MOTOROLA
intelligence everywhere™

digital dna™

Central Heating Controller using the MC68HC908LJ12 Reference Design

Designer Reference Manual

M68HC08
Microcontrollers

DRM044
Rev. 0, 09/2003

MOTOROLA.COM/SEMICONDUCTORS

Central Heating Controller using the MC68HC908LJ12 Reference Design

Designer Reference Manual — Rev 0

by: **AT Electronic Embedded Control Consultants**
e-business centre
Consett Business Park
Villa Real
Consett
Co. Durham
DH8 6BP
England

Telephone: **44(0) 1207 693920**
Fax: **44(0) 1207 693920**
Email: **enquires@ateecc.com**
Web: **www.ateecc.com**

List of Sections

List of Sections

Designer Reference Manual — DRM044

Table of Contents

Table of Contents

Appendix A. Software Routines

Table of Contents

List of Figures

List of Figures

List of Tables

List of Tables

Section 1. Introduction

This document details the hardware and software design of a functional time and temperature control system, which combines heating and cooling algorithms to implement an intelligent climate control system. The MC68HC908LJ12 has been selected to perform the control functions, as, in addition to the ample memory size, it contains a fully-featured real-time clock (RTC) and liquid crystal display (LCD) driver module, which greatly simplifies the implementation of these functions.

Emphasis has been placed throughout the design on efficient modes of operation during periods of mains failure, to demonstrate the low current consumption of this device when powered by a backup battery.

Two methods of temperature measurement have been used in order to demonstrate the flexibility of the device. The internal temperature is detected by an analog sensor, which provides an output voltage proportional to the temperature. The external temperature is measured using a 2-wire serial sensor. The resulting data is decoded by the microcontroller and displayed on the LCD in degrees Celsius.

The internal and external temperature readings are also used in an algorithm, which may be used to modify the start of an automatic timing function if desired. This gives the user the opportunity to select a **'SMART'** mode of operation, whereby the heating or cooling functions are implemented earlier than the user set-time, depending on the ambient internal and external temperatures.

Introduction

Figure 1-1. The Climate Controller

During normal operation, the internal temperature is displayed on the three right-hand digits. Time-of-day (TOD) and day-of-week (DOW) are displayed in the left-hand digits and chevrons respectively.

A 3 V lithium coin cell is used for backup of time keeping functions.

Section 2. Design Overview

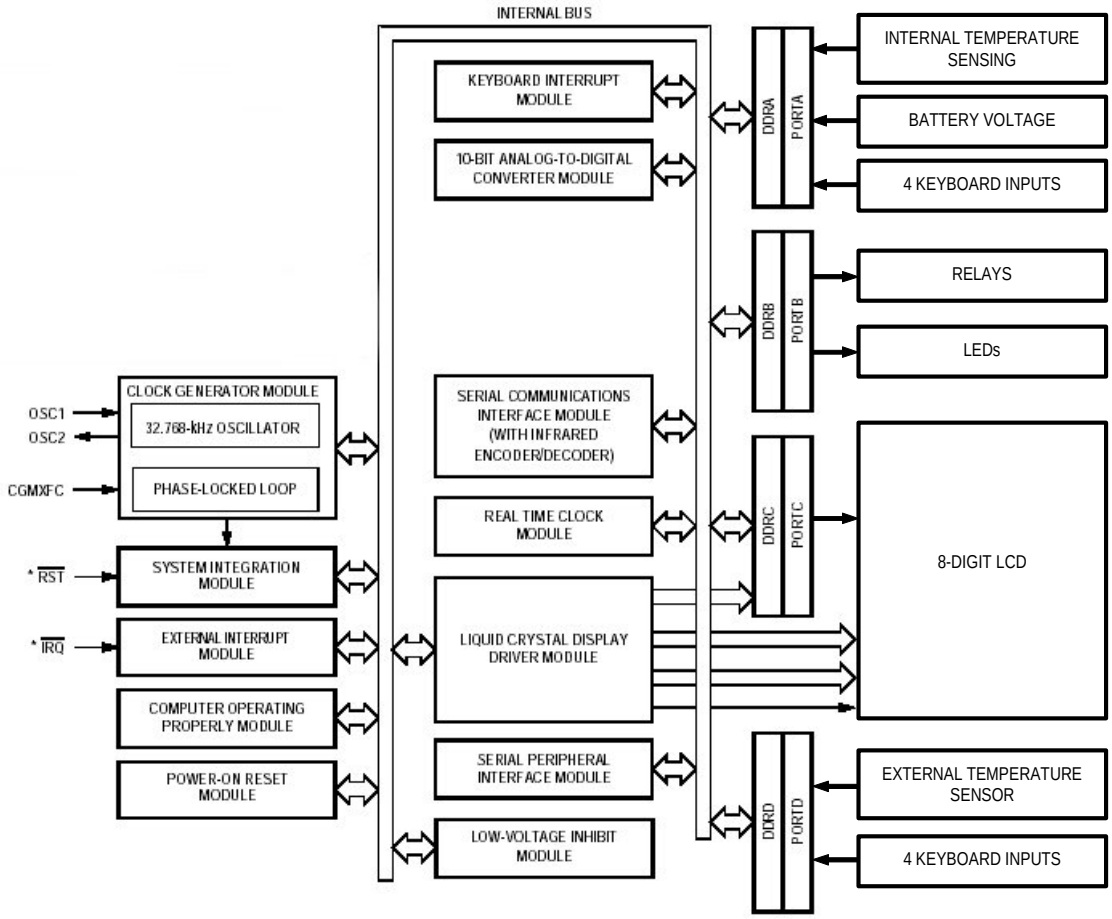


Figure 2-1. System Block Diagram

2.1 Internal Temperature Sensing

The voltage supplied by the internal temperature sensor is conditioned by an operational amplifier and scaled for the input parameters of the analog to digital converter (ADC). The ADC is then used to read this temperature value.

2.2 Battery Voltage

To keep the RTC module running during periods of mains supply failure, a 3 V lithium ‘coin’ cell is used. To minimize current drain from the battery, all non-essential outputs, i.e., relay drive, temperature measurement and light-emitting diode (LED) drive signals, are inhibited. The ADC is used to detect if the battery drops below the microcontroller’s minimum operating voltage. Under these circumstances, ‘low battery’ is displayed to the user.

2.3 Keyboard Inputs

All eight of the designated keyboard inputs are connected to buttons. These buttons are used for the general operation of the controller.

2.4 Output Drivers

Port B is used to activate the two climate control relays and two indication LEDs.

2.5 LCD

A commercially available LCD has been selected to perform the display functions. This has eight 7-segment digits, each separated by a decimal point (period). In addition to the digits, it contains eight downward-pointing chevrons that are used to indicate weekdays and the operation of the **‘SMART’** mode (please see 4.7.3 SMART — Intelligent Temperature Prediction for more information).

2.6 External Temperature Sensor

This is a 2-wire serial device. This allows the sensor to be placed at some distance from the controller with no degradation of signal.

2.7 Phase-locked Loop

The phase-locked loop (PLL) feature of the MC68HC908LJ12 enables a low-cost 32.768 kHz crystal, of the type used extensively in digital watches, to be used to obtain a bus frequency of 2 MHz.

2.8 Low Voltage Inhibit

The low voltage inhibit (LVI) is used to detect when the mains voltage fails. The microcontroller then powers down before entering STOP mode. Depending on feature implementation, current consumption can be reduced to approximately 30 µA.

Design Overview

Section 3. Hardware

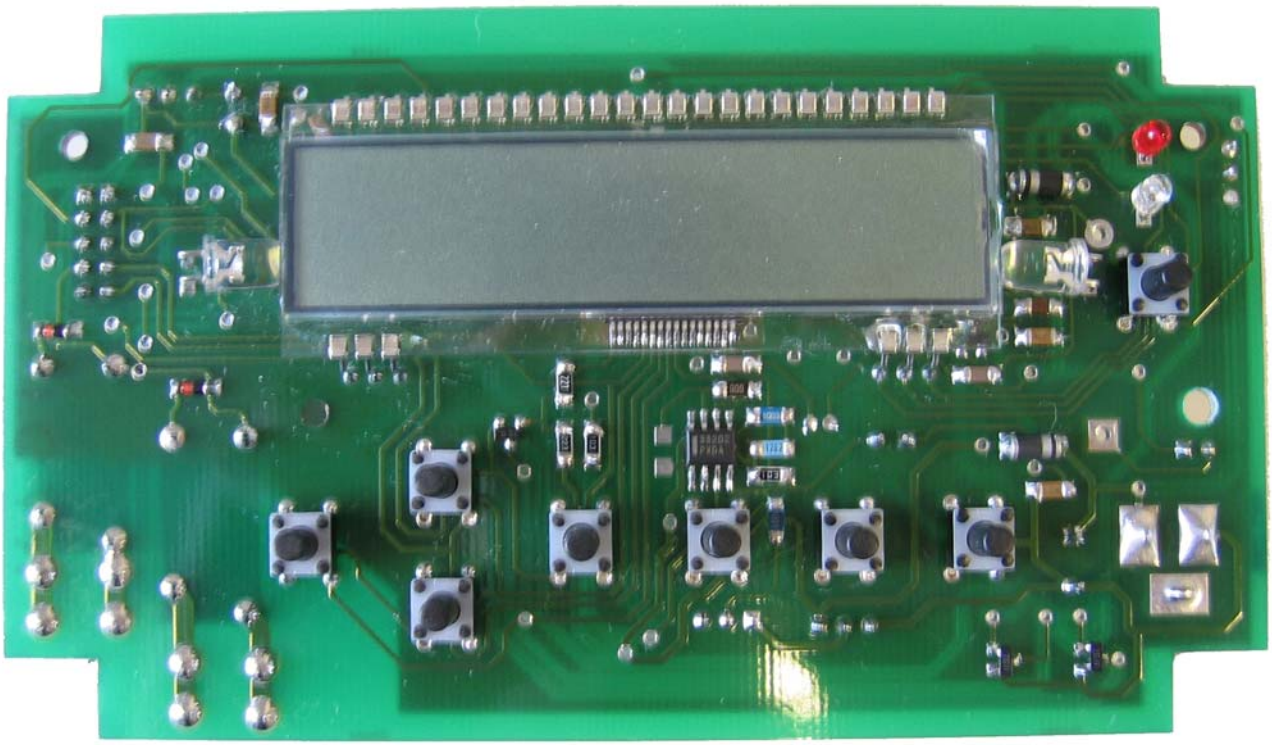


Figure 3-1. The Printed Circuit Board

3.1 Phase-locked Loop

The clock generator module (CGM) generates the system clock signal CGMXCLK, which operates at the crystal-derived frequency of 32.768 kHz. An internal PLL generates the programmable voltage controlled oscillator (VCO) frequency clock and thus determines the bus frequency. A Pierce oscillator configuration is used (**Figure 3-2. MC68HC908LJ12 PLL Connections**), which uses four external components with the crystal connected directly between the crystal

amplifier input pin (OSC1) and the crystal amplifier output pin (OSC2). The PLL analog power and ground pins VDDA and VSSA are connected to the same potential as V_{DD} and V_{SS} for correct operation.

A filter network is connected to the external capacitor pin (CGMXFC) to filter out phase corrections.

Typical values for the network are shown in **Figure 3-2. MC68HC908LJ12 PLL Connections**.

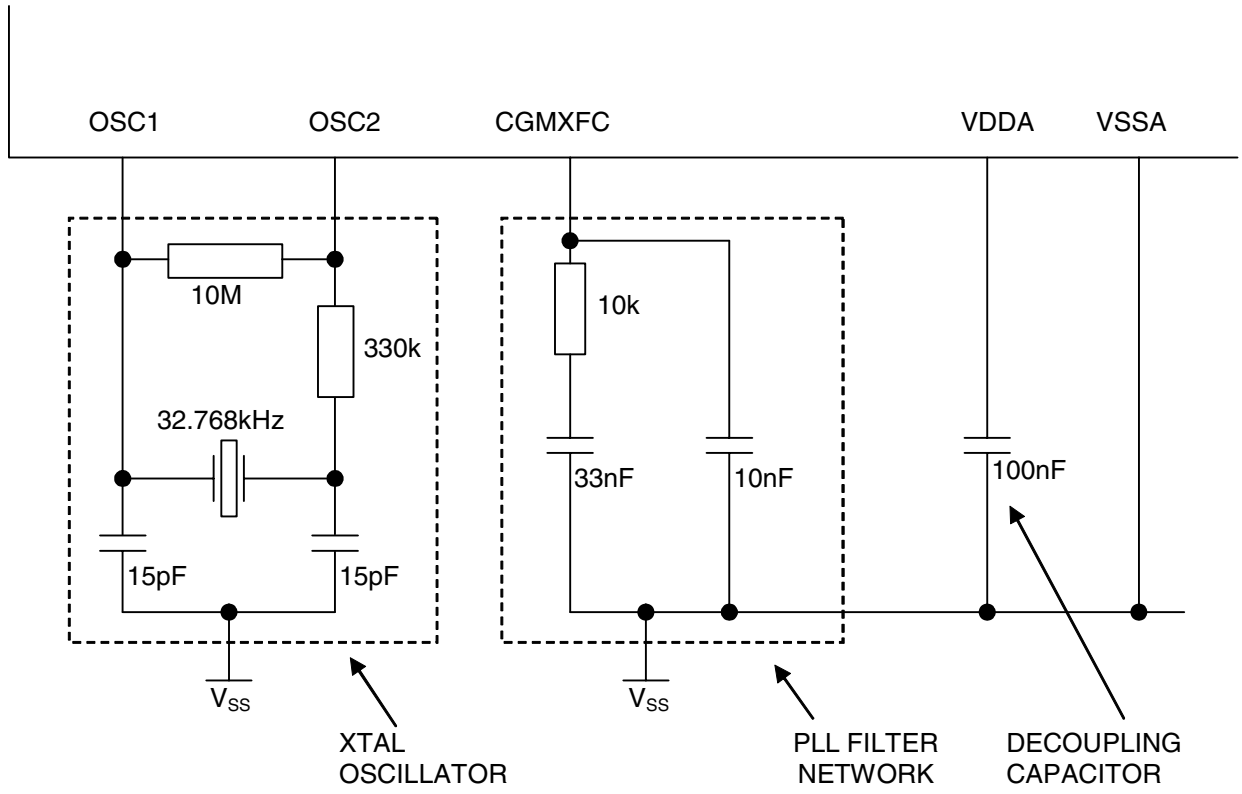


Figure 3-2. MC68HC908LJ12 PLL Connections

3.2 Temperature

3.2.1 Analog Temperature Measurement

The internal temperature sensing is achieved using an LM35. This device gives an output potential of 10 mV/°C and is available in several versions, which allow a wide range of temperature ranges to be measured. The device selected for use in the reference design is an LM35DZ which has a commercial specification of 0–100°C and an absolute accuracy of ±1°C.

A 5 Vdc supply is used for normal operation of the MC68HC908LJ12; this is derived from a mains power source, and the reference for the ADC is also set to this value.

A maximum internal measurement temperature of 40°C was considered suitable for the application, and an operational amplifier was used to scale the sensor output voltage to the necessary 0–5 V. Changing the scaling factor of the operational amplifier and making the necessary software changes alters this maximum temperature value.

The amplifier amplification factor is determined as follows:

Output of sensor @ 40°C = 40 • 10 mV = 400 mV

Required amplification at this temperature for full scale (Av):

5 V/0.40 = 12.5

A non-inverting configuration has been used, therefore:

Av = 1+ Ra/Rb

Using preferred resistor values:

Av = 1 + 11.5k /1k = 12.5

Using the nearest preferred value components, Ra consists of 10k connected in series with 1k2, Rb = 1k. These resistors have a ±1% tolerance value.

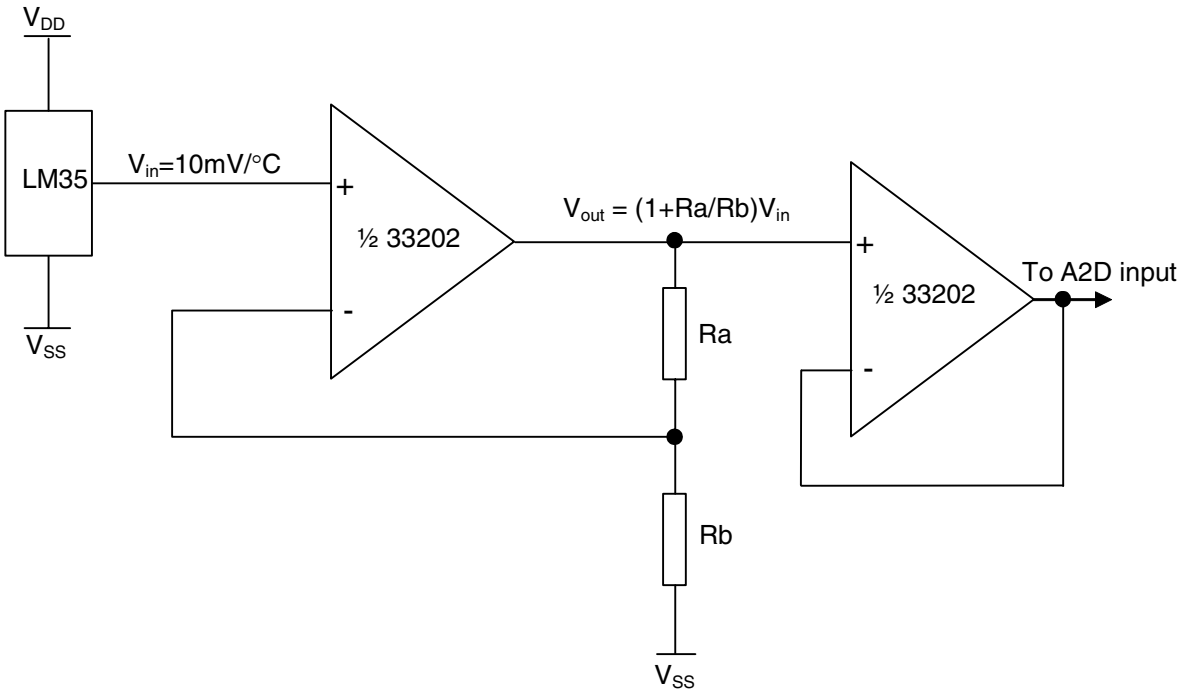


Figure 3-3. Internal Temperature Sensor and Operational Amplifier Circuit Diagram

The requirements for the operational amplifier are for single supply, rail-to-rail operation. A 33202 was selected as a suitable device for the task. This is a dual op-amp and the 'second' stage is connected as a non-inverting voltage follower.

If required, the internal LM35 may be replaced with another, mounted external to the case. A 3-pin connector provides the necessary power, ground and inputs, whilst a 3-pin, on-board header is used to select one of the two sensors. A screened cable should be used for the external analog sensor, although input signal filtering ensures that the sensor may be mounted several metres from the control unit with no undue signal degradation.

3.2.2 2-wire, Serial Temperature Sensor

The external sensor is a 2-wire serial data device, type DS1721. This provides a wide range of temperature measurement and gives a serial data output, in two's complement format.

Up to eight devices may be used on the bus, each hard-wired to a unique address set on three address pins. In this application, only one device is required, therefore address offset zero is selected by connecting all three address pins to Vss. The DS1721 acts as a slave on the bus, and data is requested by and transmitted to the MC68HC908LJ12 via the SDA line, whilst the clock is generated by the microcontroller on the CLK line.

A separate 'Thermostat' output is available from this sensor, although this feature is not used in this design.

A 4-pin Molex KK connector provides the power, ground, clock and data connections to this device.

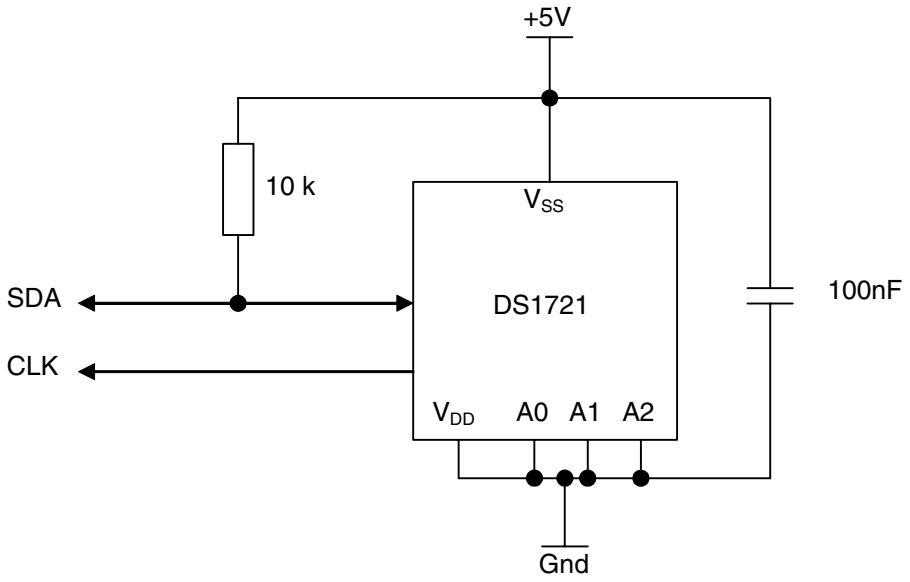


Figure 3-4. External Temperature Sensor Circuit Diagram (DS1721)

3.3 User Interface

3.3.1 Liquid Crystal Display Driver

To obtain maximum life, together with a wide viewing angle on an LCD, it is necessary to drive it using an alternating current. The LCD driver module on the MC68HC908LJ12 contains all the necessary driver circuitry to make driving a multi-segment LCD very straightforward. The MC68HC908LJ12 allows three arrangements of frontplane/backplane configurations:

26 frontplanes x 4 backplanes (104 segments)

27 frontplanes x 3 backplanes (81 segments)

27 frontplanes x 1 backplane (27 segments)

A device with a multiplex ratio of 1/3 was selected for this design, to meet the display requirements listed, together with those of the LCD driver on board the MC68HC908LJ12 (see Motorola Technical Data section 16.3).

The part selected for this design is a VARITRONIX VIM838, which has eight 7-segment digits separated by dots, with a downward pointing arrow under each digit. The display is a transflective, twisted nematic type. It is available in a ‘leaded’ format, as used in this design, or a 'zebra stripe' which would be the preferred option for volume production. The operating frequency range is 30–85 Hz.

3.3.2 Button Detection

Internal pullup resistors are enabled in software. These resistors have a nominal value of 28k. They are internally terminated to V_{SS} and are individually pulled to V_{DD} via the function switches.

3.4 Climate Control

3.4.1 Heating/Cooling

3.4.1.1 Relay Driver

Two relays are incorporated to control the output heating/cooling devices, with uncommitted contact ratings of 3 A @ 230 Vac or 30 Vdc. It is envisaged that the relays will provide ‘pilot’ control to external contactors or switching devices, to allow control of much greater electrical loads.

The relay coils are driven from output ports PB0 and PB1 of the MC68HC908LJ12 via two bipolar junction transistors (BJT), type BC850. A 'snubber' diode is incorporated in parallel with each relay coil to suppress the back emf produced by the relay coil at switch-off.

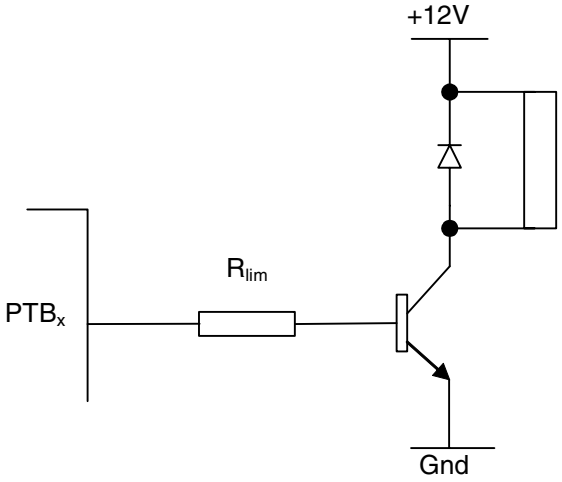


Figure 3-5. Relay Driver Circuit

R_{lim} is included to limit the transistor base drive current from the microcontroller.

3.4.2 Override

If the override button is pressed, a check is done to test whether the climate control is running. If it is running, then it is deactivated and the next alarm is loaded into the alarm register.

If a program is not running when the override button is pressed, then two variables are set to the current time plus one hour. The *OverrideCheck()* function then compares the TOD and 'override time' variables; when they match, it deactivates the override and the climate control. If a scheduled program starts while the override is running, then the override is disabled and control is passed to the program. See Appendix **A.25[operation.c]** — *OverrideCheck()* and **A.7[button.c]** — *DecodeButtons()* for a more detailed account.

3.5 Low Power Modes

Battery life is of great importance in equipment that may experience extended periods of mains failure, for example, in installations that are provided with power from a generator or static inverter. Three possible modes of operation have been considered in the design, in the event of mains supply failure.

3.5.1 Low Current Operation

The microcontroller detects the change of state on the input pin, and performs a 'low power' mode shutdown, comprising the switching off of all relays, LEDs and external sensors, allowing the display to continue displaying the TOD value. The microcontroller essentially runs in a 'reduced function' mode, testing the condition of the power-down input while in a loop cycle. On restoration of the 5 V power supply, the microcontroller returns to normal operation. The current consumption at 3 V, under these conditions, was measured at approximately 3 mA. This may be considered rather high if long periods of power-down are encountered.

3.5.2 Low Voltage Inhibit — STOP Mode

The LVI module is used to detect when the supply voltage falls below a value of 4.5 V. It generates an interrupt that is used to invoke STOP mode. TOD display updating is inhibited, although the RTC timekeeping functions are maintained and register contents preserved. The message 'bAtt' is constantly displayed, to indicate mains failure. The LVI interrupt is still enabled; therefore, when power is restored, the resultant interrupt wakes the microcontroller from STOP mode. This returns the program to normal operation once more. The measured battery current in STOP mode with the LVI module active was approximately 250µA.

3.5.3 Keyboard Interrupt — STOP Mode

The LVI module is used to detect power loss and to invoke STOP mode, in a similar manner to that described above, the difference in this case being that, once the power loss is detected, the LVI module is disabled. This results in a further power saving, although the interrupt necessary to waken the microcontroller from STOP mode does not now occur on restoration of the main supply. To circumvent this problem, a BJT has been added to the KB13 input. The base of this transistor is supplied with current, via a series resistor, from the 5 V supply rail. Thus, on restoration of the supply, the transistor simulates a button press and the resulting keyboard interrupt wakes the microcontroller and restores normal operation. On power-up, the base of the transistor is pulled low via a resistor, supplied from the PD0 pin. PD0 is made an input during STOP mode, changing to an output low on power-up. The transistor therefore turns off, allowing the button input to resume its normal function. This method results in a further reduction in standby current, which has been measured at approximately 30µA. However, this has the disadvantage of increasing the hardware count. Provision has been made on the printed circuit board for the inclusion of this transistor and its associated drive components.

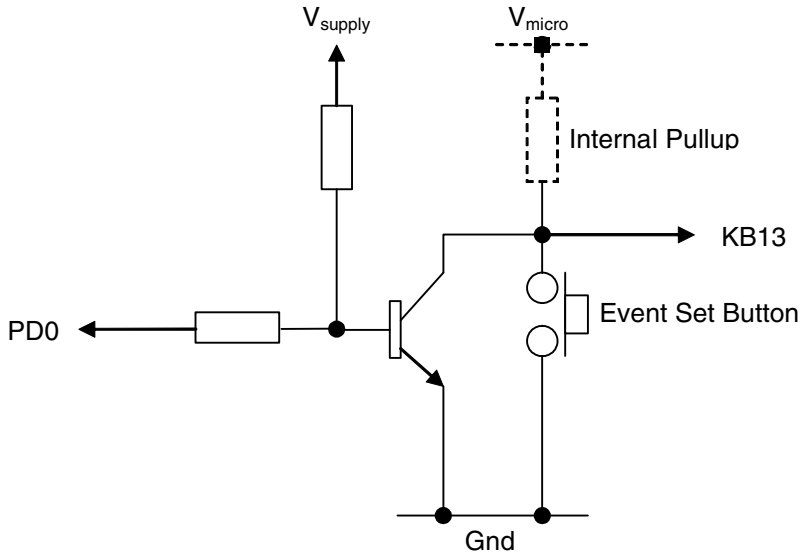


Figure 3-6. Button Simulation Circuit

3.6 Battery Backup

Since the LCD current consumption is very low, it is used to give a visual indication of mains failure by indicating ‘bAtt’. The switch-over between mains power and battery backup is achieved using two diodes. The diode connected in series with the battery was selected to be a Schottky type to minimize the voltage drop and, hence, increase the useful operating life of the battery.

