specified.

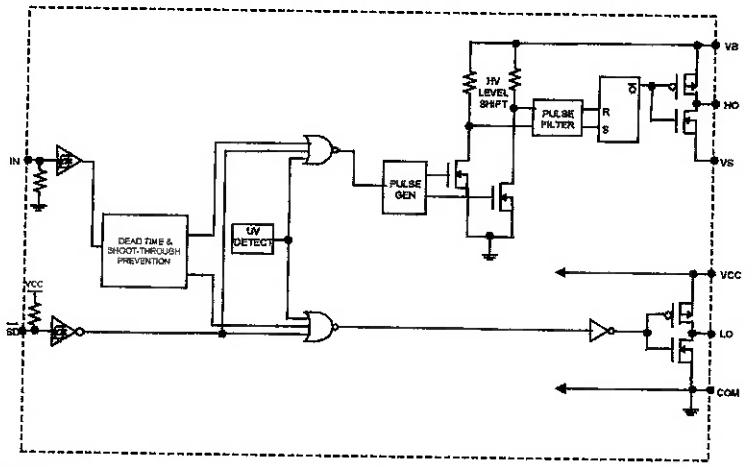
Symbol	Definition	Min.	Typ.	Max.	Units	Test Conditions
t_{on}	Turn-on propagation delay	—	680	820	ns	V_S = 0V
t_{off}	Turn-off propagation delay	—	150	220		V_S = 600V
t_{sd}	Shutdown propagation delay	—	160	220		
t_r	Turn-on rise time	—	100	170		
t_f	Turn-off fall time	—	50	90		
DT	Deadtime, LS turn-off to HS turn-on & HS turn-on to LS turn-off	400	520	650		
MT	Delay matching, HS & LS turn-on/off	—	—	60		

Static Electrical Characteristics

V_{BIAS} (V_{CC}, V_{BS}) = 15V and T_A = 25°C unless otherwise specified. The V_{IN}, V_{TH} and I_{IN} parameters are referenced to COM. The V_O and I_O parameters are referenced to COM and are applicable to the respective output leads: HO or LO.

Symbol	Definition	Min.	Typ.	Max.	Units	Test Conditions
V_{IH}	Logic "1" (HO) & Logic "0" (LO) input voltage	3	—	—	V	V_{CC} = 10V to 20V
V_{IL}	Logic "0" (HO) & Logic "1" (LO) input voltage	—	—	0.8		V_{CC} = 10V to 20V
$V_{SD,TH+}$	SD input positive going threshold	3	—	—		V_{CC} = 10V to 20V
$V_{SD,TH-}$	SD input negative going threshold	—	—	0.8		V_{CC} = 10V to 20V
V_{OH}	High level output voltage, V_{BIAS} - V_O	—	—	100	mV	I_O = 0A
V_{OL}	Low level output voltage, V_O	—	—	100		I_O = 0A
I_{LK}	Offset supply leakage current	—	—	50	µA	V_B = V_S = 600V
I_{QBS}	Quiescent V_{BS} supply current	—	30	55		V_{IN} = 0V or 5V
I_{QCC}	Quiescent V_{CC} supply current	—	150	270		V_{IN} = 0V or 5V
I_{IN+}	Logic "1" input bias current	—	3	10		V_{IN} = 5V
I_{IN-}	Logic "0" input bias current	—	—	1		V_{IN} = 0V
V_{CCUV+}	V_{CC} supply undervoltage positive going threshold	8	8.9	9.8	V	
V_{CCUV-}	V_{CC} supply undervoltage negative going threshold	7.4	8.2	9		
I_{O+}	Output high short circuit pulsed current	130	210	—	mA	V_O = 0V PW≤ 10 µs
I_{O-}	Output low short circuit pulsed current	270	360	—		V_O = 15V PW≤ 10 µs

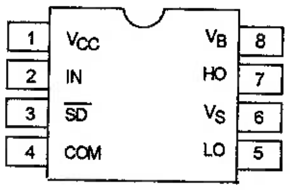
Functional Block Diagram



Lead Definitions

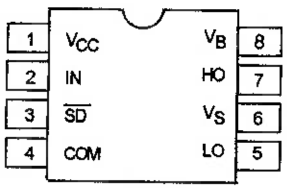
Symbol	Description
IN	Logic input for high and low side gate driver outputs (HO and LO), in phase with HO
$\overline{SD}$	Logic input for shutdown
V_B	High side floating supply
HO	High side gate drive output
V_S	High side floating supply return
V_{CC}	Low side and logic fixed supply
LO	Low side gate drive output
COM	Low side return

