Brushed DC Motor Control Using the MC68HC16Z1
by Lawrence Donahue

INTRODUCTION

The MC68HC16Z1 is a 16-bit high speed microcontroller that incorporates a number of different modules. One of these modules is the General Purpose Timer (GPT), which provides various timing functions including pulse width modulation (PWM) output. PWM is very useful for motor control. This note describes a DC motor control system that provides for constant motor speed using PWM.

The control system uses motor shaft rotation period as its input, monitors motor speed, and changes PWM output duty cycle to either speed up or slow down the motor in order to maintain constant speed. The M68HC16 interfaces to the motor via the DEVB103 Logic to Motor Interface Module, which is described in detail in Freescale Application Note AN1300, Interfacing Microcomputers to Fractional Horsepower Motors.

BACKGROUND

A DC motor is a transducer that converts electrical energy to mechanical energy. As shown in Figure 1, an ideal motor would run without loss and store no energy.

The model of an ideal motor is similar to that of an ideal transformer, but only one side has voltage (V) and current (I) variables, while the other side has velocity (U) and force (F) variables. The voltage-velocity and current-force relationships are:

\[ K I_s = F_m \]
\[ V_s = K U_m \]

However, a real motor is far from ideal, both electrically and mechanically. The electrical side consists essentially of wire wound around a core. The windings have an associated inductance, and since wire has resistivity, there is also a finite resistance. Inertia affects mechanical operation. A rotating motor does not instantaneously stop when disconnected from its power source, but rather slows down and eventually stops.
Similarly, a stopped motor does not instantaneously jump up to speed when power is applied. Because the motor resists step changes in the velocity (the across variable) there is an element that looks capacitive. Because an unpowered motor eventually slows down, it is lossy. A more realistic model of a motor includes an electrical resistance and inductance and a mechanical resistance and compliance, along with an additional load, as shown in Figure 2.

![Figure 2 Model Motor with Load](image)

Figure 2 Model Motor with Load

Figure 3 shows the electrical equivalent circuit, looking into the terminals of the ideal transducer.

![Figure 3 Electrical Equivalent of Mechanics](image)

Figure 3 Electrical Equivalent of Mechanics

Figure 4 is the second-order electronic equivalent of the motor.

![Figure 4 Electrical Equivalent of a Motor](image)

Figure 4 Electrical Equivalent of a Motor
The result is a second order model system with a natural frequency of:

\[ \omega_0 = \left[ L_e C_{eq} \right]^{-1/2} \]

The input to this model system is a pulse-width modulated signal that switches between ground and a constant voltage. Switching occurs at a constant frequency with a given duty cycle. If the switching frequency is sufficiently above the bandwidth of the motor, the motor filters out everything but the DC component of the PWM signal, thus averaging it. For example, consider a PWM that switches between 0 volts and \( V_0 \) volts with a duty cycle of 25%. The motor behaves as if it were connected to a DC supply of 0.25\( V_0 \) volts — the duty cycle of the PWM determines the speed of the motor.

In real DC motors, however, the relationship between source voltage and shaft velocity is not linear, and these relationships vary from motor to motor. Therefore, one desirable characteristic of a motor control system is the ability to control speed independent of motor characteristics. Also, in many real-world applications, motor loads vary. Hence, another desirable characteristic is the ability to provide constant motor speed under changing loads. These requirements are addressed by monitoring and controlling motor speed rather than providing a specific voltage or duty cycle.

**SYSTEM OVERVIEW**

The system has four basic elements, as shown in Figure 5.

The first block represents the (MC68HC16Z1EVB) evaluation board that provides system computing and control functions. The MCU is accessed by EVB16 software running on an IBM PC compatible connected to the EVB via a parallel port. The GPT in the MCU generates a PWM signal which is connected to the DEVB103 logic to motor interface module (Figure 6). The logic to motor interface module takes logic level PWM signals and switches the power transistors of an H-bridge to provide motor drive power up to 60 V and 3 A. The motor is the third element of the system. The fourth element consists of an opto-sensor and a sensor board that conditions sensor output so that the period of motor shaft rotation can be measured by software, to complete the feedback loop. Figure 7 shows a typical comparator with unipolar output. Comparator component values depend on sensor output level.