The diode connected in series with the mains-derived supply is a standard silicon diode, type LL4148. To negate the effect of the voltage drop across this diode, a similar device is connected in series with the common terminal of the IC regulator. The effective output from the regulator is:

$$V_{supply} = V_{out(nom)} + V_{diode} = 5.0 + 0.6 = 5.6\ V$$

$$V_{micro} = V_{supply} - V_{diode} = 5.6 - 0.6 = 5.0\ V$$

ADC channel 2 is used to monitor the condition of the battery during normal operation. The nominal terminal voltage of a new lithium cell is approximately 3.25 V. The ADC is used to detect when the terminal

voltage falls below 2.8 V. When this occurs, the display alternates between normal TOD and temperature and ‘Lo bAtt’, at 2-second intervals.

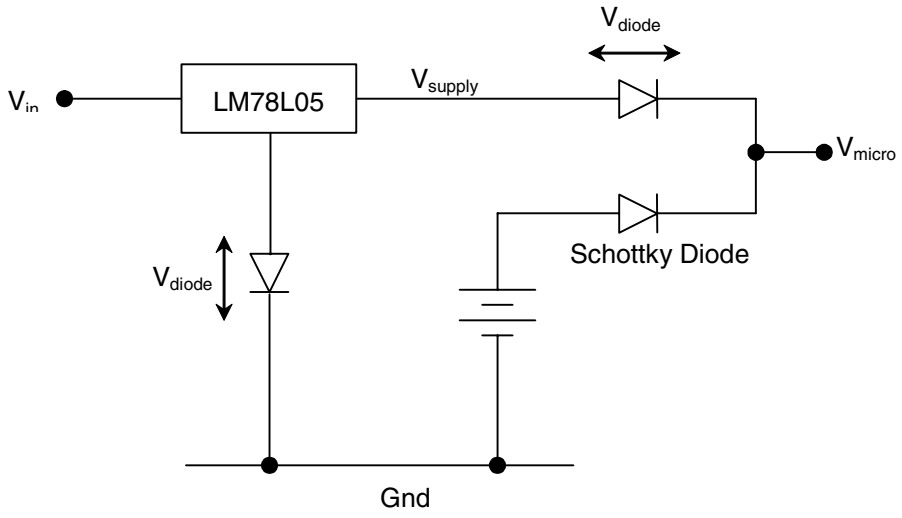


Figure 3-7. Supply Circuit

Section 4. Software

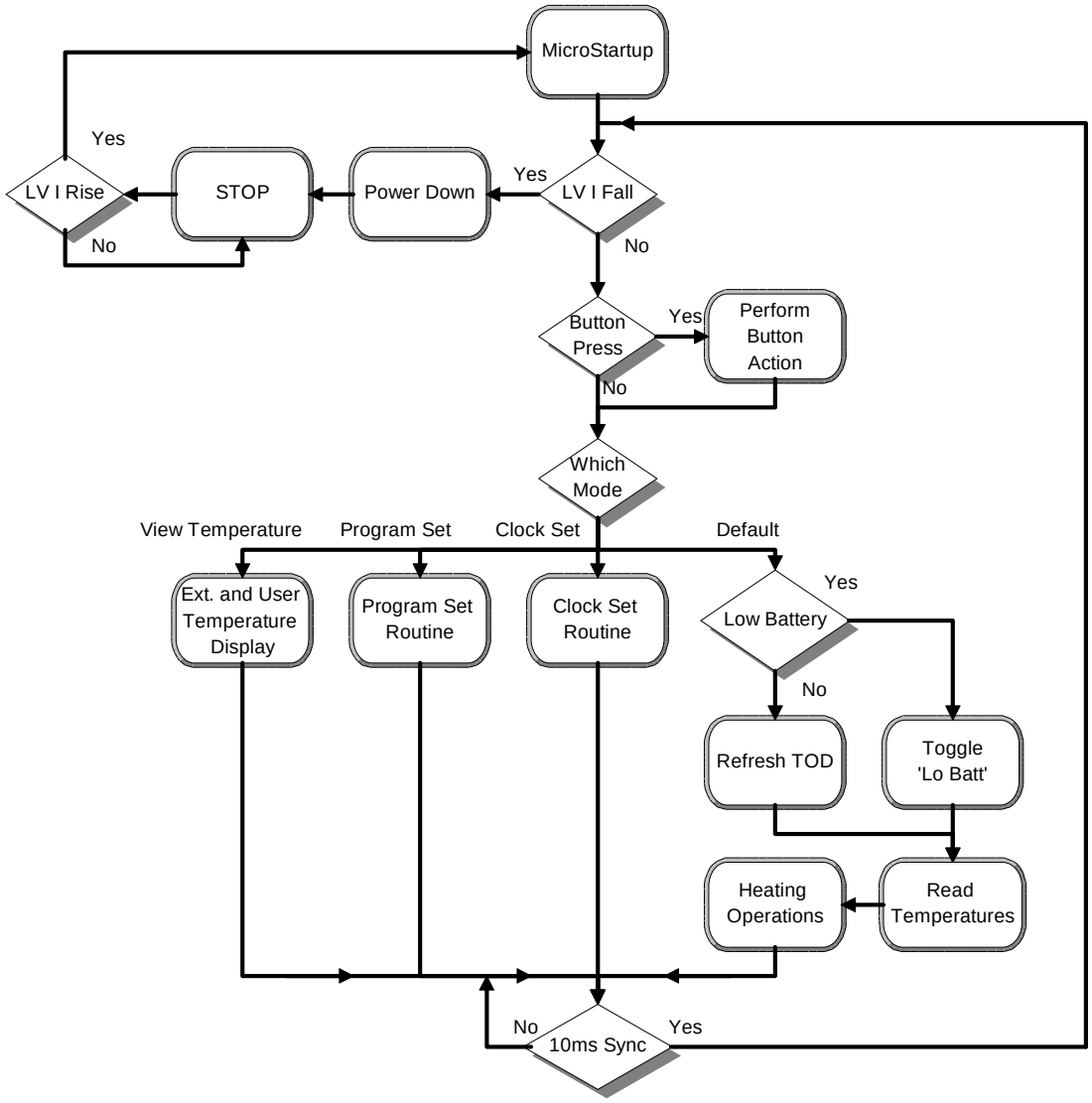


Figure 4-1. Software Overview

A flowchart of the software is shown in Figure 4-1.

4.1 Analog to Digital Converter

Figure 4-2 provides a summary of the ADC registers in the MC68HC908LJ12.

Addr.	Register Name		Bit 7	6	5	4	3	2	1	Bit 0
$003C	ADC Status and Control Register (ADSCR)	Read:	COCO	AIEN	ADCO	ADCH4	ADCH3	ADCH2	ADCH1	ADCH0
		Write:								
		Reset:	0	0	0	1	1	1	1	1
$003D	ADC Data Register High (ADRH)	Read:	ADx	ADx	ADx	ADx	ADx	ADx	ADx	ADx
		Write:	R	R	R	R	R	R	R	R
		Reset:	0	0	0	0	0	0	0	0
$003E	ADC Data Register Low (ADRL)	Read:	ADx	ADx	ADx	ADx	ADx	ADx	ADx	ADx
		Write:	R	R	R	R	R	R	R	R
		Reset:	0	0	0	0	0	0	0	0
$003F	ADC Clock Register (ADCLK)	Read:	ADIV2	ADIV1	ADIV0	ADICLK	MODE1	MODE0	0	0
		Write:								R
		Reset:	0	0	0	0	0	1	0	0

= Unimplemented R = Reserved

Figure 4-2. ADC Register Summary

The 10-bit ADC data is stored in bits 1, 0 of address $003D (ADRH) and bits 7–0 of address $003E (ADRL). The data was chosen to be right-justified to allow simple access to the full ten bits by reading from address $003D.

The code below shows how both bytes of data can be read as one sixteen bit variable (a2d_bytes).

```
a2d_bytes._8bit.hibyte = ADRH.reg;
a2d_bytes._8bit.lobyte = ADRL.reg;
a2d_total += a2d_bytes._16bit;
```

The ADC clock register is initialized in *MicroStartUp()*. The operating frequency for the ADC lies between 32 kHz and 2 MHz. Therefore, the bus speed is divided by two to give a 1 MHz clock. The ADC is set to perform a conversion only when required. The means of initiating a conversion is to write to the ADC status register. The argument channel is passed to the routine, which selects the appropriate ADC channel.

```
unsigned short int ReadA2D( unsigned char channel )
{
unsigned short int          a2d_total;
union uUNSIGNED_INTEGER     a2d_bytes;
unsigned char               ii;


for ( a2d_total = 0, ii = 0; ii < A2D_SAMPLE_COUNT; ii++ )
   {
   ADSCR.reg = 0x00|channel;           // a2d interrupts off, clears COCO bit
   while ( !ADSCR.bit.COCO ) ;         // Wait for conversion to complete
   a2d_bytes._8bit.hibyte = ADRH.reg;  // Read high reg first to latch low reg
   a2d_bytes._8bit.lobyte = ADRL.reg;  // Now read lo reg
   a2d_total += a2d_bytes._16bit;      // Update running total
   }

a2d_total /= A2D_SAMPLE_COUNT;         // Average
return a2d_total;                      // Return averaged result
}                                      // ReadA2D()
```

NOTE: *To obtain stable results, the ADC is read eight times and averaged before returning a value. This technique greatly reduces the effects of any external noise or 'jitter'.*

4.2 Serial Communications

Table 4-1 shows the order in which the master and slave must communicate with each other. The code had to be arranged so that the MC68HC908LJ12 talks to the slave and listens when necessary.

Table 4-1. DS1721 Sequence

LJ12 Mode	DS1721 Mode	Data	Comments
TX	RX	START	LJ12 initiates a START condition
TX	RX	<address, 0>	LJ12 sends DS1721 address, read = 0
RX	TX	ACK	DS1721 generates acknowledge bit
TX	RX	0x51	LJ12 sends start convert T protocol
RX	TX	ACK	DS1721 generates acknowledge bit
TX	RX	START	LJ12 initiates a repeated START condition
TX	RX	<address, 0>	LJ12 sends DS1721 address, read = 0
RX	TX	ACK	DS1721 generates acknowledge bit
TX	RX	0xAA	LJ12 sends read temperature protocol
RX	TX	ACK	DS1721 generates acknowledge bit
TX	RX	START	LJ12 initiates a repeated START condition
TX	RX	<address, 1>	LJ12 sends DS1721 address, read = 1
RX	TX	ACK	DS1721 generates acknowledge bit
RX	TX	<1 data byte>	DS1721 transmits MSB of temperature
TX	RX	ACK	LJ12 generates acknowledge bit
RX	TX	<1 data byte>	DS1721 transmits LSB of temperature

NOTE: *For more information, please refer to the DS1721 data sheet.*

A delay function provides an in-line delay while the MC68HC908LJ12 waits for an acknowledge bit or temperature data. The DS1721 operates at a maximum of 400 kHz; therefore, a delay of 5 µs was chosen to allow ample time. The main function that performs the above actions is *ExternalTemperatureRead()*. The DS1721 has a tolerance of ±1 degree; therefore, a calibration constant (EXTERNAL_CALIBRATION) exists to compensate (see Appendix A.19 [i2c.c] — *ExternalTemperatureRead()*). The Flash capabilities of the MC68HC908LJ12 can allow this calibration value to be inserted as part of the end of line test sequence in a production run, allowing the overall accuracy of the temperature reading to be improved.

4.3 Real Time Clock

The MC68HC908LJ12 has a built in RTC module comprising the registers shown in Figure 4-3.

$0044	RTC Status Register (RTCSR)	Read:	ALMF	CHRF	DAYF	HRF	MINF	SECF	TB1F	TB2F
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$0045	Alarm Minute Register (ALMR)	Read:	0	0	AM5	AM4	AM3	AM2	AM1	AM0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$0046	Alarm Hour Register (ALHR)	Read:	0	0	0	AH4	AH3	AH2	AH1	AH0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$0047	Second Register (SECR)	Read:	0	0	SEC5	SEC4	SEC3	SEC2	SEC1	SEC0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$0048	Minute Register (MINR)	Read:	0	0	MIN5	MIN4	MIN3	MIN2	MIN1	MIN0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$0049	Hour Register (HRR)	Read:	0	0	0	HR4	HR3	HR2	HR1	HR0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$004A	Day Register (DAYR)	Read:	0	0	0	DAY4	DAY3	DAY2	DAY1	DAY0
		Write:								
		Reset:	0	0	0	0	0	0	0	1
$004B	Month Register (MTHR)	Read:	0	0	0	0	MTH3	MTH2	MTH1	MTH0
		Write:								
		Reset:	0	0	0	0	0	0	0	1
$004C	Year Register (YRR)	Read:	YR7	YR6	YR5	YR4	YR3	YR2	YR1	YR0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$004D	Day-Of-Week Register (DOWR)	Read:	0	0	0	0	0	DOW2	DOW1	DOW0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$004E	Chronograph Data Register (CHRR)	Read:	0	CHR6	CHR5	CHR4	CHR3	CHR2	CHR1	CHR0
		Write:								
		Reset:	0	0	0	0	0	0	0	0

= Unimplemented R = Reserved

Figure 4-3. Real Time Clock Registers

Each register rolls over after the appropriate number has been reached. As shown in **Figure 4-3. Real Time Clock Registers**, there are also two alarm registers: minutes and hours. When the RTC hour and minute registers are equal to the alarm registers, the alarm flag is set and can generate an interrupt, if desired.

The month, year and chronograph data registers are not used in this application, but may be incorporated, if more advanced timekeeping capabilities are required.

There is a wide range of flags available with their associated interrupts, ranging from 10 ms to 1 day. However, only the 10 ms interrupt is used in this application. It is used to synchronize the main loop to 10 ms for precision in control. Although not used in this application, the 10 ms synchronization is useful if other types of 'mains synchronous' output switching devices (triacs, for example) are employed (assuming a 50 Hz mains supply).

```
///////////////
// 10ms sync //
///////////////
while ( !flags1.bit._10MS ) ;       // Waiting for interrupt
flags1.bit._10MS = 0;               // reset, for next loop iteration
```

The main program runs for about 2 ms per cycle. However, when the temperatures are refreshed (every five seconds) the period increases to about 4 ms per cycle. This is well within the 10 ms interrupt that has been used, and gives ample time for the loop synchronization to occur.

RTCCR1 controls which interrupts are active and, in this case:

```
// RTC setup
RTCCR2.reg        = 0x60;           // rtc off, set for 32.768kHz xtal...
                                    // chronograph setup
RTCCR1.reg        = 0x40;           // CHRIE set
RTCCR2.bit.RTCE = 1;                // ...all done, start RTC
```

RTCCR2 is set up for a 32.768 kHz crystal with the chronograph counter cleared and enabled. Once both registers are set, the RTC is initiated.

The *TenMsSynchronise()* function was set up to toggle variables at different time intervals based on the 10 ms main loop. Toggles are set up for 0.25, 0.5 and 1 second intervals (see Appendix **A.25 [operation.c]** — *TenMsSynchronise()* for a more detailed account).

```
void TenMsSynchronise( void )
{
if ( ++_10ms%25   == 0 )            // every 0.25s
   {                                //
   _0P25sec        ^= 0x01;         //
                                    //
   if ( _10ms%50    == 0 )          // every 0.5s
      {                             //
      SFLASH      ^= 0x01;          //
      }                             //
                                    //
   if ( _10ms%100  == 0 )           // every 1.0s
      {                             //
      one_second  ^= 0x01;          //
      _5sectick++;                  //
      }                             //
                                    //
   if ( _10ms >= 200 )              // Unsigned char, 256 max
      {                             //
      _10ms = 0;                    // reset for next
      }                             //
   }                                //
}                                   // TenMsSynchronise()
```

4.4 Phase-lock Loop

A 2 MHz bus speed was chosen to keep current drain as low as possible while still allowing the maximum main loop cycle time to remain comfortably inside the 10 ms synchronization.

Table 4-2. PLL Frequency Table

CGMPCLK (MHz)	f_{BUS} (MHz)	f_{RCLK} (kHz)	R	N	P	E	L
8.0	2.0	32.768	1	f5	0	0	d1
9.8304	2.5	32.768	1	12c	0	1	80
10.0	2.5	32.768	1	132	0	1	83
16.0	4.0	32.768	1	1e9	0	1	d1
19.6608	4.9	32.768	1	258	0	2	80
20.0	5.0	32.768	1	263	0	2	82
29.4912	7.4	32.768	1	384	0	2	c0
32.0	8.0	32.768	1	3d1	0	2	d0
16.0	4.0	32.768	1	1e9	1	2	d0
8.0	2.0	32.768	1	f5	2	2	d0
4.0	1.0	32.768	1	7b	3	2	d0

The code below initializes the PLL at the desired frequency:

```
void InitialisePLL( void )
{
                                    ////////////////////////////////
PBWC.reg       = 0x80;              // auto mode                    //
PTCL.reg       = 0x00;              // settings here...             //
PMS            = 0x00F5;            // as described in...           //
PMRS.reg       = 0xD1;              // the MC68HC908LJ12            //
PMDS.reg       = 0x01;              // Rev2.0 data book section     //
                                    // 8.4.6 page 113               //
PTCL.bit.PLLON = 1;                 // turn pll on after settings   //
                                    // 'set'                        //
                                    ////////////////////////////////

////////////////////////////////////////////////
// wait for the required frequency to be reached //
////////////////////////////////////////////////
ServiceWatchDog();
while ( !PBWC.bit.LOCK );

PTCL.bit.BCS = 1;                   // pll clock drives CGMOUT
}                                   // InitialisePLL()
```

4.5 Temperature

All temperatures are updated every five seconds to prevent display jitter, which is most noticeable with decimal values. In normal mode, the internal temperature is displayed on the right-hand side of the display. The function *InsideTemperature()* is used to calculate this value.

The ADC is read and stored in a variable. This variable is then manipulated to correspond to the maximum and minimum temperature scale. The calculations are shown below:

```
////////////////////////////////////////////////////////////////////////////
// The internal temperature calculations:                                 //
//                                                                          //
// The A2D byte is multiplied by 49 to convert it to 10000 * the input    //
// voltage. It is then / by 100 to convert it to 100 * the input voltage.//
//                                                                          //
// The result is then * by 41 which is the maximum temperature that can   //
// be read by the A2D, and / by 5 which is the maximum input voltage.     //
//                                                                          //
// Finally the result is / by 10 to leave it in a manageable form         //
//                                                                          //
// eg.   internal temperature = 26.0CA2D function returns 650              //
//       (49 * 650)/100 = 316       (316*41)/50 = 259 (0.33% error)       //
////////////////////////////////////////////////////////////////////////////

internal_temperature = ( 49 * ReadA2D( CHANNEL0 ))/100;
internal_temperature = ( internal_temperature * 41 )/( 5 * 10 );
```

NOTE: *The multiples of 10 are used to allow the integers to represent decimal values.*

This calculated internal temperature is then manipulated so that there are two variables, one stores any multiples of ten and the other stores the units and decimal values. These are then sent to the *DisplayNumbers()* function to be displayed on screen.

This temperature display can also be called before the five seconds count to refresh the display when in default mode. This is useful when returning to default mode; otherwise it could take up to five seconds to display the internal temperature. The refresh variable is set in order to refresh the display.

When the temperature button is held down, the program runs in temperature view mode. The external temperature replaces the internal

temperature and the last or current user temperature is displayed on the left-hand side of the screen. The letters ‘P’ and ‘E’ are also displayed to aid the user in temperature type recognition. While in this mode the user can manually change the desired temperature for the current override or climate cycle by using the increment and decrement buttons.

NOTE: *This does not affect the stored program in memory.*

The serial communication returns two bytes of information, the ‘hibyte’ and the ‘lobyte’. Both are in two’s complement form. After conversion, one is subtracted from the ‘hibyte’. This is because the 0.5 bit is treated as if it were a one by the DS1721.

NOTE: *For more information, please see the DS1721 data sheet.*

The ‘lobyte’ is then converted to a decimal as follows:

```
////////////////////////////////
// fractional part processing //
////////////////////////////////
dec = 0;
if ( lobyte.byte >= 0x10 )          // is bit 4 or greater, set
   {
   if ( lobyte.bit.bit7 )
      {
      dec += 5000;                  // 0.5*10000
      }
   if ( lobyte.bit.bit6 )
      {
      dec += 2500;                  // 0.25*10000
      }
   if ( lobyte.bit.bit5 )
      {
      dec += 1250;                  // 0.125*10000
      }
   if ( lobyte.bit.bit4 )
      {
      dec += 625;                   // 0.0625*10000
      }
                            // checking for '0.05' second dec place rounding
   if ( dec%1000 >= 500 )           // is remainder >= 0.05
      {
      dec += 1000;                  // 0.1 correction on first decimal
      }
                                    // div by 1000 (not 10000) to
   dec /= 1000;                     // produce integer value for first
                                    // decimal place
```

In a similar manner to the methods used in decoding the internal temperature, the values are manipulated so that they can be sent to the function *DisplayNumber()* where they are displayed.

4.6 User Interface

4.6.1 Liquid Crystal Display Driver

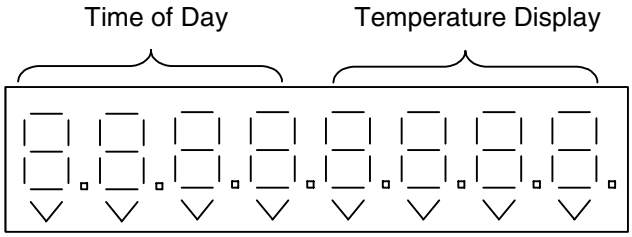


Figure 4-4. LCD Layout

The LCD has a operating frequency range of 30–85 Hz. The 1/3 duty cycle was selected as the LCD has three backplanes. The 42.7 Hz frequency was therefore selected as being the optimum choice for a 1/3 duty cycle and a 32.768 kHz crystal.

Table 4-3. LCD Frequency Table

LCLK2	LCLK1	LCLK0	Divide Ratio	LCD Waveform Base Clock Frequency LCDCLK (Hz)		LCD Frame Rate f_{XTAL} (1) = 32.768 kHz		LCD Frame Rate f_{XTAL} = 4.9152MHz	
				f_{XTAL} = 32.768 kHz	f_{XTAL} = 4.9152MHz	1/3 duty	1/4 duty	1/3 duty	1/4 duty
0	0	0	128	256	—	85.3	64	—	—
0	0	1	256	128	—	42.7	32	—	—
0	1	0	512	64	—	21.3	16	—	—
0	1	1	1024	32	—	10.7	8	—	—
1	0	0	16384	—	300	—	—	100	75
1	0	1	32768	—	150	—	—	50	37.5
1	1	0	65536	—	75	—	—	25	18.75
1	1	1	Reserved						

For optimum optical performance the fast charge selection is set to a time period of LCDCLK/64. This value was derived empirically to achieve a good contrast ratio for the display used.

```
//////////////////////
// Display Startup //
//////////////////////
LCDCLK.byte = 0x29;
//          = 00101001;
//              |||||||
//              |||||||___ Base Clock/Frame Rate
//              ||||||____   "    "     "    "
//              |||||_____   "    "     "    "
//              ||||______ LCD Duty Selection
//              |||_______  "   "      "
//              ||________ Fast Charge
//              |_________  "     "
```

For the LCD control register, the LCD, fast charge and low current options were all enabled and a high bias voltage was applied to obtain the optimum viewing angle.

```
LCDCR.reg   = 0xbc;
//          = 10111100;
//            ||||||||
//            ||||||||___ Contrast Controls
//            |||||||____    "         "
//            ||||||_____    "         "
//            |||||______    "         "
//            ||||_______  Low Current
//            |||________  Fast Charge
//            ||_________  N/A
//            |__________  LCD Enable (1 = enabled)
```

At start-up, the segments are lit for one second as a screen test to confirm that all segments are operational.

Table 4-4 and Table 4-5 show the relationship between the 'control' hexadecimal codes, and the displayed values for the LCD.

Table 4-4. Bit to Segment Correlation – 1st Digit

	LDAT1								Hex		LDAT2				Hex
	7	6	5	4	3	2	1	0			3	2	1	0	
Number	**--**	**D**	**G**	**A**	**--**	**DP**	**C**	**B**			**--**	**^**	**E**	**F**	
0	0	1	0	1	0	0	1	1	**53**		0	0	1	1	**3**
1	0	0	0	0	0	0	1	1	**03**		0	0	0	0	**0**
2	0	1	1	1	0	0	0	1	**71**		0	0	1	0	**2**
3	0	1	1	1	0	0	1	1	**73**		0	0	0	0	**0**
4	0	0	1	0	0	0	1	1	**23**		0	0	0	1	**1**
5	0	1	1	1	0	0	1	0	**72**		0	0	0	1	**1**
6	0	1	1	1	0	0	1	0	**72**		0	0	1	1	**3**
7	0	0	0	1	0	0	1	1	**13**		0	0	0	0	**0**
8	0	1	1	1	0	0	1	1	**73**		0	0	1	1	**3**
9	0	0	1	1	0	0	1	1	**33**		0	0	0	1	**1**
DP	0	0	0	0	0	1	0	0	**04**		0	0	0	0	**0**
AN	0	0	0	0	0	0	0	0	**00**		0	1	0	0	**4**

Table 4-5. Bit to Segment Correlation – 2nd Digit

	LDAT3								Hex		LDAT2				Hex
	7	6	5	4	3	2	1	0			7	6	5	4	
Number	**--**	**^**	**E**	**F**	**--**	**D**	**G**	**A**			**--**	**DP**	**C**	**B**	
0	0	0	1	1	0	1	0	1	**35**		0	0	1	1	**3**
1	0	0	0	0	0	0	0	0	**00**		0	0	1	1	**3**
2	0	0	1	0	0	1	1	1	**27**		0	0	0	1	**1**
3	0	0	0	0	0	1	1	1	**07**		0	0	1	1	**3**
4	0	0	0	1	0	0	1	0	**12**		0	0	1	1	**3**
5	0	0	0	1	0	1	1	1	**17**		0	0	1	0	**2**
6	0	0	1	1	0	1	1	1	**37**		0	0	1	0	**2**
7	0	0	0	0	0	0	0	1	**01**		0	0	1	1	**3**
8	0	0	1	1	0	1	1	1	**37**		0	0	1	1	**3**
9	0	0	0	1	0	0	1	1	**13**		0	0	1	1	**3**
DP	0	0	0	0	0	0	0	0	**00**		0	1	0	0	**4**
AN	0	1	0	0	0	0	0	0	**40**		0	0	0	0	**0**

The tables also allow direct access to individual segments for displaying a chevron or decimal point; for example:

```
LDAT1.bit.bit2 = 1;
```

This displays a decimal point on the far right hand digit.

