A DC MOTOR CONTROLLER USING A ZILOG MCU

TABLE OF CONTENTS

Abstract 3
General Overview 3
Discussion 4
Theory of Operation 4
Software Description 6
Remote Control 7
Results of Operation 7
Summary 7
Reaffirmation of Results 7
Technical Support 8
Source Code(s) 8
Flow Chart 16
Technical Drawings 19
Test Procedure 19
General Test Setup and Execution 19
References 20
Appendix 21
Schematics 21
Notes 22
Information Integrity 22
Document Disclaimer 22

Acknowledgments

Project Lead Engineer
Jim Gersbacher

Application and Support Engineer
Bob Bongiorno, Mark Thissen

System and Code Development
Jim Gersbacher

ABSTRACT

General Overview

Introduction

Many applications, such as printers, ATM machines, robotics, CD players, disk drivers, fans, and toys require DC motor control. Additionally, the demand for smaller packaging and higher reliability creates a demand for more integrated motor driver solutions. In the past, a DC motor controller was comprised of a microcontroller (MCU) supported by many discrete components. Presently, advanced IC technology reduces the drive electronics to just two chips: an MCU and an integrated motor driver. This Application Note references the ZiLOG Z86E0408PSC and the National Semiconductor LMD18200 to produce the low-part-count DC motor driver.

Motor Control Basics

Several DC motor design topologies exist; however, the most widely-used method is the *H-bridge* configuration. The H-bridge method employs four Bipolar Junction Transistors (BJT) or four Metal–Oxide Silicon Field Effect Transistors (MOSFET) devices configured in an *H* pattern (Figure 1). To drive the motor in the forward direction, current flows through Q1 and Q4, while Q2 and Q3 are turned off (path 1 in Figure 1). Drive the motor in the reverse direction, with Q2 and Q3 turned on and Q1 and Q4 turned off (path 2 in Figure 1). The LMD18200 allows operating frequencies up to 1KHz, with no external components. For faster operating frequencies (up to 500KHz), use an external 10-nF capacitor across the bootstrap pins. The average current of the H-bridge controls motor speed. The Z8 creates a Pulse-Width Modulated (PWM) signal, which is used to control the MOSFETs. Varying the duty cycle of the output, by changing the register value of the counter in the Z8, changes the average DC voltage, which is used to drive the power MOSFETs. The necessary steering logic is incorporated inside the LMD18200. The PWM does not require an external low-pass filter, unlike conventional designs which require an external low-pass filter circuit to produce a constant DC voltage.

Because the voltage drop across the collector and emitter can reach more than 1V during saturation, high heat dissipation is encountered using BJT devices. MOSFETs, which are intrinsically characterized by low drain-to-source turn-on resistance, orR_{DS}, are better suited for motor driven applications. Typically, power MOSFETs require a gate voltage of least 8V to turn on, which is a problem when using an MCU whose outputs swing from 0V–5V. Special logic MOSFETs are developed that exhibit gate turn-on voltages of 5V. The 5V gate turn-on works well for the lower legs of the H-bridge motor drive, but what about the upper legs? The upper legs of the bridge circuit require a higher gate voltage, due to the fact that the motor's winding resistance raises the MOSFETs source reference above the ground potential. A separate DC-to-DC converter chip can be used to overcome the higher voltage turn-on; however, the extra converter adds more cost and complexity to the design. The LMD18200 solves the higher voltage turn-on problem with its built-in DC-to-DC converter.

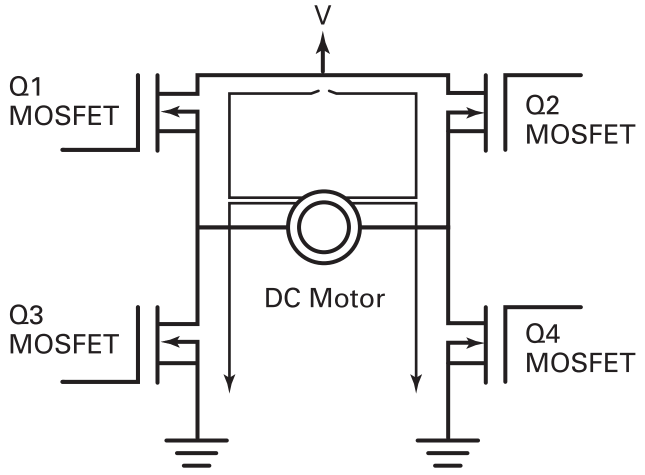

Figure 1. H-Pattern Motor Driver Configuration

DISCUSSION

Theory of Operation

Hardware Description

The heart of the motor controller is the ZiLOG Z86E0408PSC, an 18-pin, one-time programmable (OTP) MCU. The Z86E04 features 1KB of ROM, 128 Bytes of RAM, 14 I/O lines, two timers/counters, and two on-board comparators. A block diagram of the chip is illustrated in Figure 2. An 8-MHz crystal clocks the Z8 MCU, but designs using an RC, LC or ceramic resonator are possible, depending upon the requirement for precision timing. The National Semiconductor LMD 18200 controls the motor. The LMD 18200 is a 3A H-bridge driver IC with direction and braking logic; in addition, the device features current and temperature-sensing output drivers. The block diagram for the LMD 18200 is illustrated in Figure 3.

