Application Note

How to Interface a Seven-Segment Display with the Z8 Encore!® MCU

AN018901-0604

ZiLOG Worldwide Headquarters • 532 Race Street • San Jose, CA 95126
Telephone: 408.558.8500 • Fax: 408.558.8300 • www.ZiLOG.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
532 Race Street
San Jose, CA 95126
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

Table of Contents

List of Figures

List of Tables

Abstract

This Application Note demonstrates how to multiplex and interface four 7-segment LEDs with the Z8 Encore!® MCU to form a display for a digital clock.

A GPIO port on the Z8 Encore!® MCU is used to drive the seven segments of the display. Four additional GPIO ports are used to select the individual digits. In this application, real time is displayed using the seven-segment display with a feature to adjust and set the time.

Z8 Encore!® Flash Microcontrollers Overview

ZiLOG's Z8 Encore!® products are based on the eZ8 CPU and introduce Flash memory to ZiLOG's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows for faster development time and program changes in the field. The high-performance register-to-register based architecture of the eZ8 core maintains backward compatibility with ZiLOG's popular Z8 MCU.

Z8 Encore!® MCUs combine a 20 MHz core with Flash memory, linear-register SRAM, and an extensive array of on-chip peripherals. These peripherals make the Z8 Encore!® MCU suitable for a variety of applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

Discussion

Light Emitting Diodes (LED), as the name implies, are diodes that emit light when forward-biased. LEDs are used to display information. Commonly-used LED displays are offered as seven-segment displays and dot-matrix displays. The seven-segment display is arranged in the form of the digit 8 and can display numbers and hexadecimal characters (0–F).

Seven-Segment Display Construction

A seven-segment display is a group of eight LEDs arranged in segments. These segments are generally used to display the numbers 0 through 9 and the letters A through F. The display features a common-anode or common-cathode orientation as shown in Figure 1.

In this Application Note, a GPIO port on the Z8 Encore!® MCU transmits 8 bits of data. These bits are applied to the seven-segment display to cause it to illuminate the appropriate segments to display the proper number or character.

The seven-segment display used in this Application Note is of the common-cathode type. The common cathode is connected to a transistor, and the anode terminals of the LEDs are connected to Port E[0:7] through current limit resistors.

Segment intensity is dependent on the current flow and should not exceed the limit of the segment.

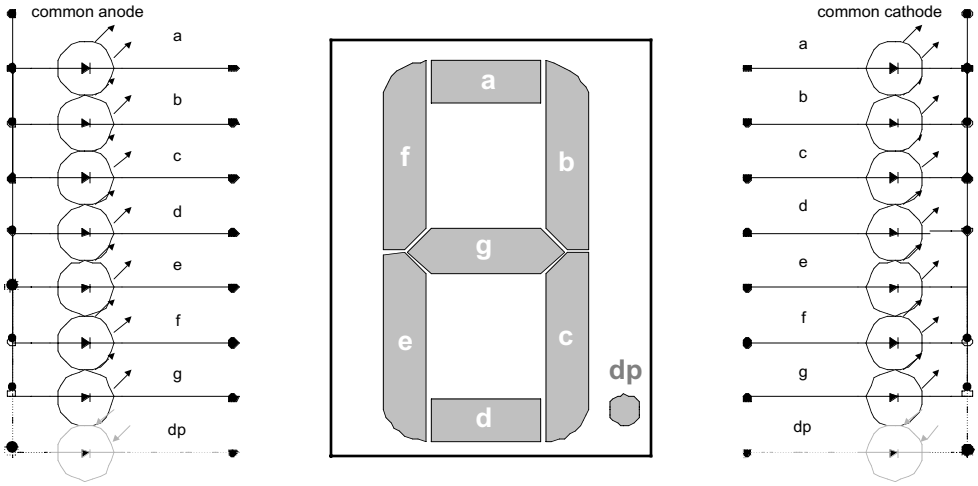

Figure 1. A Seven-Segment Display

The seven-segment unit illustrated in Figure 1 is used to display a single number (0—) or a letter (A–F). For other applications such as digital timers, clocks, and other displays, several units can be aligned next to each other to form a larger panel.

The Hardware Connection

For the seven-segment display, one GPIO pin is assigned per LED. Some examples of character display are shown in Figure 2.

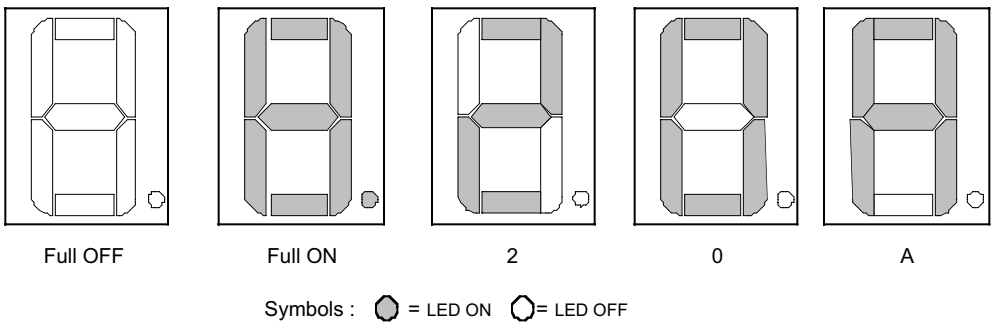

Figure 2. Displayed Characters

For a certain character, a combination of LED ON and LED OFF is generated to display the character for a short period of time. The pattern is loaded alternately to display other characters. For example, to display the number 2, LEDs **a**, **b**, **d**, **e**, and **g** are illuminated. Table 3 provides the display pattern for numbers and characters A through F.