**Figure 5 System Block Diagram**

The first block represents the (MC68HC16Z1EVB) evaluation board that provides system computing and control functions. The MCU is accessed by EVB16 software running on an IBM PC compatible connected to the EVB via a parallel port. The GPT in the MCU generates a PWM signal which is connected to the DEVB103 logic to motor interface module (Figure 6). The logic to motor interface module takes logic level PWM signals and switches the power transistors of an H-bridge to provide motor drive power up to 60 V and 3 A. The motor is the third element of the system. The fourth element consists of an opto-sensor and a sensor board that conditions sensor output so that the period of motor shaft rotation can be measured by software, to complete the feedback loop. Figure 7 shows a typical comparator with unipolar output. Comparator component values depend on sensor output level.

**Figure 10** shows the open loop system function. The input is \( D \), the duty cycle that determines the speed of the motor. The output is \( \omega_A \), the actual speed of the motor. Hence, the transfer function of the motor is:

\[ \omega_A / D = \left[ s^2 + 2as + \omega_0^2 \right]^{-1} \]

Observations of the actual motor indicate that the open loop system function is overdamped, and thus looks somewhat like a first order system.
Figure 6 Logic To Motor Interface Schematic

Figure 7 Sensor Output Conditioning Circuit
The system is to maintain the motor speed $\omega_A$ constant at the desired speed $\omega_D$. Examining the closed-loop system function diagram shown in Figure 9 yields:

$$D = D + \frac{[\omega_D - \omega_A]}{\omega_D}$$

or

$$\omega_A = \omega_D$$

Therefore the input-output relationship is independent of the motor given a constant input $\omega_D$. However, when the transfer function of the motor changes (i.e., when the load on the motor changes) or when the desired speed changes, the system must adjust the value of $D$ to make the actual motor speed equal the desired speed. Further examination of the system function yields:

$$D = D + \frac{[\omega_D - \omega_A]}{\omega_D} = \frac{\omega_A}{V_{ref}} \left( s^2 + 2\alpha s + \omega_0^2 \right)$$

which simplifies to:

$$\omega_A = V_{ref} \left( 1 + D \right) \left[ s^2 + 2\alpha s + \omega_0^2 + V_{ref}/\omega_D \right]^{-1}$$

As shown in Figure 10, the poles of the closed loop system function differ from those of the open loop function. The poles become:

$$s_1 = -\alpha + \left[ \alpha^2 - \omega_0^2 - V_{ref}/\omega_D \right]^{1/2}$$

$$s_2 = -\alpha - \left[ \alpha^2 - \omega_0^2 - V_{ref}/\omega_D \right]^{1/2}$$
The poles are thus functions of $V_{\text{ref}}/\omega_D$. If $V_{\text{ref}}/\omega_D$ gets large enough, the poles become complex, and the actual motor speed, $\omega_A$, rings as it settles to its final value of $\omega_D$. The envelope and overshoot of the ringing are both dependent on several factors, including the control algorithm, the voltage and duty cycle used for the PWM, the desired speed, the load on the motor, and the motor itself.

**SOFTWARE CONTROL**

The software that controls the motor brings the motor up to speed as fast as possible, given the voltage constraints of the system, and then maintains that desired speed. Figure 12 is an overall block diagram of the system software. Figure 13 through Figure 15 provide flow diagrams of the three major blocks of code.

*Begin* sets up the registers that control the GPT, sets the desired period of rotation and tolerances, and sets the PWM to 100%. *Startup* then monitors the period of rotation of the motor by using the subroutine *measure* (refer to Figure 13). If *measure* returns a period longer than that desired, the program loops back to *startup*. When *measure* returns a period shorter than that desired, the program continues with *new_duty* (refer to Figure 13), which calculates and sets the new duty cycle with the algorithm:

$$D_\text{new} = D_\text{old} \left[1 + \frac{(T_D - T_A)}{T_A}\right]^{-1}$$

where $T_D$ and $T_A$ are the desired and actual periods of rotation, respectively. Following *new_duty*, *tolerance* (refer to Figure 14) checks to see if the measured period is within the tolerances designated in *begin*.