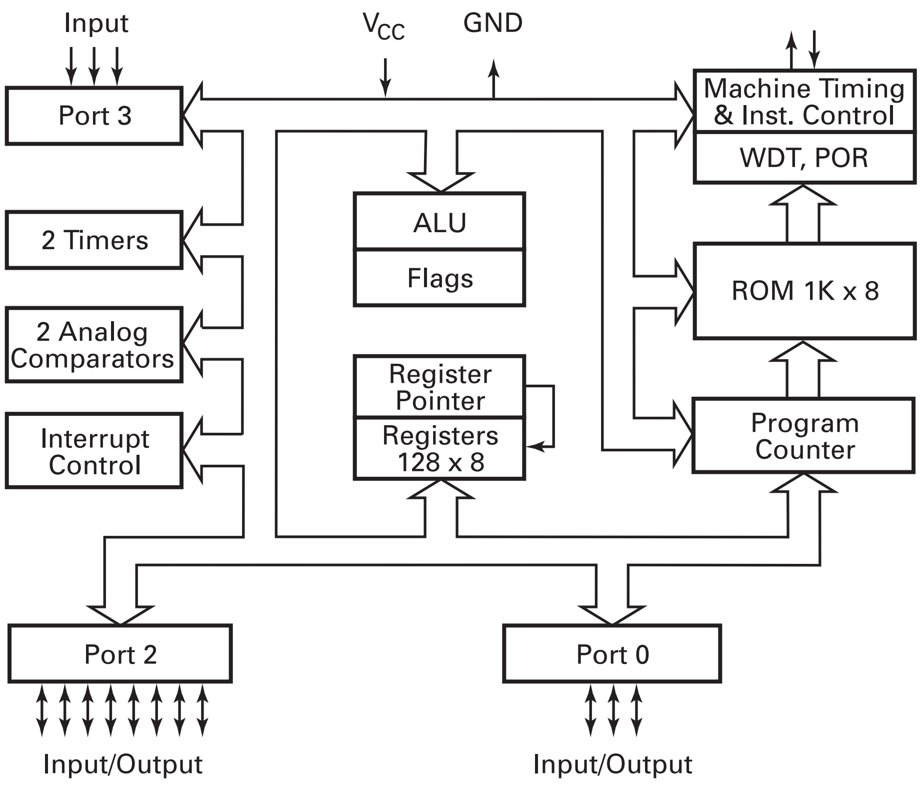

Figure 2. Z86E04 Block Diagram

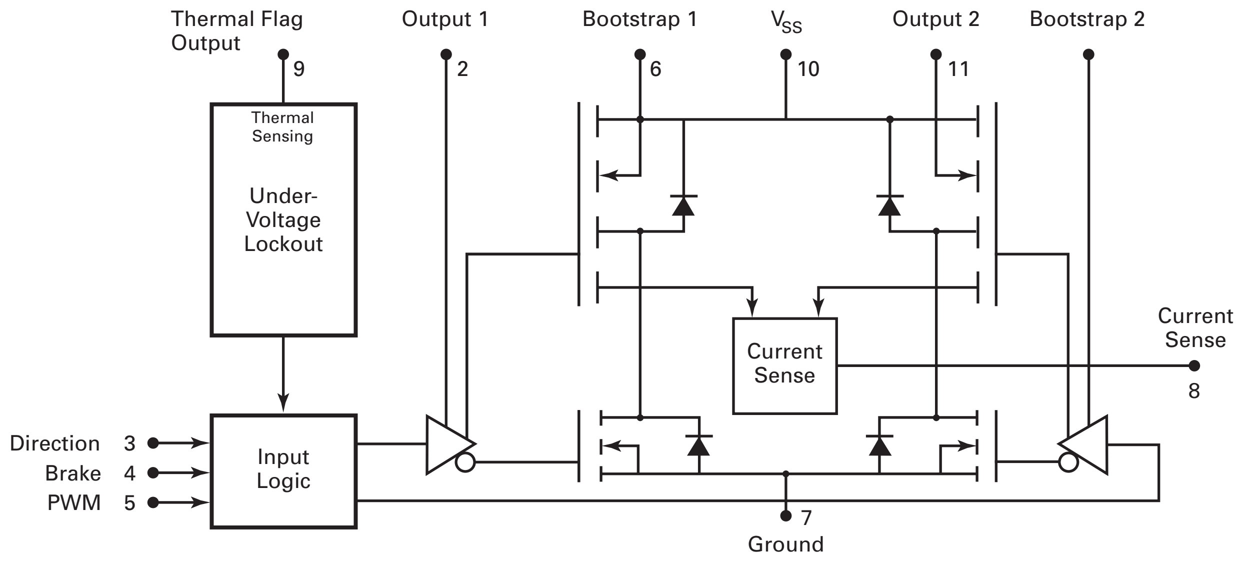

Figure 3. LMD18200 Functional Diagram

Output pins P00, P01, and P02 of the MCU provide the direction, brake and PWM signals to the LMD18200 (refer to the Z86E04 schematic diagram in the Appendix). A resistor from Pin 8 of the LMD18200 to ground provides the current-sensing circuit utilized by the Z8's comparator. The current output from the current-sensing pin is typically 377 mA/A. If the motor is rated at 1 A, a resistor value is chosen so that the voltage developed across the resistor does not exceed 3.3 V, the reference pin for the comparators, P33, is biased at 3.3 V through a voltage divider. The resister value works out to be 3.3V/377 mA = 8.8KΩ. For the circuit in this Application Note, the motor is rated at 12 V @ 1.6 A. The resistor value works out to be:

```
3.3V/(377mA x 1.6A) = 5.5KΩ
```

Any significant load on the motor increases the motor current, thereby increasing the voltage across the current-sensing resistor. If the voltage across the sensing resistor exceeds 3.3 V, a comparator interrupt is generated, and the BRAKE line to the LMD18200 is activated, stopping the motor. When the load condition of the motor is removed, pressing the START/STOP push-button reactivates the motor. The temperature flag, Pin 9 of the LMD18200, is connected to P31 of the Z8, the other comparator. The temperature pin is an open-collector output, so the circuit includes a 10-KΩ pull-up resistor. When the junction temperature of the LMD18200 exceeds 145ºC, the temperature pin goes Low, causing the Z8 to interrupt to the temperature interrupt handler routine.

There are three push-button switches and one single-pole, double-throw (SPDT) switch. The push-button switches are the START/STOP, DECREASE and INCREASE push-buttons. By holding down either of the INCREASE or DECREASE push-buttons, one can increase or decrease the motor speed to a rate in the limit range of 28% to 90%. The SPDT switch determines the FORWARD or REVERSE direction. The switches are connected to P20–P23. LEDs are associated with each of the switches. The STOP LED is red and is connected to P26. The FORWARD LED is yellow and is connected to P25. The REVERSE LED is green and is connected to P24. The debounce time for the switches is managed in the interrupt service routine (ISR). The switches must be held down for at least two passes into the ISR.