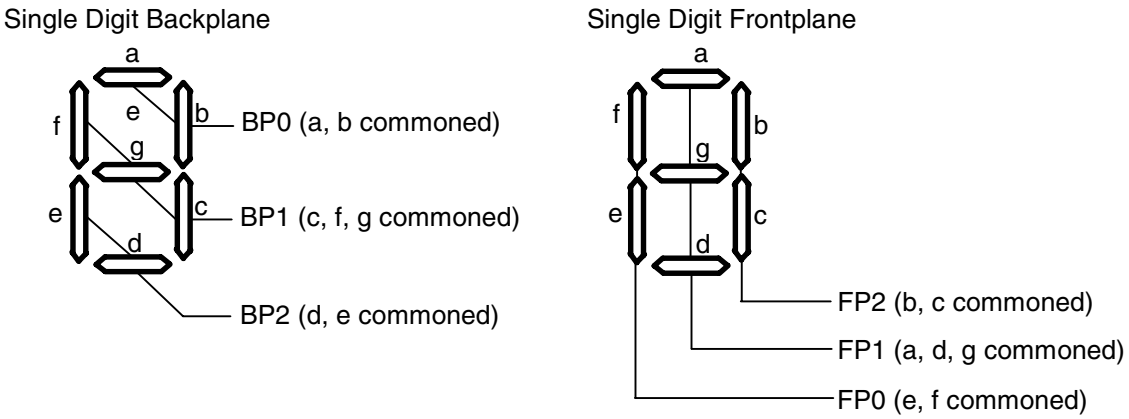


Figure 4-5. Segment Labelling

The *DisplayNumber()* function is used to decode a single integer and display it in the desired location on screen. The number to be displayed and the digit location are passed to the function when it is called. Initially, the function must determine whether the digit number is odd or even so that it can address the correct registers. This is done by taking the modulus of 2 from the number and checking for a one or zero. Once this has been established, the digit variable is altered to accommodate the non-linear relationship between the variable and the registers, resulting from each digit occupying one-and-a-half display registers.

```
if ( digit%2 )                      // odd digit check
  {
   switch ( digit )
      {
      case 1: digit--;break;        // Compensates for 1.5 bytes
      case 5: digit++;break;        // being used per digit
      case 7: digit += 2;  break;   //
      }
```

After the correct sets of registers are addressed, the numeric variable is decoded into its hexadecimal code value, which is then placed into the registers.

4.6.2 Button Detection

Eight buttons are used in this application. All eight of the designated keyboard pins (PTA bits 0–3 and PTD bits 4–7) are used for this purpose. The function *ReadButtons()* is called every 10 ms to process a button press. Each button has its own assigned pattern with two additional patterns for simultaneous button presses in temperature view mode.

```
////////////////////
// button decodes //
////////////////////
#define  DEFAULT_BUTTONS     0xff    // 11111111
#define  BUTTON_1            0xfe    // 11111110
#define  BUTTON_2            0xfd    // 11111101
#define  BUTTON_3            0xfb    // 11111011
#define  BUTTON_4            0xf7    // 11110111
#define  BUTTON_5            0xef    // 11101111
#define  BUTTON_6            0xdf    // 11011111
#define  BUTTON_7            0xbf    // 10111111
#define  BUTTON_8            0x7f    // 01111111
#define  BUTTON_5_DEC        0xed    // 11101101
#define  BUTTON_5_INC        0xeb    // 11101011
```

The algorithm used incorporates a button press and a button release routine. The ability to have an auto-scroll is included; this occurs when a button is pressed, and remains pressed for the duration of the de-bounce counter. This condition will occur when performing an adjustment of any time value. A single press and hold of the increment or decrement button will cause the auto-scroll to operate. The auto-scroll feature can be enabled/disabled on any button, as required. The flag that allows this feature is '*button_flags.bit.AUTO_SCROLL*'. When the flag is set, the auto-scroll feature is enabled.

Software

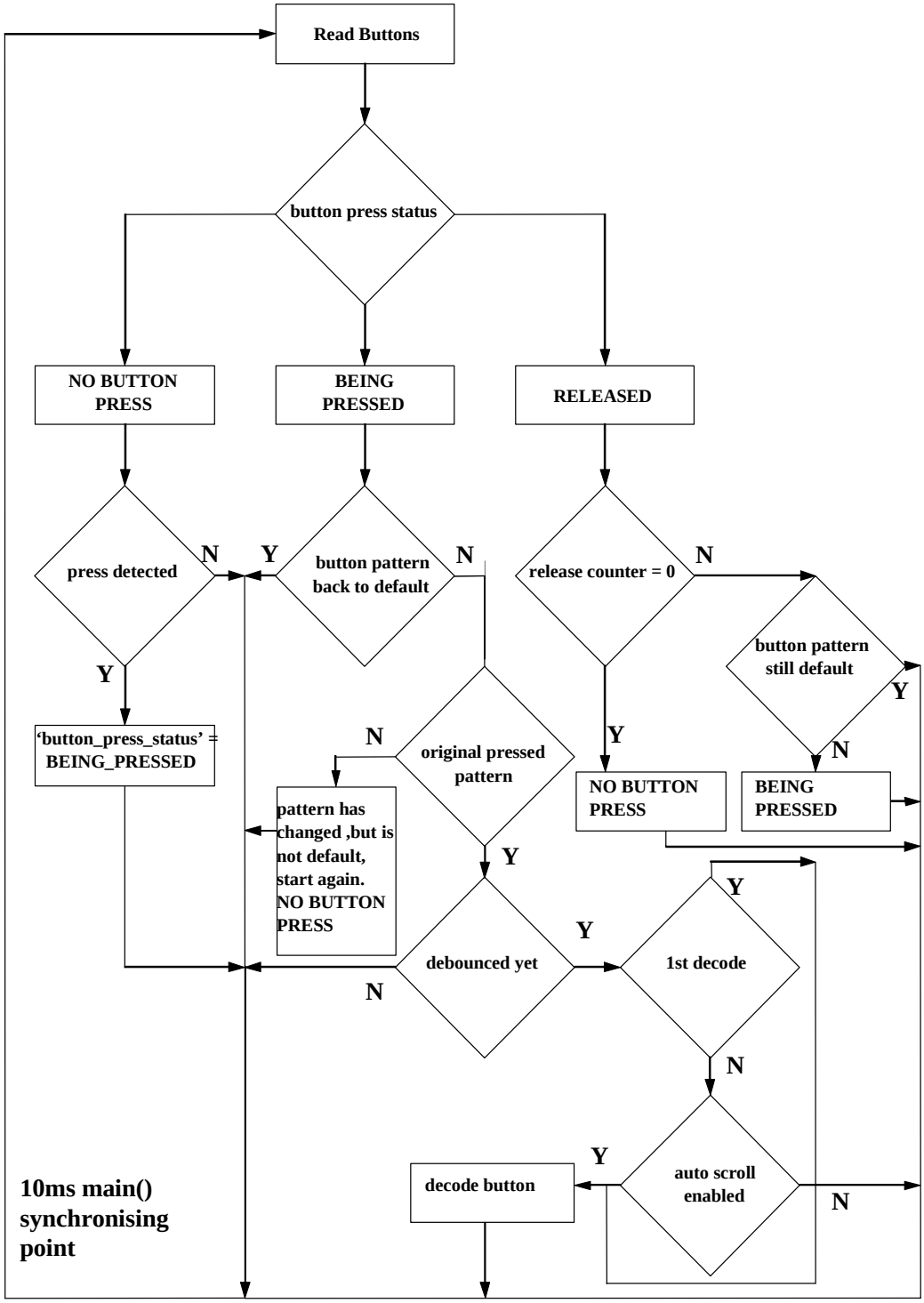


Figure 4-6. Button Processing Flow Chart

4.7 Climate Control

4.7.1 Heating/Cooling

A ‘Mode' button allows the selection of one of four modes.

- Automatic — both heating and cooling will be active
- Heat — only the heating will be active
- Cool — only the cooling will be active
- Off — climate control will never be active

The mode of operation is indicated by two LED indicators, which also serve to indicate heating/cooling activity by flashing at 1 Hz when active. Relays are used as control devices for the heating and cooling functions.

4.7.2 Override

If the override button is pressed, a check is done to test whether the climate control is running. If it is running, then it is deactivated and the next alarm is loaded into the alarm register.

If a program is not running when the override button is pressed, two variables are set to the current time plus one hour. The *OverrideCheck()* function then compares the TOD and ‘override time’ variables. When they match, it deactivates the override and the climate control. If a scheduled program starts while the override is running, then the override is disabled and control is passed to the program. See Appendix **A.26 [operation.h]** — *OverrideCheck()* and Appendix A.7 [button.c] — *DecodeButtons()* for a more detailed account.

4.7.3 SMART — Intelligent Temperature Prediction

The **'SMART'** function allows the controller to modify the user's activation times to take account of internal and external temperature variations. **'SMART'** mode aims to achieve the user temperature at the desired time of building occupation. This is independent of external temperature variations, but is limited by the **'SMART'** prediction cap.

When the climate control is activated the user temperature is compared with the internal temperature. If the internal temperature is below the hysteresis band (user temperature - one degree), the heating will switch on, as long as the correct mode is active. If the internal temperature is above or equal to the user temperature, the cooling will activate provided that the correct mode is active. See A.25 [operation.c] — *ClimateControl()* for a more detailed account.

If the SMART button is pressed, then the SMART chevron will be displayed. When in SMART mode, an algorithm examines the internal, external and desired temperatures, one hour prior to the commencement of the program. A predictive calculation is then performed, based on the following method:

> A percentage of internal temperature to desired temperature is calculated. The difference of this value from 100% (x) is used below.
>
> The ratio of internal temperature and external temperature is taken and multiplied by the previously calculated value, 'x'. This value is then multiplied by a scaling factor of 3, to enable correct operation.
>
> Finally this time is subtracted from the programmed 'on time'.

The new calculated 'on time' is then loaded into the alarm register. The maximum SMART time is capped at 50 minutes to prevent the user's heating/cooling system from being activated for extended time periods. The actual SMART algorithm is as follows.

```
smart_time = ( 1000 * internal_temperature )  // user_temperature;
                                             // Smart calculations, stage 1
if ( smart_time > 1000 )                       // For Smart cooling
   {                                           //
   smart_time -= 1000;                         //
   }                                           //
else                                           // For Smart heating
   {                                           //
   smart_time = 1000 - smart_time;             //
   }                                           //
smart_time = 3 * (( smart_time * internal_temperature ) /
                  ( 100 * external_temperature      ));
                                               // Smart calculations, stage 2
```

NOTE: *The multiples of 10 are used to allow the integers to represent decimal values. This prevents loss of accuracy in rounding errors.*

4.8 Programming Modes

4.8.1 Clock

When in 'clock set' mode, the main program branches to the function *Cset()*. A switch statement is used to determine which value is being set. The first case is the 'hour set'. Every 0.25 seconds the hour digits toggle on/off to indicate the variable, which is being set. The 0.25 second toggle is used for this purpose, see 4.3 Real Time Clock. On clearing the display, the minute digits and the chevrons are refreshed, so that they do not appear to have turned off. If the increment or decrement buttons are pressed, the *ReadButtons()* function decodes the button press (see 4.6.2 Button Detection) and alters the hour digit accordingly. As previously mentioned, the increment and decrement are capable of auto-scrolling so the user can keep the button pressed and the corresponding register will scroll round the values. A maximum and minimum cap is already in place in the RTC registers, however, when the maximum is reached, the next increment resets the RTC register to zero. When the minimum is reached, the next decrement sets the register to its maximum value.

```
case HOUR_SET:
if ( _0P25sec )                    // For every other 1/4 of a second
   {
   SFLASH = 1;                     // Displays divider decimal point
   HRClr();                        // Refreshes hour digits
   MINClr();                       // Refreshes minute digits
   ChevronRefresh();
   }
else
   {
   ClearDisplay();                 // Allows hour digits to flash
   LDAT10.bit.bit2 = 1;            // Decimal point remains constant
   MINClr();                       // Minute digits refreshed
   ChevronRefresh();
   }
break;
```

After pressing the ‘clock set’ button, the programming mode switches to ‘minute set’. This function performs in the same manner as the ‘hour set’, with the exception that the minutes now toggle instead of the hours. Again, the register ‘rolls over’ when required.

The next program mode is the Day of Week (DOW). The first thing that is done is to remove all previous chevrons to prevent ‘ghosting’. The chevron bits can be directly accessed using a ‘bit set’ or ‘bit clear’ (see 4.6.1 Liquid Crystal Display Driver). Again, the 0.25-second toggle is used to make the chevron flash. The increment and decrement buttons move the chevron from Monday through Sunday. Sunday is equal to 0x00 in the DOW register so, when Saturday is reached and the increment button is pressed, the DOW register is reset to zero. Again, a reverse process is implemented for decrement.

When the clock set button is pressed again, the set mode returns to default and the main loop no longer branches to the function. There are no variables to store or manipulate, because the clock set function changes the RTC registers directly. Whenever the clock set button is pressed, the RTC second register is reset so that the TOD can be set precisely, if required.

4.8.2 Climate Control Cycles

Up to four individual on/off time periods may be set at any point in time. Each period may be set for either a single event up to seven days in advance, or for multiple timing events at the same time each day. The daily settings can be made for weekdays, weekends or individual days.

The time setting of a program works in a very similar way to the clock set but with a few additional modes.

On entering the alarm set mode, the first screen displayed is ‘Pr’ followed by the number ‘1’ flashing. The increment and decrement buttons can be used to select a program. The maximum number of programs in this application is four. Increasing the data structure and the button cap limits can modify this.

NOTE: *Any new programs must be added to the enum in define.h before the 'NO_PROGRAM' declaration (see Appendix A.13 [define.h]).*

The user now has two options. Pressing the ‘Prog’ button will continue to set a new program. However, pressing the temperature/clear button will erase the program that is currently flashing from the stored memory.

Setting the off times equal to the on times does this. The ignore condition is implemented in Alarm.c (see Appendix A.3 [Alarm.c]).

After the user has pressed the alarm set button, the DOW selection mode is activated. The selected program number is then displayed followed by ‘On’. Once in this mode, the user can press the increment and decrement buttons in the same way as the DOW mode in clock set. There are, however, two additional selections. The user can set programs for individual days, Monday–Friday or Saturday–Sunday. This gives flexibility to users who may have the same program patterns for the weekdays but would perhaps prefer the weekends to be different.

The next mode is the ‘On Time’ set. This operates in exactly the same way as the clock set function, but it is repeated for ‘Off Time’ set also (see 4.8.1 Clock). Once ‘Off Time’ is activated, the displayed ‘On’ changes to ‘Off’.

After the on and off times are set, the’ final mode is the user temperature set. A ‘t’ is then displayed instead of ‘Off’ and a default value of ‘20’ flashes next to a ‘c’. Again, increment and decrement change this value, which has a maximum of 40 and a minimum of 0. The scaling of the measurement circuitry sets the upper limit. The minimum is set to 0, as the temperature sensor cannot detect negative values, and it is unlikely that the building temperature would fall to that value. If the maximum or minimum is reached, the rollover value is set to a default of 20°C.

At the end of each mode, the data that has just been set is stored into the ‘program’ structure. The structure stores six different variables:

- On hour
- On minute
- DOW
- Off hour
- Off minute
- Desired temperature

Once the temperature has been set, the *ProCheck()* function is called to check if the new program overlaps with any other programs that have

already been stored. The on and off data is converted into two variables by multiplying the hours by 60 and then adding the minutes to this value. These two variables are then checked with the already stored programs, which are converted into the same format (see Appendix **A.5 [Alarmset.c]** — *ProCheck()* for a more detailed account). If an overlap is found, then 'Err' is displayed, followed by the stored program, which has the conflict, then '-' followed by the new program number that has been entered. The error screen is displayed for one second. The program is then cancelled by altering the off times to the ignore condition. If no error is found then the program continues as normal.

After the *ProCheck()* function, the *Alarm()* function is called. The alarm function is used to load the next scheduled program into the alarm registers.

When the alarm function is called, a check is done to see if the climate control is running. If it is, then the corresponding 'Off time' is loaded into the alarm register in preparation for the program ending. Otherwise, a check is done to determine the next scheduled program. This is done by checking that the DOW is correct and then converting the on times into the same format as mentioned previously in *ProCheck()*. Once the next on time is loaded into the alarm register, one hour is subtracted and stored in two separate variables for use in the **'SMART'** function (see 4.7.3 SMART — Intelligent Temperature Prediction).

The Alarm function is called when: a new alarm has just been programmed, an alarm flag is detected, or at the beginning of a new day. If no programs were active over midnight, the next day's programs would not be loaded into the alarm register. For this reason, the last call has been added. The day flag is checked and, when it is set, as long as a program is not running, the alarm function is called again. The day flag is then reset in preparation for the next day.

4.9 Low Power Modes

4.9.1 Low Current Operation

Firstly, a measurement of the supply voltage should be taken using channel 1 of the ADC. A function *PowerFailure()* is then needed to detect if the supply voltage has dropped below the desired level. If a reduction is detected, the MC68HC908LJ12 is prepared for low current operation. This is done in a very similar way to preparing for STOP mode (see **4.9.2 Low Voltage Inhibit — STOP Mode**). The climate control relays and LEDs are switched off. Ports B and D are then set to be inputs to reduce any current drain. The ten millisecond interrupt is disabled and the LCD control register (LCDCR) is modified to decrease the bias voltage for battery operation. The program then remains in a loop, refreshing the TOD. Also, 'Fail' is displayed on the right-hand side of the display to show that the supply has failed. The program remains in the loop until the supply voltage rises again.

```
void PowerFailure( void )
{
if ( LowPower )
   {
   PTB.byte = 0x00;                 // Turn off heating/cooling & LED's
   DDRA.reg = 0x00;                 // Set all ports to input
   DDRB.reg = 0x00;                 //
   DDRD.reg = 0x00;                 //
   RTCCR1.bit.CHRIE = 0;            // Disables RTC 100Hz interrupt

   SFLASH = 1;                      // Decimal point after hour digits
   LCDCR.reg   = 0xb0;              // Adjust LCD to lower power mode
   while ( !PTA.bit.bit5 )          // While in low power mode
      {
      ServiceWatchDog();
      HRClr();                      // Refresh TOD
      MINClr();                     //
      LDAT6.byte = 0x33;            // Display 'FAIL'
      LDAT5.byte = 0x03;            //
      LDAT4.byte = 0x33;            //
      LDAT3.byte = 0x30;            //
      LDAT2.byte = 0x03;            //
      LDAT1.byte = 0x40;            //
      }
   RTCCR2.bit.CHRCLR = 1;           // Reset chronograph counter
   MicroStartUp();                  // Reinitialises the micro
   Alarm();                         // Load next alarm into the register
   }
}                                   // PowerFailure()
```

When the supply voltage returns, the program runs the same initialisation routines as the LVI and keyboard methods (see 4.9.2 Low Voltage Inhibit — STOP Mode). The program then returns to normal operation.

4.9.2 Low Voltage Inhibit — STOP Mode

In this application, the LVI is used to detect when the main supply voltage drops or rises. When a drop in supply voltage is detected, the LVI module generates an interrupt. Within the interrupt routine, the climate control relays and LEDs are switched off. Ports B and D are then set to be inputs to reduce current drain. The LCD is cleared and the control register (LCDCR) is modified to decrease the bias voltage, so that it can be read when operating from the low voltage of the battery. ‘bAtt’ is displayed on the right-hand side of the screen to tell the user that the controller is now running from the battery.

In addition to the above, the ten millisecond chronograph interrupt must also be disabled by changing the correct RTC control register (RTCCR1). Once the above has been completed, the MC68HC908LJ12 is brought into STOP mode by calling the function *STOP()*.

```
#pragma TRAP_PROC
void LVI_INTERRUPT( void )
{
ServiceWatchDog();
PTB.byte = 0x00;                    // Turn off heating/cooling & LED's
LVISR.bit.LVIIAK = 1;               // Resets the interrupt flag

if ( LVISR.bit.LVIOUT )
   {
   DDRB.reg = 0x00;                 // Set ports to input to reduce...
   DDRD.reg = 0x00;                 // current drain
   RTCCR1.bit.CHRIE = 0;            // Disables RTC 100Hz interrupt

   ClearDisplay();
   LCDCR.reg  = 0xb0;               // Adjust LCD to lower power mode
   LDAT6.byte = 0x36;               // Displays 'bAtt'
   LDAT5.byte = 0x23;               //
   LDAT4.byte = 0x33;               //
   LDAT3.byte = 0x36;               //
   LDAT2.byte = 0x03;               //
   LDAT1.byte = 0x60;               //

   STOP();
   }
else
```

```
    {
    RTCCR2.bit.CHRCLR = 1;          // Reset chronograph counter
    MicroStartUp();                 // Return micro to previous state
    Alarm();                        // Load next alarm into the
                                    // register.
    }
}                                   // LVI_INTERRUPT()
```

NOTE: *The LVIOUT flag is used to detect whether the supply voltage has risen or fallen.*

When the supply voltage returns, the LVI issues another interrupt to bring the MC68HC908LJ12 out of STOP mode. The interrupt routine then clears the chronograph counter (preventing a ten millisecond interrupt occurring instantly) and runs the initialization function *MicroStartUp()* (see Appendix A.27 [startup.c]). Once the registers of the MC68HC908LJ12 have been re-initialized, the *Alarm() function* is called to allow the next program to be loaded into the alarm register. Normal operation is then resumed.

4.9.3 Keyboard Interrupt — STOP Mode

If a keyboard interrupt is required to bring the MC68HC908LJ12 out of STOP mode, then the keyboard interrupts must be initialized before entering STOP mode (see MC68HC908LJ12 Technical Data, page 367). The LVI module should be used in a similar way to above. However, the module then must be disabled to save current consumption. This is done in configuration register one (CONFIG1). By clearing the LVISTOP bit, the LVI module is automatically disabled during STOP mode. See below for the LVI interrupt function.

```
#pragma TRAP_PROC
void LVI_INTERRUPT( void )
{
ServiceWatchDog();
PTB.byte = 0x00;                    // Turn off heating/cooling & LEDs
LVISR.bit.LVIIAK = 1;               // Resets the interrupt flag

DDRB.reg = 0x00;                    // Set ports to input to reduce...
DDRD.reg = 0x00;                    // current drain
RTCCR1.bit.CHRIE = 0;               // Disables RTC 100Hz interrupt

ClearDisplay();
LCDCR.reg  = 0xb0;                  // Adjust LCD to lower power mode
LDAT6.byte = 0x36;                  // Displays 'bAtt'
```

```
LDAT5.byte = 0x23;                  //
LDAT4.byte = 0x33;                  //
LDAT3.byte = 0x36;                  //
LDAT2.byte = 0x03;                  //
LDAT1.byte = 0x60;                  //

KBSCR.bit.IMASK  = 1;               // Mask interrupt flag
KBIER.byte     = 0xff;              // Enable keyboard interrupts
KBSCR.bit.ACKK  = 1;                // Clear interrupt flag
KBSCR.bit.IMASK  = 0;               // Unmask interrupt flag

STOP();
}                                   // LVI_INTERRUPT()
```

In addition to the LVI interrupt function, a keyboard interrupt function must also be added. When a keyboard interrupt is detected, the MC68HC908LJ12 is brought out of STOP mode and performs the interrupt routine. This routine must re-initialize the MC68HC908LJ12. An example routine is shown below.

```
#pragma TRAP_PROC
void KEYBOARD_INTERRUPT( void )
{
ServiceWatchDog();
KBSCR.bit.IMASK = 1;                // This prevents further interrupts
KBSCR.bit.ACKK  = 1;                //

RTCCR2.bit.CHRCLR = 1;
MicroStartUp();                     // Reinitialises the micro
Alarm();                            // Load next alarm into the register
}                                   // KEYBOARD_INTERRUPT()
```

4.10 Battery Backup

Every five seconds, the backup battery voltage is read using channel 2 of the ADC. This value is then accessed in default mode and compared with a predefined value (see Appendix A.13 [define.h] — LOWBATT). If the voltage is less than the predefined value then the display toggles every two seconds between the TOD/internal temperature and ‘Lo bAtt’.

```
void TimeRefresh( void )
{
if ( BatteryCheck <= LOWBATT && two_second )
   {
   LDAT12.byte = 0x00;              // Clears unused digit
   LDAT11.byte = 0x03;              // Displays 'Lo'
```

```
   LDAT10.byte = 0x40;              //
   LDAT9.byte  = 0x26;              //
   LDAT8.byte  = 0x20;              //
   LDAT7.byte  = 0x00;              // Clears unused digit
   LDAT6.byte  = 0x36;              // Displays 'bAtt'
   LDAT5.byte  = 0x23;              //
   LDAT4.byte  = 0x33;              //
   LDAT3.byte  = 0x36;              //
   LDAT2.byte  = 0x03;              //
   LDAT1.byte  = 0x60;              //
   }
else
   {
   LDAT6.byte  = 0x00;              // Removes the 'b'
   LDAT5.byte &= 0x0f;              //
   MINClr();                        // Refresh TOD display
   HRClr();                         //
   Refresh = 1;                     // Refresh internal temperature
   ChevronRefresh();
   }
}                                   // TimeRefresh()
```

The predefined value is determined as follows:

> The minimum recommended voltage for the MC68HC908LJ12 is 3.3 V ± 10%. Therefore ‘Lo bAtt’ should be displayed when the battery voltage is 2.7 V or less. To convert 2.7 V into ADC counts, it is divided by the reference voltage (4.8 V) and multiplied by 1024 as the ADC is set at 10-bit resolution. This gives a value of 576 counts.

4.11 On-board Flash Programming

The compiler used for the project was Metroworks Code Warrior IDE v5.2.1149. The debugging tool that is incorporated into the IDE is very useful. It allows quick and easy communication with the development kit for stepping through the code.

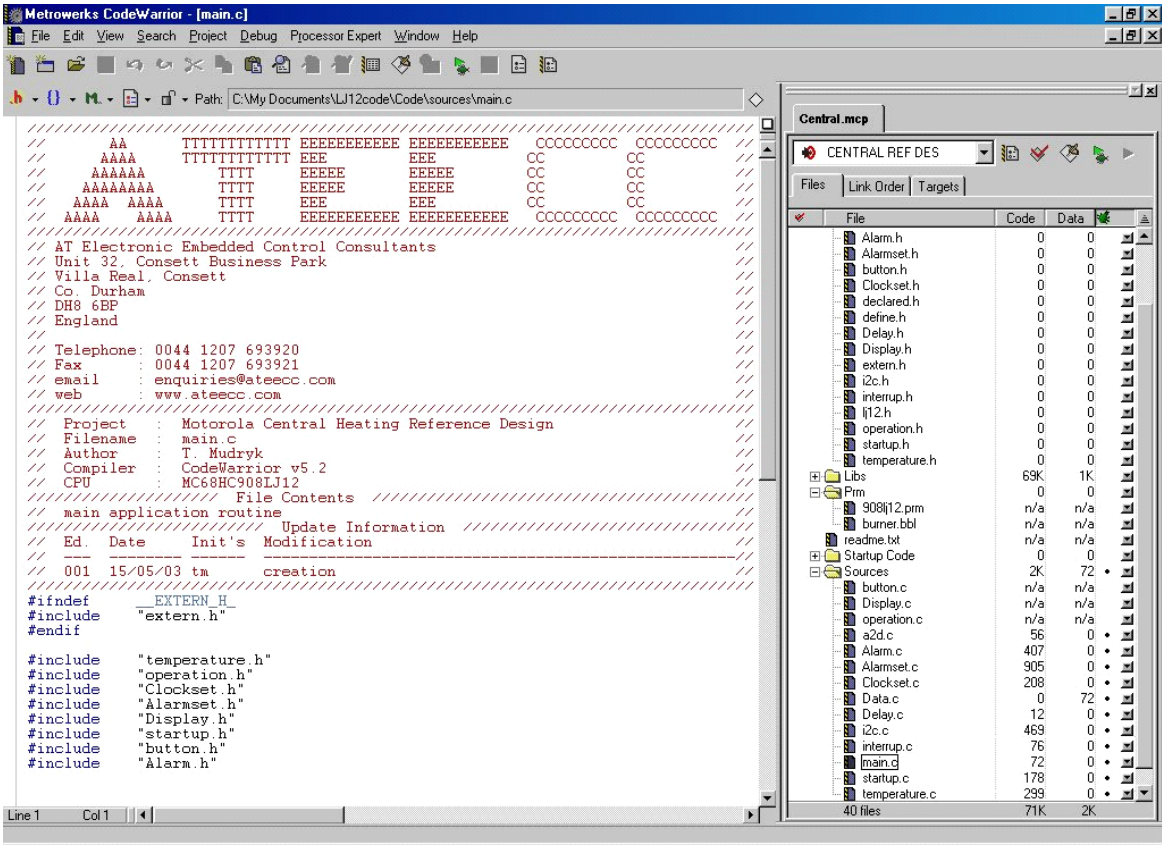


Figure 4-7. Metroworks Codewarrior Screenshot

The debugger also has the option to program the MC68HC908LJ12. The following method was used for this project. P&E's programming software was installed (see Motorola's data CD–ROM). The target settings were then changed to the P&E software. To select this, go to the menu 'component' —> 'set target'. Once there, change the target interface to the 'P&E target interface'. The P&E software is then loaded. Once complete, enter programming mode by going to the menu 'PEDebug' —> 'Mode:' and selecting 'In-Circuit Debug/Programming'. To load the project onto the microcontroller, go to 'PEDebug' —> 'load' and select the project's abs file. The screen should now look similar to the screen shot below.