Lead Assignments



8 Lead PDIP

IR2104



8 Lead SOIC

IR2104S

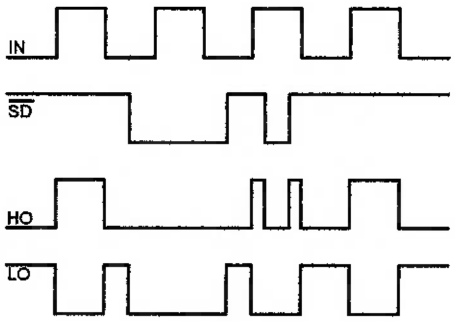


Figure 1. Input/Output Timing Diagram

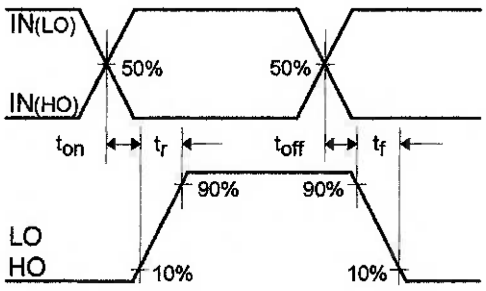


Figure 2. Switching Time Waveform Definitions

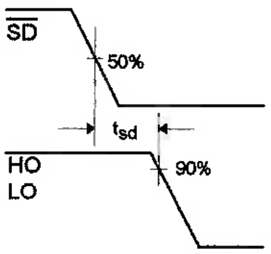


Figure 3. Shutdown Waveform Definitions

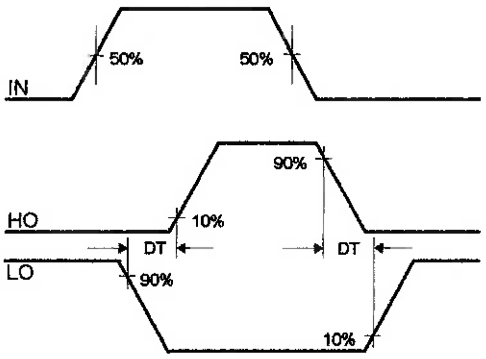


Figure 4. Deadtime Waveform Definitions

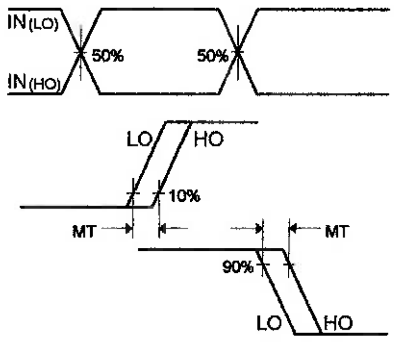


Figure 5. Delay Matching Waveform Definitions

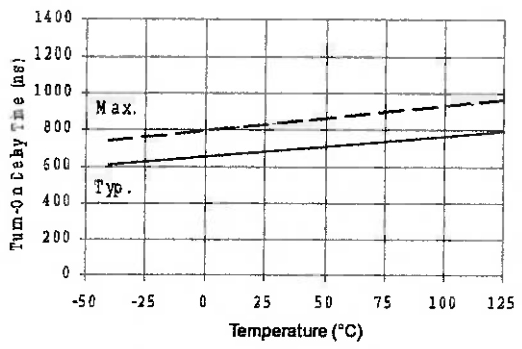


Figure 6A. Turn-On Time vs Temperature

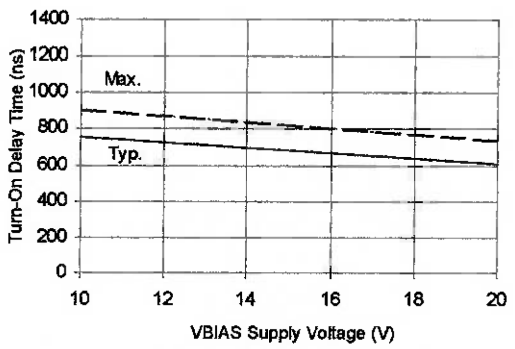


Figure 6B. Turn-On Time vs Supply Voltage

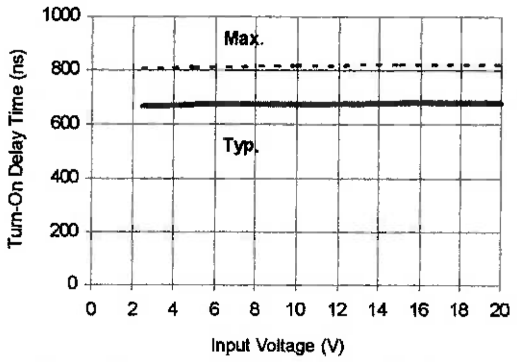


Figure 6C. Turn-On Time vs Input Voltage

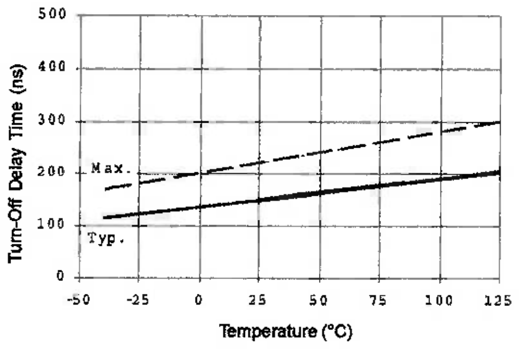


Figure 7A. Turn-Off Time vs Temperature

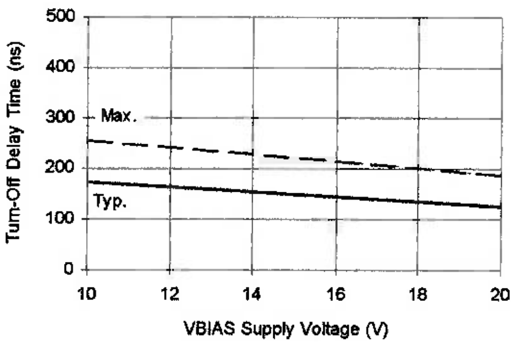


Figure 7B. Turn-Off Time vs Supply Voltage

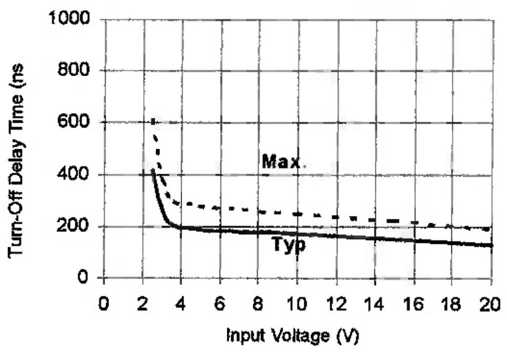


Figure 7C. Turn-Off Time vs Input Voltage

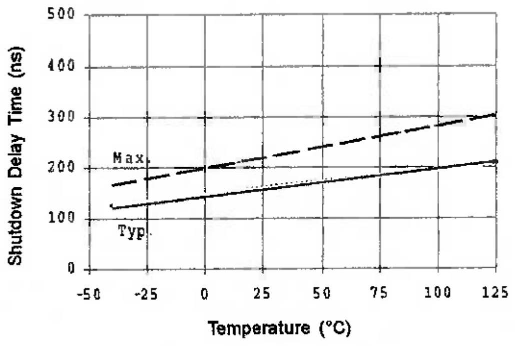


Figure 8A. Shutdown Time vs Temperature

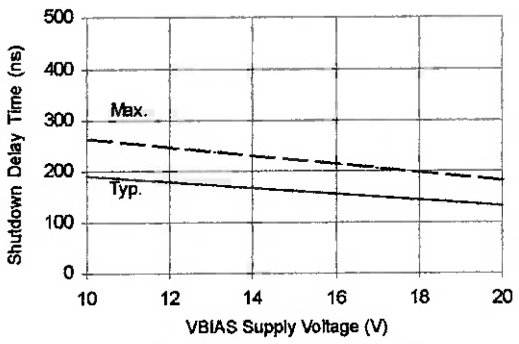


Figure 8B. Shutdown Time vs Voltage

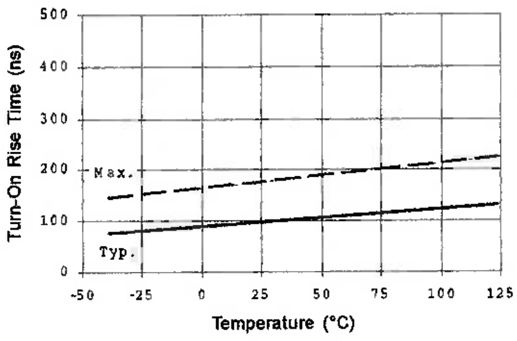


Figure 9A. Turn-On Rise Time vs Temperature

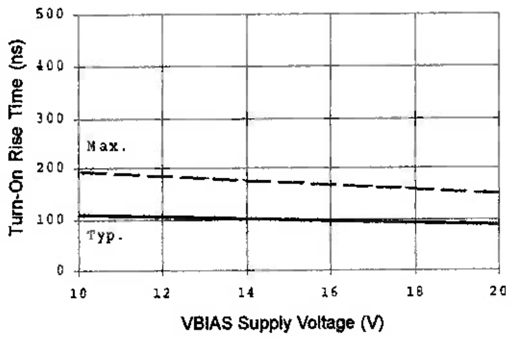


Figure 9B. Turn-On Rise Time vs Voltage

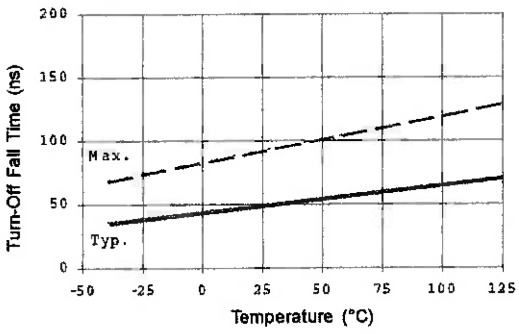


Figure 10A. Turn-Off Fall Time vs Temperature

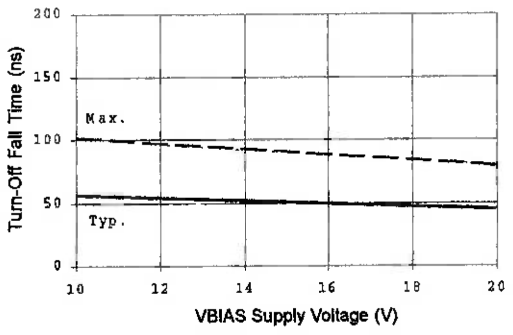


Figure 10B. Turn-Off Fall Time vs Voltage

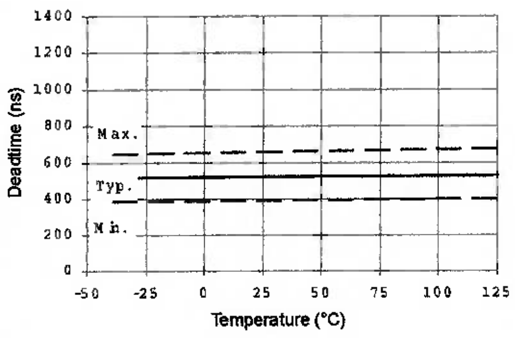


Figure 11A. Deadtime vs Temperature

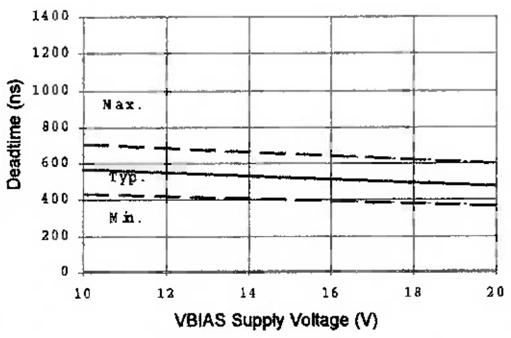


Figure 11B. Deadtime vs Voltage

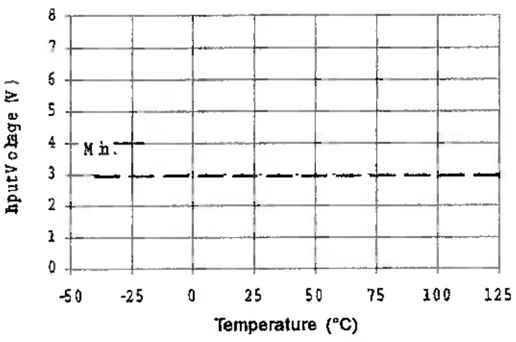


Figure 12A. Logic "1" (HO) & Logic "0" (LO) & Inactive SD Input Voltage vs Temperature

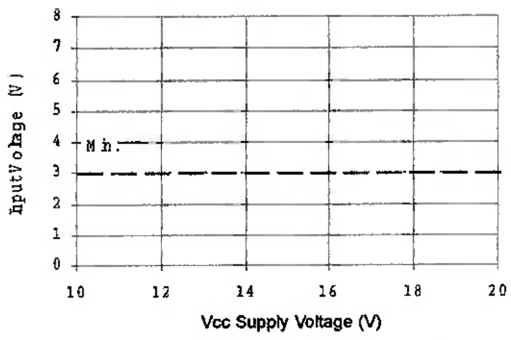


Figure 12B. Logic "1" (HO) & Logic "0" (LO) & Inactive SD Input Voltage vs Voltage

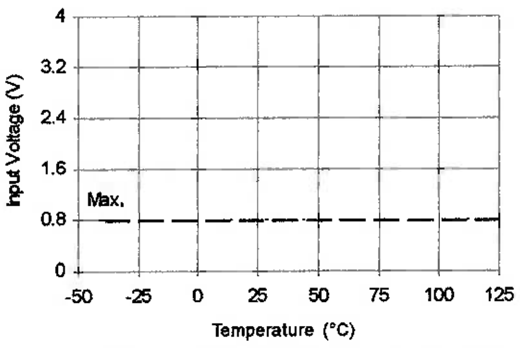


Figure 13A. Logic "0" (HO) & Logic "1" (LO) & Active SD Input Voltage vs Temperature