Software Description

The software for the motor controller is a state machine. There are three states produced by the motor controller code. The three states are STOPPED, STARTUP and RUNNING. The initial PWM is set for a 50% duty cycle. The initial state is the STOPPED state, as indicated by the red LED. The motor may be started by momentarily pressing the STOP/START push-button. The motor spins until it reaches the speed set by the PWM value in a clockwise or counter-clockwise direction, as set by the direction bit. Timer T1 controls the High time of the PWM. Timer T1 is configured to stop when reaching the zero count. When the terminal count is reached, an interrupt occurs, and in the ISR, it sets port pin P02 Low. Pressing the INCREASE push-button increases the duty cycle of the PWM; conversely, pressing the DECREASE push-button decreases the duty cycle of the PWM. The maximum limits for this application are 90% and 28%, respectively, due to no change in the DC motor speed on the high end, and stopping on the low end.

Pressing the STOP/START button toggles the BRAKE output to the LMD18200, and either brakes or enables the motor, depending on the current state.

In the STOPPED state, the Z8 sets the BRAKE bit to put the motor in the STOP condition; in addition, the STOP LED illuminates. The BRAKE condition effectively shorts the motor via the associated direction drivers.

Caution: Changing the direction of a DC motor while it is running can cause a very large surge of current. Therefore, the STOPPED state is the only time at which the motor is allowed to change direction.

The STARTUP state allows the motor to begin movement and to run for a duration prior to activating the overcurrent ISR or the PWM low ISR. This time delay allows the motor to reach its normal operating condition. When the time delay is complete, the overcurrent, overtemperature and timer T1 ISR are all enabled. The state is changed to the RUNNING state.

In the RUNNING state, timer T1 is continually restarted, and the PWM signal operates normally. Timer T0 causes the PWM to go High, and timer T1 causes the PWM to go Low. By modifying the time value of T1, the High time of the PWM is modified. Timer T1 time is changed by the increase and decrease push-buttons.

The initialization routine sets up the hardware ports, the timers and the interrupts necessary for proper motor controller operation. The I/O ports P20–P23 are arranged as inputs. Port pins P24–P27 are arranged as push-pull outputs. Port pins P00–P02 are configured for outputs. The BRAKE bit is set and the STOP LED is illuminated. The PWM is set up for a 50% duty cycle. The interrupts are set so that T0 is the only interrupt available.

Remote Control

Instead of push-button switches to control the functions and speed of the motor, an I^2C or asynchronous communications protocol can be added for remote control.

RESULTS OF OPERATION

Completing the schematic layout as shown in the Appendix and programming the Z86E04 with the system software allows the system to run with no errors. A $12\,V_{DC}$ bidirectional motor is controlled from a minimal speed to a maximum speed, in both the forward and reverse directions. All of the LEDs illuminate at the appropriate time, and the overcurrent routine tests correctly.

SUMMARY

Reaffirmation of Results

By controlling the PWM output, the H-pattern motor controller can control the speed of a DC motor. By controlling the direction pin, the direction of the motor can be controlled. Finally, a safety measure is added that stops the motor when an overcurrent condition occurs. All operations are controlled by the ZiLOG MCU.