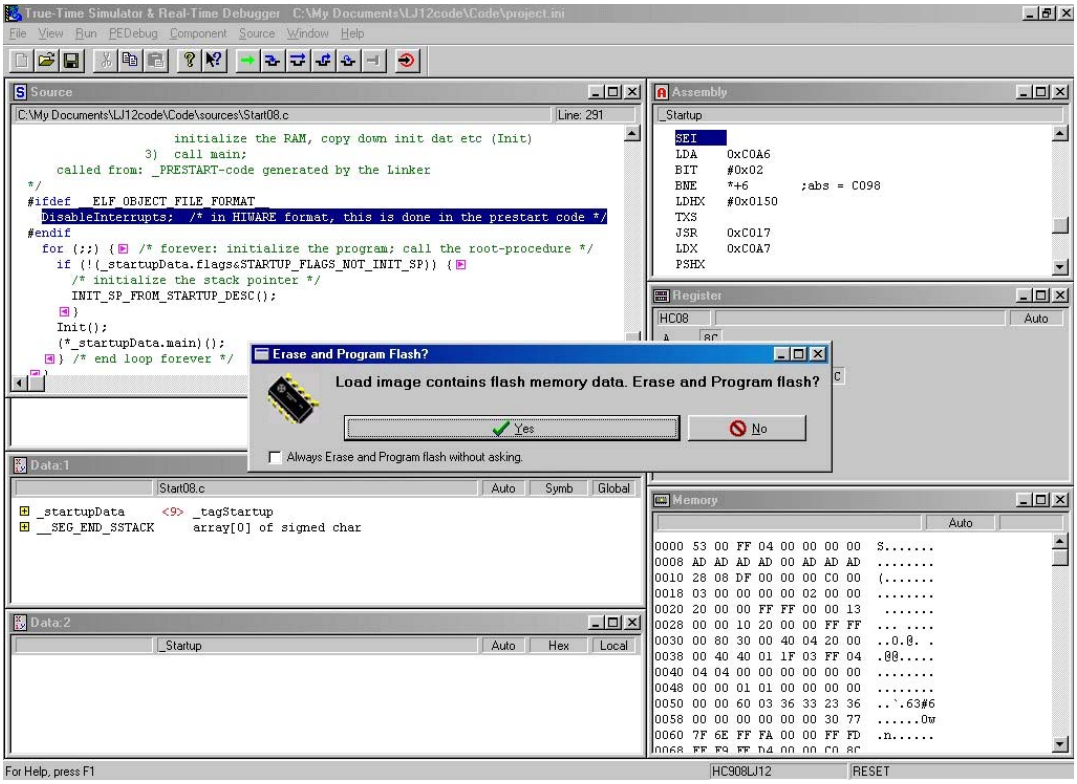


Figure 4-8. Programming Confirmation

Click ‘yes’. The program will go through the steps of programming the microcontroller as shown below.

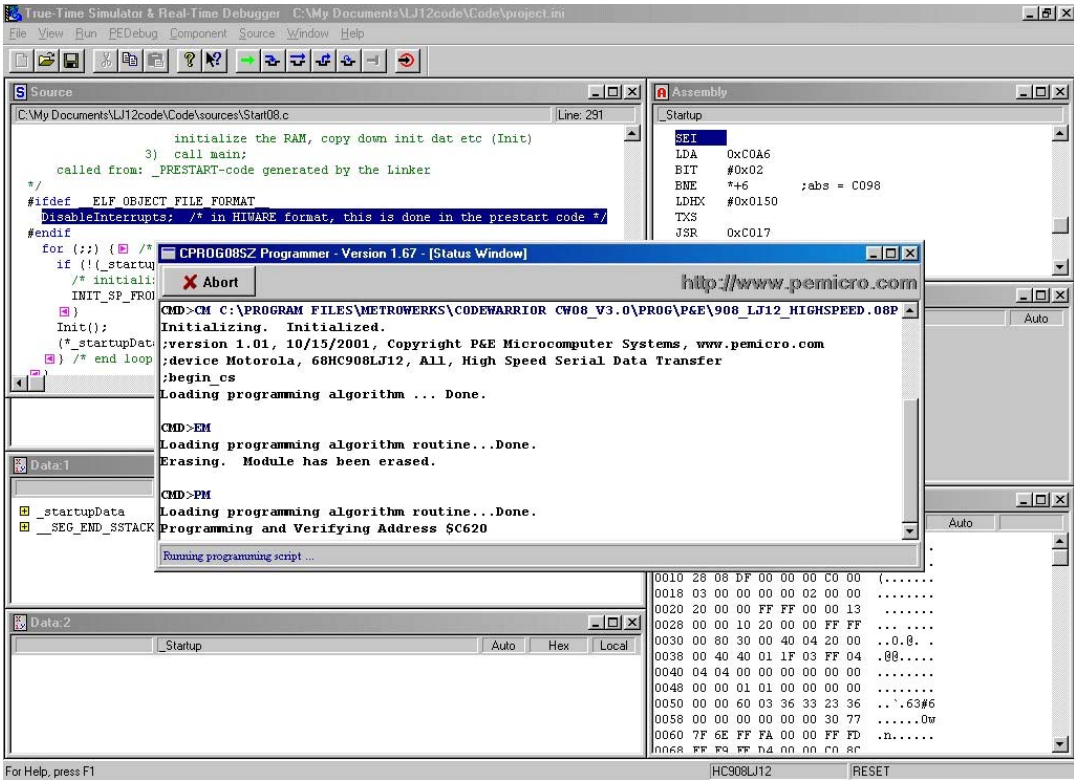


Figure 4-9. During Programming

When the programming has completed, the small window will disappear. Once the program has been set up in this way, microcontrollers can be programmed using the load command, after they have been connected to the PC's serial port. Provision is made on the printed circuit board (PCB) layout to allow the MC68HC908LJ12 to be programmed 'in circuit'. A suitable circuit to interface the device to a PC via an RS232 serial port, is given in Appendix D. Programming Circuit Diagram. This interface is connected to the main board by means of an IDC header and ribbon cable.

Software

Appendix A. Software Routines

A.1 [a2d.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEEE EEEEEEEEEEE    CCCCCCCCC  CCCCCCCCC //
//      AAAA     TTTTTTTTTTT EEE         EEE          CC         CC          //
//     AAAAAA        TTTT     EEEEE       EEEEE        CC         CC          //
//    AAAAAAAA       TTTT     EEEEE       EEEEE        CC         CC          //
//   AAAA  AAAA      TTTT     EEE         EEE          CC         CC          //
//  AAAA    AAAA     TTTT     EEEEEEEEEEE EEEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                                 //
// Unit 32, Consett Business Park                                             //
// Villa Real, Consett                                                        //
// Co. Durham                                                                 //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                    //
//  Filename  :  a2d.c                                                        //
//  Author    :  T. Mudryk                                                    //
//  Compiler  :  CodeWarrior v5.2                                             //
//  CPU       :  MC68HC908LJ12                                                //
////////////////////  File Contents  ////////////////////////////////////////
//  a2d routines                                                              //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  13/05/03 tm      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#include   "extern.h"
#endif

#include    "delay.h"
#include   "a2d.h"


////////////////////////////////////////////////////////////////////////////
// This function returns an averaged analogue value. The averaging is        //
// currently 8, determined by the value of A2D_SAMPLE_COUNT (in a2d.h)       //
```

Software Routines

```
//                                                                          //
// NOTE: the summing variable 'a2d_total' is 16 bit, this means we can store //
// 32 readings of 0x3ff ie 11 1111 1111. If you want greater averaging then  //
// make 'a2d_total' an unsigned long.                                        //
//                                                                          //
// Argument : analogue channel to read                                      //
// Returns  : averaged analogue result                                      //
//////////////////////////////////////////////////////////////////////////////
unsigned short int ReadA2D( unsigned char channel )
{
unsigned short int       a2d_total;
union uUNSIGNED_INTEGER  a2d_bytes;
unsigned char            ii;


for ( a2d_total = 0, ii = 0; ii < A2D_SAMPLE_COUNT; ii++ )
   {
   ADSCR.reg = 0x00|channel;           // a2d interrupts off, clears COCO bit
   while ( !ADSCR.bit.COCO ) ;         // Wait for conversion to complete
   a2d_bytes._8bit.hibyte = ADRH.reg;  // Read high reg first to latch low reg
   a2d_bytes._8bit.lobyte = ADRL.reg;  // Now read lo reg
   a2d_total += a2d_bytes._16bit;      // Update running total
   }

a2d_total /= A2D_SAMPLE_COUNT;         // Average
return a2d_total;                      // Return averaged result
} // ReadA2D()
//---------------------------------------------------------------------------
```

A.2 [a2d.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  a2d.h                                                      //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Header file for 'a2d.c'                                                 //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/05/00 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __A2D_H_
#define     __A2D_H_

#define  CHANNEL0               0x00
#define  CHANNEL1               0x01
#define  CHANNEL2               0x02
#define  CHANNEL3               0x03
#define  CHANNEL4               0x04
#define  CHANNEL5               0x05
#define  A2D_BANDGAP_REF        0x06
#define  A2D_VDDA               0x1d
#define  A2D_VSSA               0x1e
#define  A2D_OFF                0x1f
#define  CLEAR_CHANNEL_SEL      0xe0
#define  A2D_SAMPLE_COUNT       8
#define  ShutDownA2D()          (ADSCR.reg = A2D_OFF)   // ADC power off
#define  A2D_STABILISATION      16// approx 100us == 200 bus cycles @ 2MHz...
                                  // bus speed '16' => 15+(12*16) = 207 bus...
                                   // cycles == 103.5us

////////////////
// prototypes //
```

Software Routines

```
////////////////

unsigned short intReadA2D( unsigned char );

#endif
```

A.3 [Alarm.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                 //
//  Filename  :  Alarm.c                                                    //
//  Author    :  T. Mudryk                                                 //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Alarm routine                                                           //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  27/06/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#include   "extern.h"
#endif

#include    <stdio.h>
#include   "Alarm.h"


void Alarm( void )
{
struct sPROG_PARAMS* ptr  = NULL;
unsigned char        temp;


if ( PRNumber < NO_PROGRAM                      // If alarm is currently running
   {                                            // Set off time in alarm register
        ALMR.byte = program[PRNumber].min_off;
        ALHR.byte = program[PRNumber].hour_off;
        PRNumber  = NO_PROGRAM;
        PR2       = 1;                          // Initiate override variable
   }
```

```
else
   {
        for ( ptr = &program[0]; ptr <= &program[3]; ptr++ )
           {
                temp = ptr->dow;

                if ( temp == 8 )
                   {
                         if  ( 0 < DOWR.byte && DOWR.byte < 6 )
                            {
                                temp = DOWR.byte; // Load DOW value for Minute - Fri
                            }
                   }
                else if ( temp == 9 )
                   {
                         if ( DOWR.byte == 0 || DOWR.byte == 6 )
                         {
                                temp = DOWR.byte; // Load DOW value for Sat - Sun
                         }
                   }

                OrderCheck(ptr, temp);           // Call of function to check...
           }  // end of 'for'                    // the order of the alarms

        if ( program[PRNumber].hour_on == 0 )SHour = 23;
        else                                        SHour = program[PRNumber].hour_on - 1;
                                                 // Store Alarm hour - 1 for smart
        SMin = program[PRNumber].min_on;         // Copy of alarm minute for smart
        PR2  = 0;                                // Reset override variable
   }
} // Alarm()
//---------------------------------------------------------------------------


void OrderCheck( const struct sPROG_PARAMS* ptr, unsigned char dow_check )
{
unsigned short int   stored_total_on;
unsigned short int   stored_total_off;
unsigned short int   alarm_reg_total;
unsigned short int   current_time_total;


if ( dow_check == DOWR.byte )                    // Is it the Day of the alarm
   {
        stored_total_on    = ( 60 * ptr->hour_on ) + ptr->min_on;
        stored_total_off   = ( 60 * ptr->hour_off) + ptr->min_off;
        alarm_reg_total    = ( 60 * ALHR.byte    ) + ALMR.byte;
        current_time_total = ( 60 * HRR.byte     ) + MINR.byte;

        if ( stored_total_on == stored_total_off )   return;
        // If the on time is equal to the off time (clear/ignore condition)
        if ( current_time_total < stored_total_on )
           {
        if ( PRNumber < NO_PROGRAM )             // If a program has already been...
             {                                   // loaded into the alarm register
```

```
                  if ( stored_total_on < alarm_reg_total )
                     {
                     ALMR.byte = ptr->min_on;    // Set new on time
            ALHR.byte = ptr->hour_on;            //
                        PRNumber  = (unsigned char)(ptr - &program[0]);
            }                 // Store which program is now is the alarm reg.
         }
      else
         {
         ALMR.byte = ptr->min_on;                // Set new time
                  ALHR.byte = ptr->hour_on;      //
                  PRNumber  = (unsigned char)(ptr - &program[0]);
                  }            // Store which program is now is the alarm reg.
      }
        }
} // OrderCheck()
//---------------------------------------------------------------------------


void AlarmFlag( void )
{
unsigned char  temp;


if ( RTCSR.bit.ALMF )                           // If the Alarm flag is set
        {
        temp = ALHR.reg;                        // Clears Alarm flag
        if (PRNumber < NO_PROGRAM)              // If alarm is to be activated
        {
                Active = 1;                     // Set heating control variable
                user_temperature = program[PRNumber].temperature;
                }                               // Desired temperature is recalled
        else                                    // If a program has just ended
                {
                Active = 0;                     // Clear heating control variable
                PTB.bit.bit0 = 0x00;            // Turn off heating
                PTB.bit.bit1 = 0x00;            // Turn off Cooling
           }

        Alarm();
        }
} // AlarmFlag()
//---------------------------------------------------------------------------


void NewDayRefresh( void )
{
unsigned char temp;


if ( RTCSR.bit.DAYF )                           // If the Day flag is set
        {
   if ( PRNumber >= NO_PROGRAM )                // If a program is not running
           {
           Alarm ();                            // Refresh the alarm register
```

Software Routines

```
            }

   temp = DAYR.reg;                                  // DayFlag reset
        }
} // NewDayRefresh()
//---------------------------------------------------------------------------
```

A.4 [Alarm.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEEE EEEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE         EEE           CC         CC         //
//     AAAAAA        TTTT     EEEEE       EEEEE         CC         CC         //
//    AAAAAAAA       TTTT     EEEEE       EEEEE         CC         CC         //
//   AAAA  AAAA      TTTT     EEE         EEE           CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEEE EEEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                                 //
// Unit 32, Consett Business Park                                             //
// Villa Real, Consett                                                        //
// Co. Durham                                                                 //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                    //
//  Filename  :  Alarm.h                                                      //
//  Author    :  T. Mudryk                                                    //
//  Compiler  :  CodeWarrior v5.2                                             //
//  CPU       :  MC68HC908LJ12                                                //
////////////////////  File Contents  ////////////////////////////////////////
//  Header file for 'Alarm.c'                                                 //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ------------------------------------------------------//
//  001  27/06/03 tm      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef    __Alarm_H_
#define     __Alarm_H_

void NewDayRefresh( void );
void OrderCheck( const struct sPROG_PARAMS*, unsigned char );
void AlarmFlag( void );
void Alarm( void );

#endif
```

A.5 [Alarmset.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  Alarmset.c                                                 //
//  Author    :  T. Mudryk                                                 //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Alarm set routines                          //
//////////////////////////  Update Information  ////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  25/06/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#include   "extern.h"
#endif

#include   "Display.h"
#include   "button.h"
#include   "Delay.h"
#include    "Alarm.h"
#include   "Alarmset.h"

void ModeAlarmSet( void )
{
switch ( Aset_mode )                            // Branch to the current setting
        {
        case PROGRAM_MODE:
        Program();
        break;

        case WEEK_MODE:
        Week();
        break;
```

```
        case WEEKEND_MODE:
        Weekend();
        break;

        case DAILY_MODE:
        case DAILY_MODE_REVERSE:
        Daily();
        break;

        case HOUR_MODE:
        AlarmHset();
        break;

        case MINUTE_MODE:
        AlarmMset();
        break;

        case TEMPERATURE_MODE:
        AlarmTset();
        break;

        case PROGRAM_CHECK_MODE:
        ProCheck();
   set_mode      = MODE_DEFAULT;
   Aset_mode     = DEFAULT;
        AlarmPRNumber = NO_PROGRAM;
        if ( !PR2 )     Alarm ();               // Refresh alarm register
        Refresh       = 1;                      // Refresh temperature
        break;
        }
} // ModeAlarmSet()
//-----------------------------------------------------------------------------


void Program( void )
{
ClearDisplay();
LDAT12.byte = 0x33;                             // Displays "AlarmPRNumber"
LDAT11.byte = 0x12;                             //
LDAT10.byte = 0x20;                             //
if ( _0P25sec )                                 // For every other 1/4 of a second
   {
        LDAT9.byte  = 0x00;                     // Allows Program Number to Flash
        LDAT8.byte &= 0x0f;                     //
   }
else    DisplayNumber( AlarmPRNumber + 1, 6 );  // Displays Program Number
} // Program()
//-----------------------------------------------------------------------------


void Week( void )
{
if ( _0P25sec )
   {
```

```
        LDAT12.bit.bit6 = 1;                    // Chevrons are Added, Mon - Fri
        LDAT11.bit.bit2 = 1;                    //
        LDAT9.bit.bit6  = 1;                    //
        LDAT8.bit.bit2  = 1;                    //
        LDAT6.bit.bit6  = 1;                    //
   }
else
   {
        LDAT12.bit.bit6 = 0;                    // Chevrons are Removed, Mon - Fri
        LDAT11.bit.bit2 = 0;                    //
        LDAT9.bit.bit6  = 0;                    //
        LDAT8.bit.bit2  = 0;                    //
        LDAT6.bit.bit6  = 0;                    //
   }
} // end of week
//-----------------------------------------------------------------------------


void Weekend( void )
{
if ( _0P25sec )                                 // For every other 1/4 of a second
   {
        LDAT5.bit.bit2  = 1;                    // Add Chevrons to Sat - Sun
        LDAT3.bit.bit6  = 1;                    //
   }
else
   {
        LDAT5.bit.bit2  = 0;                    // Allows chevrons to flash
        LDAT3.bit.bit6  = 0;                    //
   }
} // end of weekend
//-----------------------------------------------------------------------------


void Daily( void )
{
volatile union uPORT*ptr = &LDAT1;


if ( _0P25sec )
   {
        if ( dow_set == MON || dow_set == WED || dow_set == FRI || dow_set == SUN )
           {                                    // Determins Even Digit
                ptr[dow_set].bit.bit6    = 1;   // Adds a Chevron
           }
        else    ptr[dow_set].bit.bit2 = 1;      // Adds a Chevron
   }
else
   {
        LDAT12.bit.bit6 = 0;                    // Clears all chevrons
        LDAT11.bit.bit2 = 0;                    //
        LDAT9.bit.bit6  = 0;                    //
        LDAT8.bit.bit2  = 0;                    //
        LDAT6.bit.bit6  = 0;                    //
        LDAT5.bit.bit2  = 0;                    //
```

```
        LDAT3.bit.bit6  = 0;                    //
   }
} // end of Daily
//---------------------------------------------------------------------------


void AlarmHset( void )
{
MinAlm();                                       // Displays minute digits
if ( _0P25sec )                                 // For every other 1/4 of a second
   {
   LDAT6.byte = 0x00;                           // Clears the other digits
        LDAT5.byte = 0x00;                      //
        LDAT4.byte = 0x00;                      //
   }
else    HrAlm();                                // Display Hour

ChevronRefresh();
} // end of AlarmHset
//---------------------------------------------------------------------------


void AlarmMset( void )
{
HrAlm();                                        // Displays hour digits
if ( _0P25sec )                                 // For every other 1/4 of a second
   {
   LDAT3.byte = 0x00;                           // Clears the other digits
        LDAT2.byte = 0x00;                      //
        LDAT1.byte = 0x00;                      //
   }
else    MinAlm();                               // Display Minutes

ChevronRefresh();
} // end of AlarmMset
//---------------------------------------------------------------------------

void AlarmTset( void )
{
if ( _0P25sec )                                 // For every other 1/4 of a second
   {
        LDAT2.byte &= 0x0f;                     //
        LDAT3.byte  = 0x00;                     // Temperature Digits are Cleared
        LDAT4.byte  = 0x00;                     //
        LDAT5.byte &= 0xf0;                     //
   }
else    TempAlm();                              // Display Temperature

ChevronRefresh();
} // end of AlarmTset
//---------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// This function checks to see if any of the programs overlap each other. If //
```

```
// they do, an error message is displayed with the numbers of the programs   //
// that clash. The latest program is then cancelled/ignored.                 //
/////////////////////////////////////////////////////////////////////////////
void ProCheck( void )
{
unsigned short int   temp;
unsigned short int   stored_total_on;
unsigned short int   stored_total_off;
unsigned short int   new_total_on;
unsigned short int   new_total_off;
unsigned char        error = NO_PROGRAM;
unsigned char        stored_program;


new_total_on  = ( 60 * program[AlarmPRNumber].hour_on  )
                + program[AlarmPRNumber].min_on;
new_total_off = ( 60 * program[AlarmPRNumber].hour_off )
                + program[AlarmPRNumber].min_off;

if ( new_total_on != new_total_off )
   {
   for ( stored_program = 0; stored_program <= 3; stored_program++ )
      {
      if ( AlarmPRNumber != stored_program )
         {
         if ( program[stored_program].dow == 8 )// Mon - Fri
            {
            if (  program[AlarmPRNumber].dow == 0 ||     // Sun
                  program[AlarmPRNumber].dow == 6 ||     // Sat
                  program[AlarmPRNumber].dow == 9  )     // Sat - Sun
               {
               continue;                          // If dow are not equal, skip...
               }                                  // the check
            }

         if ( program[stored_program].dow == 9 ) // Sat - Sun
            {
            if (( program[AlarmPRNumber].dow  > 0   &&   // DOW > Sunday and ...
                  program[AlarmPRNumber].dow  < 6 ) ||   // DOW < Saturday or...
                  program[AlarmPRNumber].dow == 8 )      // DOW = Mon - Fri
               {
               continue;                          // If dow are not equal, skip...
               }                                  // the check
            }

         if ( program[AlarmPRNumber].dow == 8 ) // Mon - Fri
            {
            if (  program[stored_program].dow == 0 ||    // Sun
                  program[stored_program].dow == 6  )    // Sat
               {
               continue;                         // If dow are not equal, skip...
               }                                 // the check
            }

         if ( program[AlarmPRNumber].dow == 9 ) // Sat - Sun
```

```
            {
            if ( program[stored_program].dow  > 0 &&// DOW > Sunday and ...
                 program[stored_program].dow  < 6  )// DOW < Saturday or...
               {
               continue;                          // If dow are not equal, skip...
               }                                  // the check
            }

          if ( program[AlarmPRNumber].dow  < 8 &&
               program[stored_program].dow < 8  )
            {
            if ( program[AlarmPRNumber].dow != program[stored_program].dow )
               {
                  continue;                       // If dow are not equal, skip...
               }                                  // the check
            }

          stored_total_on  = ( 60 * program[stored_program].hour_on  )
                                                   + program[stored_program].min_on;
          stored_total_off = ( 60 * program[stored_program].hour_off )
                                                   + program[stored_program].min_off;
          if ( stored_total_on != stored_total_off )
            {
            if (( stored_total_on > new_total_on   &&
                  stored_total_on  < new_total_off ) ||
         // If the new program starts before the stored but the off time overlaps
                ( stored_total_on < new_total_on )   &&
                ( stored_total_off > new_total_on )  ))
         // If the new program starts after the stored but the on time overlaps
                {
                error = stored_program;
                continue;
                }
            }
          }
       } // end of 'for'
    }

if ( error != NO_PROGRAM )                         // If an error is present
    {
    ClearDisplay();
    LDAT12.byte    = 0x37;                         // Displays "Err"
         LDAT11.byte    = 0x02;                    //
         LDAT10.byte    = 0x20;                    //
         LDAT9.byte     = 0x22;                    //
         LDAT4.bit.bit5 = 1;                       // Displays "-"

         DisplayNumber( AlarmPRNumber + 1, 4 );    // The program that was entered
         DisplayNumber( error + 1   , 2 );         // The conflicting program

    for ( temp = 0; temp < 1000; temp++ )
       {
       Delay( _1MS );                              // 1sec delay to show error message
       ServiceWatchDog();
       }
```

```
   program[AlarmPRNumber].hour_off = program[AlarmPRNumber].hour_on;
   program[AlarmPRNumber].min_off  = program[AlarmPRNumber].min_on;
   }                                                  // Cancels the program
ClearDisplay();
} // ProCheck()
//-------------------------------------------------------------------------


void HrAlm ( void )                                   // Hour Alarm Refresh
{
DisplayNumber(( unsigned char )( Hour%10 ), 3 );
                                                      // Second Hour Digit
if ( Hour >= 10 )                                     // If the First Hour Digit != 0
   {
        DisplayNumber(( unsigned char )( Hour/10 ), 4 );
   }                                                  // Displays the First Hour Digit
} // end of HRAlm()
//-------------------------------------------------------------------------


void MinAlm ( void )                                  // Min Alarm refresh
{
DP = 0;
DisplayNumber(( unsigned char )( Minute%10 ), 1 );
                                                      // Second Minute Digit
DisplayNumber(( unsigned char )( Minute/10 ), 2 );
                                                      // First Minute Digit
} // end of MINAlm
//-------------------------------------------------------------------------


void TempAlm ( void )
{
DisplayNumber(( unsigned char )( user_temperature%10 ), 2 );
                                                      // Second Temperature Digit
if ( user_temperature/10 )                            // First Temperature Digit != 0
   {
        DisplayNumber(( unsigned char )( user_temperature/10 ), 3 );
   }                                                  // First Temperature Digit
                                                      // Resets the Chevron Variable
} // end of TEMPAlm
//-------------------------------------------------------------------------
```

A.6 [Alarmset.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  Alarmset.h                                                 //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Header file for 'Alarmset.c'                                            //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  25/06/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef    __Alarmset_H_
#define     __Alarmset_H_

enum     { MON_FRI = 0x00, SAT_SUN, SUN, SAT = 0x04, FRI,
           THU     = 0x07, WED,     TUE = 0x0a,      MON };


void ModeAlarmSet( void );
void AlarmTset( void );
void AlarmMset( void );
void AlarmHset( void );
void ProCheck( void );
void Weekend( void );
void Program( void );
void TempAlm( void );
void MinAlm( void );
void Daily( void );
void HrAlm( void );
void Week( void );

#endif
```

A.7 [button.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEEE EEEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE         EEE           CC         CC         //
//     AAAAAA        TTTT     EEEEE       EEEEE         CC         CC         //
//    AAAAAAAA       TTTT     EEEEE       EEEEE         CC         CC         //
//   AAAA  AAAA      TTTT     EEE         EEE           CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEEE EEEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                                 //
// Unit 32, Consett Business Park                                             //
// Villa Real, Consett                                                        //
// Co. Durham                                                                 //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                    //
//  Filename  :  button.c                                                     //
//  Author    :  T. Mudryk                                                    //
//  Compiler  :  CodeWarrior v5.2                                             //
//  CPU       :  MC68HC908LJ12                                                //
////////////////////  File Contents  ////////////////////////////////////////
//  Button routine                                                          //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  11/07/03 tm      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#include   "extern.h"
#endif

#include    "Alarmset.h"
#include    "Clockset.h"
#include    "Display.h"
#include    "Alarm.h"
#include    "button.h"

////////////////////////////////////////////////////////////////////////////
// Simple 8 bit (byte) button read                                            //
//                                                                            //
// Arguments: none                                                            //
// returns  : none                                                            //
////////////////////////////////////////////////////////////////////////////


void ReadButtons( void )
{
button_pattern = ( PTD.reg & 0xf0 ) + ( PTA.reg & 0x0f );
```

