MICROCHIP

AN777

Multi-Tasking on the PIC16F877 with the Salvo™ RTOS

Authors: Chris Valenti
Microchip Technology Inc.

Andrew E. Kalman, Ph.D.
Pumpkin, Inc.

INTRODUCTION

This application note covers a Real-Time Operating System (RTOS) running on a PIC16F877. The application is written in C using the HI-TECH C compiler. MPLAB® IDE is used as the Integrated Development Environment. This RTOS is unique, in that it is intended for microcontroller applications where memory is severely limited. The application runs on a prototype PCB that monitors temperature, accepts user input and displays important temperature information.

RTOS OVERVIEW

Salvo™ is a full featured, cooperative, event driven, priority based, multi-tasking RTOS with highly efficient memory utilization. It is ideally suited for use on Microchip PICmicro® devices. Written in C, it is very easy to use, employing standardized RTOS methods and terminology. This RTOS makes PICmicro programming a breeze, and includes:

- Over 40 callable user services in its API
- Up to 16 separate dynamic task priority levels
- Support for multiple event types
- Timer-based services
- Minimal call … return stack usage
- Low interrupt latency and fast context switching

Every Salvo application must adhere to two “golden rules”:

1. Each task must have at least one context switch.
2. Context switches may only occur in tasks.

For this application, Salvo was user-configured to provide the basic multi-tasking kernel, along with binary semaphore and message event services, as well as timer based delays. It automatically manages complex issues, like task scheduling, access to shared resources, intertask communication, real-time delays, PICmicro RAM banking and interrupt control. With this multi-tasking RTOS foundation in place, the application programmer can concentrate on quickly and efficiently implementing the desired system functionality.

SYSTEM DESCRIPTION

The prototype's hardware includes a 20 MHz crystal, four thermistors, four potentiometers, a serial port, EEPROM, four 7-segment LEDs, 16-button keypad and a piezo beeper. The phrase, “normal conditions,” will be used frequently in this application note, indicating the demo board is in temperature monitoring mode with no alarm or user functions being executed.

The time-base is a 2 ms periodic interrupt derived from Timer1. There are a total of eight tasks, four of which are in the waiting state under normal conditions. There are five events, four of which are dependent upon the status of outside conditions (e.g., keypad entry, alarm) and one is required for resource control.

The thermistors are divided up into four zones (Z1, Z2, Z3, Z4). Each zone will be monitored to check if the temperature is between the low and high threshold temperature range (set by user). The user sets the low and high threshold temperatures by pressing the Low-High program button (see Figure 1).

FIGURE 1: KEYPAD CONFIGURATION

SET POT-1	**SET POT-2**	**SET POT-3**	**DISPLAY ZONE 1**
1	*2*	*3*	
SET POT-4	*5*	*6*	**DISPLAY ZONE 2**
4			
7	*8*	*9*	**DISPLAY ZONE 3**
LOW-HIGH PROGRAM	**EXIT POT SETTING**	**ZONE RECALL**	**DISPLAY ZONE 4**
	0		

The low temperature is entered first, then the high; each entry is followed by a quick display of the entered temperature. A zone that is not within these parameters will set off the Piezo alarm, simultaneously displaying the zone number that set off the alarm. An alarm condition will also signal `Task_Weeprom()` with the zone number. Under normal conditions, once selected, the LEDs will always have a zone temperature displayed. The particular zone on display is dependent upon which zone button was pressed. Buttons 1 through 4 have two functions (see Figure 1), potentiometer selection and numerical. When one of these buttons is pressed (under normal conditions), the current potentiometer value is displayed on the LEDs.

At this point, two actions can be taken: potentiometer adjustment, or press ‘0’ to exit the function. The Zone Recall button is used to display the zone that set off the alarm last. The USART is used for displaying the current temperatures on a PC monitor; this is executed by entering 'z' via the PC keyboard. The USART is configured for Master Asynchronous mode with a 9600 baud rate.

APPLICATION CONFIGURATION

The initial setup for the RTOS involves creating a configuration file and creating an MPLAB project. The Salvo user services are contained in different source files. As code development progresses, more user services are needed, resulting in additional source files being added to the application. The application includes the following files:

- `main.c`
- `binsem.c`
- `chk.c`
- `delay.c`
- `event.c`
- `init.c`
- `mem.c`
- `task.c`
- `util.c`
- `msg.c`
- `timer.c`
- `qins.c`
- `salvo.h`
- `salvocfg.h`

Keep in mind that these files are specific to this application and may not apply to others. Each Salvo application requires its own configuration file called `salvocfg.h`. The default `salvocfg.h` file contains all possible parameters. For this application, specific parameters were stripped out of the default file and put into a application specific `salvocfg.h` file. This file is automatically included via the `salvo.h` header file. The `salvocfg.h` file for this application is shown in Appendix B. Table 1 shows the node property settings in MPLAB IDE.

MEMORY

General purpose RAM is allocated to four parts of the application:

- Global variables.
- Control blocks and other variables.
- Parameter stack and auto variables maintained by the compiler.
- Interrupt saves and restores.

The memory requirements exceed the available memory in RAM Bank 0, so the global variables are placed in Bank 1, and Salvo's variables are placed in Bank 2, using configuration options in `salvocfg.h`. Salvo's message pointers can access memory in any RAM bank and anywhere in ROM. The final code consists of three roughly equal portions: one-third Salvo RTOS, one-third HI-TECH C compiler library functions and one-third application specific code.

TIME-BASE

In an RTOS environment, establishing a true time-base is critical for time-based task operations. In this application, Timer1 triggers an interrupt every 2 ms and is solely used for this periodic interrupt. The ISR calls the `OSTimer()` function and reloads Timer1 for another 2 ms. The 2 ms interrupt is also known as the “system tick rate” and forms the time basis for task delays. Six of the eight tasks rely on `OSTimer()` via `OSDelay()`. Under normal conditions, each task's run time is constant, thus the importance for a time-base. For instance, `Task_Convert()` is configured to run every 40 ms via "`OS_Delay(20);`". In the `salvocfg.h` include file, there is a configuration statement regarding the number of bytes allocated for delays. This configuration option tells the OS what the maximum delay can be:

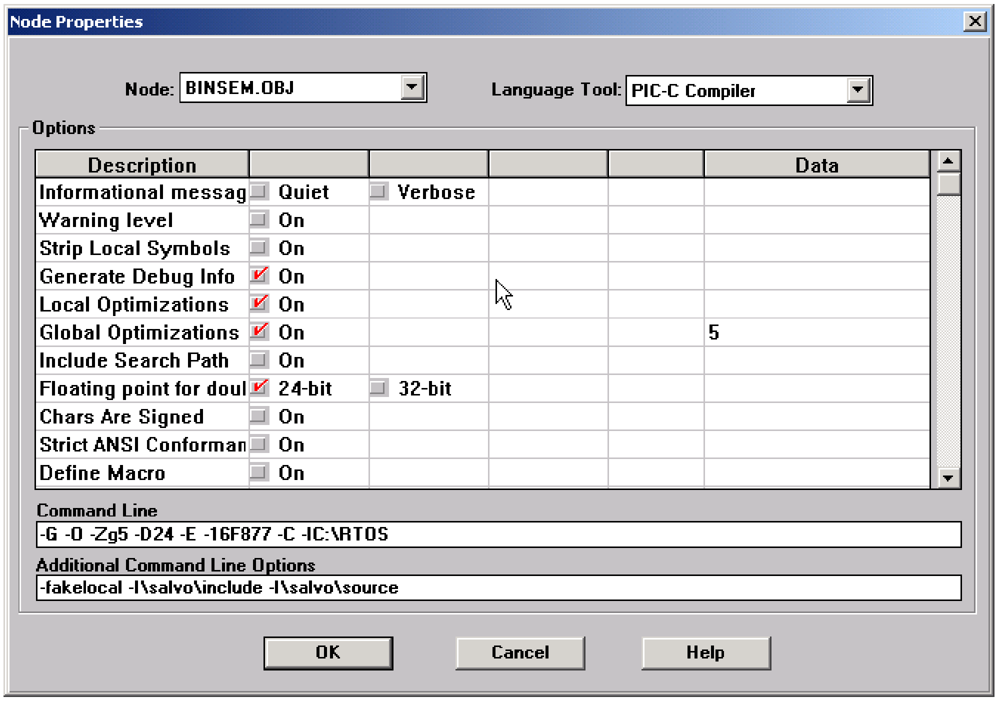
one byte = 2^8-1 ticks

two bytes = 2^{16}-1 ticks, etc.