Table 3. Display Pattern for Characters 0–F

Characters	DP	G	F	E	D	C	B	A	Hexadecimal
0	0	0	1	1	1	1	1	1	3F
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	0	1	1	1	1	6F
A	0	1	1	1	0	1	1	1	77
B	0	1	1	1	1	1	0	0	7C
C	0	0	1	1	1	0	0	1	39
D	0	1	0	1	1	1	1	0	5E
E	0	1	1	1	1	0	0	1	79
F	0	1	1	1	0	0	0	1	71

To control four 7-segment displays, multiplexing can reduce the number of GPIO pins required.

In this setup, the four multiplexed seven-segment displays are turned on one at a time to output the appropriate display. Because of the visual phenomenon known as *persistence of vision*, rapid switching of the seven-segment display can appear as if all four displays are turned on.

Figure 3 illustrates the procedure for displaying a four-digit value in the multiplexed seven-segment display. In this example, to be able to display the value 1258, number 8 is displayed first by inputting the pattern of number 8. Next, the NPN transistor connected to the corresponding seven-segment display is switched on in a short period of time while the other transistors remain turned off.

Next, the display is turned off, and the following pattern is input to turn on the second transistor. This pattern repeats until the four digits alternately display. Rapid switching produces the illusion that all four 7-segment displays are turned on.

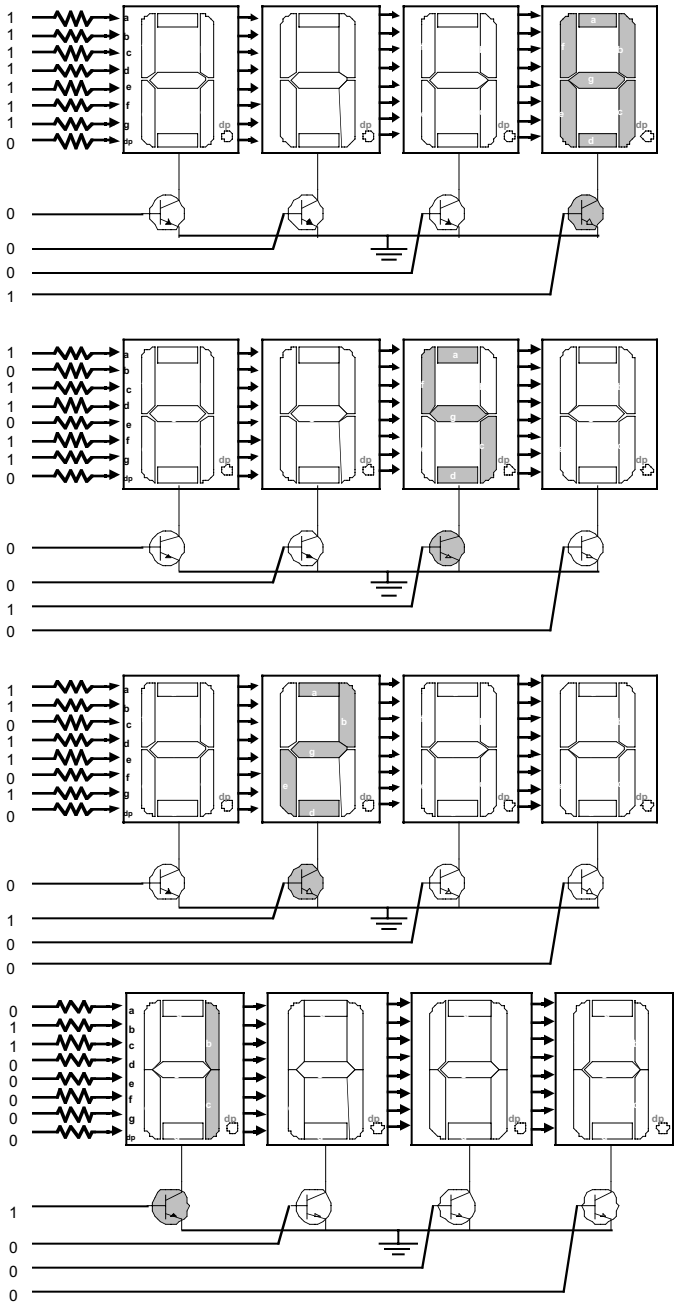

Figure 3. Displaying a Four-Digit Number in a Multiplexed Segment

Figure 3. Displaying a four digit number in a multiplexed seven-segment display

Interfacing Four 7-Segment Displays with the Z8 Encore!® MCU

The following sections describe the hardware and software components required to build the keypad routine.

Hardware Architecture

Figure 4 illustrates a block diagram of the hardware architecture featuring the Z8 Encore!® MCU and four 7-segment displays.