TECHNICAL SUPPORT

Source Code(s)

```
******************************************************************************
**
**
** Application:  Motor Controller
** Date:                                August 07, 1999
** Created by: J. D. Gersbacher for Motor Controller Application Note
** Description:The motor controller code is developed to control a
** LMD18200 National Semiconductor part, which in turn controls the operation
** of a DC motor. Through three control lines and two feedback lines the
** MCU and motor controller can increase, decrease and stop a motor. The
** stopping of the motor can occur through normal operation or through two
** real time interrupts. The two real-time interrupts become active
** if an overcurrent condition or on an overtemperature condition exist.
** The two interrupt service routines handle the error condition exactly the
** same. Both immediately stop the motor and change the condition of the software
** to the STOPPED state.
** The overall flow of the software is based on an interrupt from timer 0.
** The main loop does not execute in this example code; the software merely waits
** for the time-out of Timer 0. Timer 0 interval between interrupts is
** 500 microseconds. Timer 0's ISR is a three condition state machine. The
** three states associated with the ISR are: STOPPED, STARTUP and RUNNING.
** In the STOPPED state, the system checks for a change in direction, then
** store the direction. Storing of the direction occurs whether the
** direction changes. When the direction is determined, the STOPPED STATE
** checks for push-button activity. If any of the push-buttons are active, the
** appropriate routine is run. When the push-button routine is completed, the ISR
** exits using the IRET command. There are three push-buttons increase, decrease
** and start/stop. The start/stop bush button is a toggle button. When the motor
** is in the STOPPED condition, as it is at reset, the pressing of the start/stop
** button initiates the debounce time. When the debounce routine is complete,
** and the start/stop button is still active, the motor starts, the brake pin is
** released, and the pulse width modulation pin (PWM) is set High. The ISR condition is
** changed to the STARTUP state. During the STARTUP state there is a time delay, used
** to get the motor started. The system exits again, using the IRET. When the
** time delay expires, Timer 1 interrupt is enabled, and Timer 1 is started, and the
** ISR condition is changed to the RUNNING state.
** In the RUNNING state, the motor is running at 50% of its maximum speed, provided the
** increase or decrease button had not been pushed. The increase and decrease buttons
** change the speed of the motor. By holding one of the push-buttons down, the
** motor speeds up or slows down, depending on which button is pushed. In the
** sample_ISR routine, the change of speed is accomplished by modifying the Timer 1's
** count value. Timer 1 interrupts when the count value reaches zero. In Timer 1's
** ISR, the PWM signal is brought Low. Timer 1's count value decreases until the
** minimum value, defined in motorhead.h, is reached. Timer 1's count value
** increases until the maximum value, defined in motorhead.h, is reached.
** The third switch is the direction switch. Associated with the direction switch, are
** three LED's. Stop LED, clockwise direction LED and counter-clockwise direction LED.
** When the motor is in the STOPPED state, the Red LED is lit. As described above, the
** STOPPED state is the only time the direction switch is active. When the motor is in
** any other state other than the STOPPED state, the Red LED is turned off and the
** appropriate When Timer 1 exits out of initialization, the time is set for 250
** microseconds.direction LED is lit.
**
** The linker settings must be set accordingly:
** To setup the linker options correctly in ZDS version 2.12:
**            PROJECT → SETTINGS → LINKER
**            RANGES → NEW
**            then add (it takes 3 entries to enter all information below)
**            regfile                  0010h to 001fh
**            regfile_Motor            0020h to 002fh
**            regfile_sample_ISR       0030h to 003fh
*******************************************************************************

   subtitle "Motor Controller"
   .include "motorhead.inc"


   globals

   ;*************** Interrupt Vectors ************************************
   vector      irq0=overtemperature_ISR
   vector      irq1=IRQ1
   vector      irq2=overcurrent_ISR
   vector      irq3=IRQ3
   vector      irq4=sample_ISR
   vector      irq5=pwmlow_ISR


   ;************** Global Variables ************************************
   segment regfile set in linker to start at %10
   public State_Reg, PWM_Reg, Key_temp_reg

State_Regds    %1


   define regfile_Motor, space=RFILE
   segment regfile_Motor                ; set in linker to start at %20

PWM_Reg        ds                       %1
Key_temp_regds %1



   subtitle "Global Functions"
   .page
   ;************** Global Functions ************************************

   extern      overtemperature_ISR,overcurrent_ISR,sample_ISR,pwmlow_ISR
   extern      delay_reg
   segment code
   subtitle "Main Control"
   .page
   ;************** Main Control **********************************
```

```
IRQ1:
IRQ3:
main:
   subtitle “Initialization”
   Newpage
   .page
   ;************* Initialization ************************************
   srp         #Working_Regs
   ld          P0, #motor_stop          ;A High on the Brake pin and PWM stops the mtr
   ld          P01M, #%04               ;P00-P02 outputs controlling
                                        ;Direction,Brake,PWM respectively
   ld          P3M, #%03                ;push-pull outputs port 0&2,analog inputs port 3
   ld          P2, #(%ff^stopped_LED)   ;Stop the motor and show it.
   ld          P2M, #%0f                ;Four input switches, 3 output LED's, 1 unused
                                        ;P20 Direction, P21–P23 Increase, Decrease,
                                        ;Stop respectively
   ld          SPl,#%80                 ;Maximize the stack

   ld          PRE0,#%09                ;Restart at end of count, Must set up the
                                        ;period for 2000hz
   ld          T0,#tmr0_value           ;i=pxtxv where t is 8MHz/2/4 = 1MHz= 1µs
                                        ;where p is the prescaler with a value of 2
                                        ;where v is the interval count value of 250
                                        ;clock is 8HMz/2/4 = 1MHz → 1µs
                                        ;Must do the period or 500µs →
                                        ;500µs=250xpx1µs
   ld          IMR,#T0E                 ;setup for timer 0 to start the sampling.
                                        ;Now setup the High time, 50% default value
   ld          PWM_Reg,#PWM_Half_value  ;Time should be 250µs → 250xpx1µs
   ld          PRE1,#%0A                ;PWM pulse width of 50% and single pass
   ld          T1,PWM_Reg

   ld          IPR,#18                  ;Priority: overcurrent err>overtemperature
                                        ;err>T0 end
   ld          TMR,#%03                 ;load and go for T0. When motor must run,
                                        ;T0 starts T1
   ld          Key_temp_reg,#%FF        ;no keys are stored
   ld          State_Reg,#stopped_state
   ld          delay_reg,#%3            ;allow motor current and speed to reach steady
                                        ;state

   subtitle “Main routine”
   .page
   ;************* Main *********************************************************
   ei
main_loop:                              ;Wait for interrupts.

   jr          main_loop


   ;*******************************************************************************

**
** Application:Motor Controller
** Date:                                August 07, 1999
** Created by: J. D. Gersbacher for Motor Controller Application Note
** Description:Sample ISR is the “main” portion of the software. It
** handles three items, direction test, state operation and
** push-button check. The motor operation is based on a state machine.
** There are three states in the state machine: STOPPED,
** STARTUP and RUNNING. In the STOPPED state the motor is off, brake is set
** and the STOPPED LED is lit. All other LEDs are off. The
** STOPPED state is the only state the direction switch is active.
** The start/stop push-button transitions to the STARTUP state.
** In the STARTUP state, after the debounce count, a delay is inserted, and
** the STOPPED LED is de-energized and the appropriate direction light is lit.
** The delay is to allow the motor time to build up speed and dissipate
** its start up current. When STEADY state is reached, T1 is activated
** and the state changes to RUNNING. In the RUNNING state T1 is restarted
** every time through the ISR.
** In all states, the push-buttons are checked for activity. The increase and
** decrease buttons affect the PWM timer register. The next time the
** timer is loaded is when the new count takes effect.
**
*******************************************************************************
   subtitle “Sample Interrupt Service Routine Values”