In this application, we need two bytes.

TABLE 1: MPLAB NODE PROPERTIES

NODE PROPERTIES (.c-FILES)



Node: BINSEM.OBJ   Language Tool: PIC-C Compiler

Options

Description					Data
Informational messag	☐ Quiet	☐ Verbose			
Warning level	☐ On				
Strip Local Symbols	☐ On				
Generate Debug Info	☑ On				
Local Optimizations	☑ On				
Global Optimizations	☑ On				5
Include Search Path	☐ On				
Floating point for doul	☑ 24-bit	☐ 32-bit			
Chars Are Signed	☐ On				
Strict ANSI Conforman	☐ On				
Define Macro	☐ On				

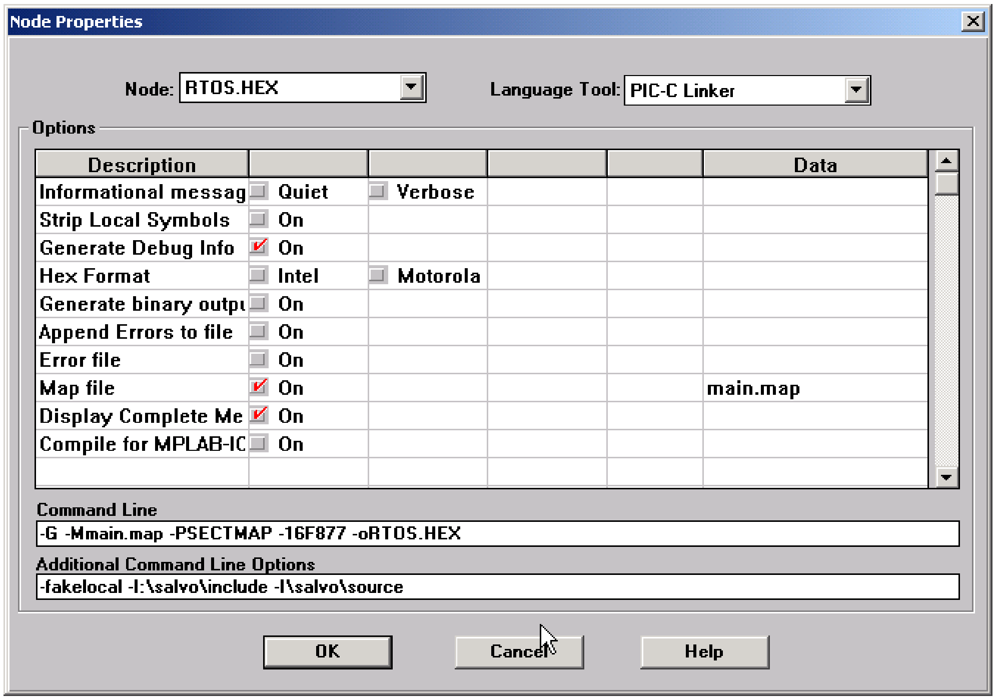
Command Line

```
-G -O -Zg5 -D24 -E -16F877 -C -IC:\RTOS
```

Additional Command Line Options

```
-fakelocal -I\salvo\include -I\salvo\source
```

NODE PROPERTIES (.hex-FILE)



Node: RTOS.HEX   Language Tool: PIC-C Linker

Options

Description					Data
Informational messag	☐ Quiet	☐ Verbose			
Strip Local Symbols	☐ On				
Generate Debug Info	☑ On				
Hex Format	☐ Intel	☐ Motorola			
Generate binary outpu	☐ On				
Append Errors to file	☐ On				
Error file	☐ On				
Map file	☑ On				main.map
Display Complete Me	☑ On				
Compile for MPLAB-IC	☐ On				

Command Line

```
-G -Mmain.map -PSECTMAP -16F877 -oRTOS.HEX
```

Additional Command Line Options

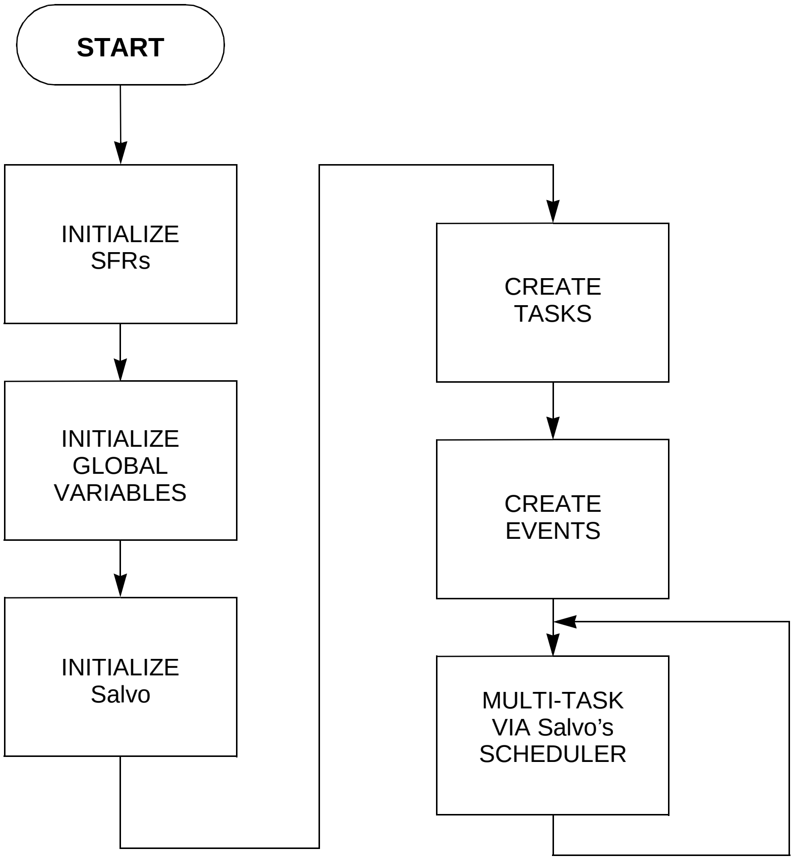
```
-fakelocal -I:\salvo\include -I\salvo\source
```

TASK CONFIGURATION

Tasks and Events are the building blocks of an RTOS. These modules can be added and deleted without affecting other parts of the code. This application is divided into eight tasks. Under normal conditions, four of the tasks are in the waiting state, while the other four run and then delay themselves repeatedly.

Figure 2 shows program execution upon power-up. An important point to realize here is that once multi-tasking begins, the four waiting tasks do not consume any processing power until they are signaled. When bringing the system online, there will be no alarms or user functions in operation. The result is, all tasks that wait for an event will go into the waiting state and become eligible only when signaled.