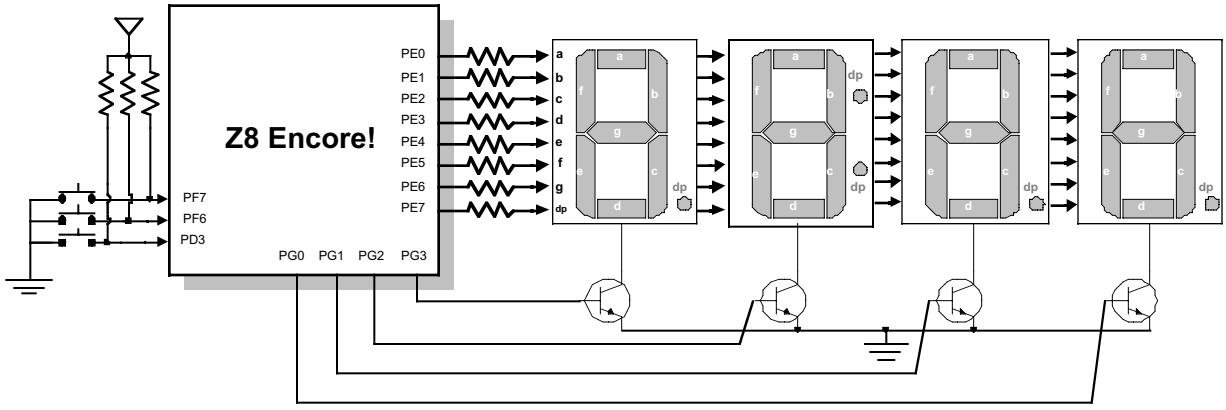

Figure 4. Block Diagram of Seven-Segment Displays with the Z8 Encore!® MCU

GPIO Port E [0:7] is used to drive a segment LED in the four display units in a multiplexed fashion. Four NPN transistors are used to turn on or off the display units individually. Three GPIOs—PD3, PF6, and PF7—are used to adjust the time as input pins and serve as a source of external interrupt. A pull-up resistor and a normally open pushbutton switch are connected at each of these pins as shown in Figure 4.

Software Implementation

In this application, the following Z8 Encore!® GPIO pins are used —8 Port E pins for driving the segments of the display, 4 Port G pins to alternately turn on or off each seven-segment display, and Port D Bit 3 as an interrupt source to select a mode that allows an adjustment of hours or minutes. Port E [0:7] outputs either 1 or 0 depending on the defined pattern to display the required character. PF6 is used to decrement the current display of hour or minutes, while PF7 is used to increment the value.

Port E and Port G are initialized as outputs and Port D Bit 3 is configured as an interrupt source and set as a high priority.

Timer 0 is programmed in to operate in a continuous mode to generate a timer output of 100 ms. The reload value is computed based on the following formula:

```
Continuous Mode Time-Out(s) = (Reload value x Prescale value) ÷ System
Clock (Hz)
```

A Reload Value High and Low Byte equal to `38h` and `40h`, respectively, are used with a prescale value of `128` for the system running on an 18.432 MHz system clock.

To produce accurate timing, a time-out of one second is required to update the display. To achieve this one-second timing, Timer 0 produces a time-out of 100ms duration, and a timer interrupt counter is used to count from 0 to 10 before it updates the display. After every change in value, the ones and tens values of the minutes and hours are extracted to get the data to be displayed. The extracted value is transferred to a register to change or update the data during every scan. After each scan, the values are updated and displayed by getting the pattern in the provided array.

Initially, the display is set to zero and waits for a falling edge interrupt in PD3. Pressing SW1 (PD3) changes the mode of the real-time clock (RTC). Three modes allow the user to configure the RTC and adjust for the correct time. Mode 0 displays the current time. Mode 1 allows the user to either increment or decrement the value of minutes displayed. The value of the hour can be modified by entering Mode 2. Pressing SW1 (PD3) three times resets the mode and returns to normal operation.

To adjust the value at any time, Timer 1 is used to monitor whether switches SW2 and SW3 (PF6 & PF7) are pressed. Timer 1 also operates in continuous mode and has a time-out of 100 ms. The formula for Timer 0 above is used. Port F bits 6 and 7 are initialized as inputs, and these bits are also used as interrupt sources. The MCU receives data from Port F inputs every two interrupts and then processes the data. Pressing the switch connected to PF6 decrements the displayed value while SW3 (PF7) increments the value.

Functions processed by the two timers work independently in the background. The main program continually refreshes and updates the display depending on the data transmitted by the two timers.

The **display()** routine called in the **main** program refreshes the display by performing the following tasks:

- Increments a scan counter that provides the timing required to shift the digits to be displayed, and resets the counter when it reaches the terminal count
- Calls the **get_data** API (extracts the digit from the hour and minute variables)
- Calls the **transfer_data** API (transfers the extracted digits to a register)

- Calls the **update()** API (gets the pattern from an array and outputs the data)

The **count()** routine is called every one second and is generated by a Timer 0 interrupt. It updates the seconds, minutes, and hours values.

The **adjust_time()** routine allows the user to adjust the current time by pressing control buttons.

Note: The parameters and returns of each of the above functions is void.

Testing

The following sections describe the equipment and procedure required to test the seven-segment display interface with the Z8 Encore!® MCU.

Equipment Used

The following equipment was used for testing:

- Z8 Encore!® Development Kit (Z8ENCORE000ZCO)
- ZDSII IDE–Z8 Encore!®
- Four SG511 7-segment displays
- NPN transistors

Test Setup

To build the application, observe the following instructions.

1. Set up the connection between the Z8 Encore!® Development Board and the seven-segment display as shown in Figure 4.
2. Run the ZDSII IDE for Z8 Encore!®. Apply power to the Z8 Encore!® Development Board and download the source code.
3. Run the program.
4. The display is initially set to zero. Press SW1 (PD3) to adjust the display. To adjust to the correct minutes value, press switches PF6 and PF7.
5. Press SW1 a second time to enter the hours mode. To adjust to the correct hours value, press switches PF6 and PF7.
6. Press SW1 a third time to resume normal operation.