```
////////////////////////////////////////////////
// any change to the current button press status //
////////////////////////////////////////////////
switch ( button_press_status )
        {
        case NO_BUTTON_PRESS:
        DefaultButtons();
        break;

        case BUTTON_PRESSED :
        PressedButtons();
        break;

        case BUTTON_RELEASED:
        ReleasedButtons();
        break;
        }
}  // ReadButtons()
//---------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// This function is called from 'ReadButtons()' which is itself called every  //
// 'main()' 10ms iteration.                                                   //
//                                                                            //
// This function is called if the value of 'button_press_status' is           //
// NO_BUTTON_PRESS. If a press is detected by the variable 'button_pattern'   //
// not being equal to DEFAULT_BUTTONS then 'button_press_status' is assigned  //
// to BUTTON_PRESSED. On the next main loop iteration then 'PressedButtons()' //
// will be called from 'ReadButtons()'.                                       //
//                                                                            //
// Arguments : none                                                           //
// Returns   : none                                                           //
////////////////////////////////////////////////////////////////////////////
void DefaultButtons( void )
{
if ( button_pattern == DEFAULT_BUTTONS )
        {
   // re-affirmation
   pressed_pattern              = DEFAULT_BUTTONS;
   button_debounce_counter      = 0;
   button_flags.bit.FIRST_PASS  = 0;
   button_flags.bit.AUTO_REPEAT = 0;
        }
else
        {
   ////////////////////////////////////////////////////////////
   // OK, a press detected, this is the first recognition of... //
   ////////////////////////////////////////////////////////////
   button_press_status = BUTTON_PRESSED;         // state change
   pressed_pattern     = button_pattern;         // store for later comparisons
   }
}  // DefaultButtons()
//---------------------------------------------------------------------------
```

Software Routines

```
////////////////////////////////////////////////////////////////////////////
// To get to this point some button activity has occured. If the button       //
// activity has been stable for DEBOUNCE_COUNTER*10ms then the button press   //
// is accepted and function decoding is performed. If a button is pressed     //
// and kept pressed further decoding will take place if the autoscroll        //
// feature has been enabled for that button. This feature is enabled by the   //
// setting of 'button_flags.bit.AUTO_REPEAT'.                                 //
//                                                                            //
// Arguments : none                                                           //
// Returns   : none                                                           //
////////////////////////////////////////////////////////////////////////////
void PressedButtons( void )
{
////////////////////////////////////////////
// No longer pressed, back to pull-up values //
////////////////////////////////////////////
if ( button_pattern == DEFAULT_BUTTONS )
   {
   button_press_status    = BUTTON_RELEASED;
   button_release_counter = 2;                  // initialise 20ms release debounce counter
   }
else
        {
   ////////////////////////////////////////////////
   // is the button pattern unchanged from last read //
   ////////////////////////////////////////////////
   if ( button_pattern == pressed_pattern )
        {
        // 10ms 'main()' synchronised 'button_debounce_counter' increments here
      if ( ++button_debounce_counter >= DEBOUNCE_COUNTER )
                        {
         if ( !button_flags.bit.FIRST_PASS )     // is this the first debounce...
            {                                    // ...of this pattern
            button_flags.bit.FIRST_PASS  = 1;    // ...yes signal this event
            button_flags.bit.AUTO_REPEAT = 0;    // reset autoscroll
            DecodeButtons();                     // respond to press
            }
         else                                    // auto repeat can now occur
                              {                  // if required
            ////////////////////////////////////////////////////////////
            // same button as for first debounce is still being pressed, //
            // after (50-DEBOUNCE_COUNTER)*10ms allow auto repeat.       //
            ////////////////////////////////////////////////////////////
            if ( button_debounce_counter >= 50 ) // (50-DEBOUNCE_COUNTER)*10ms
               {                                 // before auto repeat mode
               button_debounce_counter    = 10;  // (50-3)*10ms is the effective
                                                 // repeat speed == 470ms,
                                                 // approx 2 times per second
if (button_flags.bit.AUTO_REPEAT)// do you require auto scroll...
                                                 {
                  DecodeButtons();               // ...if so keep decoding
                                                 }
               }
```

```
            }
         }
      }
   else
      {
      ////////////////////////////////////////////////////////////////
      // pattern is different but something is pressed, start again. //
      ////////////////////////////////////////////////////////////////
      button_press_status          = NO_BUTTON_PRESS;
      pressed_pattern              = DEFAULT_BUTTONS;
      button_debounce_counter      = 0;
      button_flags.bit.FIRST_PASS  = 0;
      button_flags.bit.AUTO_REPEAT = 0;
      }
   }    // end of 'else'
}     // PressedButtons()
//-----------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// Ok, we think all the buttons are now at their default, initiate a release //
//                                                                          //
// Arguments : none                                                         //
// Returns   : none                                                         //
////////////////////////////////////////////////////////////////////////////
void ReleasedButtons( void )
{
if ( --button_release_counter == 0 )
        {
   if ( set_mode == MODE_VIEWTEMP)
      {
      set_mode = MODE_DEFAULT;
      Refresh  = 1;
      ClearDisplay();
      }
   button_press_status = NO_BUTTON_PRESS;
        }
else
   {
   // checking for noise...
   if ( button_pattern   != DEFAULT_BUTTONS )
      {
      button_press_status = BUTTON_PRESSED;      // continue as pressed...
      }
   }
}  // ReleasedButtons()
//-----------------------------------------------------------------------------


////////////////////////////////////////////////
// This function does the physical button decoding. //
//                                              //
// Arguments: none                              //
// returns  : none                              //
////////////////////////////////////////////////
```

Software Routines

```
void DecodeButtons( void )
{
unsigned char temp;


switch ( pressed_pattern )
   {
   case BUTTON_1 :                              // Clock Set Button
   if ( set_mode == DEFAULT || set_mode == MODE_CLOCKSET )
      {
      set_mode = MODE_CLOCKSET;
      if ( ++Cset_mode > DOW_SET )
         {
         set_mode  = MODE_DEFAULT;
         Cset_mode = DEFAULT_SET;
         Refresh   = 1;                         // Refresh temp and smart chevron
         }
      else if ( Cset_mode == DOW_SET)
         {
         MINClr();                              // Refresh Min
         }
      }
   SECR.reg = 0x00;                             // Resets the second register
   break;

   case BUTTON_2 :                              // Dec Button
   Decrement();
   if ( Cset_mode == DOW_SET )button_flags.bit.AUTO_REPEAT = 0;
   else                       button_flags.bit.AUTO_REPEAT = 1;
   break;

   case BUTTON_3 :                              // Inc Button
   Increment();
   if ( Cset_mode == DOW_SET ) button_flags.bit.AUTO_REPEAT = 0;
   else                       button_flags.bit.AUTO_REPEAT = 1;
   break;

   case BUTTON_4 :                              // Alarm Set
   if ( set_mode == DEFAULT || set_mode == MODE_ALARMSET )
      {
      set_mode = MODE_ALARMSET;
      switch ( Aset_mode )
         {
         case DEFAULT:
         AlarmPRNumber = PROGRAM_1;
         Aset_mode     = PROGRAM_MODE;
         break;

         case PROGRAM_MODE:
         onoff = 0;                             // Setting on/off
         Aset_mode = WEEK_MODE;
         DisplayNumber( AlarmPRNumber + 1, 6 );
         break;

         case WEEK_MODE:
```

```
Aset_mode = HOUR_MODE;
DW        = MON_FRI;
Hour      = program[AlarmPRNumber].hour_on;
Minute    = program[AlarmPRNumber].min_on;
                                        // Hour set to previous hour

LDAT11.byte &= 0xf0;                    // Clears the 7th Digit
LDAT10.byte  = 0x00;                    //
LDAT9.byte   = 0x35;                    // Displays "On"
LDAT8.byte   = 0x32;                    //
LDAT7.byte   = 0x22;                    //

DisplayNumber( AlarmPRNumber + 1, 8 );
break;

case WEEKEND_MODE:
DW = SAT_SUN;
Aset_mode = HOUR_MODE;
Hour      = program[AlarmPRNumber].hour_on;
Minute    = program[AlarmPRNumber].min_on;
                                        // Hour set to previous hour

LDAT11.byte &= 0xf0;                    // Clears the 7th Digit
LDAT10.byte  = 0x00;                    //
LDAT9.byte   = 0x35;                    // Displays "On"
LDAT8.byte   = 0x32;                    //
LDAT7.byte   = 0x22;                    //

DisplayNumber( AlarmPRNumber + 1, 8 );
break;

case DAILY_MODE:
case DAILY_MODE_REVERSE:
DW = dow_set;
Aset_mode = HOUR_MODE;
Hour      = program[AlarmPRNumber].hour_on;
Minute    = program[AlarmPRNumber].min_on;
                                        // Hour set to previous hour

LDAT11.byte &= 0xf0;                    // Clears the 7th Digit
LDAT10.byte  = 0x00;                    //
LDAT9.byte   = 0x35;                    // Displays "On"
LDAT8.byte   = 0x32;                    //
LDAT7.byte   = 0x22;                    //

DisplayNumber( AlarmPRNumber + 1, 8 );
break;

case HOUR_MODE:
if ( onoff )                            // If This is the Second Time This
   {                                    // Function has Been Ran
   program[AlarmPRNumber].hour_off = Hour;   // Stores Off Hour
   Aset_mode = MINUTE_MODE;
   }
else                                    // If This is the First Time This
```

Software Routines

```
   {                                    // Function has Been Ran
   program[AlarmPRNumber].hour_on  = Hour;   // Stores On hour
   Aset_mode = MINUTE_MODE;
   Minute = program[AlarmPRNumber].min_on;
                                        // Default Minute set to previous

   switch ( DW )                        // Stores DOW as different values
      {                                 // Mon-Fri = 0x08, Sat-Sun = 0x09
      case MON    : temp = 0x01;break;  // Sun = 0x00, Mon = 0x01, etc.
      case TUE    : temp = 0x02;break;
      case WED    : temp = 0x03;break;
      case THU    : temp = 0x04;break;
      case FRI    : temp = 0x05; break;
      case SAT    : temp = 0x06;break;
      case SUN    : temp = 0x00;break;
      case SAT_SUN: temp = 0x09; break;
      case MON_FRI: temp = 0x08;break;
      default     : temp = DOWR.byte;
      }
   program[AlarmPRNumber].dow = temp;
   }
break;

case MINUTE_MODE:
if ( onoff )                            // If this is the second time ...
   {                                    // This function has been ran
   program[AlarmPRNumber].min_off = Minute;
                                        // Stores Off Minute
   Aset_mode        = TEMPERATURE_MODE;
   user_temperature = 20;               // Default temperature is set
   ClearDisplay();
   LDAT11.byte      = 0x03;      // "t" is displayed
   LDAT10.byte      = 0x60;      //
   LDAT1.byte       = 0x60;      // "c" is displayed
   LDAT2.byte       = 0x02;     //
   DisplayNumber( AlarmPRNumber + 1, 8 );
   ChevronRefresh();
   }
else                                    // If this is the first time ...
   {                                    // This function has been ran
   program[AlarmPRNumber].min_on = Minute;
                                        // Stores on minute
   onoff = 1;
   // Sets Variable to Indicate on Times Have Been set
   Aset_mode    = HOUR_MODE;
   LDAT11.byte &= 0xf0;                 //
   LDAT11.byte += 0x03;                 // Displays "Off"
   LDAT10.byte  = 0x53;                 //
   LDAT9.byte   = 0x33;      //
   LDAT8.byte   = 0x03;      //
   LDAT7.byte   = 0x30;      //
   ChevronRefresh();
   }
break;
```

```
    case TEMPERATURE_MODE:
    program[AlarmPRNumber].temperature = user_temperature;
                                          // Stores Temperature
    Aset_mode = PROGRAM_CHECK_MODE;
    break;
    }
}
    break;

    case BUTTON_5 :                       // Temperature/Clear Button
    if ( set_mode == MODE_ALARMSET )
       {
       program[AlarmPRNumber].hour_off = program[AlarmPRNumber].hour_on;
       program[AlarmPRNumber].min_off  = program[AlarmPRNumber].min_on;
       set_mode  = MODE_DEFAULT;
       Aset_mode = DEFAULT;
       AlarmPRNumber  = NO_PROGRAM;
       Refresh   = 1;                     // Refresh temperature
       }
    else if  ( set_mode == DEFAULT )
       {
       set_mode = MODE_VIEWTEMP;
       }
    break;

    case BUTTON_5_DEC :
    // Assigned as decrement for when button_5 being held down
    Decrement();
    button_flags.bit.AUTO_REPEAT = 1;
    break;

    case BUTTON_5_INC :
    // Assigned as increment for when button_5 being held down
    Increment();
    button_flags.bit.AUTO_REPEAT = 1;
    break;

    case BUTTON_6 :                       // Override Button
    if ( set_mode == DEFAULT )
       {
       if ( PR2 == 1 || Override == 1 )   // If a program or override is ...
       {                                  // currently running
       Override     = 0;                  // Reset override
           Active       = 0;              // Turn heating/cooling off
           PR2          = 0;              // Reset program variable
           PTB.bit.bit0 = 0x00;           //
           PTB.bit.bit1 = 0x00;           //
           Alarm();                       // Refresh Alarm Register
  }
  else
           {
           if ( HRR.byte >= 23 )   OverrideH = 0;
                                          // Maximum cap for hours
           else                 OverrideH = HRR.byte + 1;
  // Increment to allow override to be on for one hour
```

```
          OverrideM = MINR.byte;               // Store current minutes
               Override  = 1;                  // Activate override
               Active    = 1;                  // Activate heating/cooling
               }
      }
   break;

        case BUTTON_7 :                        // Smart Button
        if ( set_mode == DEFAULT )
           {
      smart  ^= 0x01;
      Refresh = 1;                             // Refresh temp and smart chevron
      }
        break;

        case BUTTON_8 :                        // Mode Button
        if ( set_mode == DEFAULT )
           {
        if ( mode++ == OFF )   mode = AUTOMATIC;
      }
   break;
        }
}  // DecodeButtons()
//-----------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// The <INC> button has been pressed, decide functionality wrt current mode   //
//                                                                            //
// Arguments: none                                                            //
// returns  : none                                                            //
////////////////////////////////////////////////////////////////////////////
void Increment( void )
{
volatile union uPORT*ptr = &LDAT1;


switch ( set_mode )
   {
   case MODE_CLOCKSET:
   switch ( Cset_mode )
      {
      case HOUR_SET:
      if ( HRR.byte++  >= 23)    HRR.byte  = 0;
      break;

      case MINUTE_SET:
      if ( MINR.byte++ >= 59)    MINR.byte = 0;
      break;

      case DOW_SET:
      if ( DOWR.byte++ >= 0x06 ) DOWR.byte = 0;
      break;
      }
   break;
```

```
case MODE_ALARMSET:
switch ( Aset_mode )
   {
   case PROGRAM_MODE:
     if ( AlarmPRNumber < PROGRAM_4 )   AlarmPRNumber++;
     // Resets to Minimum if Maximum Reached
   break;

   case WEEK_MODE:
   Aset_mode  = WEEKEND_MODE;
   LDAT12.bit.bit6 = 0;                        // Chevrons are Removed, Mon - Fri
        LDAT11.bit.bit2 = 0;                   //
        LDAT9.bit.bit6  = 0;                   //
        LDAT8.bit.bit2  = 0;                   //
        LDAT6.bit.bit6  = 0;                   //
   break;

   case WEEKEND_MODE:
        Aset_mode  = DAILY_MODE;
        dow_set         = MON;                 // Set starting chevron to Mon
        LDAT5.bit.bit2  = 0;                   // Clears Sat - Sun to allow flash
        LDAT3.bit.bit6  = 0;                   //
   break;

   case DAILY_MODE:
   case DAILY_MODE_REVERSE:
     if ( dow_set == MON || dow_set == WED || dow_set == FRI || dow_set == SUN )
        {                                      // Determins Even Digit
             ptr[dow_set].bit.bit6 = 0;        // Adds a Chevron
        }
     else                                      // Determins Odd Digit
        {
             ptr[dow_set].bit.bit2 = 0;        // Adds a Chevron
        }
   dow_set -= 1;                               // DOW is Incrmented
     if ( dow_set == 9 || dow_set == 6 || dow_set == 3 )
           {
                dow_set -= 1;                  // Pointer is Incremented further..
           }                                   // due to 1/3 bytes not being used
     if ( dow_set == 1 )
           {
             Aset_mode = WEEK_MODE;
           }
   break;

   case HOUR_MODE:
     if ( Hour++ == 23 ) Hour = 0;             // Resets to Minimum
   break;

   case MINUTE_MODE:
     if ( Minute++ == 59 )Minute = 0;// Resets to Minimum
   break;

   case TEMPERATURE_MODE:
```

```
      if ( user_temperature++ == 40 )user_temperature = 20;
      break;                                    // Resets to Default

      case PROGRAM_CHECK_MODE:
      break;
      }
   break;

   case MODE_VIEWTEMP:
        if (++user_temperature > 40)      user_temperature = 20;
   break;                                    // Maximum cap for temp
   }

}  // Increment()
//---------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// The <DEC> button has been pressed, decide functionality wrt current mode   //
//                                                                            //
// Arguments: none                                                            //
// returns  : none                                                            //
////////////////////////////////////////////////////////////////////////////
void Decrement( void )
{
volatile union uPORT*ptr = &LDAT1;


switch ( set_mode )
   {
   case MODE_CLOCKSET:
   switch ( Cset_mode )
      {
      case HOUR_SET:
      if ( HRR.byte--  == 0x00 ) HRR.byte = 23;
      break;

      case MINUTE_SET:
           if ( MINR.byte-- == 0x00 ) MINR.byte = 59;
      break;

      case DOW_SET:
                if ( DOWR.byte-- == 0x00 ) DOWR.byte = 0x06;
                break;
      }
   break;

   case MODE_ALARMSET:
   switch ( Aset_mode )
      {
      case PROGRAM_MODE:
        if ( AlarmPRNumber )            AlarmPRNumber--;// Minimum cap
      break;

      case WEEK_MODE:
```

```
      Aset_mode         = DAILY_MODE_REVERSE;
      dow_set                 = SUN;          // Set starting chevron to Sun
    LDAT12.bit.bit6 = 0;                      // Chevrons are Removed, Mon - Fri
         LDAT11.bit.bit2 = 0;                 //
         LDAT9.bit.bit6  = 0;                 //
         LDAT8.bit.bit2  = 0;                 //
         LDAT6.bit.bit6  = 0;                 //
    break;

    case WEEKEND_MODE:
    Aset_mode  = WEEK_MODE;
    LDAT5.bit.bit2  = 0;                      // Clears Sat - Sun
         LDAT3.bit.bit6  = 0;                 // Chevrons to Flash
    break;

    case DAILY_MODE:
    case DAILY_MODE_REVERSE:
      if ( dow_set == MON || dow_set == WED || dow_set == FRI || dow_set == SUN )
         {                                    // Determins Even Digit
              ptr[dow_set].bit.bit6 = 0;      // Adds a Chevron
         }
      else                                    // Determins Odd Digit
         {
              ptr[dow_set].bit.bit2 = 0;      // Adds a Chevron
         }
    dow_set++;                                // DOW is Decremented
    if ( dow_set == 9 || dow_set == 6 || dow_set == 3 )
       {
      dow_set++;                              // Pointer is Decremented further
       }                                      // due to 1/3 bytes not being used
    if (dow_set == 12)
       {
                 Aset_mode = WEEKEND_MODE;    // Next function to be called is weekend
            }
    break;

    case HOUR_MODE:
    if ( Hour-- == 0 )   Hour = 23;           // Resets to Maximum
    break;

    case MINUTE_MODE:
      if ( Minute-- == 0 ) Minute = 59;       // Resets to Maximum
    break;

    case TEMPERATURE_MODE:
    if (user_temperature-- == 0)  user_temperature = 20;
    break;                                    // Resets to Default

    case PROGRAM_CHECK_MODE:
    break;
    }
  break;

  case MODE_VIEWTEMP:
       if ( !user_temperature-- )       user_temperature = 20;
```

```
   break;                                          // Resets to Default
   }

}  // Decrement()
//-----------------------------------------------------------------------------
```

A.8 [button.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  button.h                                                   //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Header file for 'button.c'                                              //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  19/06/00 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef    __BUTTON_H_
#define    __BUTTON_H_

////////////////////
// button decodes //
////////////////////
#define  DEFAULT_BUTTONS      0xff// 11111111
#define  BUTTON_1             0xfe// 11111110
#define  BUTTON_2             0xfd// 11111101
#define  BUTTON_3             0xfb// 11111011
#define  BUTTON_4             0xf7// 11110111
#define  BUTTON_5             0xef// 11101111
#define  BUTTON_6             0xdf// 11011111
#define  BUTTON_7             0xbf// 10111111
#define  BUTTON_8             0x7f// 01111111
#define  BUTTON_5_DEC         0xed  // 11101101
#define  BUTTON_5_INC         0xeb  // 11101011

#define DEBOUNCE_COUNTER      3

//////////////////////////
```

Software Routines

```
// button_flags defines //
//////////////////////////
#define FIRST_PASS            bit0
#define AUTO_REPEAT           bit1


enum { NO_CHANGE = 0x01, DECREMENT_VALUE, INCREMENT_VALUE };

//////////////////
// button states //
//////////////////
enum  { NO_BUTTON_PRESS = 0x01, BUTTON_PRESSED, BUTTON_RELEASED };

enum  {
        MODE_DEFAULT = 0x00,
        MODE_CLOCKSET,
        MODE_ALARMSET,
        MODE_VIEWTEMP
      };

///////////////
// prototypes //
///////////////
void ReleasedButtons( void );
void PressedButtons( void );
void DefaultButtons( void );
void DecodeButtons( void );
void ReadButtons( void );
void Increment( void );
void Decrement( void );

#endif
```

A.9 [Clockset.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  Clockset.c                                                 //
//  Author    :  T. Mudryk                                                 //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Clock Setting                                                          //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  13/06/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#include   "extern.h"
#endif

#include   "Display.h"
#include   "Clockset.h"


void Cset( void )
{
switch ( Cset_mode )
   {
   case HOUR_SET:
   if ( _0P25sec )                          // For every other 1/4 of a second
      {
      SFLASH = 1;                           // Displays divider decimal point
      HRClr();                              // Refreshes hour digits
      MINClr();                             // Refreshes minute digits
      ChevronRefresh();
      }
   else
```

```
   {
   ClearDisplay();                          // Allows hour digits to flash
   LDAT10.bit.bit2 = 1;                     // Decimal point remains constant
   MINClr();                                // Minute digits refreshed
   ChevronRefresh();
   }
break;

case MINUTE_SET:
if ( _0P25sec )                             // For every other 1/4 of a second
   {
   SFLASH = 1;                              // Displays divider decimal point
   MINClr();                                // Refreshes hour digits
   HRClr();                                 // Refreshes minute digits
   ChevronRefresh();
   }
else
   {
   ClearDisplay();                          // Allows hour digits to flash
   SFLASH = 1;                              // Decimal point remains constant
   HRClr();                                 // Hour digits refreshed
   ChevronRefresh();
   }
break;

case DOW_SET:
LDAT12.bit.bit6 = 0;                        //
LDAT11.bit.bit2 = 0;                        // Clear Chevrons to prevent ghost
LDAT9.bit.bit6  = 0;                        //
LDAT8.bit.bit2  = 0;                        //
LDAT6.bit.bit6  = 0;                        //
LDAT5.bit.bit2  = 0;                        //
LDAT3.bit.bit6  = 0;                        //

if ( _0P25sec )                             // For every other 1/4 of a second
   {
   switch (DOWR.byte)
        {
             case 0x00:
             LDAT3.bit.bit6  = 1;           // Set Sunday Chevron
             break;

             case 0x01:
      LDAT12.bit.bit6 = 1;                  // Set Monday Chevron
             break;

             case 0x02:
             LDAT11.bit.bit2 = 1;           // Set Tuesday Chevron
        break;

             case 0x03:
      LDAT9.bit.bit6  = 1;                  // Set Wednesday Chevron
             break;

             case 0x04:
```

```
                LDAT8.bit.bit2  = 1;             // Set Thursday Chevron
                break;

         case 0x05:
                LDAT6.bit.bit6  = 1;             // Set Friday Chevron
                break;

        case 0x06:
                LDAT5.bit.bit2  = 1;             // Set Saturday Chevron
                break;
        }
      }
   break;
   }
} // Cset()
//-------------------------------------------------------------------------


void HRClr (void)                                // Hour Clock Refresh
{
if ( HRR.byte >=10 )
        {
        DisplayNumber( (unsigned char)(HRR.byte/10), 8 );
        }
else
   {
   LDAT12.byte  = 0x00;
   LDAT11.byte &= 0x0f;
   }

DP = SFLASH;
DisplayNumber( (unsigned char)(HRR.byte%10), 7 );
DP = 0;
} // HRClr()
//-------------------------------------------------------------------------


void MINClr (void)// Min Clock refresh
{
DP = 0;
DisplayNumber( (unsigned char)(MINR.byte%10), 5 );
DisplayNumber( (unsigned char)(MINR.byte/10), 6 );
} // MINClr()
//-------------------------------------------------------------------------
```

A.10 [Clockset.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  Clockset.h                                                 //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Header file for 'Clockset.c'                                            //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  06/06/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef    __Clockset_H_
#define     __Clockset_H_


enum     { DEFAULT_SET = 0x00, HOUR_SET, MINUTE_SET, DOW_SET };


void MINClr (void);
void HRClr (void);
void Cset (void);

#endif
```

A.11 [Data.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                 //
//  Filename  :  Data.c                                                     //
//  Author    :  T. Mudryk                                                 //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  global data,                                                            //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DECLARED_H_
#include    "declared.h"
#endif


#pragma DATA_SEG __SHORT_SEG _DATA_ZEROPAGE

/////////////////////////////////////////////////////
// PAGE0 Global variables, range available 0x60 - 0xff //
/////////////////////////////////////////////////////
uBITS                  button_flags;
unsigned char                  N2;
unsigned char                  DW;
unsigned char                  DP;
unsigned char                  PR2;
unsigned char           sign;
unsigned char                  SMin;
unsigned char                  Hour;
unsigned char                  mode;
unsigned char                  smart;
unsigned char                  SHour;
```

```
unsigned char                   _10ms;
unsigned char           onoff;
unsigned char                   Minute;
unsigned char                   SFLASH;
unsigned char                   Active;
unsigned char                   dow_set;
unsigned char                   Refresh;
unsigned char                   PRNumber;
unsigned char                   low_byte;
unsigned char                   set_mode;
unsigned char           _0P25sec;
unsigned char                   Override;
unsigned char                   OverrideH;
unsigned char                   OverrideM;
unsigned char                   Cset_mode;
unsigned char                   Aset_mode;
unsigned char           _5sectick;
unsigned char           one_second;
unsigned char           two_second;
unsigned char           AlarmPRNumber;
unsigned char                   button_pattern;
unsigned char                   pressed_pattern;
unsigned char                   user_temperature;
unsigned char           button_press_status;
unsigned char                   external_temperature;
unsigned char           button_release_counter;
unsigned char           button_debounce_counter;
unsigned char                   external_temperature_decimal;
volatile uBITS                  flags1;
unsigned short intsmart_time;
unsigned short int      stop_counter;
unsigned short intBatteryCheck;
unsigned short int internal_temperature;
struct   sPROG_PARAMS   program[4];


////////////////////////////////////////////////////////
// Global non PAGE0 variables occupying 0x100 to stack top //
////////////////////////////////////////////////////////
#pragma DATA_SEG _FAR_RAM

//////////////////////////
// 'const' data ie rom //
//////////////////////////
#pragma CONST_SEG ATEECC_CONST_DATA
```

A.12 [declared.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  declared.h                                                 //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  declared data types                                                     //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DECLARED_H_
#define     __DECLARED_H_


//////////////////
// type defines //
//////////////////
typedef union uPORT         uBITS;
typedef  unsigned short intUSHORT;
typedef  unsigned char   UCHAR;


#pragma MESSAGE DISABLE C1106 // WARNING C1106: Non-standard bitfield type


//////////////////////
// bit/byte access //
//////////////////////
struct sPORT
{
UCHAR bit0 : 1;
```