For experimental purposes, a second control algorithm that modifies the duty cycle in a different manner was also implemented. The routine compares $T_D$ and $T_A$ and determines whether $T_A$ is greater or less than $T_D$, then takes one of the following actions:

- If the two periods are within 8 GPT counts, the duty cycle is not modified.
- If $T_D - 8$ is greater than $T_A$, the duty cycle is decreased.
- If $T_D + 8$ is less than $T_A$, the duty cycle is increased.

The routine then looks at the magnitude of the difference between the two periods to determine by how much the duty cycle is to be modified. Figure 16 is a flow diagram of the alternate control algorithm.

*Figure 11* is a plot of the increment by which the duty cycle is modified as a function of the difference between $T_D$ and $T_A$.

![Figure 11 Modify Value vs. Difference](image)
Figure 12  System Software Flow Diagram
Figure 13  *Startup* Routine Flow Diagram

Figure 14  *Tolerance* Subroutine Flow Diagram
Figure 15 New_duty Routine Flow Diagram
Figure 16  Alternate Control Algorithm Flow Diagram

START

SET TMSK1 TO 0100 TO MASK FOR IC1 INTERRUPT

SET A TO 0

YES

IS A < 2?

NO

TMSK1 = 0000 TO DISABLE IC1 INTERRUPT

INCREMENT A.

INC

A = 2?

YES

NO

TIC1 => E

TFLG1 = F880

RTI

YES

TIC1 => D

ACTUAL = -(E - D)

SET A TO 2.

TFLG = F880

RTI

NO

TIC1 => D

ACTUAL = -(E - D)

SET A TO 2.

TFLG = F880

RTI

Freescale Semiconductor, Inc.

For More Information On This Product, Go to: www.freescale.com
Both control routines were evaluated on two different motors. One motor was a slower, less responsive motor that worked well with the second increment/decrement approach and not so well with the first approach. The second motor was faster and more responsive and worked very well with the first approach but not so well with the second approach.

CODE LISTINGS

Listings 1 and 2 contain code for the first implementation of the control routine and the second increment/decrement implementation of the control routine, respectively.

Listing 1 First Control Implementation.

```assembly
INCLUDE 'EQUATES.ASM' ;table of EQUates for common register addr
INCLUDE 'ORG00000.ASM' ;initialize reset vector

ORG $80
DC.W PERIOD ;IC1 jumps to PERIOD

ORG $0400 ;offsets from IX for variables
DESIRED EQU $0 ;location for storing the desired period
DELTA EQU $2 ;location for storing the period tolerance
ACTUAL EQU $4 ;location for storing the measured period
DUTY EQU $6 ;location for storing the present duty cycle
TEMP1 EQU $8 ;location for storing any temporary values
TEMP2 EQU $A ;   "      "     "     "      "       "
TEMP3 EQU $C ;   "      "     "     "      "       "
TEMP4 EQU $E ;   "      "     "     "      "       "

ORG $0200 ;start program after interrupt vectors

***** Initialization Routines *****
INCLUDE 'INITRAM.ASM' ;initialize and turn on SRAM
;set stack (SK=1, SP=03FE)
INCLUDE 'INITSYS.ASM' ;initially set EK=F, XK=0, YK=0, ZK=0
;set sys clock at 16.78 MHz, disable COP

***** Here we go with motor control. *****

BEGIN LDD #$0083 ;Put the GPT into supervisor mode (the default
STD GPTMCR ;mode) and sets interrrupt priority level to 3.
LDD #$FFFF
STD CSBAR0
LDD #$7801 ;assert AVEC and other int. vector stuff
STD CSR0
LDD #$1740 ;Set IC1 to be highest GPT priority, GPT to...
STD ICR ;highest priority interrupt, and vector base...
;address to 4.
LDY #$400 ;Beginning for variables in indirect addresses.
LDD #$A00 ;Set the desired period to $A00 GPT counts.
STD DESIRED,Y
LDD #$80 ;Set the tolerance to +/- $80 GPT counts.
STD DELTA,Y
LDAA #$01 ;Set IC1 to capture only on a positive edge.
LDAA #$00 ;The A register is used to keep track of how...
MEAS_LOOP CMPA #$02 ;...many input captures have taken place--with...
BMI MEAS_LOOP ;...two, we can measure the period.
LDD #$0000 ;If two interrupts have taken place,reset for...
STD TMSK1 ;...no interrupts (disable IC1).
BMI TMS LOOP ;Loop if two measurements have not been made.
LDD #$0000 ;Enable the IC1 interrupt and the TCNT clock...
STD TMSK1 ;...to be the OCI pin.
LDAA #$50 ;The A register is used to keep track of how...

TOLERANCE BSR MEASURE ;Branch to "measure" subroutine.
```