Note: The seven-segment display used in this application is a common cathode. If you are using a common anode, please refer to the source code for some modifications in the program.

Test Results

The results were as expected, as accurate timing was achieved in displaying the time on the four 7-segment displays. The time was entered and adjusted correctly using the provided control switches.

Summary

This Application Note explains how to control a multiplexed seven-segment display along with a feature to set the time of the clock using the GPIO ports, the built-in timers, and the peripherals of the Z8 Encore!® MCU with minimal external circuitry.

Appendix A—Reference

Related documents that may be useful are listed in Table 4.

Table 4. Related Document References

Topic	Document
eZ8 CPU	eZ8 CPU User Manual (UM0128)
Z8 Encore!® CPU	Z8 Encore!® Microcontrollers with Flash Memory and 10-Bit A/D Converter Product Specification (PS0176)
ELS-511 Display	S511GWA.pdf from www.everlight.com

Appendix B—Flowcharts

Figure 5 illustrates the flow of the main routine that interfaces a seven-segment display to the Z8 Encore!® MCU's GPIO ports, which are initialized to output the seven-segment display. Timer 0 is configured in continuous mode to set the timing of the clock. Port D is also initialized to detect a falling edge interrupt to set the mode of the real-time clock. Port F inputs are used to enter the correct time.Timer 1 also operates in continuous mode to monitor whether the switches are pressed to adjust the time. The display is updated at one second intervals.

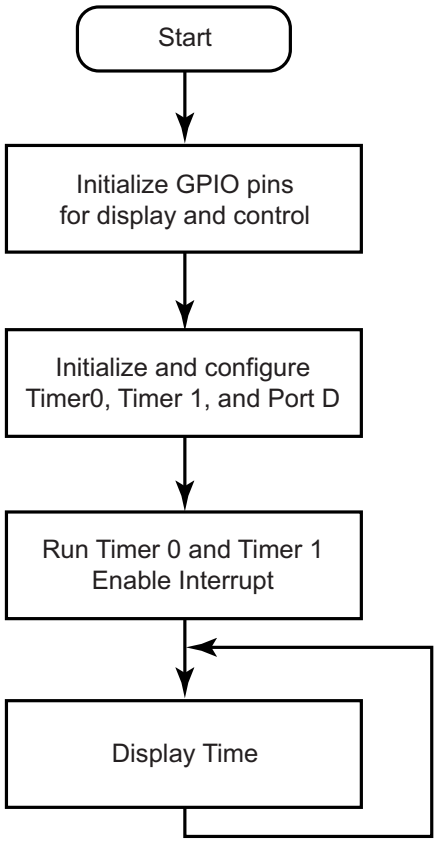

Figure 5. Main Routine Flow

Figure 6 illustrates the flow of the Port D interrupt. When a falling edge is detected, the mode is incremented. On the third interrupt, the mode resets to zero.

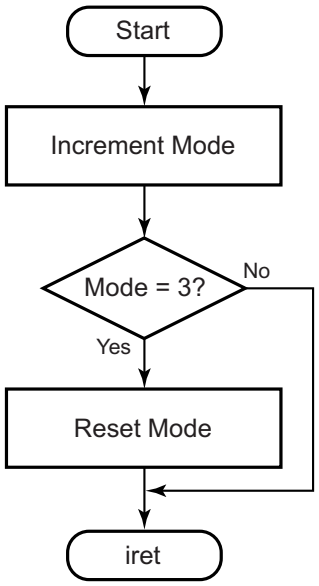

Figure 6. Port D3 Interrupt Flow

Figure 7 illustrates the flow of the Timer 0 routine. Timer 0 operates in continuous mode to produce accurate clock timing. A counter is incremented every 100 ms, and counts 10 occurrences to result in a one-second time-out. Next, it increments the seconds counter. Every 5 counts, the **blink_colon** flag is toggled to produced a blinking display. When the seconds counter reaches 60 counts, the minute counter is incremented. On the 60th count, the hour counter is incremented until it reaches 24 counts and the cycle continues.