Software Routines

```
UCHAR bit1 : 1;
UCHAR bit2 : 1;
UCHAR bit3 : 1;
UCHAR bit4 : 1;
UCHAR bit5 : 1;
UCHAR bit6 : 1;
UCHAR bit7 : 1;
};
union uPORT
{
UCHAR                       byte;
UCHAR                       reg;
struct sPORT   bit;
};

//////////////////////
// 16 bit data type //
//////////////////////
struct sUNSIGNED_INTEGER
{
UCHAR   hibyte;                                 // 0x12XX
UCHAR   lobyte;                                 // 0xXX34
};
union uUNSIGNED_INTEGER
{
USHORT                                           _16bit;
struct sUNSIGNED_INTEGER _8bit;
};

////////////////////////////
// 16 bit 'bit' data type //
////////////////////////////
struct sUNSIGNED_INTEGER_BIT
{
union uPORT    hibyte;                          // 0x12XX
union uPORT    lobyte;                          // 0xXX34
};
union uUNSIGNED_INTEGER_BIT
{
USHORT                                            _16bit;
struct sUNSIGNED_INTEGER_BIT _8bit;
};

struct sPROG_PARAMS
{
UCHAR   hour_on;
UCHAR   min_on;
UCHAR   dow;
UCHAR   hour_off;
UCHAR   min_off;
UCHAR   temperature;
};

#endif
```

A.13 [define.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  define.h                                                   //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  global defines                                                          //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DEFINE_H_
#define     __DEFINE_H_


/////////////////
// I/O defines //
/////////////////
#define  TIMING_PIN           PTD.bit.bit5       // for debugging
#define  TIMING_PIN_DDR       DDRD.bit.bit5      // for debugging

//////////////////////////////
// general numerical defines //
//////////////////////////////
#define _1MS                  165                // from delay.c, 15+(12*165)== 1995 bus
                                                 // cycles == 2000*(1/2.0E6) = 0.9975ms
#define _50US                 7                  // from delay.c, 15+(12*7)== 99 bus cycles
                                                 // == 99*(1/2.0E6) = 49.5us
#define _100US                16                 // from delay.c, 15+(12*16)==207 bus cycles
                                                 // == 207*(1/2.0E6) = 103.5us
#define LOWBATT               576                // 2.7V is the low battery condition, so...
                                                 // (2.7/4.8)*1024 == 576.0 counts
```

Software Routines

```
#define EXTERNAL_CALIBRATION  -1                // Tolerance of DS1721 is +- 1 degree

enum     { PROGRAM_1 = 0x00, PROGRAM_2, PROGRAM_3, PROGRAM_4, NO_PROGRAM };
                                                // 0 - 3 are valid programs
enum     { DEFAULT   = 0x00  ,
           PROGRAM_MODE      ,
           WEEK_MODE         ,
           WEEKEND_MODE      ,
           DAILY_MODE        ,
           DAILY_MODE_REVERSE,
           HOUR_MODE         ,
           MINUTE_MODE       ,
           TEMPERATURE_MODE  ,
           PROGRAM_CHECK_MODE
         };

enum     { AUTOMATIC = 0x00, HEAT, COOL, OFF };

////////////////////
// flags1 defines //
////////////////////
#define _10MS                   bit0
#define _1S                     bit1
#define TO_BE_ASSIGNED_2        bit2            // this is free for use
#define TO_BE_ASSIGNED_3        bit3            // this is free for use
#define TO_BE_ASSIGNED_4        bit4            // this is free for use
#define TO_BE_ASSIGNED_5        bit5            // this is free for use
#define TO_BE_ASSIGNED_6        bit6            // this is free for use
#define TO_BE_ASSIGNED_7        bit7            // this is free for use

////////////////////
// Assembler 'C' //
////////////////////
#define  RSP()                asm rsp
#define  SEI()                asm sei
#define  CLI()                asm cli
#define  STOP()               asm stop
#define  WAIT()               asm wait
#define  NOP()                asm nop
#define  ServiceWatchDog()    COPCTL.reg = 0

#endif
```

A.14 [Delay.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  Delay.c                                                    //
//  Author    :  T. Mudryk                                                 //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  delay routines                                                          //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#include    "delay.h"


////////////////////////////////////////////////////////////////////////////
// The total delay consists of loading the accumulator with the delay       //
// argument, branching to the delay routine and lastly returning from the   //
// routine, this is shown below:                                            //
//                                                                          //
//       lda      #D             ; delay arg sent to function  [2] cycles   //
//       jsr      Delay          ; branches to this function   [5] cycles   //
//                       psha                   :              [2] cycles  //
// DELAY1:                                                                  //
//       nop                     :                             [1] cycle    //
//       nop                     :                             [1] cycle    //
//       nop                     :                             [1] cycle    //
//       nop                     :                             [1] cycle    //
//       nop                     :                             [1] cycle    //
//       nop                     :                             [1] cycle    //
//       tsx                                   :           [2] cycles     //
//       dbnz    X, *-5          :                           [4] cycles     //
//       pulh                    :                           [2] cycles     //
```

Software Routines

```
//       rts                    :                              [4] cycles     //
//                                                                            //
// This gives a total delay of 13+12*D cycles, where D is the arg sent.       //
// With a 2MHZ bus speed, say we require a 1ms delay, then we have:           //
// 1E-3/(1/2E6) = 2000 bus cycles => 2000 = 15 + 12*D, => D = 165.41          //
// approx  = 165                                                              //
// 'Delay( 165 )' to get 1ms delay.                                           //
//                                                                            //
// Arguments: 'D' delay value as calculated from 'cyles = 15 + 12D'           //
// Returns  : none                                                            //
////////////////////////////////////////////////////////////////////////////////
void Delay( unsigned char delay_value )
{
#asm
DELAY1:
   nop
   nop
   nop
   nop
   nop
   nop
        tsx
   dbnz delay_value, DELAY1
#endasm
} // Delay()
//------------------------------------------------------------------------------
```

A.15 [Delay.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  Delay.h                                                    //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Header file for Delay.c                                                 //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/00 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DELAY_H_
#define     __DELAY_H_

//////////////////////////
// function prototypes //
//////////////////////////
void Delay( unsigned char );

#endif
```

A.16 [Display.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  Display.c                                                  //
//  Author    :  T. Mudryk                                                 //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Display Functions                                                       //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  06/06/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef         __EXTERN_H_
#include        "extern.h"
#endif

#include        "button.h"
#include        "delay.h"
#include        "Display.h"


void InitialDisplay( void )
{
unsigned short int temp;


for ( temp = 0; temp < 1000; temp++ )
   {
   Delay( _1MS );
   ServiceWatchDog();
   }

ClearDisplay();
```

```
Refresh = 1;

} // Display
//-----------------------------------------------------------------------------


void ClearDisplay( void )
{
volatile union uPORT*    ptr = &LDAT1;


for ( ; ptr <= &LDAT12; ptr++ ) ptr->reg = 0x00;
                                                  // All Segments Off
} // ClearDisplay
//-----------------------------------------------------------------------------


void DisplayNumber( unsigned char N, unsigned char digit )
{
volatile union uPORT*ptr = &LDAT1;


if ( digit%2 )                                    // odd digit check
  {
   switch ( digit )
      {
      case 1:  digit--;    break;                 // Compensates for 1.5 bytes
           case 5:digit++;   break;                // being used per digit
      case 7: digit += 2; break;                  //
           }

        switch ( N )
        {
                case 0x00:                        // '0'
                ptr[digit].byte        = 0x53;
                ptr[(digit + 1)].byte &= 0xf0;
                ptr[(digit + 1)].byte += 0x03;
                break;

                case 0x01:                        // '1'
                ptr[digit].byte        = 0x03;
                ptr[(digit + 1)].byte &= 0xf0;
                break;

                case 0x02:                        // '2'
                ptr[digit].byte        = 0x71;
                ptr[(digit + 1)].byte &= 0xf0;
                ptr[(digit + 1)].byte += 0x02;
                break;

                case 0x03:                        // '3'
                ptr[digit].byte        = 0x73;
                ptr[(digit + 1)].byte &= 0xf0;
                break;
```

```
                case 0x04:                          // '4'
                ptr[digit].byte        = 0x23;
                ptr[(digit + 1)].byte &= 0xf0;
                ptr[(digit + 1)].byte += 0x01;
                break;

                case 0x05:                          // '5'
                ptr[digit].byte        = 0x72;
                ptr[(digit + 1)].byte &= 0xf0;
                ptr[(digit + 1)].byte += 0x01;
                break;

                case 0x06:                          // '6'
                ptr[digit].byte        = 0x72;
                ptr[(digit + 1)].byte &= 0xf0;
                ptr[(digit + 1)].byte += 0x03;
                break;

                case 0x07:                          // '7'
                ptr[digit].byte        = 0x13;
                ptr[(digit + 1)].byte &= 0xf0;
                break;

                case 0x08:                          // '8'
      ptr[digit].byte        = 0x73;
                ptr[(digit + 1)].byte &= 0xf0;
                ptr[(digit + 1)].byte += 0x03;
                break;

                case 0x09:                          // '9'
                ptr[digit].byte        = 0x33;
      ptr[(digit + 1)].byte &= 0xf0;
                ptr[(digit + 1)].byte += 0x01;
                break;
           }
   if ( DP )   ptr[digit].bit.bit2       = 1;     // Set Decimal Point
   }
else                                                // even digit
   {
   switch ( digit )
      {
      case 2:  digit--;    break;                  // Compensates for 1.5 bytes...
           case 6:digit++;   break;                 // being used per digit
      case 8: digit += 2; break;                    //
        }
        switch (N)
           {
                case 0x00:                          // '0'
                ptr[(digit + 1)].byte  = 0x35;
                ptr[digit].byte       &= 0x0f;
                ptr[digit].byte       += 0x30;
                break;

                case 0x01:                          // '1'
                ptr[(digit + 1)].byte  = 0x00;
```

```
                ptr[digit].byte       &= 0x0f;
                ptr[digit].byte       += 0x30;
                break;

                case 0x02:                          // '2'
                ptr[(digit + 1)].byte  = 0x27;
                ptr[digit].byte       &= 0x0f;
                ptr[digit].byte       += 0x10;
                break;

                case 0x03:                          // '3'
                ptr[(digit + 1)].byte  = 0x07;
                ptr[digit].byte       &= 0x0f;
                ptr[digit].byte       += 0x30;
                break;

                case 0x04:                          // '4'
                ptr[(digit + 1)].byte  = 0x12;
                ptr[digit].byte       &= 0x0f;
                ptr[digit].byte       += 0x30;
                break;

                case 0x05:                          // '5'
                ptr[(digit + 1)].byte  = 0x17;
                ptr[digit].byte       &= 0x0f;
                ptr[digit].byte       += 0x20;
                break;

                case 0x06:                          // '6'
                ptr[(digit + 1)].byte  = 0x37;
                ptr[digit].byte       &= 0x0f;
                ptr[digit].byte       += 0x20;
                break;

                case 0x07:                          // '7'
                ptr[(digit + 1)].byte  = 0x01;
                ptr[digit].byte       &= 0x0f;
                ptr[digit].byte       += 0x30;
                break;

        case 0x08:                                  // '8'
                ptr[(digit + 1)].byte  = 0x37;
                ptr[digit].byte       &= 0x0f;
                ptr[digit].byte       += 0x30;
                break;

                case 0x09:                          // '9'
                ptr[(digit + 1)].byte  = 0x13;
                ptr[digit].byte       &= 0x0f;
                ptr[digit].byte       += 0x30;
                break;
                }
        if ( DP )   ptr[digit].bit.bit6 = 1;      // Set Decimal Point
  }
} // DisplayNumber()
```

```
//---------------------------------------------------------------------------


void ChevronRefresh( void )
{
if ( set_mode == MODE_ALARMSET )
   {
   switch ( DW )
      {
      case 0:                                  // If DOW = Mon - Fri
        LDAT6.bit.bit6  = 1;                   //
        LDAT8.bit.bit2  = 1;                   // Add Chevrons
        LDAT9.bit.bit6  = 1;                   //
        LDAT11.bit.bit2 = 1;                   //
        LDAT12.bit.bit6 = 1;                   //
        break;

        case 1:                                // If DOW = Sat - Sun
        LDAT3.bit.bit6  = 1;                   // Add Chevron
        LDAT5.bit.bit2  = 1;                   //
      break;

      case 2:                                  // If DOW = Sun
      LDAT3.bit.bit6  = 1;                     // Add Chevron
      break;

      case 4:                                  // If DOW = Sat
      LDAT5.bit.bit2  = 1;                     // Add Chevron
      break;

      case 5:                                  // If DOW = Fri
      LDAT6.bit.bit6  = 1;                     // Add Chevron
      break;

      case 7:                                  // If DOW = Thu
        LDAT8.bit.bit2  = 1;                   // Add Chevron
        break;

        case 8:                                // If DOW = Wed
        LDAT9.bit.bit6  = 1;                   // Add Chevron
        break;

        case 10:                               // If DOW = Tue
        LDAT11.bit.bit2 = 1;                   // Add Chevron
        break;

        case 11:                               // If DOW = Mon
        LDAT12.bit.bit6 = 1;                   // Add Chevron
        break;
      }
   }
else
   {
   switch ( DOWR.byte )
      {
```

```
      case 0:                                   // If DOW = Sun
        LDAT3.bit.bit6  = 1;                    // Add Chevron
        break;

        case 1:                                 // If DOW = Mon
        LDAT12.bit.bit6 = 1;                    // Add Chevron
      break;

      case 2:                                   // If DOW = Tue
      LDAT11.bit.bit2 = 1;                      // Add Chevron
      break;

      case 3:                                   // If DOW = Wed
      LDAT9.bit.bit6  = 1;                      // Add Chevron
      break;

      case 4:                                   // If DOW = Thu
      LDAT8.bit.bit2  = 1;                      // Add Chevron
      break;

      case 5:                                   // If DOW = Fri
        LDAT6.bit.bit6  = 1;                    // Add Chevron
        break;

        case 6:                                 // If DOW = Sat
        LDAT5.bit.bit2  = 1;                    // Add Chevron
        break;
      }
   }
} // ChevronRefresh()
//-----------------------------------------------------------------------------
```

A.17 [Display.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  Display.h                                                  //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Header file for 'Display.c'                                             //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  06/06/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef    __Display_H_
#define     __Display_H_


void InitialDisplay( void );
void ChevronRefresh( void );
void DisplayNumber( unsigned char, unsigned char );
void ClearDisplay( void );

#endif
```

A.18 [extern.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  extern.h                                                   //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  'extern' declarations for global variables declared in 'data.c'         //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#define     __EXTERN_H_

#ifndef     __DECLARED_H_
#include    "declared.h"
#endif

#ifndef     __LJ12_H_
#include    "lj12.h"
#endif

#ifndef     __DEFINE_H_
#include    "define.h"
#endif


#pragma DATA_SEG __SHORT_SEG _DATA_ZEROPAGE

//////////////////////////
// PAGE0 Global variables //
//////////////////////////
```

```
extern uBITS                    button_flags;
extern unsigned char            N2;
extern unsigned char            DW;
extern unsigned char            DP;
extern unsigned char            PR2;
extern unsigned char            sign;
extern unsigned char            SMin;
extern unsigned char            Hour;
extern unsigned char            mode;
extern unsigned char            smart;
extern unsigned char            SHour;
extern unsigned char            _10ms;
extern unsigned char            onoff;
extern unsigned char            Minute;
extern unsigned char            SFLASH;
extern unsigned char            Active;
extern unsigned char            dow_set;
extern unsigned char            Refresh;
extern unsigned char            PRNumber;
extern unsigned char            low_byte;
extern unsigned char            set_mode;
extern unsigned char            _0P25sec;
extern unsigned char            Override;
extern unsigned char            OverrideH;
extern unsigned char            OverrideM;
extern unsigned char            Cset_mode;
extern unsigned char            Aset_mode;
extern unsigned char            _5sectick;
extern unsigned char            one_second;
extern unsigned char            two_second;
extern unsigned char            AlarmPRNumber;
extern unsigned char            button_pattern;
extern unsigned char            pressed_pattern;
extern unsigned char            user_temperature;
extern unsigned char            button_press_status;
extern unsigned char            external_temperature;
extern unsigned char            button_release_counter;
extern unsigned char            button_debounce_counter;
extern unsigned char            external_temperature_decimal;
extern volatile uBITS           flags1;
extern unsigned short int       smart_time;
extern unsigned short int       stop_counter;
extern unsigned short int       BatteryCheck;
extern unsigned short int       internal_temperature;
extern struct   sPROG_PARAMS    program[4];


////////////////////////////////
// Global non PAGE0 variables //
////////////////////////////////
#pragma DATA_SEG _FAR_RAM

//////////////////////////
// 'const' data ie rom //
//////////////////////////
```

```
#pragma CONST_SEG ATEECC_CONST_DATA

#endif
```

A.19 [i2c.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                 //
//  Filename  :  i2c.c                                                      //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  i2c routines for accessing external temperature                         //
//////////////////////////  Update Information  ////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  17/06/00 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef         __EXTERN_H_
#include        "extern.h"
#endif

#include        "i2c.h"


unsigned char WaitForI2CAcknowledge( void )
{
unsigned char   temp = 0;


SET_DATA_TO_OUTPUT;

//////////////////////////////////////////////////////////
// set SDA hi because during the 9th clock the SLAVE will //
// pull the SDA line lo                                   //
//////////////////////////////////////////////////////////
SET_SDA;

//////////////////////////////////////////
```

```
// data line now input so we can see go lo //
//////////////////////////////////////////
SET_DATA_TO_INPUT;
SetUpAndHoldTimingDelay();

//////////////////////////////////////////
// SLAVE should pull line lo anytime       //
//////////////////////////////////////////
SET_SCL;
SetUpAndHoldTimingDelay();

while ( READ_SDA )
   {                                        ////////////////////////////
   if ( ++temp >= 250 )                     // basic error check here //
      {                                     ////////////////////////////
      SET_DATA_TO_OUTPUT;                   // back to output
      SetUpAndHoldTimingDelay();
      RESET_SCL;                            // 9th clock bit complete
        return 0;
      }
   }

SET_DATA_TO_OUTPUT;                         // back to output
SetUpAndHoldTimingDelay();
RESET_SCL;                                  // 9th clock bit complete
return 1;
}  // WaitForI2CAcknowledge()
//-----------------------------------------------------------------------------


void SendI2CAcknowledge( void )
{
//////////////////////////////////////////
// the slave RTC has left the SDA line high //
// for us to send an ACKNOWLEDGE           //
//////////////////////////////////////////

                            ////////////////////////////////////////////
SET_SDA;                    // ensure output transistor is '1' before   //
                            // making an output                         //
SET_DATA_TO_OUTPUT;         // take control of the SDA line             //
RESET_SCL;                  // an ACKNOWLEGE occurs                     //
SetUpAndHoldTimingDelay();  //                                         //
RESET_SDA;                  // when the SDA is stable lo                //
SetUpAndHoldTimingDelay();  //                                         //
SET_SCL;                    // when the clock                           //
SetUpAndHoldTimingDelay();/ /                                         //
RESET_SCL;                  // goes hi->lo                              //
SetUpAndHoldTimingDelay();/ /                                         //
SET_DATA_TO_INPUT;          // relinquish control back to the slave RTC //
                            ////////////////////////////////////////////
} // SendI2CAcknowledge()
//-----------------------------------------------------------------------------
```

Software Routines

```
unsigned char InClock( void )
{
unsigned char   temp;


SET_SCL;

SET_DATA_TO_INPUT;

SetUpAndHoldTimingDelay();
if (READ_SDA)   temp = 1;
else            temp = 0;

RESET_SCL;                                      // reset clock lo to complete read
SetUpAndHoldTimingDelay();

return temp;
} // InClock()
//-------------------------------------------------------------------------


void OutClock( void )
{
SET_SCL;
SetUpAndHoldTimingDelay();
RESET_SCL;
SetUpAndHoldTimingDelay();
} // OutClock()
//-------------------------------------------------------------------------


void StartBit( void )
{
////////////////////////////////
// bus inactive conditions here //
////////////////////////////////
SET_SDA;
SET_SCL;
SET_CLOCK_TO_OUTPUT;
SET_DATA_TO_OUTPUT;
SetUpAndHoldTimingDelay();

/////////////////
// apply START //
/////////////////
SET_SDA;
SetUpAndHoldTimingDelay();
SET_SCL;
SetUpAndHoldTimingDelay();
RESET_SDA;
SetUpAndHoldTimingDelay();
RESET_SCL;
SetUpAndHoldTimingDelay();
} // StartBit()
//-------------------------------------------------------------------------
```

```
void StopBit( void )
{
RESET_SDA;
SetUpAndHoldTimingDelay();
SET_SCL;
SetUpAndHoldTimingDelay();
SET_SDA;
SetUpAndHoldTimingDelay();
} // StopBit()
//-------------------------------------------------------------------------------


void SendI2CByte( unsigned char value )
{
unsigned char     loop;


SET_DATA_TO_OUTPUT;

//////////////////////////////////
// clock is reset from start bit //
//////////////////////////////////
for ( loop = 0; loop < 8; loop++ )
   {
   value <<= 1;                                 // load carry flag with bit7

   if ( CarryFlagCheck() )
      {
        SET_SDA;
      }
   else
      {
      RESET_SDA;
      }
                                                ////////////////////////////////////////
   OutClock();                                  // data is ready now generate the clock //
   }                                            ////////////////////////////////////////
}  // SendI2CByte()
//-------------------------------------------------------------------------


unsigned char GetI2CByte( void )
{
unsigned char loop;
unsigned char receiving_value;


SET_DATA_TO_INPUT;

receiving_value = 0;

for ( loop = 0; loop < 8; loop++ )
   {
```

```
   receiving_value <<= 1;                          // shifting data left

   if ( InClock() )                                // get next bit sample...
      {                                            // returns either 0 or 1
      receiving_value |= 1;                        // setting bit0 if hi
      }
   }

return receiving_value;
} // GetI2CByte()
//-------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// We require function call-content-return to take 5us (worst case timing).   //
// 5us/(1/2.0E6) == 10 bus cycles                                             //
//                                                                            //
// The call (jsr) is [5] cycles, the return (rts) is [4] cycles leaving the   //
// function body to occupy 10-(5+4) == 1 cycle                                //
////////////////////////////////////////////////////////////////////////////
void SetUpAndHoldTimingDelay( void )
{
NOP();
} // SetUpAndHoldTimingDelay()
//-------------------------------------------------------------------------


void ExternalTemperatureRead( void )
{
union uPORT                     lobyte;
char                                            hibyte;
unsigned short int   dec;


SET_DATA_TO_OUTPUT;

StartBit();
SendI2CByte(BASE_ADDRESS_0);                    // device address - R/W bit = 0
                                                //
if ( WaitForI2CAcknowledge() )                  //
   {                                            //
   SendI2CByte(START);                          // start conversion
                                                //
   if ( WaitForI2CAcknowledge() )               //
      {                                         //
      StartBit();                               // repeated start bit
      SendI2CByte(BASE_ADDRESS_0);              // re send device address
                                                //
      if ( WaitForI2CAcknowledge() )            //
         {                                      //
         SendI2CByte(READ);                     // command to read double byte of data
                                                //
         if ( WaitForI2CAcknowledge() )         //
            {                                   //
            StartBit();                         // re send start condition
```

```
SendI2CByte(BASE_ADDRESS_1);          // re send device address
                                      //
if ( WaitForI2CAcknowledge() )        //
   {                                  //
   SET_DATA_TO_INPUT;                 // prepare to receive high byte of data
   hibyte = ( char )GetI2CByte();     // get high byte
   SendI2CAcknowledge();              //
   lobyte.byte = GetI2CByte();        // get low byte

   sign = 0;                          // assume temp're will be +ve

                                      ////////////////////////////
                                      // -ve temperature check //
                                      ////////////////////////////
                                      if ( hibyte < 0 )
                                      {
                                      sign = 1;

    ////////////////////////////////////////////////////////
    // integer result, convert from 2's complement to     //
    // magnitude.                                         //
    // NOTE: no +1 required for 2's complement conversion //
    // as 2's complement value is 1 lower than actual    //
    //                                                   //
    // This next line produces Lint Warning 502:          //
    // Expected  unsigned type  -- Unary ~ being  a  bit  //
    // operator would  more  logically  be applied to     //
    // unsigned  quantities rather than signed quantities //
    //                                                    //
    // The 'com' instruction is used and the result is    //
    // correct.                                           //
    ////////////////////////////////////////////////////////

    hibyte = ~hibyte;

    // fractional result, convert from 2's com to magnitude
    lobyte.byte = ~lobyte.byte + 1;
                                      }

    ////////////////////////////////
    // fractional part processing //
    ////////////////////////////////
    dec = 0;

    if ( lobyte.byte >= 0x10 )// is bit 4 or greater, set
              {
              if ( lobyte.bit.bit7 )
                 {
                 dec += 5000;         // 0.5*10000
                 }

              if ( lobyte.bit.bit6 )
                 {
                 dec += 2500;         // 0.25*10000
                 }
```

```
                        if ( lobyte.bit.bit5 )
                           {
                           dec += 1250;         // 0.125*10000
                           }

                        if ( lobyte.bit.bit4 )
                           {
                           dec += 625;          // 0.0625*10000
                           }

                        // checking for '0.05' second dec place rounding
                        if ( dec%1000 >= 500 )  // is remainder >= 0.05
                           {
                           dec += 1000;         // 0.1 correction on first decimal
                           }
                                                // div by 1000 (not 10000) to
                           dec /= 1000;         // produce integer value for first
                                                // decimal place

                        ////////////////////////////////
                        // assignment to global space //
                        ////////////////////////////////
                        external_temperature         = ( unsigned char )hibyte
                                                      + EXTERNAL_CALIBRATION;
                        external_temperature_decimal = ( unsigned char )dec;
                        }
                  }
               }
            }
         }
      }
   }
} // ExternalTemperatureRead()
//-------------------------------------------------------------------------------


unsigned char CarryFlagCheck( void )
{
unsigned char  result = 0;                      // assume carry clear


#asm
bcc     CARRY_CHECK_COMPLETE
lda     #$01
sta     result
CARRY_CHECK_COMPLETE:
#endasm

return result;
} // CarryFlagCheck()
//-------------------------------------------------------------------------------
```

A.20 [i2c.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  i2c.h                                                      //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Header file for 'i2c.c'                                                 //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  17/06/00 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __I2C_H_
#define     __I2C_H_


/////////////////
// I2C defines //
/////////////////
// data
#define SET_DATA_TO_OUTPUT    DDRD.bit.bit2  = 1
#define SET_DATA_TO_INPUT     DDRD.bit.bit2  = 0
#define RESET_SDA             PTD.bit.bit2   = 0
#define SET_SDA               PTD.bit.bit2   = 1
#define READ_SDA              PTD.bit.bit2

// clock
#define SET_CLOCK_TO_OUTPUT   DDRD.bit.bit3  = 1
#define RESET_SCL             PTD.bit.bit3   = 0
#define SET_SCL               PTD.bit.bit3   = 1

// temperature commands
#define BASE_ADDRESS_0        0x90
```

Software Routines

```
#define BASE_ADDRESS_1        0x91
#define START                 0x51
#define READ                  0xaa


void           SetUpAndHoldTimingDelay( void );
void                  ExternalTemperatureRead( void );
unsigned char  WaitForI2CAcknowledge( void );
void           SendI2CAcknowledge( void );
unsigned char  CarryFlagCheck( void );
void           SendI2CByte( unsigned char );
unsigned char  GetI2CByte( void );
void           OutClock( void );
void           StartBit( void );
unsigned char  InClock( void );
void           StopBit( void );

#endif
```

A.21 [interrupt.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  interrup.c                                                 //
//  Author    :  T. Mudryk                                                 //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Interrupt routines                                                      //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#include    "extern.h"
#endif

#include    "interrup.h"
#include    "Display.h"
#include    "startup.h"
#include    "Alarm.h"


#pragma TRAP_PROC
void RTC_INTERRUPT( void )
{
unsigned char temp;