For More Information On This Product, Go to: www.freescale.com
LDE ACTUAL,Y ;Load the measured period in E  
LDD DESIRED,Y ;Load the desired period in D.  
ASLD ;Double the desired period and put in D.  
SDE ;Subtract twice the desired period from the...  
...actual period, and if the actual period is...  
BPL TOO_SLOW ;...more than twice the desired, branch to the...  
...the "too_slow" code.  
LDD DESIRED,Y ;Load the desired period in D.  
LDE ACTUAL,Y ;Load the measured period in E.  
ASLE ;Half the desired period and put in D.  
SDE ;Subtract half the desired period from the...  
...actual period, and if the actual period is...  
BMI TOO_FAST ;...less than half the desired, branch to the...  
...the "too_fast" code.  
BRA WITHIN ;Otherwise branch to the "within" code.  
TOO_SLOW  
LDD #$00FF  
STD DUTY,Y ;Set the user defined duty cycle location to $FF.  
STAB PWMA ;Set the PWMA register for a $FF/$100 duty cycle.  
BRA WAIT ;Branch to the "wait" code.  
TOO_FAST  
LDD #$0000  
STD DUTY,Y ;Set the user defined duty cycle location to 0.  
STAB PWMA ;Set the PWMA register for a 0% duty cycle.  
BRA WAIT ;Branch to the "wait" code.  
WITHIN  
LDD DESIRED,Y ;Load the desired period in D.  
LDE DELTA,Y ;Load the period tolerance in E.  
ADE ;Add the two together to get the maximum...  
...period allowed and store in E.  
SDE ;Subtract it from the max period allowed.  
BMI JUMP ;Branch to get new duty cycle if slow.  
LDD DESIRED,Y ;Load the desired period in D.  
LDE DELTA,Y ;Load the period tolerance in E.  
NEGE ;Negate the period tolerance.  
ADE ;Add the two together to get the minimum...  
...period allowed and store in E.  
SDE ;Subtract it from the min period allowed.  
BPL JUMP ;Branch to get new duty cycle if fast.  
JUMP  
JMP NEW_DUTY ;Jump to "new_duty" code.  
WAIT  
LDD #$2 #2 ;Load the number of loops to wait for in E.  
STD TEMP1,Y ;...because it will be used to count the number...  
...of loops.  
WAIT_LOOP  
INCW TEMP1,Y ;Increment the counter loop.  
CPE TEMP1,Y ;Has the count (TEMP1,Y) reached the value in E?  
BNE WAIT_LOOP ;Loop if the count hasn't reached the value in E.  
BRA TOLERANCE ;Branch back to tolerance once the count has...  
...reached the specified value.  
PERIOD  
INCA ;This code is executed when the I1 interrupt...  
...takes place. The code first increments A...  
...which is used to count how many edges have...  
...been detected in the present "measure" routine.  
CMPA #$02 ;Compare the number of edge detections to 2.  
BEQ DELTA_T ;If second edge, jump to DELTA_T routine.  
LDD TIC1 ;Load the TCNT input capture value in E.  
BCLR TFLG1,#501 ;Clear the I1 interrupt flag.  
RTI ;Return from the I1 interrupt.  
DELTA_T  
LDD TIC1 ;Load the TCNT input capture value in D.  
SDE ;Subtract the second time (D) from first (E)...  