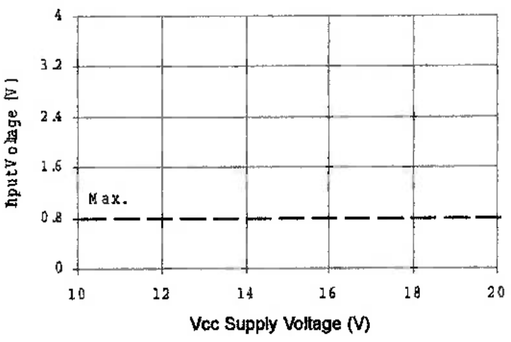


Figure 13B. Logic "0" (HO) & Logic "1" (LO) & Active SD Input Voltage vs Voltage

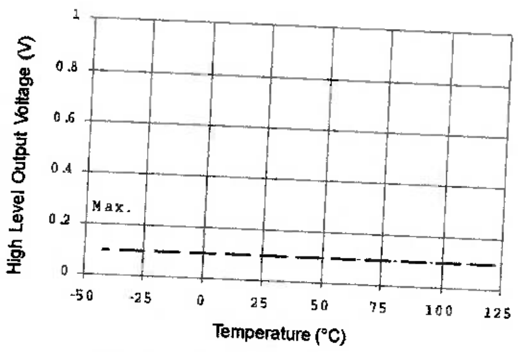


Figure 14A. High Level Output vs Temperature

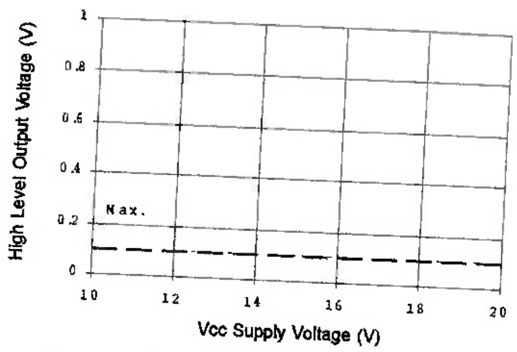


Figure 14B. High Level Output vs Voltage

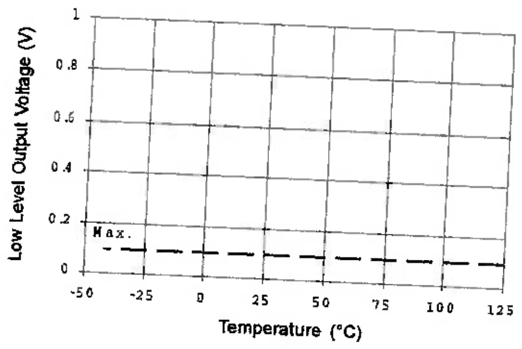


Figure 15A. Low Level Output vs Temperature

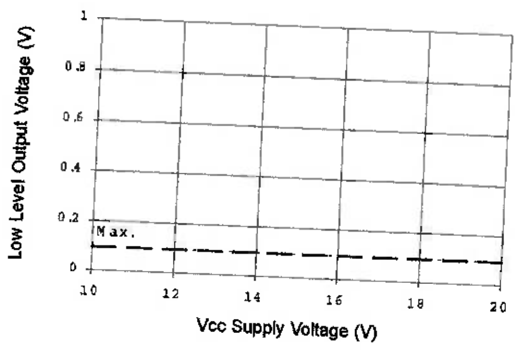


Figure 15B. Low level Output vs Voltage

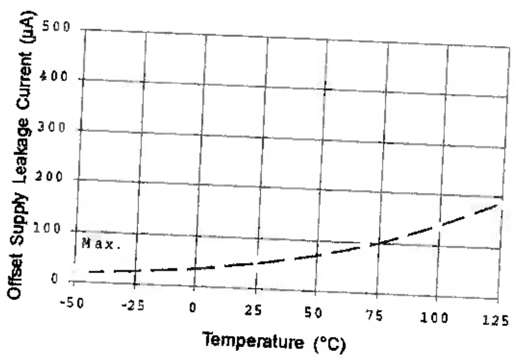


Figure 16A. Offset Supply Current vs Temperature

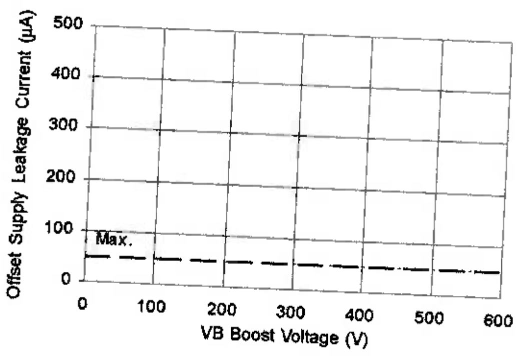


Figure 16B. Offset Supply Current vs Voltage

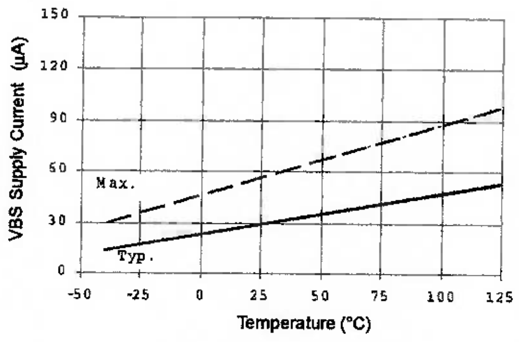


Figure 17A. VBS Supply Current vs Temperature

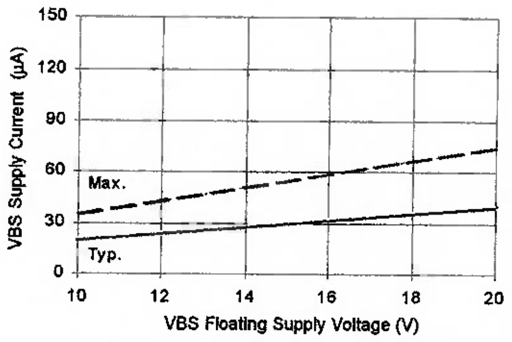


Figure 17B. VBS Supply Current vs Voltage

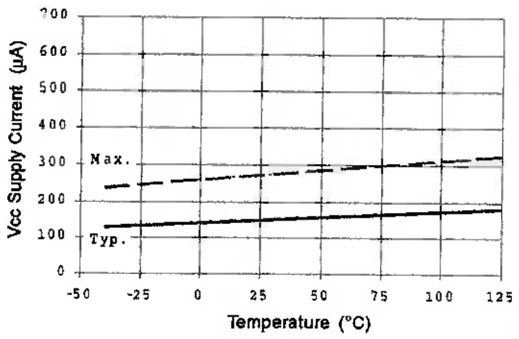


Figure 18A. Vcc Supply Current vs Temperature

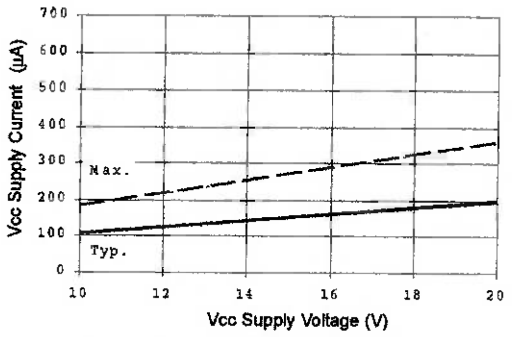


Figure 18B. Vcc Supply Current vs Voltage

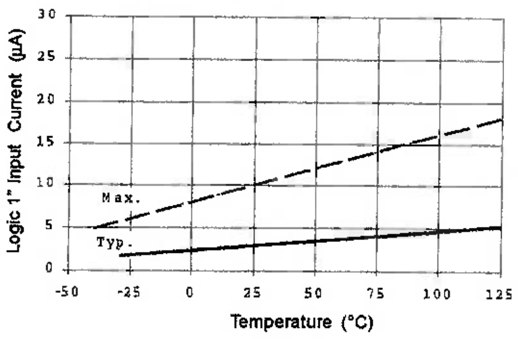


Figure 19A. Logic"1" Input Current vs Temperature

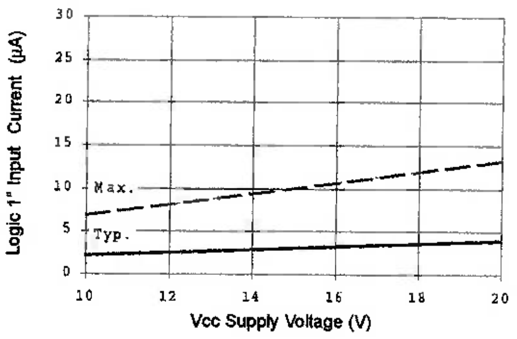


Figure 19B. Logic"1" Input Current vs Voltage

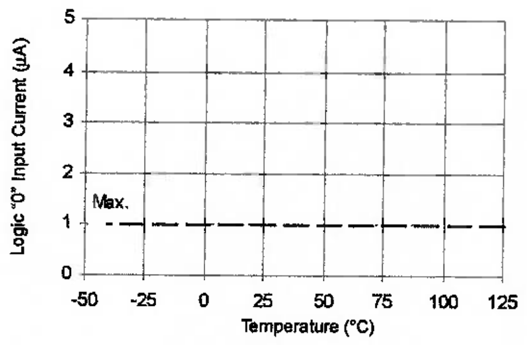


Figure 20A. Logic "0" Input Current vs Temperature

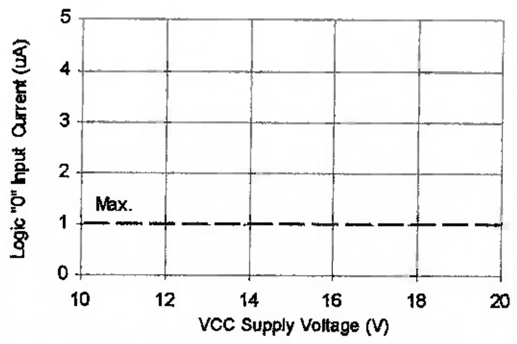


Figure 20B. Logic "0" Input Current vs Voltage

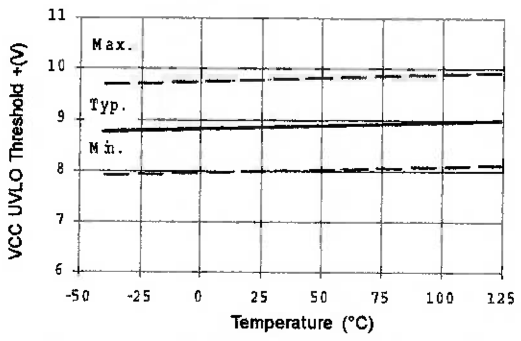


Figure 21A. Vcc Undervoltage Threshold(+) vs Temperature

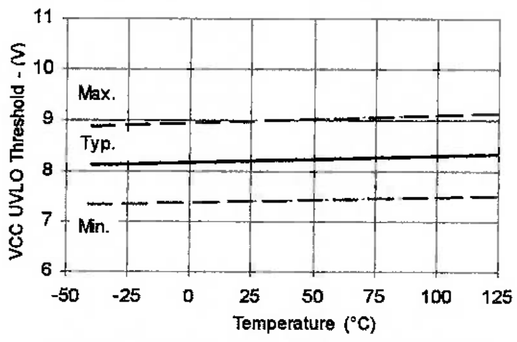


Figure 21B. Vcc Undervoltage Threshold(-) vs Temperature

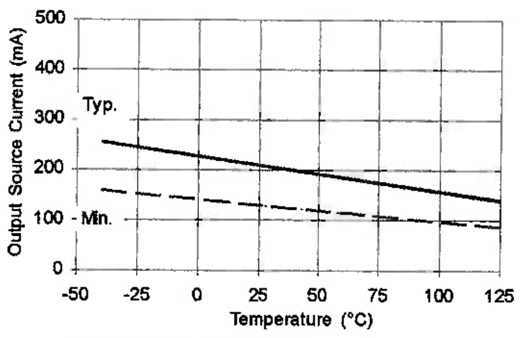


Figure 22A. Output Source Current vs Temperature

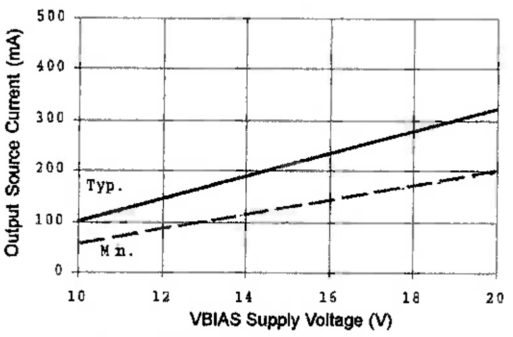


Figure 22B. Output Source Current vs Voltage

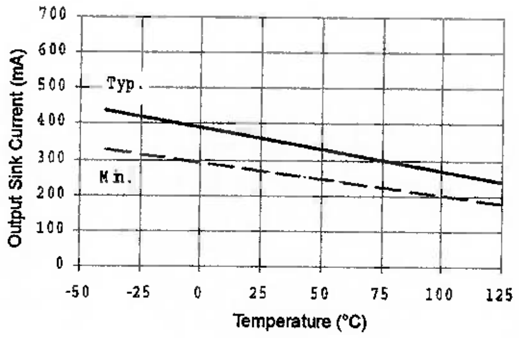


Figure 23A. Output Sink Current vs Temperature

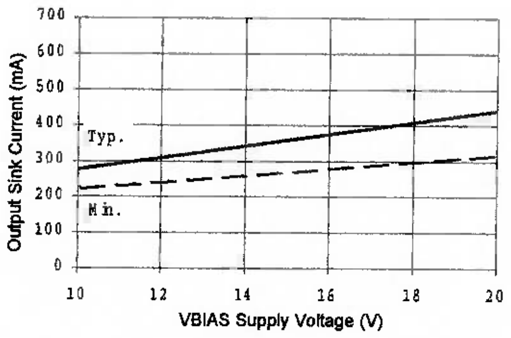
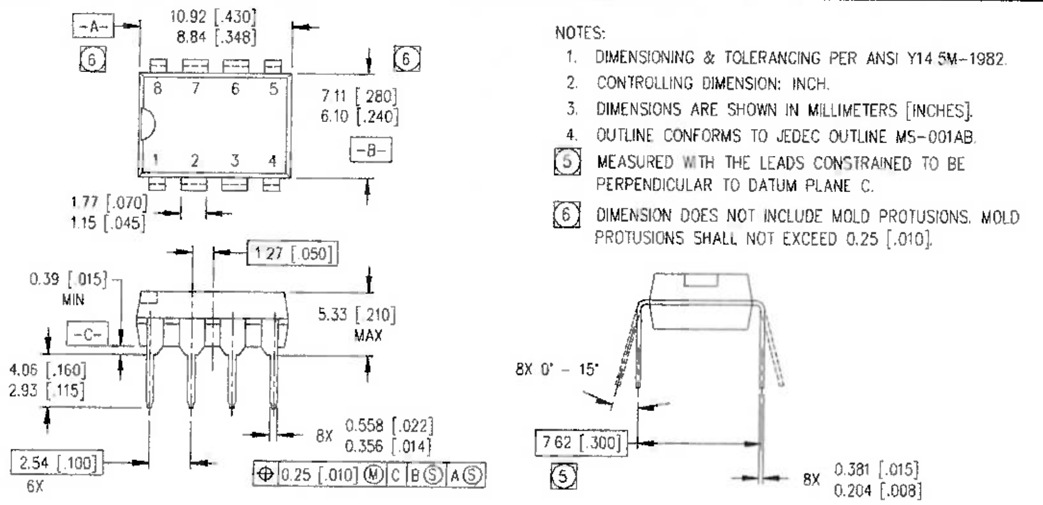


Figure 23B. Output Sink Current vs Voltage

Case Outlines

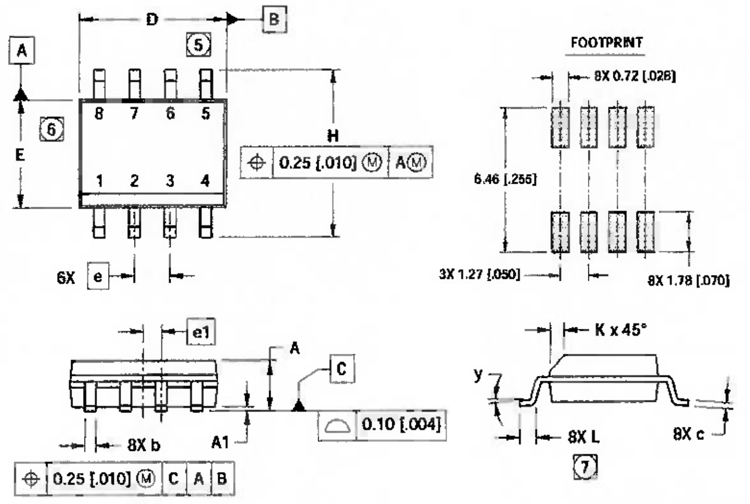


NOTES:
1. DIMENSIONING & TOLERANCING PER ANSI Y14.5M-1982.
2. CONTROLLING DIMENSION: INCH.
3. DIMENSIONS ARE SHOWN IN MILLIMETERS [INCHES].