////////////////////////////////
// 100Hz chronograph interrupt //
////////////////////////////////
if ( RTCSR.bit.CHRF && RTCCR1.bit.CHRIE )
   {
```

```
   temp             = CHRR.reg;                  // clear interrupt by reading CHRR reg
   flags1.bit._10MS = 1;                         // main() sequencer
   }
   // RTC_INTERRUPT()
//---------------------------------------------------------------------------


#pragma TRAP_PROC
void LVI_INTERRUPT( void )
{
ServiceWatchDog();
PTB.byte = 0x00;                                 // Turn off heating/cooling & LED's
LVISR.bit.LVIIAK = 1;                            // Resets the interrupt flag

if ( LVISR.bit.LVIOUT )
   {
   DDRB.reg = 0x00;                              // Set ports to input to reduce...
   DDRD.reg = 0x00;                              // current drain
   RTCCR1.bit.CHRIE = 0;                         // Disables RTC 100Hz interrupt

   ClearDisplay();
   LCDCR.reg  = 0xb0;                            // Adjust LCD to lower power mode
   LDAT6.byte = 0x36;                            // Displays 'bAtt'
   LDAT5.byte = 0x23;                            //
   LDAT4.byte = 0x33;                            //
   LDAT3.byte = 0x36;                            //
   LDAT2.byte = 0x03;                            //
   LDAT1.byte = 0x60;                            //

   STOP();
   }
else
   {
   RTCCR2.bit.CHRCLR = 1;                        // Reset chronograph counter
   MicroStartUp();                               // Reinitialises the micro
   Alarm();                                      // Load next alarm into the register
   }
}       // LVI_INTERRUPT()
//---------------------------------------------------------------------------
```

A.22 [interrupt.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  interrup.h                                                 //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Header file for interrup.c                                              //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __INTERRUP_H_
#define     __INTERRUP_H_


///////////////
// prototypes //
///////////////
void RTC_INTERRUPT( void );

#endif
```

A.23 [lj12.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  lj12.h                                                     //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  register definitions for MC68HC908LJ12                                  //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __LJ12_H_
#define     __LJ12_H_

#ifndef     __DECLARED_H_
#include    "declared.h"
#endif


//////////////////
// CPU Registers //
//////////////////
#define PTA       (*(volatile union uPORT*)0x0000)
#define PTB       (*(volatile union uPORT*)0x0001)
#define PTC       (*(volatile union uPORT*)0x0002)
#define PTD       (*(volatile union uPORT*)0x0003)
#define DDRA      (*(volatile union uPORT*)0x0004)
#define DDRB      (*(volatile union uPORT*)0x0005)
#define DDRC      (*(volatile union uPORT*)0x0006)
#define DDRD      (*(volatile union uPORT*)0x0007)
#define LEDB      (*(volatile union uPORT*)0x000C)
#define SPCR      (*(volatile union uPORT*)0x0010)
```

```
#define SPSCR     (*(volatile union uPORT*)0x0011)
#define SPDR      (*(volatile union uPORT*)0x0012)
#define SCC1      (*(volatile union uPORT*)0x0013)
#define SCC2      (*(volatile union uPORT*)0x0014)
#define SCC3      (*(volatile union uPORT*)0x0015)
#define SCS1      (*(volatile union uPORT*)0x0016)
#define SCS2      (*(volatile union uPORT*)0x0017)
#define SCDR      (*(volatile union uPORT*)0x0018)
#define SCBR      (*(volatile union uPORT*)0x0019)
#define SCIRCR    (*(volatile union uPORT*)0x001A)
#define KBSCR     (*(volatile union uPORT*)0x001B)
#define KBIER     (*(volatile union uPORT*)0x001C)
#define CONFIG2   (*(volatile union uPORT*)0x001D)
#define INTSCR    (*(volatile union uPORT*)0x001E)
#define CONFIG1   (*(volatile union uPORT*)0x001F)
#define T1SC      (*(volatile union uPORT*)0x0020)
#define T1CNTH    (*(volatile union uPORT*)0x0021)
#define T1CNT     (*(volatile USHORT*)0x0021)
#define T1CNTL    (*(volatile union uPORT*)0x0022)
#define T1MODH    (*(volatile union uPORT*)0x0023)
#define T1MOD     (*(volatile USHORT*)0x0023)
#define T1MODL    (*(volatile union uPORT*)0x0024)
#define T1SC0     (*(volatile union uPORT*)0x0025)
#define T1CH0H    (*(volatile union uPORT*)0x0026)
#define T1CH0     (*(volatile USHORT *)0x0026)
#define T1CH0L    (*(volatile union uPORT*)0x0027)
#define T1SC1     (*(volatile union uPORT*)0x0028)
#define T1CH1H    (*(volatile union uPORT*)0x0029)
#define T1CH1     (*(volatile USHORT*)0x0029)
#define T1CH1L    (*(volatile union uPORT*)0x002A)
#define T2SC      (*(volatile union uPORT*)0x002B)
#define T2CNTH    (*(volatile union uPORT*)0x002C)
#define T2CNT     (*(volatile USHORT*)0x002C)
#define T2CNTL    (*(volatile union uPORT*)0x002D)
#define T2MODH    (*(volatile union uPORT*)0x002E)
#define T2MOD     (*(volatile USHORT*)0x002E)
#define T2MODL    (*(volatile union uPORT*)0x002F)
#define T2SC0     (*(volatile union uPORT*)0x0030)
#define T2CH0H    (*(volatile union uPORT*)0x0031)
#define T2CH0     (*(volatile USHORT*)0x0031)
#define T2CH0L    (*(volatile union uPORT*)0x0032)
#define T2SC1     (*(volatile union uPORT*)0x0033)
#define T2CH1H    (*(volatile union uPORT*)0x0034)
#define T2CH1     (*(volatile USHORT*)0x0034)
#define T2CH1L    (*(volatile union uPORT*)0x0035)
#define PTCL      (*(volatile union uPORT*)0x0036)
#define PBWC      (*(volatile union uPORT*)0x0037)
#define PMSH      (*(volatile union uPORT*)0x0038)
#define PMS       (*(volatile USHORT*)0x0038)
#define PMSL      (*(volatile union uPORT*)0x0039)
#define PMRS      (*(volatile union uPORT*)0x003A)
#define PMDS      (*(volatile union uPORT*)0x003B)
#define ADSCR     (*(volatile union uPORT*)0x003C)
#define ADRH      (*(volatile union uPORT*)0x003D)
#define ADR       (*(volatile USHORT*)0x003D)
```

```
#define ADRL      (*(volatile union uPORT*)0x003E)
#define ADCLK     (*(volatile union uPORT*)0x003F)
#define RTCCR1    (*(volatile union uPORT*)0x0042)
#define RTCCR2    (*(volatile union uPORT*)0x0043)
#define RTCSR     (*(volatile union uPORT*)0x0044)
#define ALMR      (*(volatile union uPORT*)0x0045)
#define ALHR      (*(volatile union uPORT*)0x0046)
#define SECR      (*(volatile union uPORT*)0x0047)
#define MINR      (*(volatile union uPORT*)0x0048)
#define HRR       (*(volatile union uPORT*)0x0049)
#define DAYR      (*(volatile union uPORT*)0x004A)
#define MTHR      (*(volatile union uPORT*)0x004B)
#define YRR       (*(volatile union uPORT*)0x004C)
#define DOWR      (*(volatile union uPORT*)0x004D)
#define CHRR      (*(volatile union uPORT*)0x004E)
#define LCDCLK    (*(volatile union uPORT*)0x004F)
#define LCDCR     (*(volatile union uPORT*)0x0051)
#define LDAT1     (*(volatile union uPORT*)0x0052)
#define LDAT2     (*(volatile union uPORT*)0x0053)
#define LDAT3     (*(volatile union uPORT*)0x0054)
#define LDAT4     (*(volatile union uPORT*)0x0055)
#define LDAT5     (*(volatile union uPORT*)0x0056)
#define LDAT6     (*(volatile union uPORT*)0x0057)
#define LDAT7     (*(volatile union uPORT*)0x0058)
#define LDAT8     (*(volatile union uPORT*)0x0059)
#define LDAT9     (*(volatile union uPORT*)0x005A)
#define LDAT10    (*(volatile union uPORT*)0x005B)
#define LDAT11    (*(volatile union uPORT*)0x005C)
#define LDAT12    (*(volatile union uPORT*)0x005D)
#define LDAT13    (*(volatile union uPORT*)0x005E)
#define LDAT14    (*(volatile union uPORT*)0x005F)
#define SBSR      (*(volatile union uPORT far *)0xFE00)
#define SRSR      (*(volatile union uPORT far *)0xFE01)
#define SBFCR     (*(volatile union uPORT far *)0xFE03)
#define INT1      (*(volatile union uPORT far *)0xFE04)
#define INT2      (*(volatile union uPORT far *)0xFE05)
#define INT3      (*(volatile union uPORT far *)0xFE06)
#define FLCR      (*(volatile union uPORT far *)0xFE08)
#define FLBPR     (*(volatile union uPORT far *)0xFE09)
#define BRKH      (*(volatile union uPORT far *)0xFE0C)
#define BRK       (*(volatile USHORT far *)0xFE0C)
#define BRKL      (*(volatile union uPORT far *)0xFE0D)
#define BRKSCR    (*(volatile union uPORT far *)0xFE0E)
#define LVISR     (*(volatile union uPORT far *)0xFE0F)
#define COPCTL    (*(volatile union uPORT far *)0xFFFF)

//////////////////////////////
// CPU register bit defines //
//////////////////////////////

/////////
// INT1 //
/////////
#define  IF1      bit2
#define  IF2      bit3
```

```
#define  IF3      bit4
#define  IF4      bit5
#define  IF5      bit6
#define  IF6      bit7

/////////
// INT2 //
/////////
#define  IF7      bit0
#define  IF8      bit1
#define  IF9      bit2
#define  IF10     bit3
#define  IF11     bit4
#define  IF12     bit5
#define  IF13     bit6
#define  IF14     bit7

/////////
// INT3 //
/////////
#define  IF15     bit0
#define  IF16     bit1
#define  IF17     bit2

////////////
// T1SC reg //
////////////
#define PS0       bit0
#define PS1       bit1
#define PS2       bit2
#define TRST      bit4
#define TSTOP     bit5
#define TOIE      bit6
#define TOF       bit7

//////////////
// T1SC0 reg //
//////////////
#define CH0MAX    bit0
#define TOV0      bit1
#define ELS0A     bit2
#define ELS0B     bit3
#define MS0A      bit4
#define MS0B      bit5
#define CH0IE     bit6
#define CH0F      bit7

////////////
// TSC1 reg //
////////////
#define CH1MAX    bit0
#define TOV1      bit1
#define ELS1A     bit2
#define ELS1B     bit3
#define MS1A      bit4
```

```
#define CH1IE     bit6
#define CH1F      bit7

////////////////////////////////////
// A2D status & control reg (ADSCR) //
////////////////////////////////////
#define ADCH0     bit0
#define ADCH1     bit1
#define ADCH2     bit2
#define ADCH3     bit3
#define ADCH4     bit4
#define ADCO      bit5
#define AIEN      bit6
#define COCO      bit7

//////////////////////////////////
// A2D input clock reg (ADICLK) //
//////////////////////////////////
#define MODE0     bit2
#define MODE1     bit3
#define ADICLK    bit4
#define ADIV0     bit5
#define ADIV1     bit6
#define ADIV2     bit7

//////////////////////////
// FLASH control (FLCR) //
//////////////////////////
#define PGM       bit0
#define ERASE     bit1
#define MASS      bit2
#define HVEN      bit3

////////////////////////////
// KEYBOARD status/control //
////////////////////////////
#define MODEK     bit0
#define IMASKK    bit1
#define ACKK      bit2
#define KEYF      bit3

//////////////////////////////
// KEYBOARD interrupt enable //
//////////////////////////////
#define KBIE0     bit0
#define KBIE1     bit1
#define KBIE2     bit2
#define KBIE3     bit3
#define KBIE4     bit4
#define KBIE5     bit5
#define KBIE6     bit6
#define KBIE7     bit7

//////////////////////////
// PLL bandwith control //
```

```
//////////////////////////
#define AUTO      bit7
#define LOCK      bit6
#define ACQ       bit5

/////////////////
// PLL control //
/////////////////
#define PLLIE     bit7
#define PLLF      bit6
#define PLLON     bit5
#define BCS       bit4
#define PRE1      bit3
#define PRE0      bit2
#define VPR1      bit1
#define VPR0      bit0

//////////
// SCS1 //
//////////
#define SCTE      bit7
#define TC        bit6
#define SCRF      bit5
#define IDLE      bit4
#define OR        bit3
#define NF        bit2
#define FE        bit1
#define PE        bit0

//////////
// SCC2 //
//////////
#define SCTIE     bit7
#define TCIE      bit6
#define SCRIE     bit5
#define ILIE      bit4
#define TE        bit3
#define RE        bit2
#define RWU       bit1
#define SBK       bit0

//////////////////////////
// RTC control1 (RTCCR1) //
//////////////////////////
#define TB2IE     bit0
#define TB1IE     bit1
#define SECIE     bit2
#define MINIE     bit3
#define HRIE      bit4
#define DAYIE     bit5
#define CHRIE     bit6
#define ALMIE     bit7

//////////////////////////
// RTC control (RTCCR2) //
```

Software Routines

```
//////////////////////////
#define XTL0      bit0
#define XTL1      bit1
#define XTL2      bit2
#define RTCE      bit4
#define CHRE      bit5
#define CHRCLR    bit6

////////////////////////
// RTC Status (RTCSR) //
////////////////////////
#define TB2F      bit0
#define TB1F      bit1
#define SECF      bit2
#define MINF      bit3
#define HRF       bit4
#define DAYF      bit5
#define CHRF      bit6
#define ALMF      bit7

/////////////////////////////
// IRQ status and control //
/////////////////////////////
#define MODE      bit0
#define IMASK     bit1
#define ACK       bit2
#define IRQF      bit3

////////////////////////
// LVI Status (LVISR) //
////////////////////////
#define LVIIAK     bit4
#define LVIIF      bit5
#define LVIIE      bit6
#define LVIOUT     bit7

#endif
```

A.24 [main.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  main.c                                                     //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  main application routine                                                //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef    __EXTERN_H_
#include   "extern.h"
#endif

#include   "temperature.h"
#include   "operation.h"
#include   "Clockset.h"
#include   "Alarmset.h"
#include   "Display.h"
#include   "startup.h"
#include   "button.h"
#include   "Alarm.h"


void main( void )
{
MicroStartUp();                                 // Initialise micro
DOWR.byte = 0x01;                               // Set DOW to Mon on first startup

for ( ;; )
   {
```

Software Routines

```
   ServiceWatchDog();
        TenMsSynchronise();

        ReadButtons();
        NewDayRefresh();

        switch ( set_mode )
           {
           case MODE_DEFAULT:
           TimeRefresh();
           InsideTemperature();
           OverrideCheck();
           AlarmFlag();
           ClimateControl();
           break;

           case MODE_CLOCKSET:
           Cset();
           break;

           case MODE_ALARMSET:
           ModeAlarmSet();
           break;

           case MODE_VIEWTEMP:
           TemperatureView();
           break;
           }
   ///////////////
   // 10ms sync //
   ///////////////
   while ( !flags1.bit._10MS ) ;                  // Waiting for interrupt
   flags1.bit._10MS = 0;                          // reset, for next loop iteration
        }       // end of 'for( ;; )'
}               // end of main
//---------------------------------------------------------------------------
```

A.25 [operation.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTTT EEEEEEEEEEE EEEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTTT EEE         EEE          CC         CC          //
//     AAAAAA        TTTT     EEEEE       EEEEE        CC         CC          //
//    AAAAAAAA       TTTT     EEEEE       EEEEE        CC         CC          //
//   AAAA  AAAA      TTTT     EEE         EEE          CC         CC          //
//  AAAA    AAAA     TTTT     EEEEEEEEEEE EEEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                                 //
// Unit 32, Consett Business Park                                             //
// Villa Real, Consett                                                        //
// Co. Durham                                                                 //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                    //
//  Filename  :  operation.c                                                  //
//  Author    :  T. Mudryk                                                    //
//  Compiler  :  CodeWarrior v5.2                                             //
//  CPU       :  MC68HC908LJ12                                                //
////////////////////  File Contents  ////////////////////////////////////////
//  General Operations                                                        //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  17/07/03 tm      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#include    "extern.h"
#endif

#include    "Clockset.h"
#include    "Display.h"
#include    "Alarm.h"


void ClimateControl( void )
{
switch ( mode )
        {
        case AUTOMATIC:
        PTB.bit.bit4 = 1;
        PTB.bit.bit5 = 1;
        break;

        case HEAT:
        PTB.bit.bit4 = 0;
        PTB.bit.bit5 = 1;
```

Software Routines

```
        break;

        case COOL:
        PTB.bit.bit4 = 1;
        PTB.bit.bit5 = 0;
        break;

        case OFF:
        PTB.bit.bit4 = 0;
        PTB.bit.bit5 = 0;
        break;
   }

if ( Active )                                   // If heating control is on
   {
   if ( user_temperature > (( internal_temperature/10 ) + 1 ))
        {                                       // Divide by 10 for integer value
        if ( mode == AUTOMATIC || mode == HEAT )
                {                               // If heating is needed
                PTB.bit.bit0 = 0x01;            // Heating on
                PTB.bit.bit1 = 0x00;            // Cooling off
                if ( one_second ) PTB.bit.bit5 = 0;// Set mode LED to flash on heating
                else              PTB.bit.bit5 = 1;//
                }
                else if ( mode == COOL || mode == OFF )
                   {                            // If heating is not needed
                   PTB.bit.bit0 = 0x00;         // Heating off
                   }
                }
        else if (user_temperature < (( internal_temperature/10 ) + 1 ))
                {                               // Divide by 10 for integer value
                if ( mode == AUTOMATIC || mode == COOL )
                   {                            // If cooling is needed
                   PTB.bit.bit0 = 0x00;         // Heating off
                   PTB.bit.bit1 = 0x01;         // Cooling on
                   if ( one_second ) PTB.bit.bit4 = 0;// Set mode LED to flash on cooling
                   else              PTB.bit.bit4 = 1;//
                   }
                else if ( mode == HEAT || mode == OFF )
                   {                            // If cooling is not needed
                   PTB.bit.bit1 = 0x00;         // Cooling off
                   }
                }
        else
                {
                PTB.bit.bit0 = 0x00;            // Heating off
                PTB.bit.bit1 = 0x00;            // Cooling off
                }
        }
} // ClimateControl()
//---------------------------------------------------------------------------


void SmartTimeCheck( void )
{
```

```
if ( HRR.byte == SHour && MINR.byte == SMin )// One hour before on time
   {
   if ( smart )                                    // Is smart mode active
      {
      smart_time = (( 1000 * internal_temperature )/10 )/user_temperature;
                                                   // Divide by 10 for integer value
                                                   // Smart calculations, stage 1
      if ( smart_time > 1000 )                     // For Smart cooling
         {                                         //
         smart_time -= 1000;                       //
         }                                         //
      else                                         // For Smart heating
         {                                         //
         smart_time = 1000 - smart_time;//
         }                                         //
      smart_time = 3 * (( smart_time * ( internal_temperature/10 )) /
                        ( 100 * external_temperature      ));
                                                   // Divide by 10 for integer value
                                                   // Smart calculations, stage 2
      if ( smart_time > 50 )                       //
         {                                         // Maximum cap for smart minutes
         smart_time = 50;                          //
         }                                         //
      if ( SMin < smart_time )                     // Prevention for negative minutes
         {                                         //
         SMin += 50;                               // Maximum cap
         }                                         //
      else
         {
         SHour += 1;                               // Adds 1 hour back on to Shour
         }
      SMin -= (unsigned char)smart_time;// Alarm minutes reduced by smart
      program[PRNumber].min_on  = SMin;// Loads new on minutes into array
      program[PRNumber].hour_on = SHour;// Loads new on hour into array
      PRNumber = NO_PROGRAM;
      Alarm();                                     // New times loaded into Alarm reg
      }
   }
} // SmartTimeCheck()
//-----------------------------------------------------------------------------


void OverrideCheck( void )
{
if ( Override == 1 )                               // If override is active
   {
   if ( OverrideH == HRR.byte && OverrideM == MINR.byte )
      {                                            // If override period expired
      Override = 0;                                // Reset override
      if ( !PR2 )                                  // Program is not currently running
         {
         Active = 0;                               // Turn heating/cooling off
         PTB.bit.bit0 = 0x00;                      //
         PTB.bit.bit1 = 0x00;                      //
         }
```

```
      }
   }
} // OverrideCheck()
//---------------------------------------------------------------------------


void TimeRefresh( void )
{
if ( BatteryCheck <= LOWBATT && two_second )
   {
   LDAT12.byte = 0x00;                          // Clears unused digit
   LDAT11.byte = 0x03;                          // Displays 'Lo'
   LDAT10.byte = 0x40;                          //
   LDAT9.byte  = 0x26;                          //
   LDAT8.byte  = 0x20;                          //
   LDAT7.byte  = 0x00;                          // Clears unused digit
   LDAT6.byte  = 0x36;                          // Displays 'bAtt'
   LDAT5.byte  = 0x23;                          //
   LDAT4.byte  = 0x33;                          //
   LDAT3.byte  = 0x36;                          /
   LDAT2.byte  = 0x03;                          //
   LDAT1.byte  = 0x60;                          //
   }
else
   {
   LDAT6.byte  = 0x00;                          // Removes the 'b'
   LDAT5.byte &= 0x0f;                          //
        MINClr();                               // Refresh TOD display
        HRClr();                                //
        Refresh = 1;                            // Refresh internal temperature
        ChevronRefresh();
        }
} // TimeRefresh()
//---------------------------------------------------------------------------


void TenMsSynchronise( void )
{
if ( ++_10ms%25   == 0 )                        // every 0.25s
   {                                            //
   _0P25sec       ^= 0x01;                      //
                                                //
   if ( _10ms%50  == 0 )                        // every 0.5s
      {                                         //
      SFLASH      ^= 0x01;                      //
      }                                         //
                                                //
   if ( _10ms%100 == 0 )                        // every 1.0s
      {                                         //
      one_second  ^= 0x01;                      //
      _5sectick++;                              //
      }                                         //
                                                //
   if ( _10ms >= 200 )                          // Unsigned char, 256 max
      {                                         //
```

```
      two_second  ^= 0x01;                    // every 2.0s
      10ms = 0;                                    // reset for next
      }                                            //
   }                                               //
}  // TenMsSynchronise()
//-------------------------------------------------------------------------
```

A.26 [operation.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  operation.h                                                //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Header file for 'operation.c'                                           //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  17/07/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __OPERATION_H_
#define     __OPERATION_H_


void TenMsSynchronise( void );
void SmartTimeCheck( void );
void ClimateControl( void );
void OverrideCheck( void );
void TimeRefresh( void );

#endif
```

A.27 [startup.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTTT EEEEEEEEEEE EEEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTTT EEE         EEE          CC         CC          //
//     AAAAAA        TTTT     EEEEE       EEEEE        CC         CC          //
//    AAAAAAAA       TTTT     EEEEE       EEEEE        CC         CC          //
//   AAAA  AAAA      TTTT     EEE         EEE          CC         CC          //
//  AAAA    AAAA     TTTT     EEEEEEEEEEE EEEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                                 //
// Unit 32, Consett Business Park                                             //
// Villa Real, Consett                                                        //
// Co. Durham                                                                 //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  startup.c                                                    //
//  Author    :  T. Mudryk                                                    //
//  Compiler  :  CodeWarrior v5.2                                             //
//  CPU       :  MC68HC908LJ12                                                //
////////////////////  File Contents  ////////////////////////////////////////
//  startup routines                                                          //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/03 tm      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#include    "extern.h"
#endif

#include    "Display.h"
#include    "button.h"
#include    "startup.h"


////////////////////////////////////////////////////////////////////////////
// This function is the first ATEECC function called out of reset, after the  //
// Metrowerks startup code.                                                   //
//                                                                            //
// A basic initialisation takes place as we enter STOP mode after this        //
// function awaiting a button press.                                          //
//                                                                            //
// Args   : none                                                              //
// Returns: none                                                              //
////////////////////////////////////////////////////////////////////////////
void MicroStartUp( void )
{
```

Software Routines

```
volatile union uPORT*ptr = &LDAT1;


CONFIG1.reg = 0xe2;                              // (2^13 - 2^4)/47^3 (ICLK) = 174ms
                                                 // watchdog, LVI reset disabled
ServiceWatchDog();
CONFIG2.reg = 0x2e;                              // XTAL will still run in STOP()
                                                 // 5V LVI trip, Enables Port C...
                                                 // to drive LCD (FP19 - FP23)
INTSCR.reg  = 0x02;                              // IRQ interrupts disabled
SEI();                                           // re-affirm

InitialisePLL();

// LVI setup
LVISR.bit.LVIIAK = 1;                    // Clears Interrupt Acknowledge bit
LVISR.bit.LVIIE  = 1;                    // Enables LVI interrupt
LVISR.bit.LVIIAK = 1;                    // Clears Interrupt Acknowledge bit

////////////////////////////////////////////////////////////////////////////
// Motorola data book MC68HC908LJ12 Rev. 2.0 page 314, states :               //
// "the ADC clock should be set to between 32kHz and 2MHz". Aiming for 1MHz.  //
// Therefore divide by 2, since using a 8.00MHz Xtal the bus speed will be    //
// 2MHz. Check this for final pll frequency and adjust accordingly.           //
// Using RIGHT JUSTIFIED mode to store raw unsigned data.                     //
////////////////////////////////////////////////////////////////////////////

// A2D setup
ADCLK.reg = 0x34;

// RTC setup
RTCCR2.reg      = 0x60;                          // rtc off, set for 32.768kHz xtal...
                                                 // chronograph setup
RTCCR1.reg      = 0x40;                          // CHRIE set
RTCCR2.bit.RTCE = 1;                             // ...all done, start RTC

////////////////////
// micro reg setup //
////////////////////
DDRA.byte           = 0x00;                      // Sets button pins to be inputs
DDRD.byte          &= 0x0f;
DDRD.bit.bit0       = 1;                         // PTD.bit.bit0 is set to output, ...
PTD.bit.bit0        = 0;                         // low for button release on wakeup
KBSCR.bit.IMASK     = 1;                         // Masks keyboard interrups
KBIER.byte          = 0xff;
KBSCR.bit.ACKK      = 1;
// Activates keyboard interrups and pull up resistors
DDRB.byte           = 0xff;                      // Sets port B to output

////////////////////////////////
// user variable initialisation //
////////////////////////////////
_5sectick           = 5;
PR2                 = 0;
user_temperature    = 20;
```

```
Override            = 0;
PRNumber            = NO_PROGRAM;
_10ms               = 0;
Aset_mode           = 0;
Cset_mode           = 0;
set_mode            = MODE_DEFAULT;
button_press_status = NO_BUTTON_PRESS;
SMin                = 60;
ALHR.reg            = 23;                       // Prevents Alarm being activated...
ALMR.reg            = 59;                       // accidentaly in the first minute...
                                                // of power up
////////////////////
// Display Startup //
////////////////////
LCDCLK.byte = 0x29;
//          = 00101001;
//             |||||||
//             |||||||___ Base Clock/Frame Rate
//             ||||||____   "    "     "     "
//             |||||_____   "    "     "     "
//             ||||______ LCD Duty Selection
//             |||_______  "   "       "
//             ||________ Fast Charge
//             |_________  "     "

LCDCR.reg   = 0xbc;
//          = 10111100;
//            ||||||||
//            ||||||||___ Contrast Controls
//            |||||||____   "         "
//            ||||||_____   "         "
//            |||||______   "         "
//            ||||_______ Low Current
//            |||________ Fast Charge
//            ||_________
//            |__________ LCD Enable (1 = enabled)

// displays all segments, (NOTE: ptr initialised in declaration)
for ( ; ptr <= &LDAT12; ptr++ )
   {
   ptr->reg = 0xff;
   }

InitialDisplay();

//////////////////
// Interputs on //
//////////////////
CLI();
} // MicroStartUp()
//-------------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// This function initialises the onboard pll to provide a bus frequency of    //
```

Software Routines