   .page
   .include “motorhead.inc”
   .include “sampleISR.inc”

   public bounce_reg,count_reg,key_cnt_reg,make_reg,delay_reg,temp_1_reg,temp2_reg
   extern State_Reg, PWM_Reg, Key_temp_reg
   segment regfile_sample_ISR;set in the Linker to start at %30.

bounce_reg   ds    %1
count_reg    ds    %1
key_cnt_reg  ds    %1
make_reg     ds    %1
delay_reg    ds    %1
temp_1_reg   ds    %1
temp2_reg    ds    %1

   subtitle “Global Functions”
   .page
   ;************** Global Functions ************************************
   segment code

   public sample_ISR
   subtitle “Motor Control”
   .page

sample_ISR:
   push        RP
   ld          RP,#Working_ISRRs        ;working registers 30–3f
   cp          State_Reg,#stopped_state
   jr          ne,test_startup_state_label
; Know the motor is stopped; therefore, check for a change in direction.
   tm          P2,#dir_switch
   jr          z,ccw_direction_label
   or          P0,#forward_direction
   ld          temp2_reg, #(%ff^forward_LED)   ;Store the forward LED for use later.
                                               ;The register, temp2_reg, does not
                                               ;change, even if there is another
                                               ;routine, if the other routine
                                               ;uses a different set of working
                                               ;registers. The changing of the working
                                               ;registers is necessary for proper
                                               ;operation of this routine.
   jr          continue_label
ccw_direction_label:
   and         P0, #reverse_direction
   ld          temp2_reg,#(%ff^reverse_LED)
   jr          continue_label
test_startup_state_label:
   cp          State_Reg,#startup_state
   jr          ne,continue_label
   ld          temp_1_reg,P0                   ;Start the motor.
   or          temp_1_reg,#%04
   ld          P0,temp_1_reg
   dec         delay_reg                       ;Leave the system running until current
reaches
   jr          nz,continue_label               ;normal operating level.
   ld          IMR,#(T0E+OTE+OCE+T1E)          ;Start the PWM and allow overcurrent
ISR.
   ld          State_Reg,#running_state
continue_label:
   cp          State_Reg,#running_state
   jr          ne,keyscan_label
   ld          temp_1_reg,P0                   ;keep the systems PWM running
   or          temp_1_reg,#%04                 ;with PWM High and T1 timer starting.
   ld          P0,temp_1_reg
   nop
   nop
   ld          TMR,#%0E                        ;Enable T0, T1, load T1 for counting.
                                               ;T0 is always counting, so keep the
                                               ;same time for T0.
;*************************************************************************************
   subtitle "Key Scan Operation"
   .page
keyscan_label:
   clr         key_cnt_reg
   ld          temp_1_reg,P2
   and         temp_1_reg,#switches
   cp          temp_1_reg,#%0E                 ;check for any activity on push-buttons.
   jp          eq,make_exit_label
   or          temp_1_reg,#%F1                 ;find out which one.
   rr          temp_1_reg
   ld          count_reg,#%03
keyloop_label:
   inc key_cnt_reg
   rrc         temp_1_reg
   jp          c,NotThisKey_label
   cp          Key_temp_reg,key_cnt_reg        ;first time delay for debounce
   jr          ne,store_key_label
   inc         bounce_reg
   cp          bounce_reg,#bounce_count
   jr          ult,store_key_label
;****** Increase check
   cp          key_cnt_reg,#increase
   jr          ne,try_decrease_label
   inc         PWM_Reg
   cp          PWM_Reg,#maximum
   jr          ult,do_load_label
   ld          PWM_Reg,#maximum
do_load_label:
   ld          T1,PWM_Reg
   jr          make_exit_label
try_decrease_label:
   cp          key_cnt_reg,#decrease
   jr          ne,try_brake_label
   dec         PWM_Reg
   cp          PWM_Reg,#minimum
   jr          ugt,do_load_label
   ld          PWM_Reg,#minimum
   jr          do_load_label
store_key_label:
   ld          Key_temp_reg,key_cnt_reg
   clr         key_cnt_reg
   jr          exit_label
NotThisKey_label:
   djnz        r1,keyloop_label
try_brake_label:
   cp          key_cnt_reg,#start_stop
   jr          ne,make_exit_label
   cp          make_reg,#%ff
   jr          eq,normal_exit_label
   xor         P0,#brake_bit
   ld          make_reg,#%ff
   tcm         P0,#brake_bit
   jr          z,test_dir_label
   ld          P2,temp2_reg
   ld          State_Reg,#startup_state
   jr          normal_exit_label
test_dir_label:
   ld          State_Reg,#stopped_state
   ld          IMR,#T0E
   ld          delay_reg,#%3
   ld          P2,#(%ff^stopped_LED)
   jr          normal_exit_label
make_exit_label:
   clr         make_reg
normal_exit_label
   clr         bounce_reg
   ld          Key_temp_reg,#%FF
   clr         count_reg
exit_label:
   pop         RP
   iret

   .end
 *******************************************************************************
**
**
** Application:Motor Controller
** Date:       August 07, 1999
** Created by: J. D. Gersbacher for Motor Controller Application Note
** Description:When the motor is running, The sample ISR causes the
** PWM cycle to go High. T1 ISR causes the PWM signal to go Low.
**
**                                          _________                _______
**            PWM:                         |         |XXX|_____________|
**                                         sample   T1
**  ISR
*******************************************************************************
   subtitle “T1 Interrupt Service Routine Values”
   .page

   ;************** Low PWM routine **********************************
   subtitle “Global Functions”
   .page
   ;************** Global Functions ************************************

   public pwmlow_ISR
   segment code

   ;************** ISR Routine  *************************************
pwmlow_ISR:
   and         P0,#%FB                  ;1111_1011 which clears the PWM bit
   iret
 *******************************************************************************
**
**
** Application:Motor Controller
** Date:       August 07, 1999
** Created by: J. D. Gersbacher for Motor Controller Application Note
** Description:When an error condition exist, either the overcurrent or
** the overtemperature, the motor must stop. Stopping
** the motor allows the error condition to correct itself,
** and the system continues normally, from a stop state.
** The most recent Timer 1 value is the starting value for the PWM.
**
*******************************************************************************
   subtitle “ERROR Interrupt Service Routine Values”
   .page
   .include “motorhead.inc”

   ;************** Over current routine *********************************
   extern State_Reg

   subtitle “Global Functions”
   .page
   ;************** Global Functions ************************************

   public overtemperature_ISR,overcurrent_ISR
   segment code

   ;************** ISR Routines  *************************************
overtemperature_ISR:
overcurrent_ISR:

   ld          P0, #motor_stop          ;A High on the Brake pin and PWM stops the mtr
   ld          State_Reg, stopped_state
   ld          P2, #(%ff^stopped_LED)   ;Stop the motor and show it.
   ld          IMR,#tmr0_irq

   iret
;*******************************************************************************
;**
;**FILE:       sampleISR.inc
;**DATE:       August 7, 1999
;**Author: J. D. Gersbacher
;**Description:
;**


   define regfile_sample_ISR, space=RFILE

;*******************************************************************************
;**
;**FILE:       Motorhead.inc
;**DATE:       August 7, 1999
;**Author: J. D. Gersbacher
;**Description:Motorhead.inc contains the variables used in the Motor Controller
;**            Application Note.

   title       “Z8 Motor Controller Application Note”

   define regfile, space=RFILE

Working_Regs.equ%20
Working_ISRRs.equ%30 ;working Interrupt Status Routine Registers
bounce_count.equ%ff

increase.equ   %01
decrease.equ   %02
start_stop.equ %03
dir_switch.equ %01
switches.equ   %0E

minimum             .equ     %13
maximum             .equ     %D5
brake_bit           .equ     %02
forward_direction   .equ     %01
reverse_direction   .equ     %FE
```

```
stopped_LED         .equ     %40
reverse_LED         .equ     %20
forward_LED         .equ     %10
stopped_state       .equ     %00
startup_state       .equ     %01
running_state       .equ     %02
motor_stop          .equ     %07

PWM_Half_value      .equ     %80
tmr0_value          .equ     250
tmr0_irq            .equ     %04
OCE                 .equ     %04        ;OverCurrent Enable
OTE                 .equ     %08        ;OverTemperature Enable, falling edge
T0E                 .equ     %10        ;Timer 0 Enable
T1E                 .equ     %20        ;Timer 1 Enable
```