4. OUTLINE CONFORMS TO JEDEC OUTLINE MS-001AB.
5. MEASURED WITH THE LEADS CONSTRAINED TO BE PERPENDICULAR TO DATUM PLANE C.
6. DIMENSION DOES NOT INCLUDE MOLD PROTUSIONS. MOLD PROTUSIONS SHALL NOT EXCEED 0.25 [.010].

8 Lead PDIP

01-3003 01

01-6014
(MS-001AB)



DIM	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	.0532	.0688	1.35	1.75
A1	.0040	.0098	0.10	0.25
b	.013	.020	0.33	0.51
c	.0075	.0098	0.19	0.25
D	.189	.1968	4.80	5.00
E	.1497	.1574	3.80	4.00
e	.050 BASIC		1.27 BASIC	
e 1	.025 BASIC		0.635 BASIC	
H	.2284	.2440	5.80	6.20
K	.0099	.0196	0.25	0.50
L	.016	.050	0.40	1.27
y	0°	8°	0°	8°

NOTES:

1. DIMENSIONING & TOLERANCING PER ASME Y14.5M-1994.
2. CONTROLLING DIMENSION: MILLIMETER
3. DIMENSIONS ARE SHOWN IN MILLIMETERS [INCHES].
4. OUTLINE CONFORMS TO JEDEC OUTLINE MS-012AA.

$\boxed{5}$ DIMENSION DOES NOT INCLUDE MOLD PROTRUSIONS. MOLD PROTRUSIONS NOT TO EXCEED 0.15 [.006].

$\boxed{6}$ DIMENSION DOES NOT INCLUDE MOLD PROTRUSIONS. MOLD PROTRUSIONS NOT TO EXCEED 0.25 [.010].

$\boxed{7}$ DIMENSION IS THE LENGTH OF LEAD FOR SOLDERING TO A SUBSTRATE.

8 Lead SOIC

01-6027
01-0021 11 (MS-012AA)

International
IOR Rectifier

IR WORLD HEADQUARTERS: 233 Kansas St., El Segundo, California 90245 Tel: (310) 252-7105

Data and specifications subject to change without notice. 6/30/2003

APÊNDICE E – DATASHEET IRF840 – MOSFET

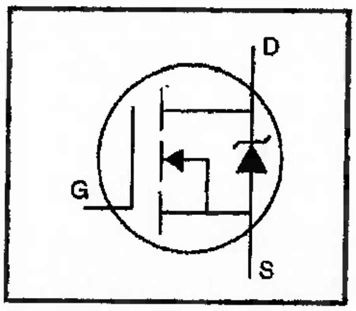
International
IOR Rectifier

PD-9.376H

IRF840

IEXFET® Power MOSFET

- Dynamic dv/dt Rating
- Repetitive Avalanche Rated
- Fast Switching
- Ease of Paralleling
- Simple Drive Requirements



V_{DSS} = 500V

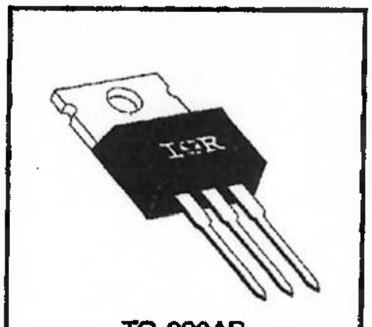
$R_{DS(on)}$ = 0.85Ω

I_D = 8.0A

escription

ıird Generation HEXFETs from International Rectifier provide the designer ith the best combination of fast switching, ruggedized device design, low ı-resistance and cost-effectiveness.

ıe TO-220 package is universally preferred for all commercial-industrial ıplications at power dissipation levels to approximately 50 watts. The low ermal resistance and low package cost of the TO-220 contribute to its wide ceptance throughout the industry.