```
// 2.0MHz using a 32.768kHz xtal.                                              //
//                                                                             //
// Args   : none                                                               //
// Returns: none                                                               //
/////////////////////////////////////////////////////////////////////////////////
void InitialisePLL( void )
{
                          ////////////////////////////////////////////
PBWC.reg       = 0x80;    // auto mode                                //
PTCL.reg       = 0x00;    // settings here...                         //
PMS            = 0x00F5;  // as described in...                       //
PMRS.reg       = 0xD1;    // the MC68HC908LJ12                        //
PMDS.reg       = 0x01;    // Rev2.0 data book section 8.4.6 page 113 //
PTCL.bit.PLLON = 1;       // turn pll on after settings 'set'         //
                          ////////////////////////////////////////////

////////////////////////////////////////////////
// wait for the required frequency to be reached //
////////////////////////////////////////////////
ServiceWatchDog();
while ( !PBWC.bit.LOCK );

PTCL.bit.BCS = 1;         // pll clock drives CGMOUT
} // InitialisePLL()
//---------------------------------------------------------------------------
```

A.28 [startup.h]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  startup.h                                                  //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Header file for 'startup.c'                                             //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __STARTUP_H_
#define     __STARTUP_H_


///////////////
// prototypes //
///////////////
void MicroStartUp( void );
void InitialisePLL( void );

#endif
```

A.29 [temperature.c]

```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE        EEE          CC         CC         //
//     AAAAAA        TTTT     EEEEE      EEEEE        CC         CC         //
//    AAAAAAAA       TTTT     EEEEE      EEEEE        CC         CC         //
//   AAAA  AAAA      TTTT     EEE        EEE          CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Co. Durham                                                               //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                  //
//  Filename  :  temperature.c                                              //
//  Author    :  T. Mudryk                                                  //
//  Compiler  :  CodeWarrior v5.2                                           //
//  CPU       :  MC68HC908LJ12                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  Temperature Operations                                                  //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  17/07/03 tm      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#include    "extern.h"
#endif

#include    "Display.h"
#include    "i2c.h"
#include    "a2d.h"


void TemperatureView( void )
{
ClearDisplay();
LDAT12.byte = 0x33;                             // Displays "P"
LDAT11.byte = 0x10;                             //
if ( sign )
   {
   LDAT8.byte     = 0x03;                       // Displays "E" in digit 5
   LDAT7.byte     = 0x70;                       //
   LDAT6.bit.bit1 = 1;                          // Displays "-"
   }
else  LDAT6.byte  = 0x37;
```

```
DisplayNumber( external_temperature_decimal, 1 );
DP = 1;                                  // Display inside temperature decimal
DisplayNumber(( unsigned char )(external_temperature%10 ), 2 );
DP = 0;

if ( external_temperature >= 10 )// Clears 'tens' digit when not needed
   {                                     //
   DisplayNumber(( unsigned char )( external_temperature/10 ), 3 );
   }

DisplayNumber( (unsigned char)(user_temperature/10), 7 );
DisplayNumber( (unsigned char)(user_temperature%10), 6);

if ( smart )LDAT2.bit.bit2 = 1;                  // Displays smart chevron if needed
else                           LDAT2.bit.bit2 = 0;
Refresh = 1;                                     // Refresh the inside temp
} // TemperatureView()
//---------------------------------------------------------------------------


void InsideTemperature( void )
{
unsigned char temp, temp2;


if ( _5sectick >= 5 )
   {
   Refresh = 1;
   5sectick = 0;
   ExternalTemperatureRead();

   ////////////////////////////////////////////////////////////////////////
   // The internal temperature calculations:                             //
   //                                                                    //
   // The A2D byte is multiplied by 4.8V (ref V) to convert it to 10000 * the //
   // i/p voltage, it is then / by 100 to convert it to 100 * the i/p voltage //
   //                                                                    //
   // The result is then * by 40 which is the desired maximum temperature,    //
   // and / by 5 which is the maximum input voltage from the opamp.      //
   //                                                                    //
   // Finally the result is / by 10 to leave it in a manageable form     //
   ////////////////////////////////////////////////////////////////////////

   internal_temperature = ( 48 * ReadA2D( CHANNEL0 ))/100;
   internal_temperature = ( internal_temperature * 40 )/( 5 * 10 );

   BatteryCheck = ReadA2D( CHANNEL2 );
   }

if ( Refresh )
   {
   Refresh = 0;
   temp  = ( unsigned char )( internal_temperature/100 );
   temp2 = ( unsigned char )( internal_temperature - ( 100 * temp ));
```

Software Routines

```
   if ( temp )                                    // If number is >= 10
      {
      DisplayNumber(( unsigned char )( temp ), 3 );
      }                                           // Tens displayed
   else
      {
      LDAT4.byte  = 0x00;                         // Clear tens number
      LDAT5.byte &= 0xf0;                         //
      }

   DP = 1;
   DisplayNumber(( unsigned char )( temp2/10 ), 2 );
   DP = 0;                                        // Units displayed with decimal point
   DisplayNumber(( unsigned char )( temp2%10 ), 1);
   if ( smart )LDAT2.bit.bit2 = 1;     // Displays smart chevron if needed
   else                          LDAT2.bit.bit2 = 0;
   ChevronRefresh();
   }
} // InsideTemperature()
//---------------------------------------------------------------------------
```

A.30 [temperature.h]

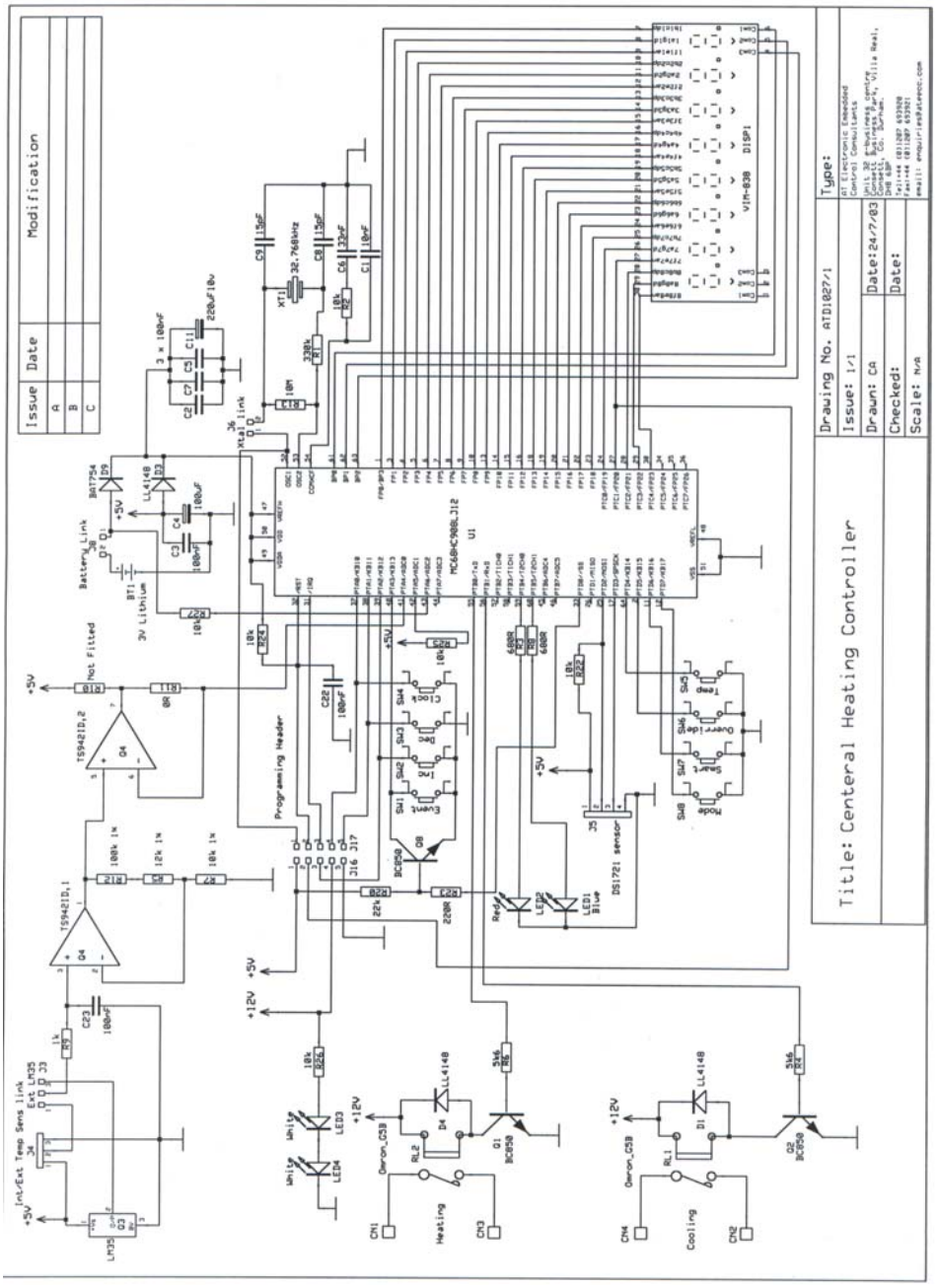
```
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTT EEEEEEEEEEE EEEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
//      AAAA     TTTTTTTTTTT EEE         EEE           CC         CC         //
//     AAAAAA        TTTT     EEEEE       EEEEE         CC         CC         //
//    AAAAAAAA       TTTT     EEEEE       EEEEE         CC         CC         //
//   AAAA  AAAA      TTTT     EEE         EEE           CC         CC         //
//  AAAA    AAAA     TTTT     EEEEEEEEEEE EEEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                                 //
// Unit 32, Consett Business Park                                             //
// Villa Real, Consett                                                        //
// Co. Durham                                                                 //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Central Heating Reference Design                    //
//  Filename  :  temperature.h                                                //
//  Author    :  T. Mudryk                                                    //
//  Compiler  :  CodeWarrior v5.2                                             //
//  CPU       :  MC68HC908LJ12                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  Header file for 'temperature.c'                                           //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  17/07/03 tm      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __TEMPERATURE_H_
#define     __TEMPERATURE_H_


void InsideTemperature( void );
void TemperatureView( void );

#endif
```

Software Routines

Appendix B. Main Circuit Diagram



Main Circuit Diagram

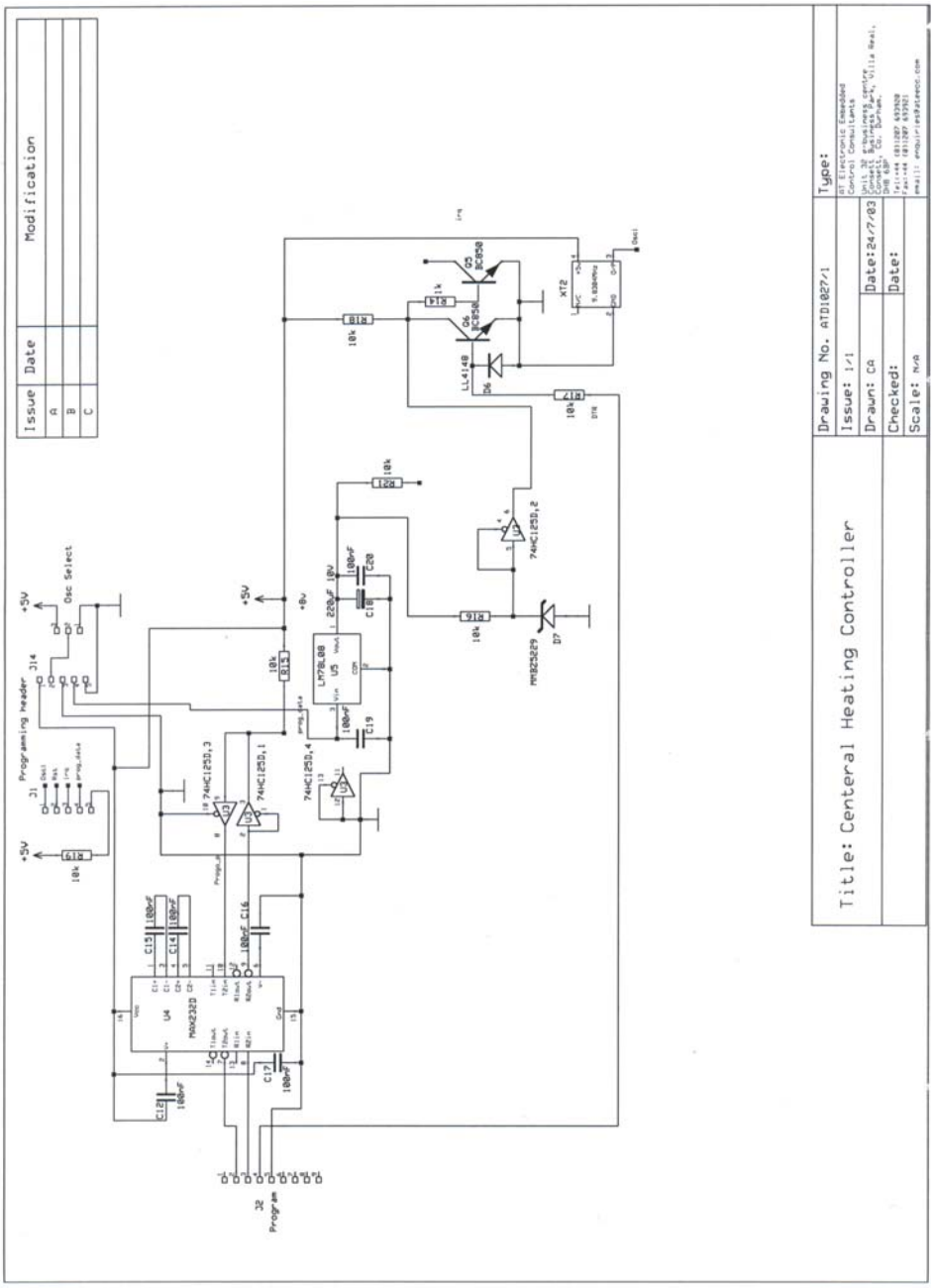
Appendix C. Main Circuit Bill of Materials

Resistors		
R1	330k +/-5%	1206
R2	10k+/-5%	1206
R3	680R+/-5%	1206
R4	5k6 +/-5%	1206
R5	12k +/-1%	1206
R6	5k6 +/-5%	1206
R7	10k +/-1%	1206
R8	680R +/-5%	1206
R9	1k +/-5%	1206
R10	0R (see text)	1206
R11	0R (see text)	1206
R12	100k +/-1%	1206
R13	10M +/-5%	1206
R20	22k +/-5%	1206
R22	10k +/-5%	1206
R23	220R +/-5%	1206
R24	10k +/-5%	1206
R25	10k +/-5%	1206
R26	10k +/-5%	1206
R27	10k +/-5%	1206
Capacitors		
C1	10nF 30V	1206
C2	100nF 30V	1206
C3	100nF 30V	1206
C4	100uF 10V	Radial
C5	100nF 30V	1206
C6	33nF 30V	1206
C7	100nF 30V	1206
C8	15pF 30V	1206
C9	15pF 30V	1206
C11	220uF 10v	Radial
C13	220uF 25v	Radial
C23	100nF 30V	1206
C22	100nF 30V	1206

Main Circuit Bill of Materials

Semiconductors		
D1	LL4148	MINI MELF
D3	LL4148	MINI MELF
D4	LL4148	MINI MELF
D5	LL4001	MELF
D8	LL4001	MELF
D9	BAT754	SOT-23
Q1	BC850	SOT-23
Q2	BC850	SOT-23
Q3	LM35CZ	TO92
Q4	TS942ID	SO8
Q7	DS1721	SO-8
Q8	BC850	SOT-23
U2	LM78L05	TO92
U1	MC68HC908LJ12	64pin QFP
LED1	Blue	3mm
LED2	Red	3mm
LED3	White	5mm
LED4	White	5mm
Miscellaneous		
XT1	32.768kHz	Radial
BT1	Coin Cell Battery Holder	
	3V Lithium	Coin
RL1	Omron_G5B 12VDC Coil	
RL2	Omron_G5B 12VDC Coil	
DISP1	Varitronix VIM-838	
PCB	FR4 PTH 130mm x 75mm	Main PCB
PCB	FR4 15mm x 25mm	Remote sensor
Connectors		
J3	3 -pin 0.1" pitch header	
J4	3-pin Molex kk connector	
J5	4-pin Molex kk connector	
J6	2-pin 0.1" pitch header	
J8	2-pin 0.1" pitch header	
J9	12VDC	
J15	4-pin Molex kk connector	
J16 - J17	10 -way 0.1" pitch box header	
CN1-CN4	0.625" Faston Blade	

Appendix D. Programming Circuit Diagram



Programming Circuit Diagram

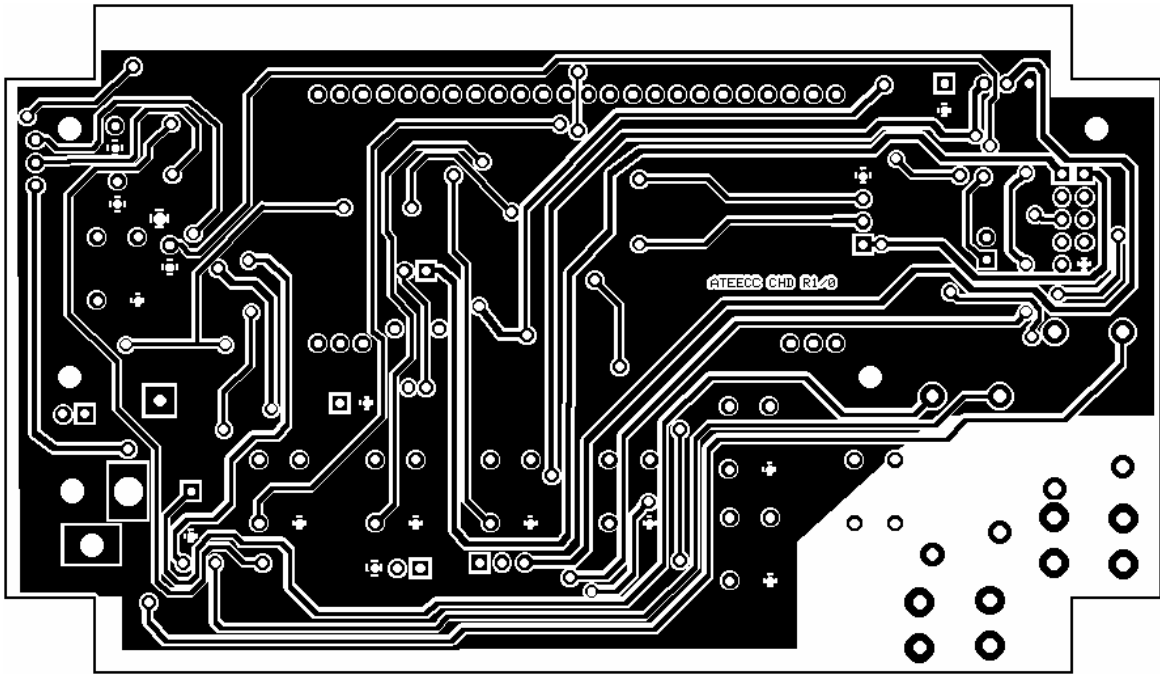
Appendix E. Programming Circuit Bill of Materials

Resistors		**Package**
R14	1k +/-5%	1206
R15	10k +/-5%	1206
R16	10k +/-5%	1206
R17	10k +/-5%	1206
R18	10k +/-5%	1206
R19	10k +/-5%	1206
R21	10k +/-5%	1206
Capacitors		
C12	100nF 30V	1206
C14	100nF 30V	1206
C15	100nF 30V	1206
C16	100nF 30V	1206
C17	100nF 30V	1206
C18	220µF 10V	Radial
C19	100nF 30V	1206
C20	100nF 30V	1206
C21	100nF 30V	1206
Semiconductors		
Q5	BC850	SOT23
Q6	BC850	SOT23
D6	LL4148	MINI MELF
D7	MMBZ5229	SOT23
U3	74HC125D	SO-14
U4	MAX232D	SO-16 Wide
U5	LM78L08	TO92
XT2	9.8304 MHz	Oscillator Module
Connectors		
J1, J7	10 -way 0.1" pitch box header	
J2	9-way 'D' type	Female
J14	2-pin 0.1" pitch header	

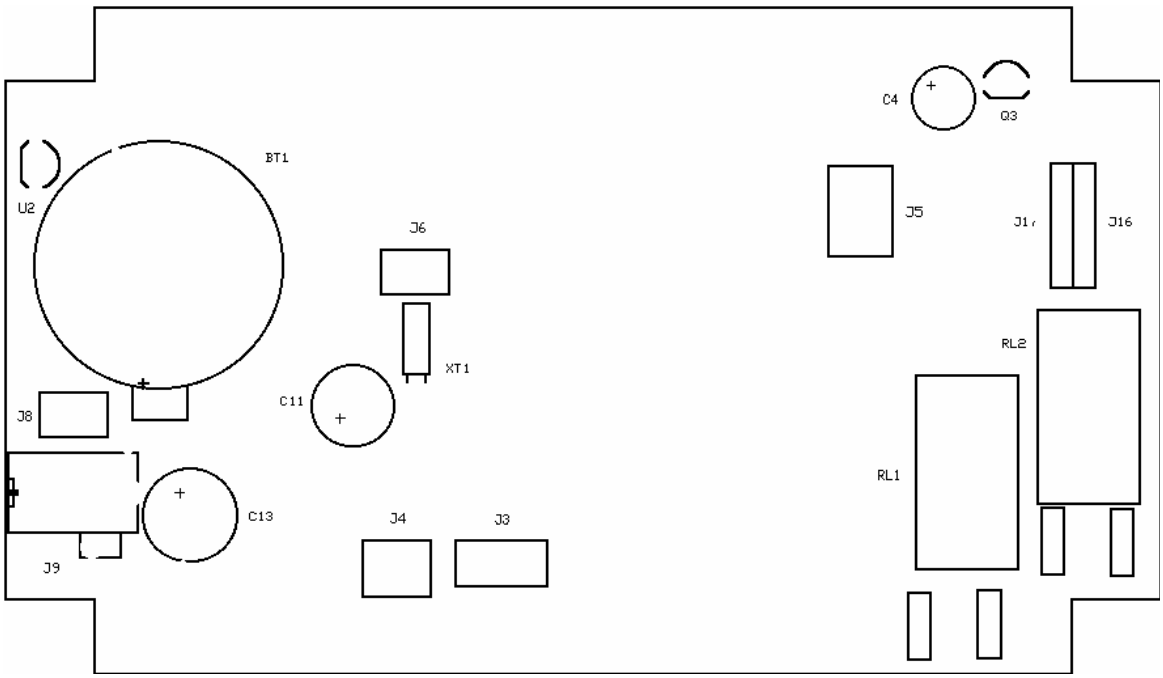
Programming Circuit Bill of Materials

Appendix F. PCB Details

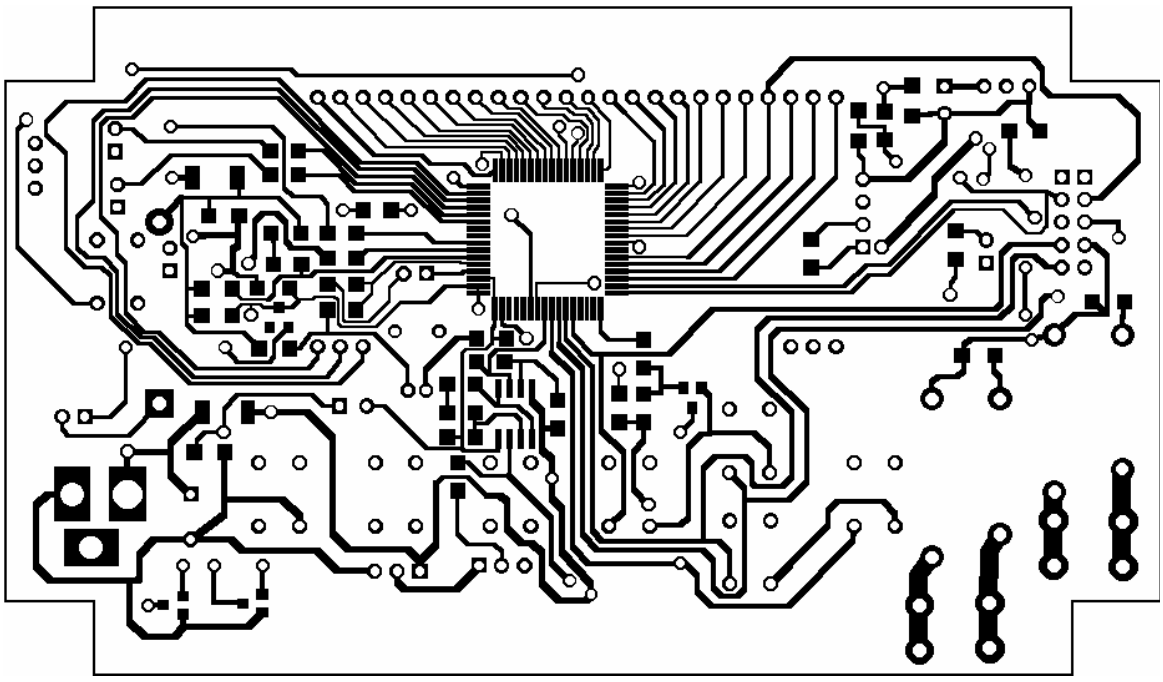
F.1 Component Trace



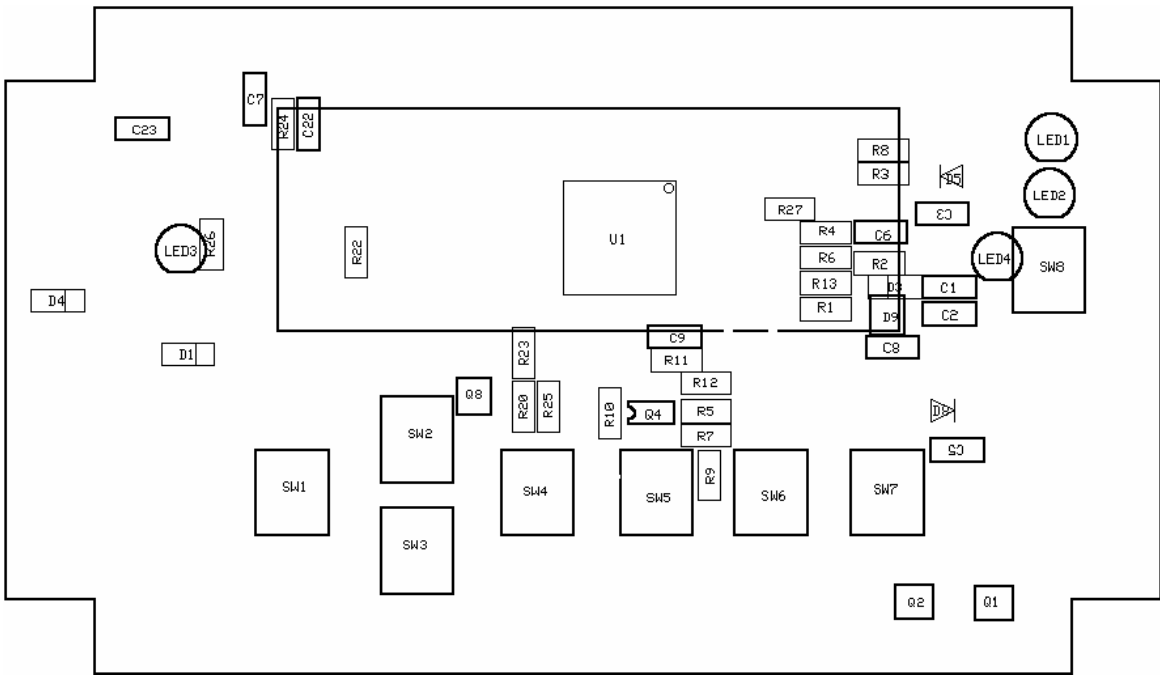
F.2 Component Silk Screen



F.3 Solder Trace



F.4 Solder Silk Screen



HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

DRM044

Freescale Semiconductor, Inc.