NEGE ;...and change the sign to get positive value...  
...for the period.  
STE ACTUAL,Y ;Store the period in the appropriate location...  
...of user specified memory (ACTUAL,Y).  
LDAA #$02 ;Set A to 2 to break out of MEAS_LOOP above.  
BCLR TFLG1,#501 ;Clear the I1 interrupt flag.  
RTI ;Return from the I1 interrupt.  
NEW_DUTY  
LDD ACTUAL,Y ;Load the measured period into D.  
LDE DESIRED,Y ;Load the desired period into E.  
SDE ;Subtract the desired period from the measured...  
...and put result in E.  
BPL PLUS ;Branch to the "plus" code if the desired period...  
...is greater than the measured period.  
MINUS  
NEGE ;-(Desired-Measured) --> E  
TD ;-(Desired-Measured) --> E  
LDD ACTUAL,Y ;Load the measured period into the IX register.  
FDIV ;(Actual-Desired)/Actual --> IX  
STX TEMP1,Y ;Store the result in user defined memory...  
LDD TEMP1,Y ;...TEMP1,Y and store the result in D.  
LD $8000 ;Shift bits one place to the right because in...  
...this section, the convention changes and...  
...$8000 becomes equal to 1 instead of 0.5.  
SDE ;Subtract the word shifted above from $8000 thus...  
...performing: 1-(Actual-Desired)/Actual --> E  
BRA FACTOR ;Branch to section of code that will scale the...  
...present duty cycle by the result stored in E.
LDX  ACTUAL,Y ;Load the measured period into the IX register.
TED  ;(Desired-Actual) --&gt; E
FDIV  ;(Desired-Actual)/Actual --&gt; IX
STX  TEMP1,Y ;Store the result in user defined memory...
LDD  TEMP1,Y ;...TEMP1,Y then load the result in D.
LSRD  ;Shift bits one place to the right because in...
LDE  #$8000  ;...this section, the convention changes and...
ADE  ;Add the word shifted above to $8000 thus...
FACTOR  STE  TEMP1,Y  ;...performing: 1+(Desired-Actual)/Actual --&gt; E.
STE  TEMP1,Y  ;Store the result from "minus" or "plus" in...
LDD  TEMP1,Y ;...user defined memory TEMP1,Y then load the...
LDE  #$8000  ;...result in IX.
LDD  DUTY,Y  ;Load D with the present duty cycle.
FDIV  ;Divide the present duty cycle by the factor...
STX  TEMP1,Y  ;...calculated in "minus" or "plus" above...
LDD  TEMP1,Y ;...thus performing the operation:...
LDE  #$8000  ;...DUTY/(1+(Desired-Actual)/Actual) --&gt; IX.
LSRD  ;Shift the result one place to the right to...
STD  DUTY,Y  ;...compensate for the shift from above.
STD  DUTY,Y  ;Load this result in the memory location...
LDE  #$00FF  ;...designated to be the duty cycle--DUTY,Y.
SDE  ;...load E with $FF and compare with the result...
BPL  OK  ;...as a sanity check to make sure that the...
LDD  #$00FF  ;...resulting duty cycle makes sense.
LDE  #$00FF  ;...to OK code, ...
LDD  #$00FF  ;...otherwise, load the maximum allowable duty...
STD  DUTY,Y  ;...cycle ($FF/$100) into DUTY,Y.
OK  STA  PWMA  ;Store the new duty cycle in the GPT PWMA register.
JMP  TOLERANCE ;Loop back to TOLERANCE.