Flow Chart

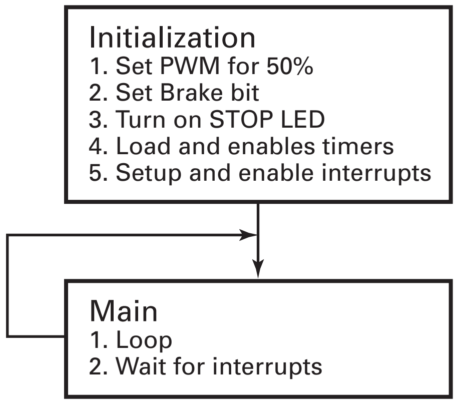

Figure 4. Main Routine

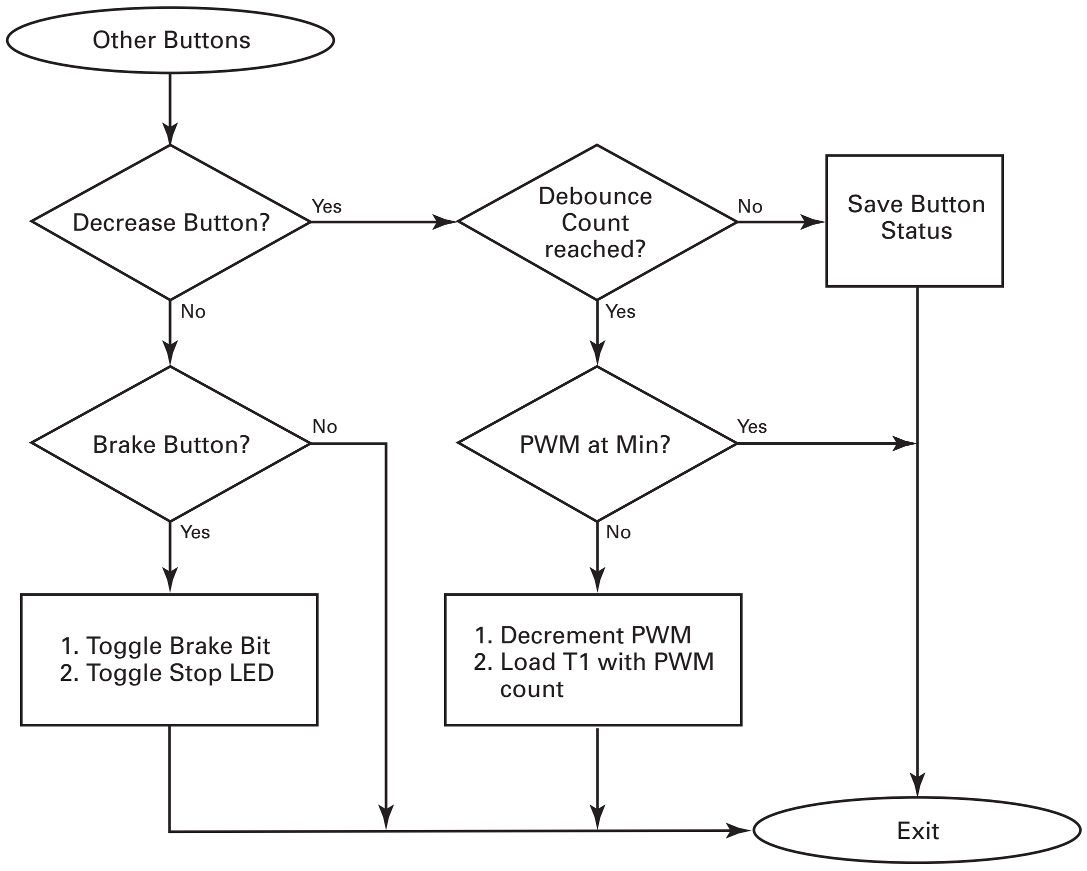

Figure 5. T0 Interrupt: Other Button Routine

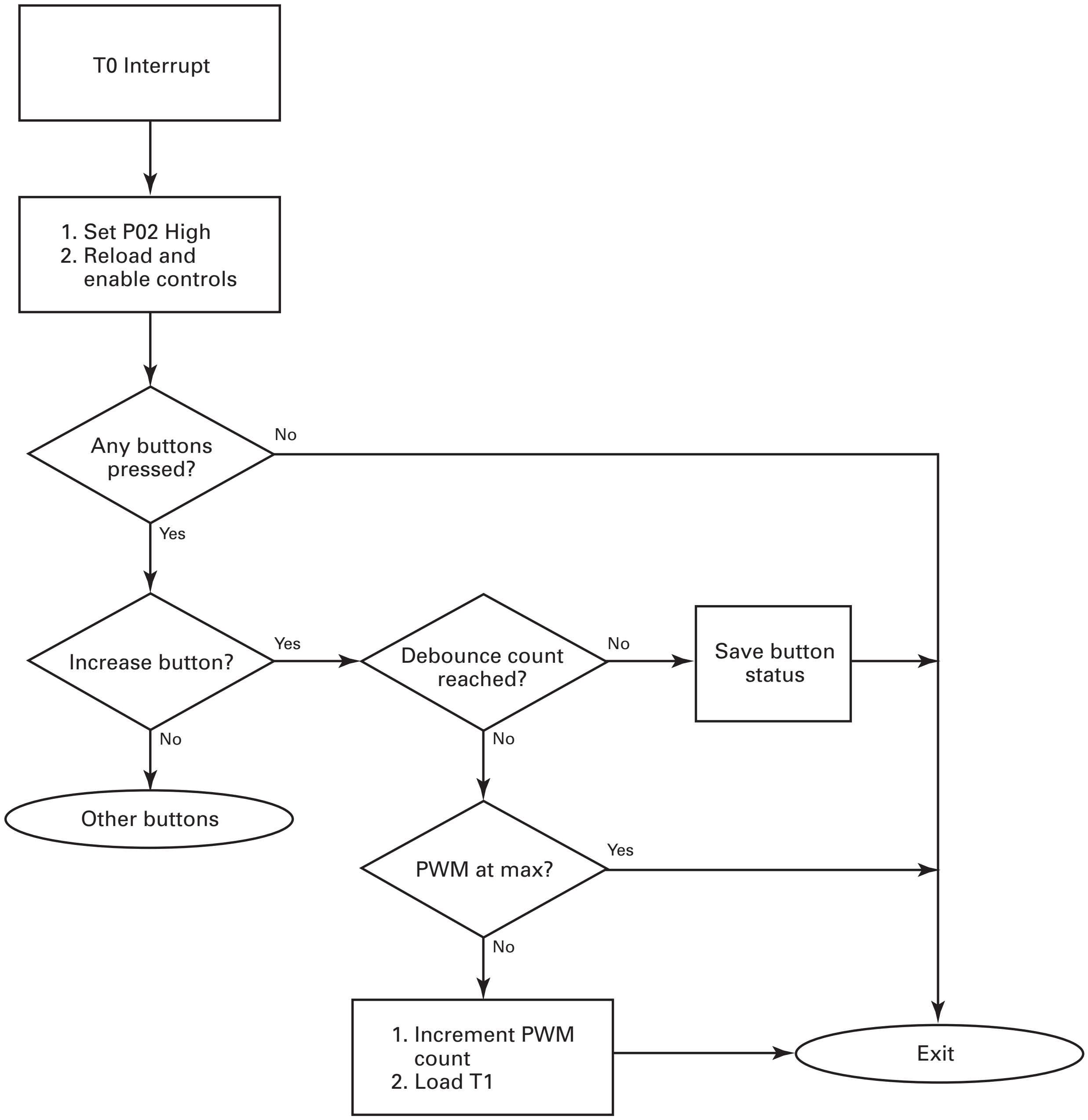

Figure 6. T0 Interrupt

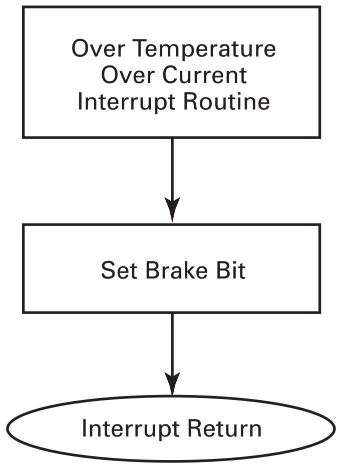

Figure 7. Error Routine

TECHNICAL DRAWINGS

Test Procedure

Equipment Used

- Z86CCP01ZEM—ZiLOG emulator or equivalent
- ZDS version 2.12 or later—ZiLOG assembler, linker, debug environment
- Tektronix 2230 digital storage oscilloscope
- PC running Windows 95

General Test Setup and Execution

1. Follow the ZiLOG Developer Studio (ZDS) setup procedures for installation onto the PC. Follow the Z86CCP01ZEM connection directions for proper operation.
2. When ZDS is running and the emulators are operating, assemble and link the code for debug operation. Use the default settings for assembly and linking. In addition to the default settings, use the toolbar icon for Project → Settings → Linker. Select the Ranges category.

Enter:

a.	regfile	address `0010h–001Fh`
b.	regfile_Motor	address `0020h–002Fh`
c.	regfile_sample_ISR	address `0030h–003Fh`

3. Use an oscilloscope to ensure the proper operation of the PWM, stop bits and the LED output pins.

4. When the signals are verified, connect the emulator to the system board. Run the software and verify that the LEDs illuminate, the push-buttons control the motor, and the motor rotates in the forward and reverse directions.
5. Place a brake point in the overtemperature_ISR. On the unit under test (UUT), momentarily place a ground level at the temperature input, P31, Pin 8. The grounded pin should cause the emulator to stop at the break point. The grounded pin represents LMD18200 operation during an overtemperature condition. Remove the short, remove the break point, and reset the software.
6. To test the overcurrent condition, place a break point in the overcurrent_ISR routine. Start the emulator running and start the motor. Increase the motor to its maximum speed. Stop the motor. When the motor winds to a stop, grasp the end of the motor shaft, and start the motor. The emulator should stop in the overcurrent_ISR. Remove the load from the shaft, remove the breakpoint, and reset the software.
7. Upon completion of the above checks, program a Z86E0408PSC, using the Z86CCP01ZEM. To program the Z86E04 from the CCP emulator, remove the UUT from the CCP emulator, and download the code into the CCP emulator. Place the Z86E04 in the 18-pin ZIF socket. From the toolbar on the top of the ZDS environment, select the OTP icon. Select the programming option icon, and begin programming. When the part is programmed, put the new part into the UUT, and repeat test steps 4–7. After the part is programmed, the steps that require setting breakpoints no longer apply. Instead of looking for the breakpoint to be reached, the motor stops operating, and the STOP LED illuminates.