FIGURE 2: MAIN( )



The following is a detailed description of each task's priorities, status, and responsibilities.

`Task_Convert()`

Priority: 1
Task has a priority of '1' because we must determine thermistor temperatures to decide whether an alarm condition exists.

Status: Runs every 40 milliseconds.

Responsibilities:

1. Converts the analog thermistor voltage into a digital value, then translates this value into a Fahrenheit temperature.
2. This value is compared against the low and high threshold temperatures [via `ConvertTemp()`] to determine if an alarm is necessary.
3. If no alarm is called then the other thermistor zones are converted.

`Task_Alarm_On()`

Priority: 1
This task also has a priority of '1', but runs after `Task_Convert()` in a round-robin fashion. After determining temperature, checking for zone alarms is most important.

Status: Waits for an event.

Responsibilities:

1. Has the same priority as, and runs immediately after, `Task_Convert()`at start-up.
2. Displays the zone number in alarm.
3. Turns the piezo beeper on and off.

`Task_Display()`

Priority: 2
Enables temperatures to be read from the display.

Status: Runs every 2 milliseconds.

Responsibilities:

1. Converts the temperature value to a format necessary for displaying on the LEDs.
2. Displays each converted digit.

`Task_KeyPad()`

Priority: 3
Keypad entry is infrequent and should not supercede the prior tasks.

Status: Runs every 20 milliseconds.

Responsibilities:

1. Scans for the low-high entry.
2. Scans for potentiometer adjustment entry.
3. Scans for EEPROM recall entry.
4. Scans for zone display entry.

`Task_Usart()`

Priority: 4
Remote PC monitoring is only performed occasionally because usage is low.

Status: Runs every 800 milliseconds.

Responsibilities:

1. Scans for a PC keyboard entry (z).
2. Prepares each zone temperature for PC monitor display.
3. Writes the Z1 string out to the HyperTerminal via the USART.

`Task_Weeprom()`

Priority: 5
This task is only active when an alarm has occurred; therefore, it is used very little.

Status: Waits for an event.

Responsibilities:

1. Receives the zone number in alarm.
2. Writes zone number to EEPROM.
3. I^2C communication between the microcontroller and EEPROM.

`Task_Reeprom()`

Priority: 6
This task is dependent upon `Task_KeyPad()` and is independent of temperature and alarm status; therefore, it is a very low priority.

Status: Waits for an event.

Responsibilities:

1. Reads the last address that `Task_Weeprom()` wrote to.
2. Reads the data within this address.
3. Displays the contents of the EEPROM address on the LEDs (zone number).

`Task_Pots()`

Priority: 7
This task is least important because it is only used for setting potentiometers, which do not affect any temperature or alarm statuses.

Status: Waits for an event.

Responsibilities:

1. According to the value passed to the local variable pot_val, the appropriate pot is selected for adjustment while displaying the current potentiometer A/D value on the LEDs.

EVENT CONFIGURATION

Semaphores and messages can represent events and these methods of intertask communication are used in two ways. The first and more obvious is done by signaling tasks. When a task is signaled, it transitions from a waiting state to an eligible state and finally a running state. ALARM, REEPROM, POTVAL and WEEPROM are used in this fashion. The DISPLAY event is used to control a resource, quite different from the other events. Because the LED display is used by multiple tasks and the LEDs and keypad both operate out of PORTB on the microcontroller, PORTB has to be configured differently for both. The DISPLAY event is used to manage access to PORTB. When control of DISPLAY is placed around a group of statements, it creates a sequence whereby a resource is acquired, used, and then released.

The process flow for Task_Alarm_On(), has the task in one of three states: running, delayed, or waiting for an event. Salvo manages task execution so the PICmicro always runs the highest priority, eligible task. Whenever a particular task is running in this application, all other tasks are either delayed, waiting for an event, or eligible to run.

Looking at Task_Alarm_On() when the code reaches OS_WaitBinSem (DISPLAY), if DISPLAY = 1, then OS_WaitBinSem() flips it to ‘0’, and the following code is executed. When Salvo context switches via OS_Delay(), any piece of the code that waits for DISPLAY will not run (DISPLAY = 0). After both Task_Alarm_ON() and OS_Delay() are completed, DISPLAY is signaled (DISPLAY = 1) and allows the next piece of code waiting for DISPLAY to run.

ALARM

Type: Message

Purpose: Signal Task_Alarm_On() from within Task_Convert()(ConvertTemp()), with a message containing the zone number in alarm.

WEEPROM

Type: Message

Purpose: Signal Task_Weeprom() with a message containing the zone number in alarm. This message only happens if there is an alarm and after the signaling of Task_Alarm_On().

REEPROM

Type: Binary Semaphore

Purpose: Signal Task_Reeprom() from within Task_KeyPad() that the read EEPROM button has been pressed. Signaling the binary semaphore causes the waiting task to run.

POTVAL

Type: Message

Purpose: Signal Task_Pots() from within Task_KeyPad() that a potentiometer adjustment button has been pressed. Passes information containing the potentiometer number to set for adjustment mode.

DISPLAY

Type: Binary Semaphore

Purpose: This semaphore is used to control a resource, this may be the function of the LEDs or the keypad.

TIMING PERFORMANCE

Time management is a major responsibility for an RTOS. An application's response is dependent upon task execution times. The actual time between successive executions of Task_Convert() was measured as 40 milliseconds, with less than one system tick (2 ms) of timing jitter. When task delay times are calculated, the time necessary for instructions within the task must also be taken into consideration.

SUMMARY

This application note demonstrates how easy it is to implement a common embedded application into an RTOS environment. The temperature application shown here is just one of the many ways in which an RTOS can be applied. Some RTOS features that have not been discussed may be what your application requires. This includes counting semaphores and message queues, which are extended versions of the user services used in this application. Only one interrupt was used (to maintain a time-base), but additional interrupt sources can be included for added real-time response. After establishing an understanding of RTOS user services, it's just a matter of adding more tasks and events to suit the demands of your application.

WEBSITES

Microchip Technology Inc.	www.microchip.com
Pumpkin, Inc.	www.pumpkininc.com
HI-TECH Software	www.htsoft.com

APPENDIX A: FLOW CHARTS

FIGURE A-1: SCHEMATIC (SHEET 1 OF 3)