Listing 2  Second (Increment/Decrement) Control Implementation

INCLUDE 'EQUATES.ASM'  ;table of EQUates for common register addr
INCLUDE 'ORG00000.ASM' ;initialize reset vector
ORG  $80
DC.W PERIOD  ;IC1 jumps to PERIOD
ORG  $0400  ;Offsets from IX for variables
DESIRED EQU  $0  ;Location for storing the desired period.
DELTA EQU  $2  ;Location for storing the period tolerance.
ACTUAL EQU  $4  ;Location for storing the measured period.
DUTY EQU  $6  ;Location for storing the present duty cycle.
TEMP1 EQU  $8  ;Location for storing any temporary values.
TEMP2 EQU  $A  ;Location for storing any temporary values.
TEMP3 EQU  $C  ;Location for storing any temporary values.
TEMP4 EQU  $E  ;Location for storing any temporary values.
ORG  $0200 ;start program after interrupt vectors
***** Initialization Routines *****
INCLUDE 'INITRAM.ASM'  ;initialize and turn on SRAM
;set stack (SK=1, SP=03FE)
INCLUDE 'INITSYS.ASM' ;initially set EK=F, XK=0, YK=0, ZK=0
;set sys clock at 16.78 MHz, disable COP
*****  Here we go with motor control.  *****
BEGIN  LDD  #$0083  ;Put the GPT into supervisor mode (the default...
STD  GPTMCR  ;...and sets interrupt priority level to 3.
LDD  #$FFF9
STD  CSBAR0
LDD  #$7801  ;assert AVEC and other int. vector stuff
STD  CSOR0
LDD  #$1740  ;Set IC1 to be highest GPT priority, GPT to...
STD  ICR  ;...highest priority interrupt, and vector base...
STD  DESIRED,Y  ;...base address to 4.
LDY  #$400  ;Beginning for variable in indirect addresses.
LDD  #$A000  ;Set the desired period to $A000 GPT counts.
STD  DESIRED,Y
LDD  #$68  ;Set the period tolerance to +/- $8 GPT counts.
STD  DELTA,Y
LDD  #$501  ;Set IC1 to capture only on a positive edge.
STA  TCTL2
LDD  #$0062  ;Set the input of PWMCNT to the system clock...
STD  PWMC  ;...divided by 128 and set PWMA to be high always.
STARTUP BSR MEASURE  ;Branch to measure period of revolution.
CPE  DESIRED,Y  ;Compare the measured period with the desired...
BPL STARTUP  ;...and loop if motor is slower than desired
LDD  #$00FF  ;Store the value $FF in...
STD    DUTY,Y ;...the user defined duty cycle location and...
STAB   PWMA ;...in the GPT PWMA register thus setting the...
LDD    #$0060 ;Change PWMA's output to be from a constant...
STD    PWMC ;...output to a PWM output with duty cycle...
JMP    NEW_DUTY ;Branch to get new duty cycle.
MEASURE LDD    #$0104 ;Branch to "measure" subroutine.
STD    TMSK1 ;...be the OCl pin (as a reference for debug).
LDAA   #$500 ;The A register os used to keep track of how...
;...two, we can measure the period.
BMFI    MEAS_LOOP ;If two measurements have not been made.
LDD    #$0000 ;If two interrupts have taken place, reset for...
STD    TMSK1 ;...no interrupts (disable IC1).
RTS              ;Return from "measure" subroutine.
TOLERANCE BSR    MEASURE ;Branch to "measure" subroutine.
LDD    DESIRED,Y ;Load the desired period into D.
LDE    DELTA,Y ;Load the period tolerance into E.
AEE ;...period allowed and store the value in E.
LDD    ACTUAL,Y ;Load the measured period in D and...
SDE ;...subtract it from the max period allowed.
JMP    NEW_DUTY ;Branch to get new duty cycle if slow.
LDD    DESIRED,Y ;Load the desired period into D.
LDE    DELTA,Y ;Load the period tolerance into E.
NEGEE ;Negate the period tolerance.
AEE ;Add the two together to get the maximum...
;...period allowed and store it in E.
LDD    ACTUAL,Y ;Load the measured period in D and...
SDE ;...subtract it from the min period allowed.
BPL    JUMP ;Branch to get new duty cycle if fast.
JUMP          JMP    NEW_DUTY ;Jump to "new_duty" code which calculates the...
;...new duty cycle.
WAIT        LDE    #$2 ;Load the number of loops to wait for in E.
LDD    #$0000 ;Initialize temporary location 41 with zero...
STD    TEMP1,Y ;...because it will be used to keep track of...
;...the number of loops.
WAIT_LOOP   INCA             ;This count is executed when the IC1 interrupt...
CMPA   #$02      ;Compare the number of edge detections to 2.
BEQ    DELTA_T   ;If second edge, jump to DELTA_T routine.
LDE    TIC1      ;Load the TCNT input capture value in E.
BCLR   TFLG1,#$01 ;Clear the IC1 interrupt flag
RTI              ;Return from the IC1 interrupt.
DELTA_T       LDD    TIC1      ;Load the TCNT input capture value in D.