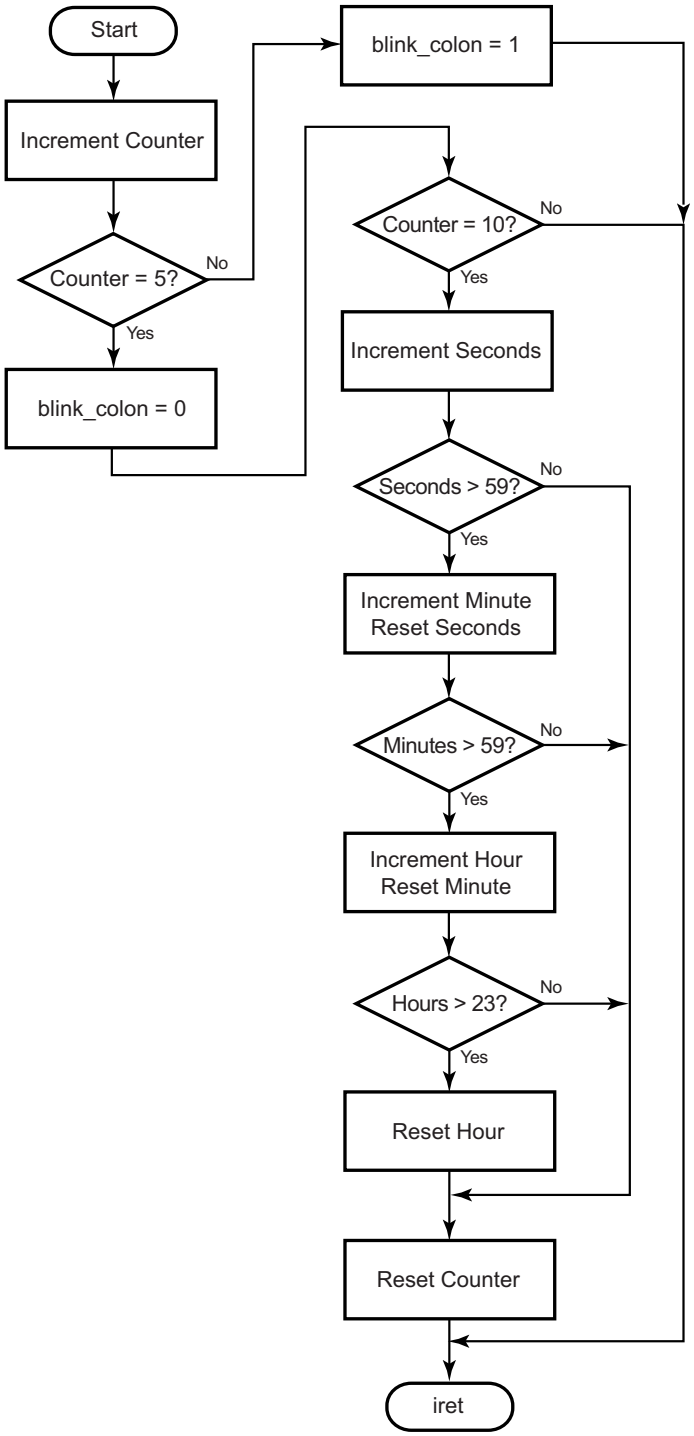

Figure 7. Timer 0 Flow

Figure 8 illustrates the flow of the Timer 1 routine. Timer 1 operates in continuous mode with a time-out value of 100 ms. For every two interrupts, it checks the mode of the real-time clock. The displayed time can be either incremented or decremented using the switches.

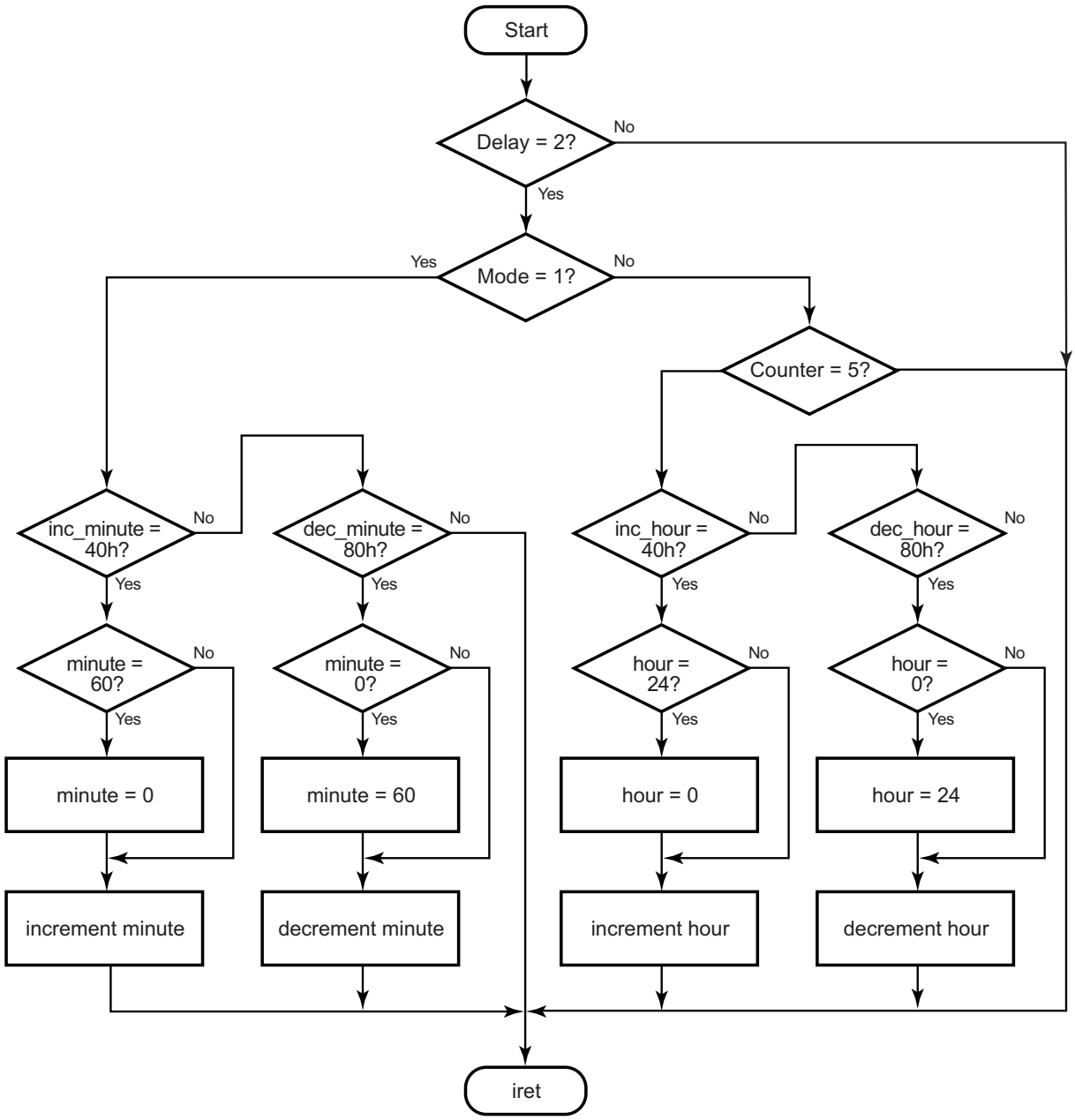

Figure 8. Timer 1 Flow

Appendix C—Source Code

This appendix contains a listing of the C and header code for the keypad routine discussed in this document. A complete listing is available in the AN0200-SC01.zip file.