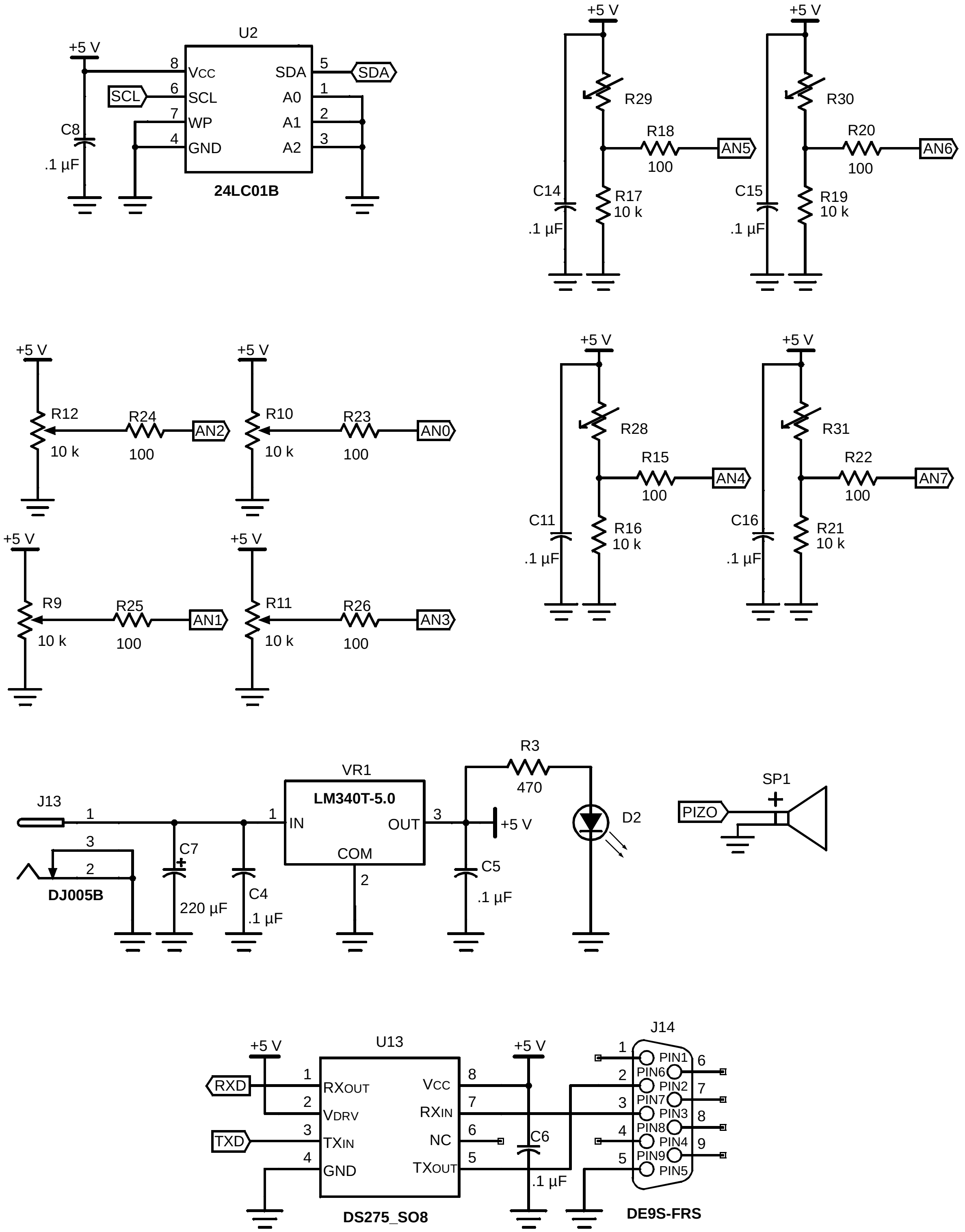


FIGURE A-2: SCHEMATIC (SHEET 2 OF 3)

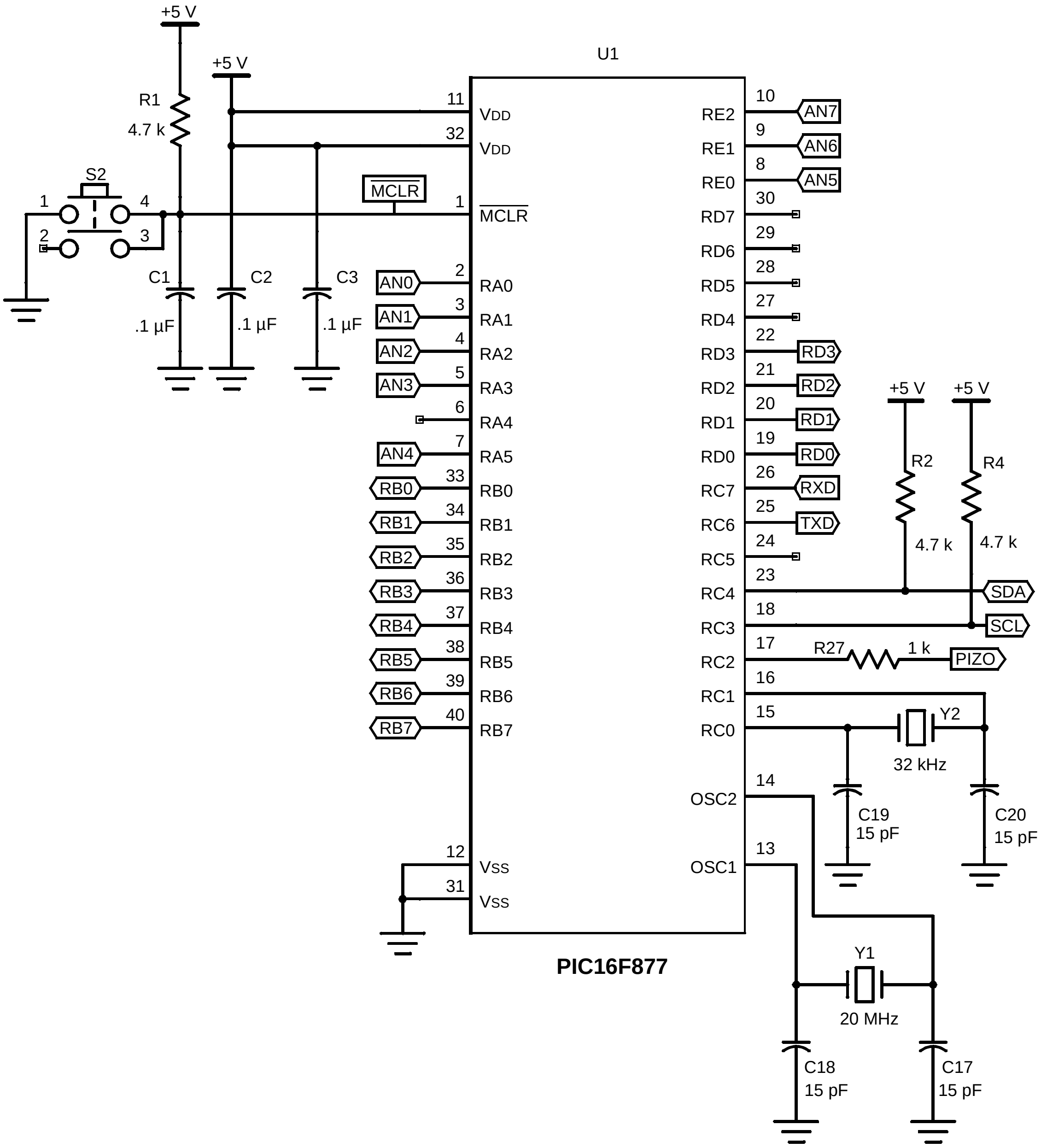
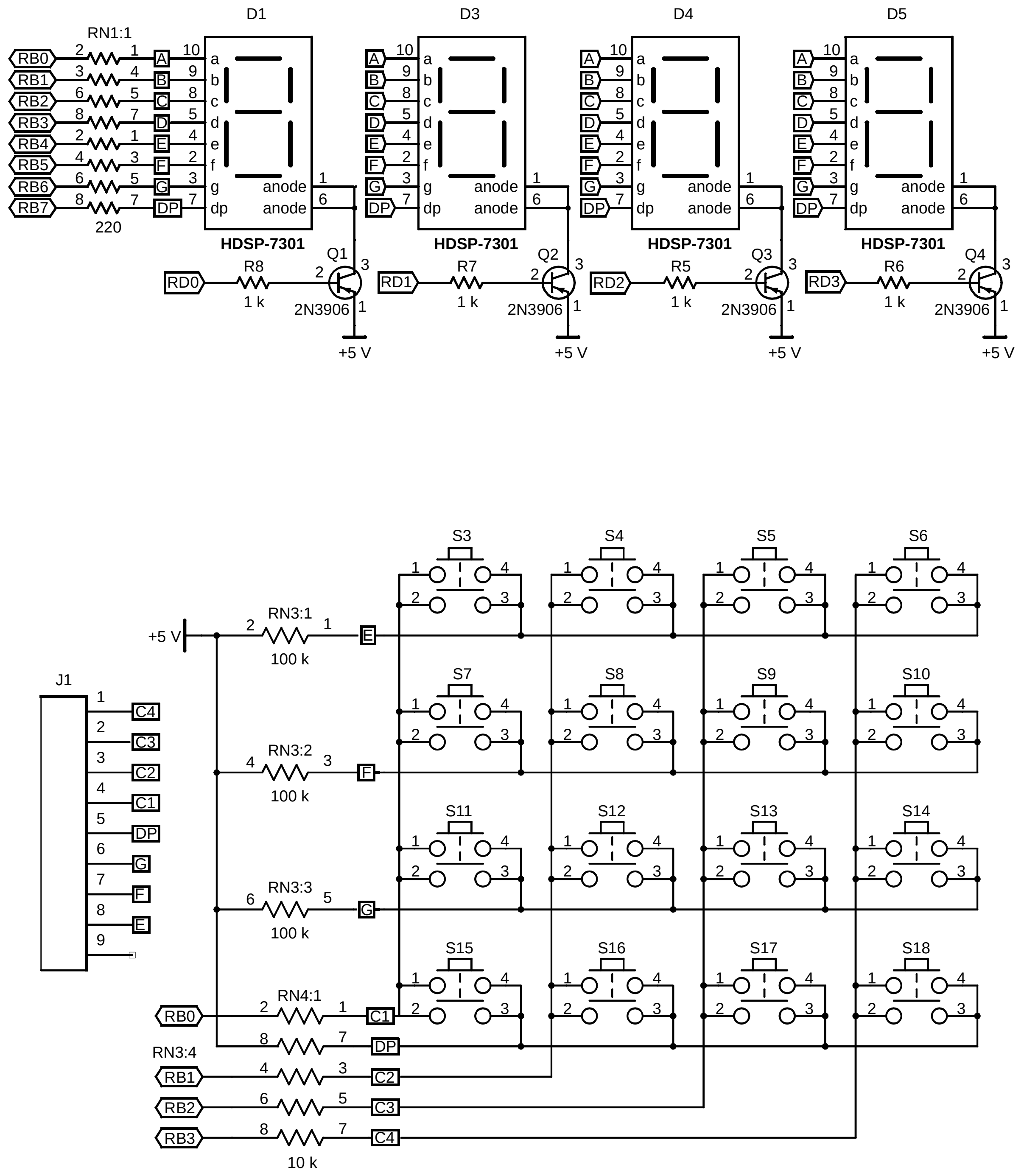