SDE ;...subtract the second time (D) from first (E)...
NEGE ;...and change the sign to get positive value...
STA    ACTUAL,Y ;Store the measured period in the appropriate...
;...location of user specified memory (ACTUAL,Y).
LDAE    #$02 ;Set A to 2 to break out of MEAS_LOOP above.
BCLR   TFLG1,#$01 ;Clear the IC1 interrupt flag.
RTI              ;Return from the IC1 interrupt.
NEW_DUTY      LDD    ACTUAL,Y ;Load the measured period into D.
LDE    DESIRED,Y ;Load the desired period into E.
SDE ;Subtract the desired period from the measured...
;...and put the result in E.
BPL    PLUS ;Branch to the "plus" code if the desired...
;...period is greater than the measured period.
NEGE ;Load -(Desired-Measured) into register E.
MINUS        STE    TEMP1,Y ;Store -(Desired-Measured) into location TEMP1,Y.
LDD    #$0010 ;Load the value $10 into register D.
SDE ;Subtract $10 from the discrepancy between the...
;...measured period and the desired period.
BPL    TWO_PL   ;If the discrepancy is greater than $10, branch...
;...to the "two_pl" code.
LDD    #$0001 ;If the discrepancy is less than $10, load 1...
;...into D to be the value to add to the present...
;...duty cycle.
TWO_PL        LDD    TEMP1,Y ;Load -(Desired-Measured) from TEMP1,Y into E.
LDD    #$0040 ;Load the value $40 into register D.
SDE ;Subtract $40 from the discrepancy between the...
;...measured period and the desired period.
BPL    FOUR_PL ;If the discrepancy is greater than $40, branch...
;...to the "four_pl" code.
LDD    #$0002 ;If the discrepancy is less than $40, load 2...
JMP OK_PL     ; Jump to the "ok_pl" code
LDE TEMP1,Y  ; Load -(Desired-Measured) from TEMP1,Y into E.
LDD #$0100   ; Load the value $100 into register D.
SDE          ; Subtract $100 from the discrepancy between the...
BPL EIGHT_PL ; If the discrepancy is greater than $100,...
LDD #$0004   ; If the discrepancy is less than $100, load 4...
LDD DUTY,Y   ; Load the present duty cycle into register E...;
ADE          ; and add it to the value obtained above...
STE DUTY,Y   ; Store the new duty cycle in DUTY,Y.
LDD DUTY,Y   ; Load the new duty cycle from DUTY,Y into D.
LDE #$00FF   ; Load E with $FF and compare with the result...
BPL OK_MI    ; If the result passes the sanity check, branch...
LDD #$00FF   ;...otherwise, load the maximum allowable duty...
STE TEMP1,Y  ; Store (Desired-Measured) into location TEMP1,Y.
BRA OK       ; Branch to "ok" code.
LDE TEMP1,Y  ; Load (Desired-Measured) from TEMP1,Y into E.
SDE          ; Subtract $40 from the discrepancy between the...
BPL FOUR_MI  ; If the discrepancy is greater than $40,...
LDD #$0002   ; If the discrepancy is less than $40, load 2...
LDD DUTY,Y   ; Load the present duty cycle into register E...;
STD DUTY,Y   ; Store the new duty cycle in DUTY,Y.
LDD DUTY,Y   ; Load the new duty cycle from DUTY,Y into D.
LDE 0        ; Load E with 0 and compare with the result...
BMI OK       ; If the result passes the sanity check, branch...
LDD #$0      ;...otherwise, load the minimum allowable duty...
STD DUTY,Y   ; Store the new duty cycle in the GPT PWMA register.
JMP TOLERANCE ; Loop back to "tolerance" code.
ADDITIONAL INFORMATION

The following Freescale publications contain additional information that may be of use to the reader.

The motor control system described in this note is based on an M68HC11-based system discussed in Application Note AN1311, Software for an 8-bit Microcontroller Based Brushed DC Motor Drive. However, the M68HC11 system uses a PWM signal as its input.

The DEVB103 Logic to Motor Interface Module is completely described in Application Note AN1300, Interfacing Microcomputers to Fractional Horsepower Motors.

The MC68HC16Z1 User’s Manual (MC68HC16Z1UM/AD) contains comprehensive information concerning the MC68HC16Z1 microcontroller.

The GPT Reference Manual (GPTRM/AD) contains detailed information concerning the General-Purpose Timer (GPT) module in the MC68HC16Z1.

These publications are available through Freescale sales offices and Literature Distribution Centers.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document. Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by the customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

For More Information On This Product, Go to: www.freescale.com