TO-220AB

DATA SHEETS

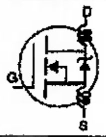
bsolute Maximum Ratings

	Parameter	Max.	Units
@ T_C = 25°C	Continuous Drain Current, V_{GS} @ 10 V	8.0	A
@ T_C = 100°C	Continuous Drain Current, V_{GS} @ 10 V	5.1	
M	Pulsed Drain Current ①	32	
D @ T_C = 25°C	Power Dissipation	125	W
	Linear Derating Factor	1.0	W/°C
3S	Gate-to-Source Voltage	±20	V
AS	Single Pulse Avalanche Energy ②	510	mJ
R	Avalanche Current ①	8.0	A
AR	Repetitive Avalanche Energy ①	13	mJ
/dt	Peak Diode Recovery dv/dt ③	3.5	V/ns
STG	Operating Junction and Storage Temperature Range	-55 to +150	°C
	Soldering Temperature, for 10 seconds	300 (1.6mm from case)	
	Mounting Torque, 6-32 or M3 screw	10 lbf•in (1.1 N•m)	

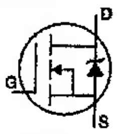
ermal Resistance

	Parameter	Min.	Typ.	Max.	Units
JC	Junction-to-Case	—	—	1.0	°C/W
CS	Case-to-Sink, Flat, Greased Surface	—	0.50	—	
JA	Junction-to-Ambient	—	—	62	

Electrical Characteristics @ T_J = 25°C (unless otherwise specified)

	Parameter	Min.	Typ.	Max.	Units	Test Conditions
$V_{(BR)DSS}$	Drain-to-Source Breakdown Voltage	500	—	—	V	V_{GS}=0V, I_D= 250μA
$\Delta V_{(BR)DSS}/\Delta T_J$	Breakdown Voltage Temp. Coefficient	—	0.78	—	V/°C	Reference to 25°C, I_D= 1mA
$R_{DS(on)}$	Static Drain-to-Source On-Resistance	—	—	0.85	Ω	V_{GS}=10V, I_D=4.8A ④
$V_{GS(th)}$	Gate Threshold Voltage	2.0	—	4.0	V	$V_{DS}=V_{GS}$, I_D= 250μA
g_{fs}	Forward Transconductance	4.9	—	—	S	V_{DS}=50V, I_D=4.8A ④
I_{DSS}	Drain-to-Source Leakage Current	—	—	25	μA	V_{DS}=500V, V_{GS}=0V
		—	—	250		V_{DS}=400V, V_{GS}=0V, T_J=125°C
I_{GSS}	Gate-to-Source Forward Leakage	—	—	100	nA	V_{GS}=20V
	Gate-to-Source Reverse Leakage	—	—	-100		V_{GS}=-20V
Q_g	Total Gate Charge	—	—	63	nC	I_D=8.0A
Q_{gs}	Gate-to-Source Charge	—	—	9.3		V_{DS}=400V
Q_{gd}	Gate-to-Drain ("Miller") Charge	—	—	32		V_{GS}=10V See Fig. 6 and 13 ④
$t_{d(on)}$	Turn-On Delay Time	—	14	—	ns	V_{DD}=250V
t_r	Rise Time	—	23	—		I_D=8.0A
$t_{d(off)}$	Turn-Off Delay Time	—	49	—		R_G=9.1Ω
t_f	Fall Time	—	20	—		R_D=31Ω See Figure 10 ④
L_D	Internal Drain Inductance	—	4.5	—	nH	Between lead, 6 mm (0.25in.) from package and center of die contact 
L_S	Internal Source Inductance	—	7.5	—		
C_{iss}	Input Capacitance	—	1300	—	pF	V_{GS}=0V
C_{oss}	Output Capacitance	—	310	—		V_{DS}=25V
C_{rss}	Reverse Transfer Capacitance	—	120	—		f=1.0MHz See Figure 5

Source-Drain Ratings and Characteristics

	Parameter	Min.	Typ.	Max.	Units	Test Conditions
I_S	Continuous Source Current (Body Diode)	—	—	8.0	A	MOSFET symbol showing the integral reverse p-n junction diode. 
I_{SM}	Pulsed Source Current (Body Diode) ①	—	—	32		
V_{SD}	Diode Forward Voltage	—	—	2.0	V	T_J=25°C, I_S=8.0A, V_{GS}=0V ④
t_{rr}	Reverse Recovery Time	—	460	970	ns	T_J=25°C, I_F=8.0A
Q_{rr}	Reverse Recovery Charge	—	4.2	8.9	μC	di/dt=100A/μs ④
t_{on}	Forward Turn-On Time	Intrinsic turn-on time is negligible (turn-on is dominated by L_S+L_D)				

Notes:

① Repetitive rating; pulse width limited by max. junction temperature (See Figure 11)

② V_{DD}=50V, starting T_J=25°C, L=14mH R_G=25Ω, I_{AS}=8.0A (See Figure 12)

③ I_{SD}≤8.0A, di/dt≤100A/μs, V_{DD}≤$V_{(BR)DSS}$, T_J≤150°C

④ Pulse width ≤ 300 μs; duty cycle ≤2%.