FIGURE A-3: SCHEMATIC (SHEET 3 OF 3)



APPENDIX B: SOURCE CODE

salvocfg.h

```
#define OSCOMPILER                      OSHT_PICC
#define OSTARGET                        OSPIC16

#define OSBYTES_OF_DELAYS               2

#define OSLOC_ECB                       bank2
#define OSLOC_TCB                       bank2

#define OSEVENTS                        5
#define OSTASKS                         8

#define OSENABLE_BINARY_SEMAPHORES      TRUE
#define OSENABLE_MESSAGES               TRUE
#define OSBIG_MESSAGE_POINTERS          TRUE
```

main.c

```
/*
This program is based on the Salvo RTOS (v2.1). Its function is to scan
four thermistors and report their temperatures. If any of reported temperatures
are not within a preset range, the alarm will sound. Four potentiometers adjustments
are accessed via keypad entry. Two of them will be used to determine the Piezo tone and
duty cycle, while these pots are being set their A/D values will appear on the LED display.
The four thermistor are divided up into 4 zones, each zone can be displayed on the
4-digit LED display via a keypad entry. The defined temperature range can be entered by keypad
entry, entering the LOW temp first followed by the HIGH temp. Zone temperatures can be recalled
onto a PC monitor via the HyperTerminal by pressing ’z’ on a PC keyboard.
Every time a zone goes into alarm, the alarm zone number will be written to the
EEPROM. The zone that last set off an alarm can be recalled via keypad entry and the
zone number will be displayed.
*/

#include <salvo.h>

#define ALARM      0
#define WEEPROM    1
#define REEPROM    2
#define POTVAL     3
#define DISPLAY    4

static volatile unsigned int TMR1 @ 0x0E;

bank1   unsigned char Low_Hi;
bank1   signed   char data_address;              //EEPROM ADDRESS
bank1   unsigned char *zone_dis;                 //ZONE DISPLAY
bank1   unsigned char temp1, temp2, temp3, temp4;//ALARM & ZONE TEMPS
bank1   unsigned char low, high;                 //LOW & HIGH TEMP THRESHOLD
```

```
bank1   unsigned char Z1[39] = "ZONE Temps: z1-xx z2-xx z3-xx z4-xx\n\r\v";//RS-232 DISPLAY

const   char SevenSegmentTable[] =               //DIGIT SEGMENTS
    {   0b11000000,                              // 0
        0b11111001,                              // 1
        0b10100100,                              // 2
        0b10110000,                              // 3
        0b10011001,                              // 4
        0b10010010,                              // 5
        0b10000010,                              // 6
        0b11111000,                              // 7
        0b10000000,                              // 8
        0b10010000                               // 9
    };

const unsigned char CHSmask[] =                  //A/D CHS BITS
    {   0b00100000,
        0b00101000,
        0b00110000,
        0b00111000
    };

const unsigned char zones[] =                    //TEMPERATURE ZONE NUMBERS
    {
        1,
        2,
        3,
        4
    };

bank1 unsigned char * const tempPArray [] =      //ZONE TEMPERATURES
    {
        &temp1,
        &temp2,
        &temp3,
        &temp4
    };

               //PROTOTYPES
void    Delay(unsigned char tmr);
void    interrupt isr(void);
void    ConvertAD(void);
char    ButtonPress(unsigned char buttons);
char    Keys(void);
void    BcdConv(char);
void    WriteSevenSegment( unsigned char segment, unsigned char digit);
char    ReadUSART(void);
void    WriteUSART(char data);
void    WriteUSARTBuffer(unsigned char *data, unsigned char len);
void    Idle(void);
void    Display(unsigned char lo_hi);
void    PotDisplay(void);
void    ConvertTemp( bank1 unsigned char * const temp,
           const unsigned char * zone );

    _OSLabel (task_convert1)
    _OSLabel (task_alarm_on1)
    _OSLabel (task_alarm_on2)
    _OSLabel (task_alarm_on3)
    _OSLabel (task_alarm_on4)
    _OSLabel (task_keypad1)
    _OSLabel (task_keypad2)
    _OSLabel (task_keypad3)
    _OSLabel (task_display1)
    _OSLabel (task_display2)
    _OSLabel (task_usart1)
    _OSLabel (task_weeprom1)
    _OSLabel (task_reeprom1)
    _OSLabel (task_reeprom2)
    _OSLabel (task_reeprom3)
    _OSLabel (task_pots1)
    _OSLabel (task_pots2)

//**************************(  FUNCTIONS  )****************************************

void  Delay(unsigned char tmr)                             //TIMER0 MAX TIMEOUT = 13ms
{
      TMR0 = 255 - tmr;
      T0IF = 0;
      while(T0IF==0);
}


#pragma interrupt_level 0
void  interrupt isr(void)                                  //TIMER1 2ms PERIODIC INTERRUPT
{
      if(TMR1IF)
      {
          TMR1IF = 0;
          TMR1 -= 5000;
          OSTimer();
      }
}


void  ConvertAD(void)                                      //A/D CONVERSION
{
      Delay(1);
      ADGO = 1;
      while(ADGO);
}


char    ButtonPress(unsigned char buttons)
{
      unsigned char Col_Row;                               //FIND BUTTON PRESS
      PORTB = buttons;
      Delay(55);
      Col_Row = PORTB;
      return Col_Row;
}


char  Keys(void)                                           //NUMBER SELECTION
{
          char KeyVal = 10;                                //BUTTON NUMBER PRESSED
          PORTD = 0x0F;                                    //LEDs OFF
          TRISB = 0xF0;                                    //RB7:RB4=INPUTS,RB3:RB0=OUTPUTS

      while(KeyVal == 10)
      {
          switch(ButtonPress(0b00001110))
          {
          case 0xEE:
              KeyVal = 0b00000001;                         //#1
              break;

          case 0xDE:
              KeyVal = 0b00000100;                         //#4
              break;
```

```
         case 0xBE:
            KeyVal = 0b00000111;                        //#7
            break;

         default:
            break;
         }

         switch(ButtonPress(0b00001101))
         {
         case 0xED:
            KeyVal = 0b00000010;                        //#2
            break;

         case 0xDD:
            KeyVal = 0b00000101;                        //#5
            break;

         case 0xBD:
            KeyVal = 0b00001000;                        //#8
            break;

         case 0x7D:
            KeyVal = 0;                                 //#0
            break;

         default:
            break;
         }

         switch(ButtonPress(0b00001011))
         {
         case 0xEB:
            KeyVal = 0b00000011;                        //#3
            break;

         case 0xDB:
            KeyVal = 0b00000110;                        //#6
            break;

         case 0xBB:
            KeyVal = 0b00001001;                        //#9
            break;

         default:
            break;

         }
      PORTB = 0b00000000;
      }return KeyVal;
}

void  BcdConv(char KeyVal)                              //BCD CONVERSION
      {
         Low_Hi *= 10;
         Low_Hi += KeyVal;
      }

void  WriteSevenSegment(unsigned char segment, unsigned char digit)
{                                                       //LED VALUE DISPLAY
      switch(digit)
      {
      case 1:
         PORTD = 0x0E;                                  //FIRST DIGIT
         break;
      case 2:
          PORTD = 0x0D;                                 //SECOND DIGIT
          break;

      case 3:
          PORTD = 0x0B;                                 //THIRD DIGIT
          break;

      case 4:
          PORTD = 0x07;                                 //FOURTH DIGIT
          break;
      }

      TRISB = 0x00;
      PORTB = SevenSegmentTable[segment];               //SEND SEGMENT NUMBER TO PORTB
}


char  ReadUSART(void)                                   //READ SERIAL DATA ENTRY
{
      unsigned char rdata;
      if(RCIF)                                          //RECEPTION COMPLETE
      rdata = RCREG;
      return rdata;
}


void  WriteUSART(char data)                             //WRITE SERIAL DATA
{
      while(!TRMT);
      TXREG = data;
}


void  WriteUSARTBuffer(unsigned char *data, unsigned char len)
{
      unsigned char i;

      for ( i = 0; i < len; i++ )
          WriteUSART(data[i]);                          //WRITE STRING
}


void  Idle(void)                                        //I2C IDLE FUNCTION
{
      while((SSPCON2 & 0x1F)|(STAT_RW))
      continue;
}


void  Display(unsigned char lo_hi)                      //DISPLAY LOW & HIGH INPUT
{