C Files

This section lists the code for the following C files:

- main.c
- function.c
- gpio.c
- interrupt.c

```
/*****************************************************
* File: main.c
* Description: This file contains the main program of
* controlling the seven-segment display

*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However, users are cautioned to authenticate the
* code contained herein.

* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*****************************************************/

#include <stdio.h>
#include <ez8.h>
#include "gpio.h"
#include "interrupt.h"
#include "functions.h"


main()
{
```

```
  init_led_gpio();                  //Initialize Port E for 7-segment
                                    //display, Port G for scanning
                                    //Port F for input
  init_timer0();                    //initialize Timer 0
  init_timer1();                    //initialize Timer 1
  init_p3ad();                      //initialized PD3 for interrupt

  EI();                             //Enable Interrupt
  T1CTL |= 0x80;                    //Run Timer 1

  while(1)                          //infinite loop
  {
   display();                       // update display
  }
}

/*****************************************************************
*********************End of main.c file***************************
*****************************************************************/



/*****************************************************
* File: functions.c
* Description: This file contains all the functions used in the
* program
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.

* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*****************************************************/
#include <stdio.h>
#include <ez8.h>
#include "functions.h"
```

```
/*Values loaded to the 7-segment display(common cathode)*/

const char value[13] =
{0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x80,0x00,0x40};
//values 0 1 2 3 4 5 6 7 8 9 . "off" -


/*Indicates the position of the seven-segment display to be scan*/

const char position[5] = {0x01,0x02,0x04,0x08,0x04};

//position(T-transistor) T1 T2 T3 T4 T3

/*Note: If you are using a common anode, use the following values:

{0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90}
0 1 2 3 4 5 6 7 8 9 */

/************************************************************
 Initialized variables
************************************************************/

int minute_L = 0;            //initialized minute_L to zero
int minute_H = 0;            //initialized minute_H to zero
int hour_L = 0;              //initialized hour_L to zero
int hour_H = 0;              //initialized hour_H to zero

int seconds = 0;             //initialized seconds to zero
int minute = 0;              //initialized minute to zero
int hour = 0;                //initialized hour to zero

int scan = 0;                //initialized scan to zero
int mode = 0;                //initialized mode to zero

int inc_hour, dec_hour;      // initialized inc_hour, dec_hour
int inc_minute, dec_minute;  // initialized inc_minute, dec_minute
int data;                    // initialized data

extern int blink_colon;      //defined in interrupt.c


/*****************************************************************
 This function counts seconds, minutes, and hours to be displayed
```

```
*****************************************************************/
void count (void)
{
  seconds++;                          // increment seconds counter
  if (seconds > 59)                   // check if seconds counter is
                                      // greater than 59
  {
   minute++;                          // increment minute counter
   seconds = 0;                       // reset seconds
  }
  if (minute > 59)                    // check if minute counter is
                                      // greater than 59
  {
   hour++;                            // increment hour counter
   minute = 0;                        // reset minute
  }
  if (hour > 23)                      // check if minute counter is
                                      // greater than 23
  {
   hour = 0;                          // reset hour
  }
}

/*****************************************************************
  This function displays the time
*****************************************************************/

void display (void)
{
  scan ++;                            // increment scan counter
  if (scan >= 6)                      // check if already equal to 6
  {
   scan &= 0;                         // reset scan counter
  }
  else
  {
   get_data();                        // call get_data function
   transfer_data();                   // call transfer_data function
   update();                          // call update function
  }
}
```

```
/*************************************************************
 This function extracts the data from a register
*************************************************************/

void get_data (void)
{
  minute_L = (minute)%10;           // gets the LSB of minute
  minute_H = (minute)/10;           // gets the MSB of minute
  hour_L = (hour)%10;               // gets the LSB of hour
  hour_H = (hour)/10;               // gets the MSB of hour
}



/*************************************************************
 This function transfers the data to be displayed, every after scan
*************************************************************/

void transfer_data (void)
{
  if((scan == 1)&&(mode == 0))      // if scan equals 1 and mode = 0
  data = minute_L;                  // transfer the value of minute_L
                                    // to data register
  else if((scan == 1)&&(mode == 1)) // if scan equals 1 and mode = 1
  data = minute_L;                  // transfer the value of minute_L
                                    // to data register
  else if((scan == 1)&&(mode == 2)) // if scan equals 1 and mode = 2
  data = 12;                        // this displays "--" to the minute
                                    // display
  else if((scan == 2)&&(mode == 0)) // if scan equals 2 and mode = 0
  data = minute_H;                  // transfer the value of minute_H
                                    // to data register
  else if((scan == 2)&&(mode == 1)) // if scan equals 1 and mode = 1
  data = minute_H;                  // transfer the value of minute_L
                                    // to data register
  else if((scan == 2)&&(mode == 2)) // if scan equals 1 and mode = 2
  data = 12;                        // this displays "--" to the minute
                                    // display
  else if((scan == 3)&&(mode == 0)) // if scan equals 3 and mode = 0
  data = hour_L;                    // transfer the value of hour_L to
                                    // data register
  else if((scan == 3)&&(mode == 1)) // if scan equals 3 and mode = 1
  data = 12;                        // this displays "--" to the hour
                                    // display
```

```
  else if((scan == 3)&&(mode == 2))  // if scan equals 3 and mode = 2
  data = hour_L;                     // transfer the value of minute_L
                                     // to data register
  else if((scan == 4)&&(mode == 0))  // if scan equals 4 and mode = 0
  data = hour_H;                     // transfer the value of hour_H to
                                     // data register
  else if((scan == 4)&&(mode == 1))  // if scan equals 4 and mode = 1
  data = 12;                         // this displays "--" to the hour
                                     // display
  else if((scan == 4)&&(mode == 2))  // if scan equals 4 and mode = 2
  data = hour_H;                     // transfer the value of minute_L
                                     // to data register
  else if((scan == 5)&& (blink_colon == 1))   // if scan equals 5 and
                                              // blink_colon equals 1
  data = 10;                                  // turn on colon
  else if((scan == 5) && (blink_colon == 0))  // if scan equals 5 and
                                              // blink_colon equals 0
  data = 11;                                  // turn off colon
}


/*****************************************************************
  This function updates the value to be displayed
*****************************************************************/

void update (void)
{
  PEOUT &= 0x00;                 // reset the values loaded to port E
  PGOUT = position[scan - 1];    // turn on the seven-segment display
  PEOUT = value[data];           // outputs the updated data
}


/*****************************************************************
  This function is use to update the minute display
*****************************************************************/
void update_minute(void)
{
 if (mode == 1)                      // if mode equals 1
 {
   inc_minute = (PFIN & 0x40);       // check if PF6 is press
   dec_minute = (PFIN & 0x80);       // check if PF7 is press

   if (inc_minute == 0x40)           // if PF6 is pressed
```

```
  {
   if (minute > 59)                 // check if minute is greater than