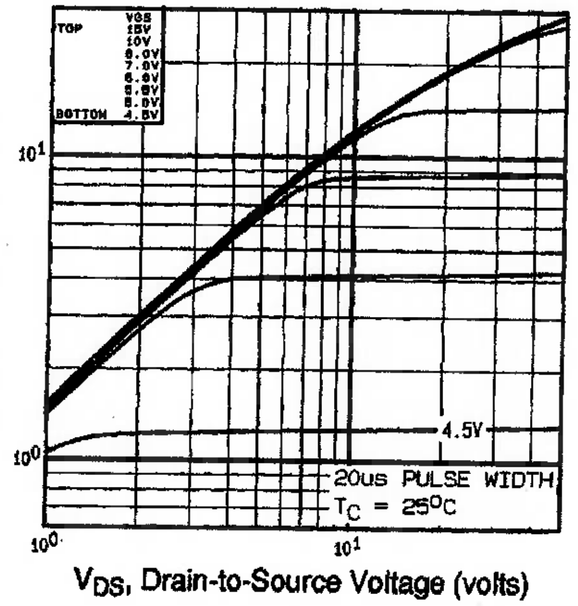


Fig 1. Typical Output Characteristics, T_C=25°C

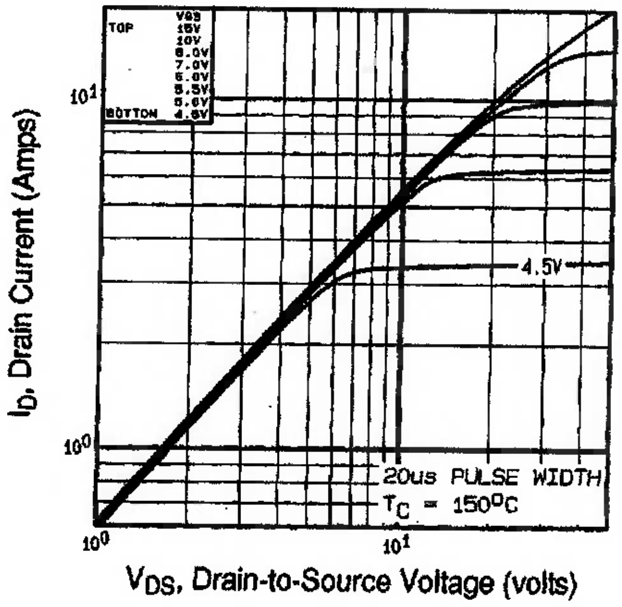


Fig 2. Typical Output Characteristics, T_C=150°C

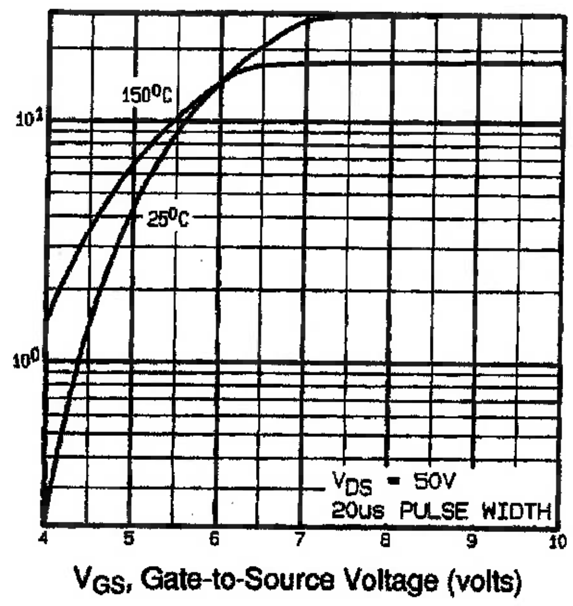


Fig 3. Typical Transfer Characteristics

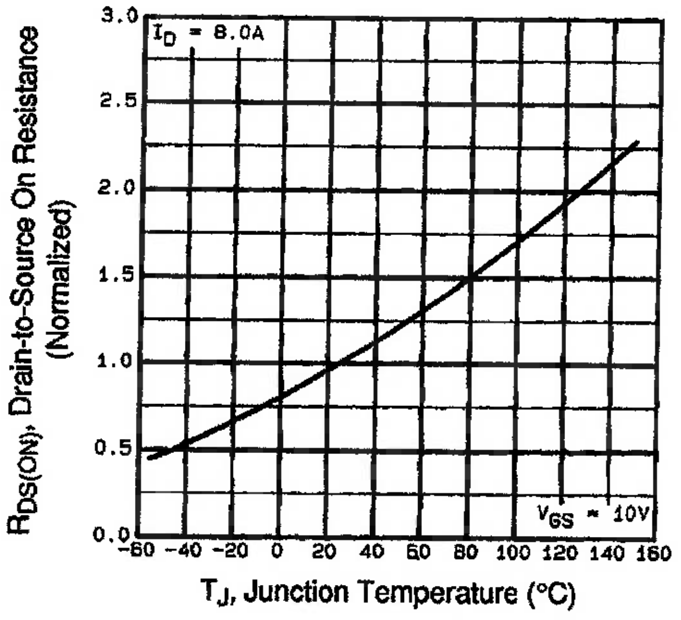


Fig 4. Normalized On-Resisance Vs. Temperature

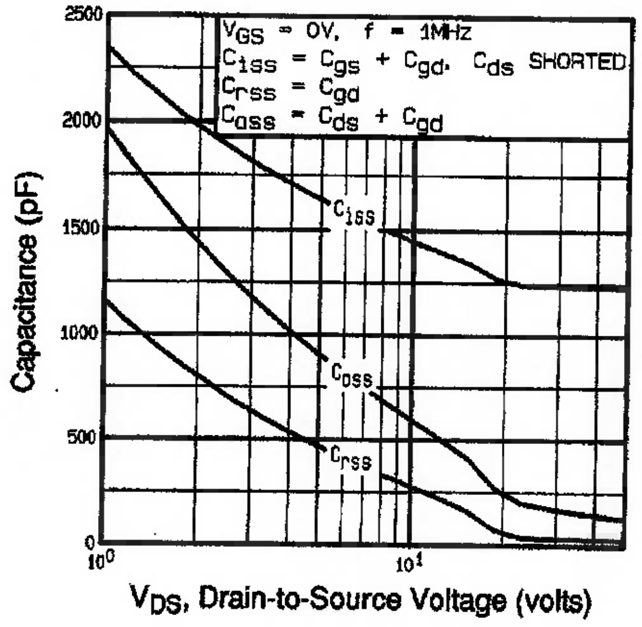


Fig 5. Typical Capacitance Vs. Drain-to-Source Voltage

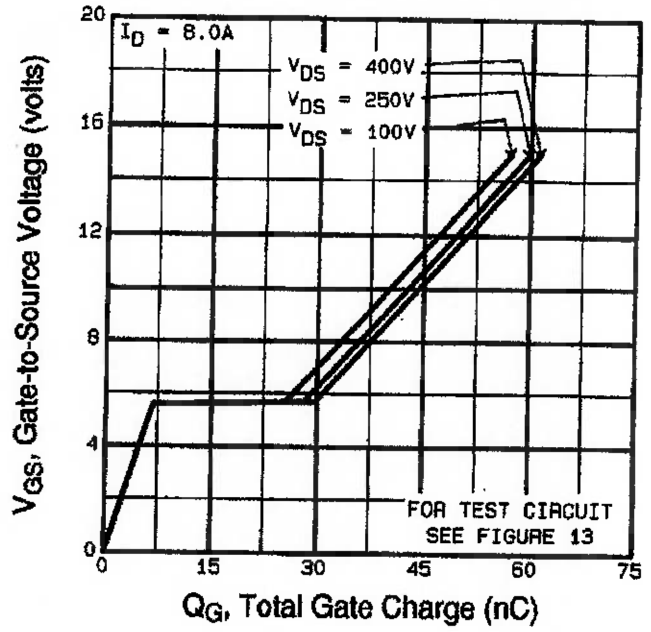


Fig 6. Typical Gate Charge Vs. Gate-to-Source Voltage

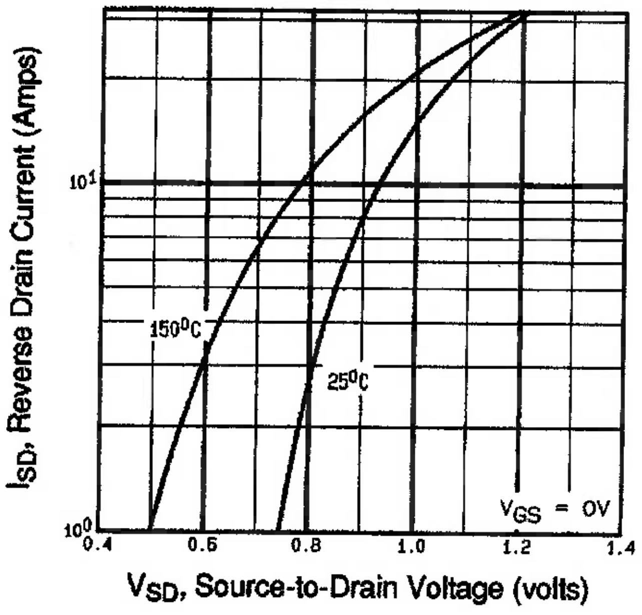


Fig 7. Typical Source-Drain Diode Forward Voltage

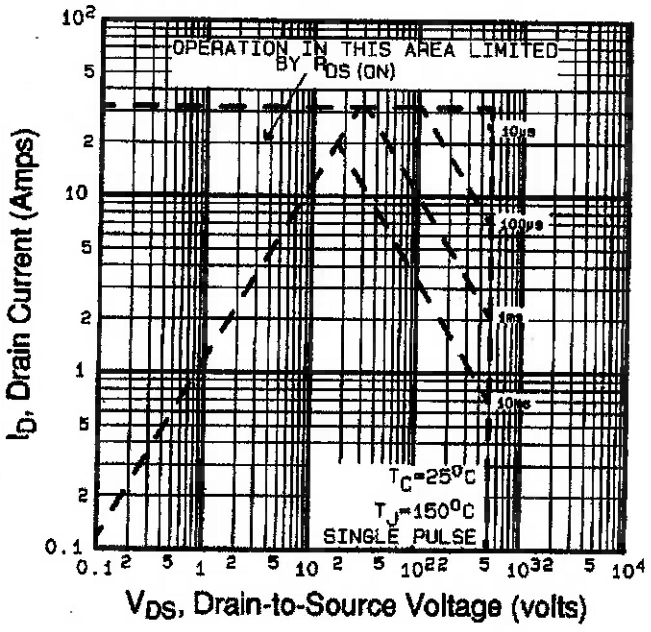


Fig 8. Maximum Safe Operating Area

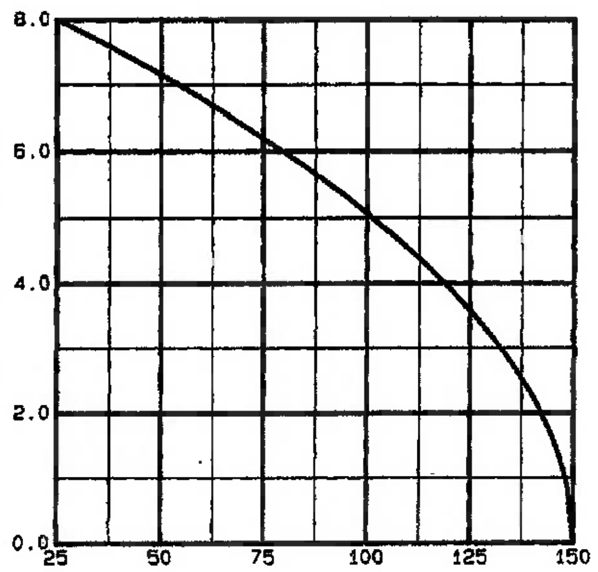


Fig 9. Maximum Drain Current Vs. Case Temperature

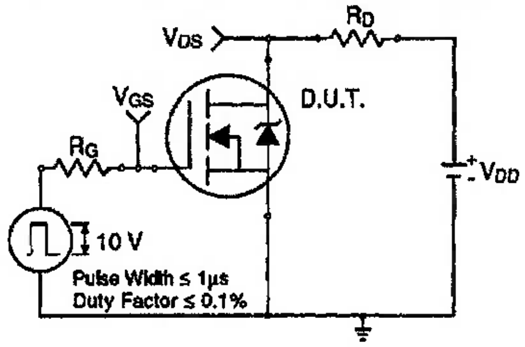


Fig 10a. Switching Time Test Circuit

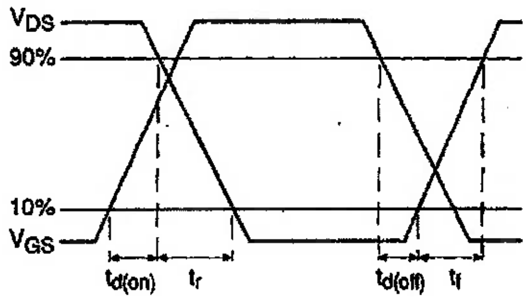


Fig 10b. Switching Time Waveforms

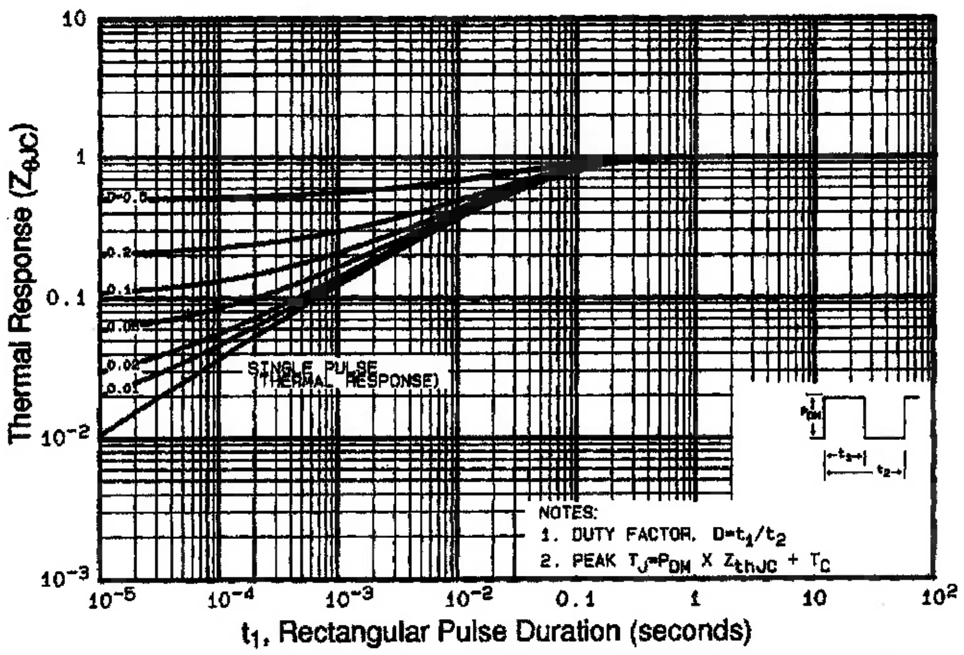


Fig 11. Maximum Effective Transient Thermal Impedance, Junction-to-Case

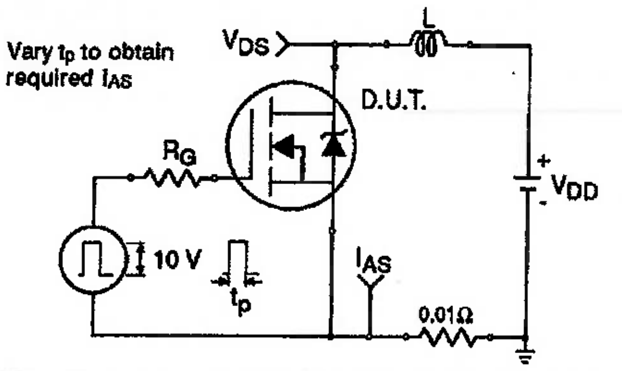


Fig 12a. Unclamped Inductive Test Circuit

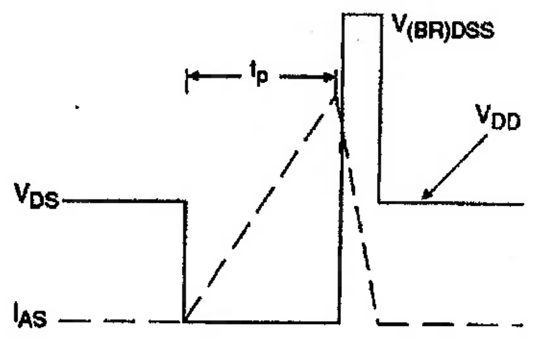


Fig 12b. Unclamped Inductive Waveforms

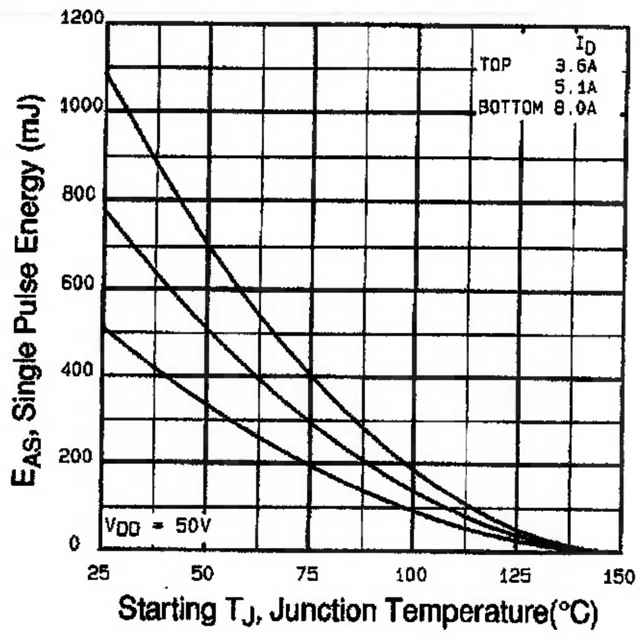


Fig 12c. Maximum Avalanche Energy Vs. Drain Current

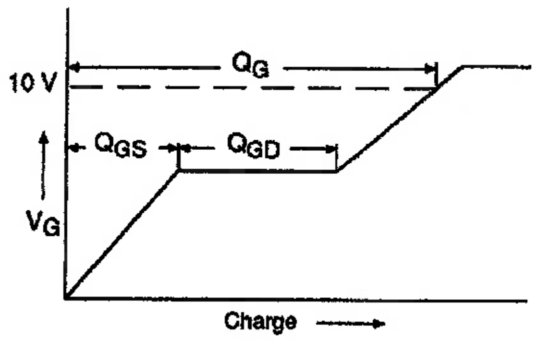


Fig 13a. Basic Gate Charge Waveform

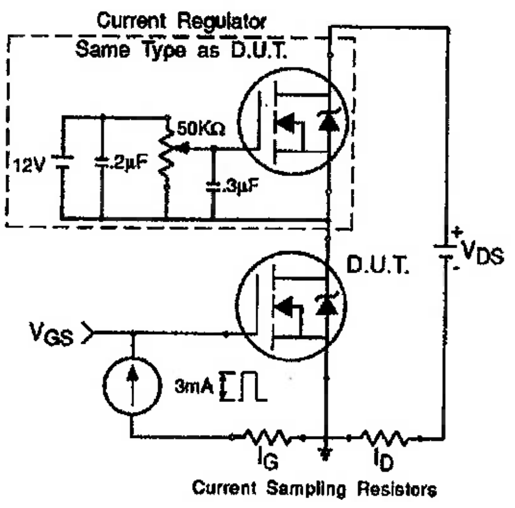


Fig 13b. Gate Charge Test Circuit

Appendix A: Figure 14, Peak Diode Recovery dv/dt Test Circuit – See page 1505

Appendix B: Package Outline Mechanical Drawing – See page 1509

Appendix C: Part Marking Information – See page 1516

Appendix E: Optional Leadforms – See page 1525

International IOR Rectifier

APÊNDICE F – DATASHEET PIC16F87X

MICROCHIP

PIC16F87XA Data Sheet

28/40/44-Pin Enhanced Flash Microcontrollers

MICROCHIP

PIC16F87XA

28/40/44-Pin Enhanced Flash Microcontrollers

Devices Included in this Data Sheet:

- PIC16F873A
- PIC16F874A
- PIC16F876A
- PIC16F877A

High-Performance RISC CPU:

- Only 35 single-word instructions to learn
- All single-cycle instructions except for program branches, which are two-cycle
- Operating speed: DC – 20 MHz clock input
  DC – 200 ns instruction cycle
- Up to 8K x 14 words of Flash Program Memory,
  Up to 368 x 8 bytes of Data Memory (RAM),
  Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to other 28-pin or 40/44-pin PIC16CXXX and PIC16FXXX microcontrollers

Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during Sleep via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I^{2}C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) – 8 bits wide with external $\overline{\text{RD}}$, $\overline{\text{WR}}$ and $\overline{\text{CS}}$ controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

Analog Features:

- 10-bit, up to 8-channel Analog-to-Digital Converter (A/D)
- Brown-out Reset (BOR)
- Analog Comparator module with:
  - Two analog comparators
  - Programmable on-chip voltage reference (VREF) module
  - Programmable input multiplexing from device inputs and internal voltage reference
  - Comparator outputs are externally accessible

Special Microcontroller Features:

- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Data EEPROM Retention > 40 years
- Self-reprogrammable under software control
- In-Circuit Serial Programming™ (ICSP™) via two pins
- Single-supply 5V In-Circuit Serial Programming
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving Sleep mode
- Selectable oscillator options
- In-Circuit Debug (ICD) via two pins

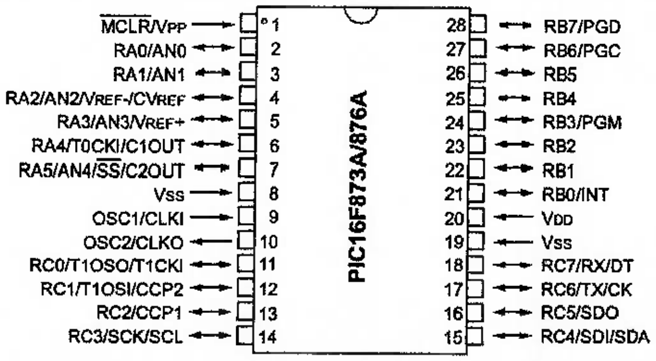
CMOS Technology:

- Low-power, high-speed Flash/EEPROM technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Commercial and Industrial temperature ranges
- Low-power consumption

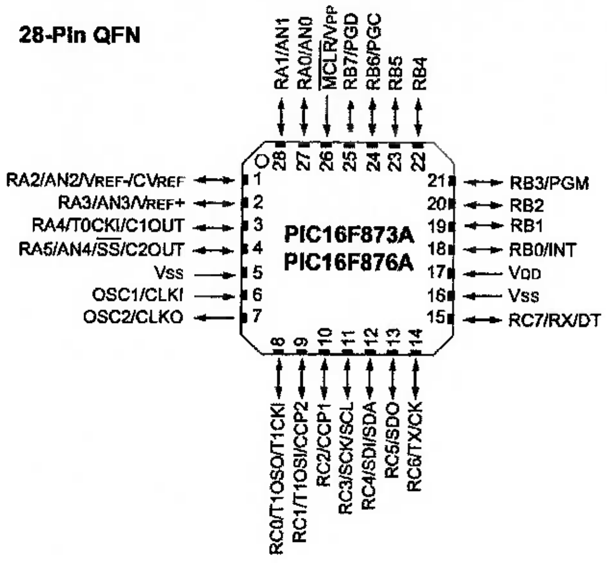
Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master I^{2}C			
PIC16F873A	7.2K	4096	192	128	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F874A	7.2K	4096	192	128	33	8	2	Yes	Yes	Yes	2/1	2
PIC16F876A	14.3K	8192	368	256	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F877A	14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