          unsigned char v1,v2,v3;
          unsigned char i;

      for(i=1; i<200; i++)
          {
          v1 = lo_hi/0x64;                              //FIND FIRST DISPLAY DIGIT
          v2 = (lo_hi-(v1*0x64))/10;                    //FIND SECOND DIGIT
          v3 = (lo_hi-(v1*0x64)-(v2*10));               //FIND THIRD DIGIT
          WriteSevenSegment(0, 1);                      //SEND SEGMENT VALUE AND DIGIT 1
          Delay(55);                                    //DIGIT DELAY
          WriteSevenSegment(v1, 2);
          Delay(55);
          WriteSevenSegment(v2, 3);
          Delay(55);
          WriteSevenSegment(v3, 4);
          Delay(55);
          }
}


void  PotDisplay(void)
{
          unsigned char v1,v2,v3;
      for(;;)
      {
          ConvertAD();
          v1 = ADRESH/0x64;                             //FIND FIRST DISPLAY DIGIT
          v2 = (ADRESH-(v1*0x64))/10;                   //FIND SECOND DIGIT
          v3 = (ADRESH-(v1*0x64)-(v2*10));              //FIND THIRD DIGIT ;
          WriteSevenSegment(v1, 2);                     //SEND SEGMENT VALUE AND DIGIT 2
          Delay(15);
          WriteSevenSegment(v2, 3);                     //SEND SEGMENT VALUE AND DIGIT 3
          Delay(15);
          WriteSevenSegment(v3, 4);                     //SEND SEGMENT VALUE AND DIGIT 4
          Delay(15);

          PORTD = 0x0F;                                 //PREPARE FOR KEYPAD USE
          TRISB = 0xF0;
          if(ButtonPress(0b00001101) == 0x7D)
          break;
      }
}


void ConvertTemp( bank1 unsigned char * const temp, const unsigned char * zone )
{     float adresh;
      adresh = ADRESH;
      *temp = ( (.538) + (.444*(adresh) ) + (.001*(adresh)*(adresh) ) );

      if ( ( low > *temp ) || ( *temp > high ) )
       {
          OSSignalMsg(ALARM, (OStypeMsgP) zone);        //SIGNAL task_alarm() W/ ZONE #
          OSSignalMsg(WEEPROM, (OStypeMsgP) zone);      //SIGNAL task_weeprom() W/ ZONE #
       }
}

//**************************(   TASKS   )*******************************************
//**********************************************************************************

void  Task_Convert(void)
{
          static unsigned char i = 0;

      for(;;)
      {
          ADCON0 &= ~0b00111000;                        //CLEAR CHS BITS
          ADCON0 |=  CHSmask[i];                        //SELECT CHS
          ConvertAD();                                  //CONVERT CHS
          ConvertTemp(tempPArray[i], &zones[i] );

          if ( ++i > 3 ) i = 0;

          OS_Delay(20,task_convert1);                   //DELAYED FOR 40ms
      }
}
```

```
void  Task_Alarm_On(void)                                       //WAITING TASK
{
         OStypeMsgP msgP;

      for(;;)
      {
         OS_WaitMsg(ALARM, &msgP, task_alarm_on1);
         OS_WaitBinSem(DISPLAY, task_alarm_on2);
         WriteSevenSegment(* ( const unsigned char *) msgP, 4);//DISPLAY ALARM ZONE
         CCP1CON = 0x0F;
         OS_Delay(200, task_alarm_on3);
         CCP1CON = 0;
         OS_Delay(200, task_alarm_on4);
         OSSignalBinSem(DISPLAY);
      }
}


void  Task_Keypad(void)
{
         static char pot;
      for(;;)
      {
         OS_WaitBinSem(DISPLAY, task_keypad1);
         PORTD = 0x0F;                                      //LEDs OFF
         TRISB = 0xF0;                                      //RB7:RB4 = INPUTS,RB3:RB0 = OUTPUTS

         switch(ButtonPress(0b00001110) )
         {
         case 0x7E:                                         //SET LOW AND HIGH TEMPS
             PORTD = 0x00;                                  //TURN ON DIGITS TO
             TRISB = 0x00;                                  //  SHOW TEMP SETTING
             PORTB = 0x00;                                  //   ACTIVATION
             OS_Delay(200, task_keypad2);

         //GET LOW TEMPERATURE LIMIT
             PEIE = 0;                                      //INTERRUPT DISABLED

             Low_Hi = 0;
             BcdConv(Keys());                               //GET 1ST DIGIT
             while( PORTB != 0xF0 );

             BcdConv(Keys());                               //GET 2ND DIGIT
             while( PORTB != 0xF0 );

             BcdConv(Keys());                               //GET 3RD DIGIT
             low = Low_Hi;

             Display(low);                                  //DISPLAY LOW TEMP
             PORTD = 0x0F;                                  //LEDs OFF
             TRISB = 0xF0;                                  //RB7:RB4 = INPUTS,RB3:RB0 = OUTPUTS

         //GET HIGH TEMPERATURE LIMIT

             Low_Hi = 0;
             BcdConv(Keys());                               //GET 1ST DIGIT
             while( PORTB != 0xF0 );

             BcdConv(Keys());                               //GET 2ND DIGIT
             while( PORTB != 0xF0 );

             BcdConv(Keys());                               //GET 3RD DIGIT
             high = Low_Hi;
             Display(high);                                 //DISPLAY HIGH TEMP
             PEIE = 1;                                      //INTERRUPT RE-ENABLED
             break;
```

```
                                                   //POTENTIOMETER SELECTION

case 0xEE:                                         //#1
    pot = 1;
    OSSignalMsg(POTVAL,(OStypeMsgP)&pot);          //SIGNAL task_pots() W/ POT-1
    break;

case 0xDE:                                         //#4
    pot = 4;
    OSSignalMsg(POTVAL,(OStypeMsgP)&pot);          //SIGNAL task_pots() W/ POT-4
    break;

default:
    break;
}


if(ButtonPress(0b00001101) == 0xED)                //#2
{

    pot = 2;
    OSSignalMsg(POTVAL,(OStypeMsgP)&pot);          //SIGNAL task_pots() W/ POT-2
}


switch(ButtonPress(0b00001011) )
{
case 0xEB:
    pot = 3;                                       //#3
    OSSignalMsg(POTVAL,(OStypeMsgP)&pot);          //SIGNAL task_pots() W/ POT-3
    break;

                                                   //EEPROM BUTTON
case 0x7B:
    OSSignalBinSem(REEPROM);                       //SIGNAL task_reeprom()
    break;

default:
    break;
}


                                                   //ZONE BUTTONS

switch(ButtonPress(0b00000111))
{
case 0xE7:
    zone_dis = &temp1;                             //ZONE 1 BUTTON
    break;

case 0xD7:
    zone_dis = &temp2;                             //ZONE 2 BUTTON
    break;

case 0xB7:
    zone_dis = &temp3;                             //ZONE 3 BUTTON
    break;

case 0x77:
    zone_dis = &temp4;                             //ZONE 4 BUTTON
    break;
default:
    break;
}
```

```
         OSSignalBinSem(DISPLAY);
         OS_Delay(10,task_keypad3);                     //DELAYED FOR 20ms
      }
}


void   Task_Display(void)
{
         unsigned char v1,v2,v3;
         unsigned char dis_temp;

      for(;;)
      {
         OS_WaitBinSem(DISPLAY, task_display1);

         dis_temp = *zone_dis;
         v1 = dis_temp/0x64;                            //FIND FIRST DISPLAY DIGIT
         v2 = (dis_temp-(v1*0x64))/10;                  //FIND SECOND DIGIT
         v3 = (dis_temp-(v1*0x64)-(v2*10));             //FIND THIRD DIGIT
         WriteSevenSegment(0, 1);                       //SEND SEGMENT VALUE AND DIGIT 1
         Delay(100);                                    //DIGIT-ON DELAY
         WriteSevenSegment(v1, 2);
         Delay(100);
         WriteSevenSegment(v2, 3);
         Delay(100);
         WriteSevenSegment(v3, 4);
         Delay(100);
         PORTB = 0xFF;                                  // TURN OFF LAST DIGIT

         OSSignalBinSem(DISPLAY);
         OS_Delay(1, task_display2);                    // DELAYED FOR 2ms

      }
}


void  Task_Usart(void)
{
         unsigned char v1,v2,v3,v2A,v3A,v2B,v3B,v2C,v3C,v2D,v3D;
      for(;;)
      {
         ReadUSART();
         if(ReadUSART() == 0x7A)                        // ASCII CHARACTER z
         {
         v1 = temp1 / 0x64;                             // CONVERT TEMP1 FOR DISPLAY
         v2 = (temp1 - (v1*0x64))/10;
         v3 = (temp1 - (v1*0x64) - (v2*10));
         v2A = v2, v3A = v3;

         v1 = temp2 / 0x64;                             //   TEMP2
         v2 = (temp2 - (v1*0x64))/10;
         v3 = (temp2 - (v1*0x64) - (v2*10));
         v2B = v2, v3B = v3;

         v1 = temp3 / 0x64;                             //   TEMP3
         v2 = (temp3 - (v1*0x64))/10;