                                    // 59
   {
    minute = 0;                     // reset minute
    minute++;                       // increment minute
   }
   else
   minute++;                        // increment minute
  }
  if (dec_minute == 0x80)           // if PF7 is pressed
  {
   if (minute == 0)                 // check if minute equals zero
   {
    minute = 59;                    // minute equals 59
   }
   else
   minute--;                        // decrement minute
  }
 }
}

/*************************************************************
  This function is use to update the hour display
*************************************************************/


void update_hour(void)
{
 if (mode == 2)
 {
   inc_hour = (PFIN & 0x40);        // check if PF6 is press
   dec_hour = (PFIN & 0x80);        // check if PF7 is press

   if (inc_hour == 0x40)            // if PF6 is pressed
  {
   if (hour >23)                    // if hour greater than 23
   {
    hour = 0;                       // reset hour
    hour++;                         // increment hour
   }
   else
   hour++;                          // increment hour
  }
```

```
  if (dec_hour == 0x80)             // if PF7 is pressed
  {
   if (hour == 0)                   // if hour equals zero
   {
    hour = 23;                      // hour equals 23
   }
   else
   hour--;                          // decrement hour
  }
 }
}

/*************************************************************
******************End of function.c file**********************
*************************************************************/



/*******************************************************
* File: gpio.c
* Description: This file contains the initialization of
   GPIO ports used
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.

* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*******************************************************/
#include <stdio.h>
#include <ez8.h>
#include "definition.h"


void init_led_gpio(void)     //Initialize Port E for 7-segment display
{
 PEADDR = ALT_FUN;           //Port E Alternate Function
```

```
 PECTL = 0x00;              //No Alternate Function

 PEADDR = OUT_CTL;          //Port E Output Control
 PECTL = 0x00;              //Output is configured as push pull

 PEADDR = DATA_DIR;         //Port E Data Direction
 PECTL = 0x00;              //Port E is Output

 PEADDR = 0x00;             //Reset, protection against accidental
                            //port reconfiguration


 PGADDR = ALT_FUN;          //Port G Alternate Function
 PGCTL = 0x00;              //No Alternate Function

 PGADDR = OUT_CTL;          //Port G Output Control
 PGCTL = 0x00;              //Output is configured as push pull

 PGADDR = DATA_DIR;         //Port G Data Direction
 PGCTL = 0x00;              //Port G is Output

 PGADDR = 0x00;             //Reset,protection against accidental port
                            //reconfiguration


 PFADDR = ALT_FUN;          //Port F Alternate Function
 PFCTL = 0x00;              //No Alternate Function

 PFADDR = DATA_DIR;         //Port F Data Direction
 PFCTL = 0xC0;              //Port F bits 6 & 7 as inputs

 PFADDR = 0x00;             //Reset,protection against accidental port
                            //reconfiguration

 }

/*************************************************************/
/*********************End of gpio.c file**********************/
/*************************************************************/



/*****************************************************
* File: interrupt.c
```

```
* Description: This file contains the initialization of Timer 0
* and Port D used as an interrupt
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.

* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*************************************************************/

#include <stdio.h>
#include <ez8.h>
#include "interrupt.h"
#include "functions.h"
#include "definition.h"

extern void isr_timer0(void);
extern void isr_timer1(void);
extern void isr_p3ad(void);

extern int minute, hour;     //defined in functions.c
extern int mode;             //defined in functions.c


int counter = 0;             //initialized counter to zero
int blink_colon = 0;         //initialized blink_colon to zero
int delay = 0;               //initialized delay to zero
int check;                   //initialized check variable


/****************************************
 Initialization of Port D bit 3
****************************************/
void init_p3ad(void)
  {
  SET_VECTOR(P3AD, isr_p3ad);    // Pass the vector number and the
                                 // ISR address to be placed into
```

```
                                  // the interrupt table */

  PDADDR = ALT_FUN;               // Port D Address, Alternate
                                  // Function.
  PDCTL = 0x00;                   // Port D Control, No Alternate
                                  // Function.

  PDADDR = DATA_DIR;              // Port D Address, Data Direction.
  PDCTL |= 0x08;                  // Port D Control, bit 3 use as input

  IRQ1ENH |= 0x08;                // Highest Priority
  IRQ1ENL |= 0x08;

  IRQES &= 0xF7;                  // Edge select for Interrupt is
                                  // Falling Edge
  IRQPS |= 0x08;                  // Port D3 is selected over A3

  }

/
*********************************************************************/
#pragma interrupt
void isr_p3ad(void)
  {
   check = PDIN;                      // get the data in Port D
   if ((check &= 0x08)== 0x00)        // check if the switch is still
                                      // down
   {
    mode++;                           // increment mode
    T0CTL &= 0x7F;                    // stop Timer 0
   if(mode == 3)                      // if the switch is pressed for the
                                      // third time...
    {
     mode = 0;                        // reset mode to zero
     T0CTL |= 0x80;                   // run Timer 0
    }
   }
  }

/*********************************************************************
  Initialization of Timer 0
*********************************************************************/
void init_timer0(void)
  {
```

```
   SET_VECTOR(TIMER0, isr_timer0);//Pass the vector number and the
                                  //ISR address to be placed into
                                  //the interrupt table */

   T0CTL = 0x39;                  //Timer 0 is in continuous mode
                                  //operation and prescale of 128*/

   T0H = 0x00;                    //Timer 0 High byte register
   T0L = 0x01;                    //Timer 0 Low byte register

   T0RH = 0x38;                   //Reload High
   T0RL = 0x40;                   //Reload Low, time-out = 100ms

   IRQ0ENH |= 0x20;               //Timer 0 is configured
   IRQ0ENL &= 0xDF;               //for nominal priority interrupt
  }

#pragma interrupt
void isr_timer0(void)
  {
   counter ++;                    // counter is incremented
   if (counter <= 5)              // check if counter is less than or
                                  // equal to 5 (500ms)
   {
    blink_colon = 1;              // set blink_colon to 1
   }
   if (counter > 5)               // check if counter is greater than 5
   {
    blink_colon = 0;              // reset blink_colon
   }

   if (counter == 10)             // check if counter is equal to 10
   {
    count();                      // call count function
    counter = 0;                  // reset counter
   }
  }

/*********************************************************************
  Initialization of Timer 1
*********************************************************************/

void init_timer1(void)
{
```

```
  SET_VECTOR(TIMER1, isr_timer1);    // Pass the vector number and the
                                     // ISR address to be placed
                                     // into the interrupt table */

  T1CTL = 0x39;                      // Timer 1 is in continuous mode
                                     // operation and prescale of 128*/

  T1RH = 0x38;                       // Reload High
  T1RL = 0x40;                       // Reload Low, time-out 100ms

  T1H = 0x00;                        // Timer High
  T1L = 0x01;                        // Timer Low

  IRQ0ENH &= 0xBF;                   // IRQ0 Enable High
  IRQ0ENL |= 0x40;                   // IRQ0 Enable Low, Timer1 is low
                                     // priority
}

#pragma interrupt
void isr_timer1(void)
{
  delay++;                           // increment delay
  if (delay == 2)                    // check if delay equals to 2 (scan
                                     // port F every 200ms)
  {
   update_minute();                  // call update_minute function
   update_hour();                    // call update_hour function
   delay = 0;                        // reset delay
  }
}


/*****************************************************************
*********************End of interrupt.c file*********************
*****************************************************************/
```