Pin Diagrams

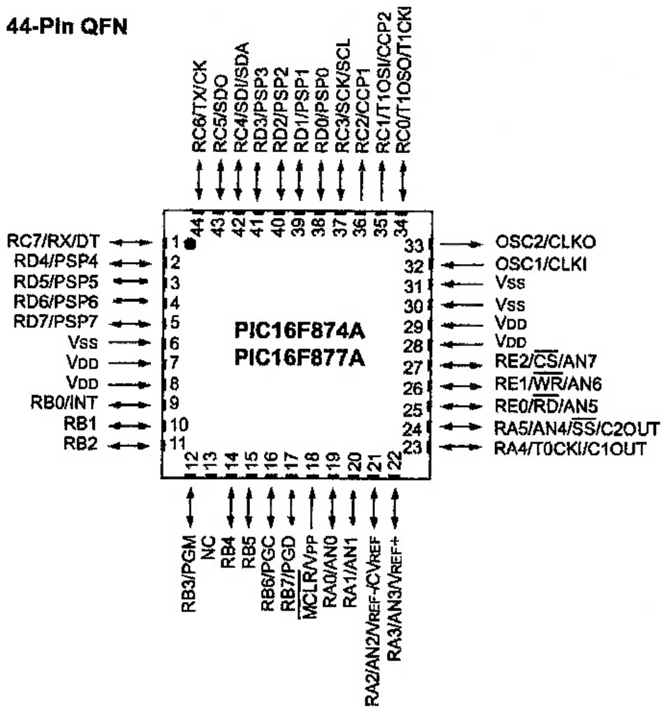
28-Pin PDIP, SOIC, SSOP



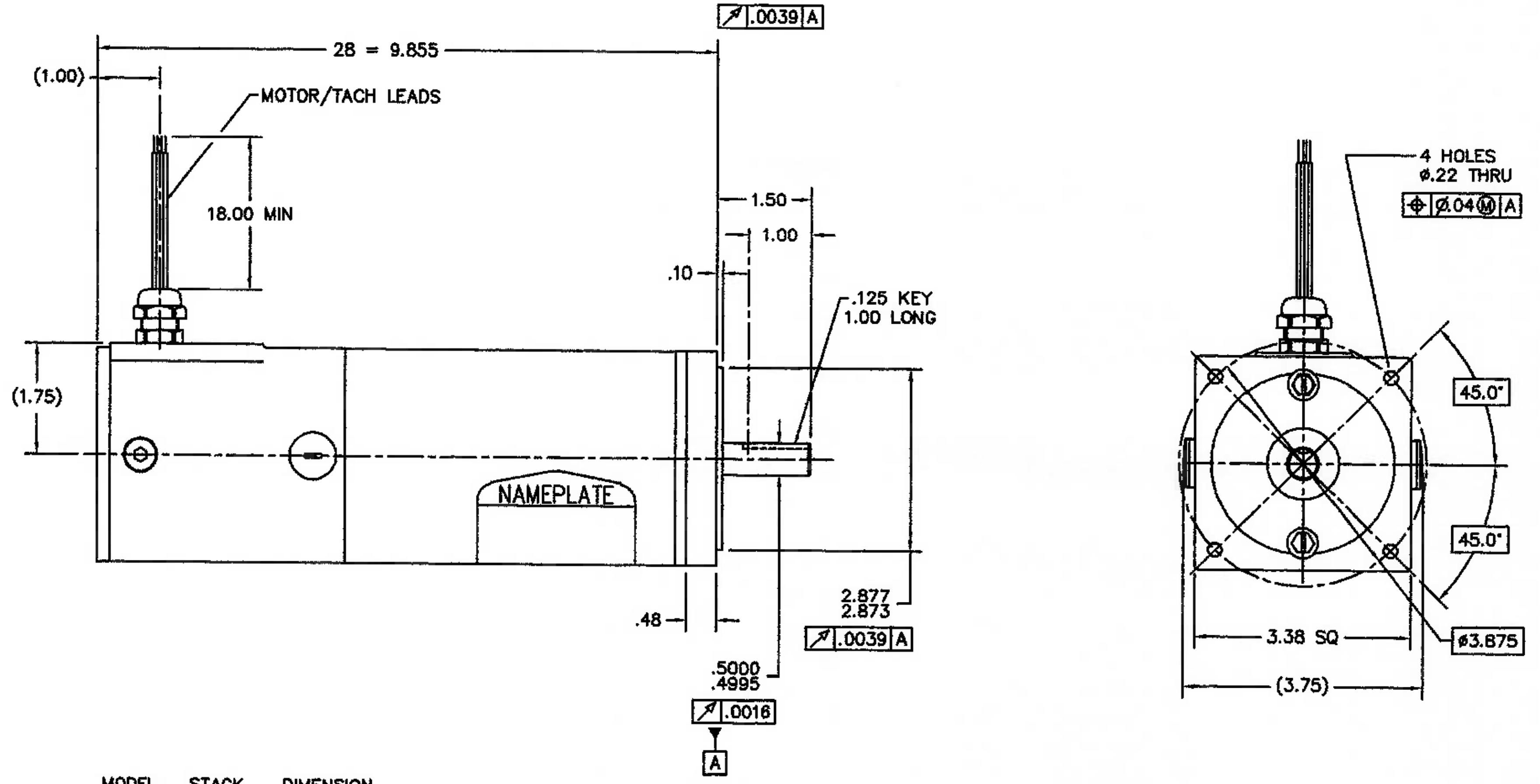
28-Pin QFN



44-Pin QFN



APÊNDICE G – DESENHOS DOS MOTORES



MODEL	STACK	DIMENSION
3353	12	7.855
3358	20	8.855
3363	28	9.855

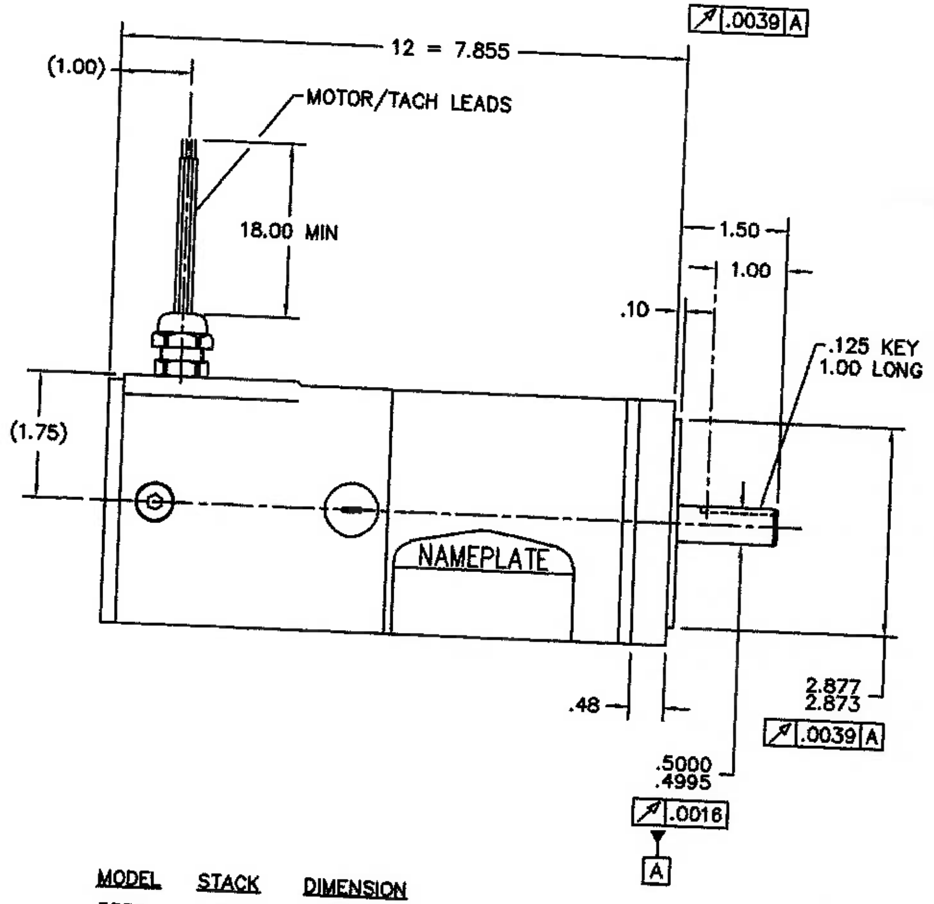
NOTE:
UNSPECIFIED TOL TO BE ±.06

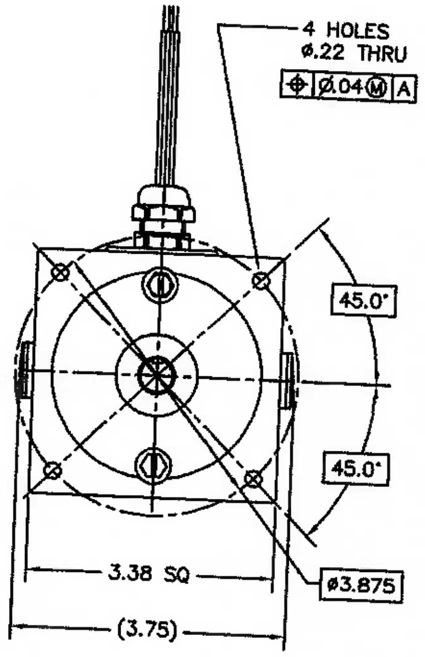
MATL: –			
REV: A	ADDED BRUSH CAPS, PIPE PLUG, & KEYWAY LENGTH		
53P125	SCALE: .5	BY: TDJ	REVISED: 07/03/97
	FILE: AAA00040773		TDR: 0125759

BALDOR ELECTRIC Co.

FINAL ASSY M/MT-33**-*L*A* METRIC BEARINGS

53P125





MODEL	STACK	DIMENSION
3353	12	7.855
3358	20	8.855
3363	28	9.855

NOTE:
UNSPECIFIED TOL TO BE ±.06

MATL: –			
REV: A	ADDED BRUSH CAPS, PIPE PLUG, & KEYWAY LENGTH		
53P125	SCALE: .5	BY: TDJ	REVISED: 07/03/97
	FILE: AAA00040773		TDR: 0125759

BALDOR ELECTRIC Co.

FINAL ASSY M/MT-33**-*L*A* METRIC BEARINGS

53P125