         v3 = (temp3 - (v1*0x64) - (v2*10));
         v2C = v2, v3C = v3;

         v1 = temp4 / 0x64;                             //   TEMP4
         v2 = (temp4 - (v1*0x64))/10;
         v3 = (temp4 - (v1*0x64) - (v2*10));
         v2D = v2, v3D = v3;
```

```
             Z1[15] = v2A + ’0’;
             Z1[16] = v3A + ’0’;
             Z1[21] = v2B + ’0’;
             Z1[22] = v3B + ’0’;
             Z1[27] = v2C + ’0’;
             Z1[28] = v3C + ’0’;
             Z1[33] = v2D + ’0’;
             Z1[34] = v3D + ’0’;
             WriteUSARTBuffer(Z1,39);                   //WRITE STRING Z1 FOR 39 BYTES
         }

      OS_Delay(400, task_usart1);                       //DELAYED FOR 800ms
      }
}

void Task_Weeprom(void)                                 //WAITING TASK
{
         OStypeMsgPalarm_zoneP;
         char word;

      for(;;)
      {
         OS_WaitMsg(WEEPROM, &alarm_zoneP, task_weeprom1);
         word = *(const unsigned char*) alarm_zoneP;

         SEN = 1;                                       //START ENABLED
         while(SEN);                                    //WAIT UNTIL START IS OVER
         SSPBUF = 0b10100000;                           //CONTROL BYTE
         Idle();                                        //ENSURE MODULE IS IDLE
         if(!ACKSTAT);                                  //LOOK FOR ACK
         else
             break;

         SSPBUF = data_address;                         //ADDRESS BYTE
         Idle();                                        //ENSURE MODULE IS IDLE
         if(!ACKSTAT);                                  //LOOK FOR ACK
         else
             break;

         SSPBUF = word;                                 //DATA BYTE (ZONES: 1,2,3 or 4)
         Idle();                                        //ENSURE MODULE IS IDLE
         if(!ACKSTAT)                                   //LOOK FOR ACK
         {   PEN = 1;                                   //STOP ENABLED
             while(PEN);                                //WAIT UNTIL STOP IS OVER
         }
         else
             break;
      }
}


void Task_Reeprom(void)
{
         char word;
      for(;;)                                           //WAITING TASK
      {
         OS_WaitBinSem(REEPROM,task_reeprom1);

         Idle();                                        //ENSURE MODULE IS IDLE
         SEN = 1;                                       //START ENABLED
         while(SEN);                                    //WAIT UNTIL START IS OVER

         SSPBUF = 0b10100000;                           //CONTROL BYTE (write)
         Idle();                                        //ENSURE MODULE IS IDLE
         if(!ACKSTAT);                                  //LOOK FOR ACK
         else
            break;

         SSPBUF = data_address;                         //ADDRESS BYTE (write)
         Idle();                                        //ENSURE MODULE IS IDLE
         if(!ACKSTAT);                                  //LOOK FOR ACK
         else
            break;
         RSEN = 1;                                      //REPEAT START CONDITION
         while(RSEN);                                   //WAIT UNTIL RESTART IS OVER

         SSPBUF = 0b10100001;                           //CONTROL BYTE (read)
         Idle();                                        //ENSURE MODULE IS IDLE
         if(!ACKSTAT);                                  //LOOK FOR ACK
         else
            break;

         RCEN = 1;                                      //ENABLE RECEIVE
         while(RCEN);                                   //WAIT UNTIL RECEIVE IS OVER

         ACKDT = 1;                                     //NO ACK
         ACKEN = 1;
         while(ACKEN);                                  //WAIT UNTIL ACK IS FINISHED

         PEN = 1;                                       //STOP ENABLED
         while(PEN);                                    //WAIT UNTIL STOP IS OVER

         word = SSPBUF;                                 //WRITE DATA TO VARIABLE
         ++data_address;                                //MOVE ADDRESS TO NEXT SPACE

         OS_WaitBinSem(DISPLAY, task_reeprom2);
         WriteSevenSegment(word, 3);                    //DISPLAY ZONE OF LAST ALARM
         OS_Delay(200, task_reeprom3);
         OSSignalBinSem(DISPLAY);
      }
}
void Task_Pots(void)                                    //WAITING TASK
{
         OStypeMsgP pot_valP;
         char pot_val;
      for(;;)
      {
         OS_WaitMsg(POTVAL, &pot_valP, task_pots1);
         pot_val = *(char*) pot_valP;
         OS_WaitBinSem(DISPLAY, task_pots2);

         switch(pot_val)
            {
         case 1:
            CHS2=0, CHS1=0, CHS0=0;                     //AN0 - PIEZO "TONE" (PWM PERIOD)
            PotDisplay();
            PR2 = ADRESH;
            break;
         case 2:
            CHS2=0, CHS1=0, CHS0=1;                     //DISPLAY A/D VALUE
            PotDisplay();
            break;
         case 3:
            CHS2=0, CHS1=1, CHS0=0;                     //DISPLAY A/D VALUE
            PotDisplay();
            break;
         case 4:
            CHS2=0, CHS1=1, CHS0=1;                     // AN3 - FOR PIEZO DUTY CYCLE
            PotDisplay();
            CCPR1L = ADRESH;
```

```
             break;
             }
         OSSignalBinSem(DISPLAY);
      }
}

//*****************************(  MAIN  )****************************************
//*******************************************************************************

void  main(void)
{
      TXSTA = 0b10100100;                               //TRANSMIT
      RCSTA = 0b10010000;                               //RECEIVE
      SPBRG = 0x81;                                     //BAUD RATE
      TRISC6 = 0,TRISC7 = 1;                            //TXD OUTPUT & RXD INPUT

      TRISC3 = 1,TRISC4 = 1;                            //SCL & SDA - I2C
      SSPADD = 0x32;                                    //I2C BAUD RATE (MASTER MODE)
      SSPCON = 0b00101000;                              //ENABLE SDA & SCL, S-PORT MODE-MASTER

      ADCON0 = 0b01000001;                              //A/D CONFIG

      OPTION = 0b10000101;                              //TIMER0 CONFIG

      T1CON = 0b00010101;                               //TIMER1 CONFIG (system tick rate)
      TMR1IE = 1;                                       //ENABLE INTERRUPT
      TMR1IF = 0;                                       //CLEAR FLAG

      TRISC2 = 0;                                       //PIEZO
      CCPR1L = 0x80,CCP1X=0,CCP1Y=0;                    //PWM DUTY CYCLE
      T2CON = 0b00000101;                               //TIMER2 PRESCALE = 4 (PWM)

      GIE = 1, PEIE = 1;                                //ENABLE GLOBAL & PERIPHERAL INTERRUPTS

      TRISD = 0x00;                                     //PORTD OUTPUT-DIGITS
      low=20,high=170;                                  //INITIAL TEMPERATURE RANGE
      data_address = 0x00;                              //FIRST EEPROM WRITE

      OSInit();
                                             //ID       PRIORITY
      OSCreateTask(Task_Convert,             0,         1);
      OSCreateTask(Task_Alarm_On,            1,         1);
      OSCreateTask(Task_Keypad,              2,         3);
      OSCreateTask(Task_Display,             3,         2);
      OSCreateTask(Task_Usart,               4,         4);
      OSCreateTask(Task_Weeprom,             5,         5);
      OSCreateTask(Task_Reeprom,             6,         6);
      OSCreateTask(Task_Pots,                7,         7);

      OSCreateMsg(ALARM,  (OStypeMsgP)       0);
      OSCreateMsg(WEEPROM,(OStypeMsgP)       0);
      OSCreateBinSem(REEPROM,                0);
      OSCreateMsg(POTVAL, (OStypeMsgP)       0);
      OSCreateBinSem(DISPLAY,                1);

      for(;;)
         OSSched();
}
```

```
Memory Usage Map:

Program ROM   $0000 - $0819  $081A (  2074) words
Program ROM   $0AAC - $0FFF  $0554 (  1364) words
                             $0D6E (  3438) words total Program ROM

Bank 0 RAM    $0020 - $004C  $002D (    45) bytes
Bank 0 RAM    $0070 - $007C  $000D (    13) bytes
                             $003A (    58) bytes total Bank 0 RAM

Bank 1 RAM    $00A0 - $00CE  $002F (    47) bytes total Bank 1 RAM
Bank 2 RAM    $0110 - $0156  $0047 (    71) bytes total Bank 2 RAM

Build completed successfully.
```