Test Results

The motor controller controls a 12V_{DC} motor. When power is applied, and the power switch moved to the ON position, the motor controller places the motor into a STOPPED condition and illuminates the RED LED. When the START/STOP button is pushed, the motor applies a 50% duty cycle to the motor. As the INCREASE button is pushed, the motor speeds up until a maximum speed is attained. The duty cycle is 570ms of 630ms. As the DECREASE button is pushed, the motor slows until a minimum speed is reached. The duty cycle is 180ms of 630ms. For an overtemperature condition, a short to ground on the P32 port pin causes the motor to stop running, and the STOP LED to illuminate. For the OVERCURRENT condition, placing a very large load on the shaft causes the motor to stop running and the STOP LED to illuminate.

REFERENCES

1. Chuck McManis, *A PIC-Based Motor Speed Controller*, Circuit Cellar Ink, July 1995.
2. Jeff Bachiacchi, *Creating the Smart-MD*, Circuit Cellar Ink, September-October, 1995.
3. National Semiconductor, LMD18200 data sheet.
4. ZiLOG, *ZiLOG Discrete Z8® Microcontroller Product Specifications Databooks*, DB97Z8X0200.
5. ZiLOG, *ZiLOG Z8® Microcontroller User's Manual*, AP96DZ80500.

APPENDIX

Schematics

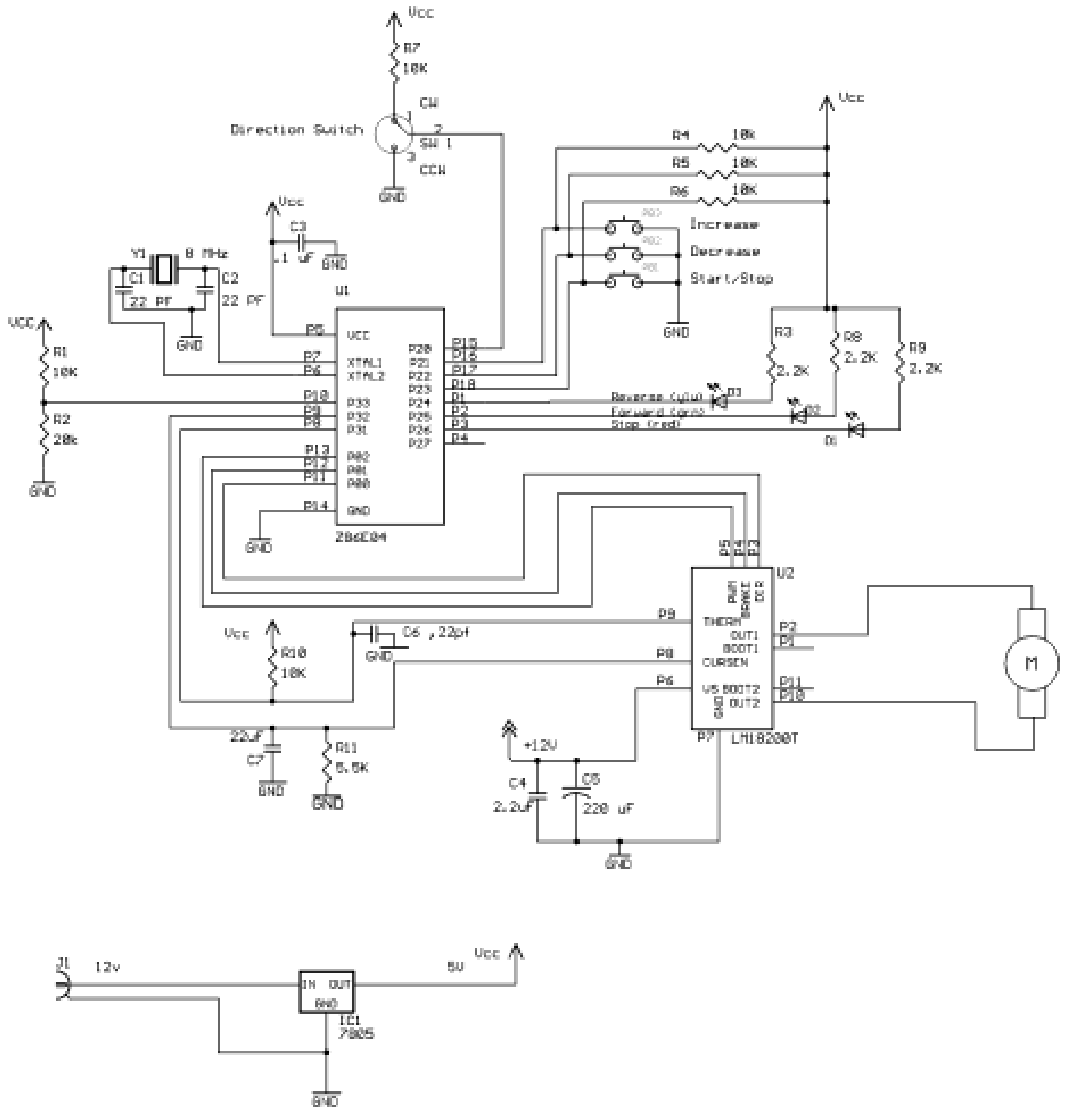

Figure 8. Z86E04 Schematic Diagram

NOTES