Header Files

This section lists the code for the following header files:

- definition.h
- functions.h
- gpio.h
- interrupt.h

```
/*************************************************************
* File: definition.h
* Description: contains defined values for sub-registers
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.

* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*************************************************************/

#define DATA_DIR 0x01        // Data Direction
#define ALT_FUN 0x02         // Alternate Function
#define OUT_CTL 0x03         // Output Control
#define HDR_EN 0x04          // High Drive Enable
#define SMRS_EN 0x05         // Stop Mode Recovery Source Enable

/*************************************************************
*********************End of definition.h file*****************
*************************************************************/



/*************************************************************
* File: functions.h
* Description: This file contains prototype declarations.
*
```

```
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.

* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*************************************************************/

void count (void);
void display (void);
void transfer_data (void);
void update (void);
void get_data (void);
void update_minute(void);
void update_hour(void);

/*************************************************************
*********************End of functions.h file******************
*************************************************************/



/*******************************************************
* File: gpio.h
* Description: This file contains prototype declarations.
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.

* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
```

```
*************************************************************/

void init_led_gpio(void);

/*************************************************************
*********************End of gpio.h file**********************
*************************************************************/



/*******************************************************
* File: interrupt.h
* Description: This file contains prototype declarations.
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.

* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*******************************************************/

void init_timer0(void);
void init_p3ad(void);
void init_timer1(void);


/*************************************************************
*********************End of interrupt.h file*****************
*************************************************************/
```