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, KEELOQ, SEEVAL, MPLAB and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Total Endurance, ICSP, In-Circuit Serial Programming, FilterLab, MXDEV, microID, *Flex*ROM, *fuzzy*LAB, MPASM, MPLINK, MPLIB, PICC, PICDEM, PICDEM.net, ICEPIC, Migratable Memory, FanSense, ECONOMONITOR, Select Mode, dsPIC, rfPIC and microPort are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Term Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

♻ Printed on recycled paper.

DNV Certification, Inc.
USA

ANSI • RAB
QMS
ACCREDITED

DNV

DNV MSC
The Netherlands
Accredited by the RvA

ISO 9001 / QS-9000
REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.

MICROCHIP

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Austin - Analog
13740 North Highway 183
Building J, Suite 4
Austin, TX 78750
Tel: 512-257-3370 Fax: 512-257-8526

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Boston - Analog
Unit A-8-1 Millbrook Tarry Condominium
97 Lowell Road
Concord, MA 01742
Tel: 978-371-6400 Fax: 978-371-0050

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Two Prestige Place, Suite 130
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai)
Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai)
Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou
Microchip Technology Consulting (Shanghai)
Co., Ltd., Fuzhou Liaison Office
Rm. 531, North Building
Fujian Foreign Trade Center Hotel
73 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7557563 Fax: 86-591-7557572

China - Shanghai
Microchip Technology Consulting (Shanghai)
Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai)
Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan
Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Germany - Analog
Lochhamer Strasse 13
D-82152 Martinsried, Germany
Tel: 49-89-895650-0 Fax: 49-89-895650-22

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

08/01/01