AN1853

Embedding Microcontrollers in Domestic Refrigeration Appliances

by William Mackay
Freescale Microcontroller Division
East Kilbride, Scotland.

1 Introduction

This Application Note describes and demonstrates the extensive capabilities of another comprehensive and economic 8 bit Microcontroller from the Freescale 68HC08 portfolio. The device is the HC908KX8, a very low cost, high performance 16-pin flash device with a user selectable Internal Oscillator and on-board Reset Circuitry. This document details how the HC908KX8 controls a domestic fridge appliance and implements control of the Fridge Compressor Induction Motor based on air temperature measurement, including some energy and cost saving features. The microcontroller and associated application hardware have been developed and embedded in a Fridge appliance, with application code written in 'C'.

Embedding a Freescale Microcontroller into any domestic appliance has numerous advantages, both through the development life cycle and production environment. A common hardware and software development platform can be established which can support a range of appliances for present and potential future needs.

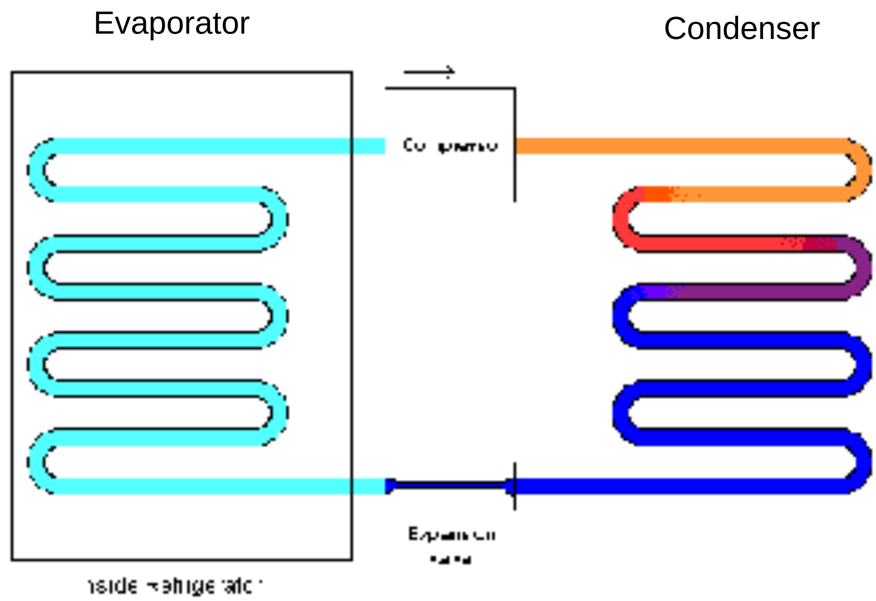
The programmability of the device provides a flexible software development environment that accommodates low-end through mid-range appliance model software versions, and the 8K of flash user space allows for future application functionality enhancements, along with the additional time saving and development advantages of the re-programmable flash technology. These attributes increase the convenience for planning future appliance developments, and in terms of hardware, standardisation of Printed Circuit Board design and manufacturing practices can be achieved with less risk and lower component count than discrete or Application Specific Integrated Circuit solutions. In short, development time is reduced, production costs are minimised and time-to-market place can be reduced significantly with better product flexibility and reliability.

Domestic appliances are the subjects of strict European regulations, with similar constraints imposed in the USA. These regulations result in demanding operational constraints on Refrigeration appliances. Predominately, the challenge is to improve the energy efficiency and electromagnetic compatibility (EMC), and enhance the marketable features of the appliance. The internal oscillator and on-board reset circuit features make for an improved EMC performance and better reliability in electrically noisy environments. With these enhancements, flash programmability, and adaptable feature set, Freescale continue to strive to meet global industry challenges with our leading system solutions.

2 Basic Refrigeration

The main electrical components required for a domestic refrigeration system are some means of temperature control and a Refrigerant Compressor.

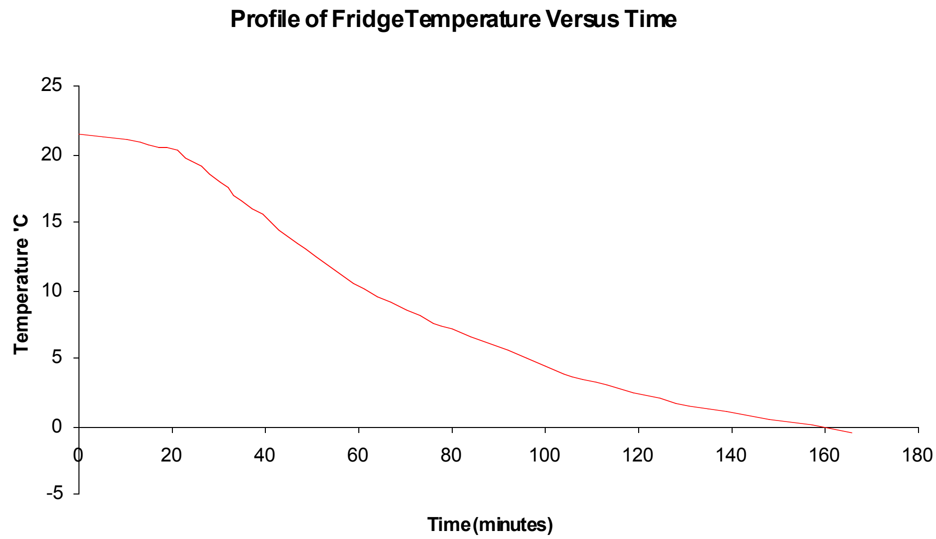
Embedded within a domestic Fridge compartment is an Evaporator, and on the outside a Condenser, heat exchanging coils and the refrigerant compressor. The compressor is driven by an electrical motor. When power is applied to the compressor the pressure of the refrigerant is increased. This increase in pressure causes an increase in refrigerant temperature and the heat produced by this action is dissipated through the heat exchanging coils at the rear of the appliance. This action is illustrated in the following diagram.



The refrigerant then condenses and passes through from the high-pressure environment of the condenser through an expansion valve to the low-pressure evaporation system inside the Fridge compartment. On evaporating, the refrigerant absorbs heat and subsequently reduces the enclosure temperature. The warmer refrigerant is circulated to the outside of the compartment where the cycle repeats under thermal control.

From initial power-on this cyclic cooling process can take some time to reach an acceptable operating temperature range, this is usually around 6 to 8°C. The following plot is an example of the behavior of the ambient air temperature within a domestic fridge compartment from power-on at

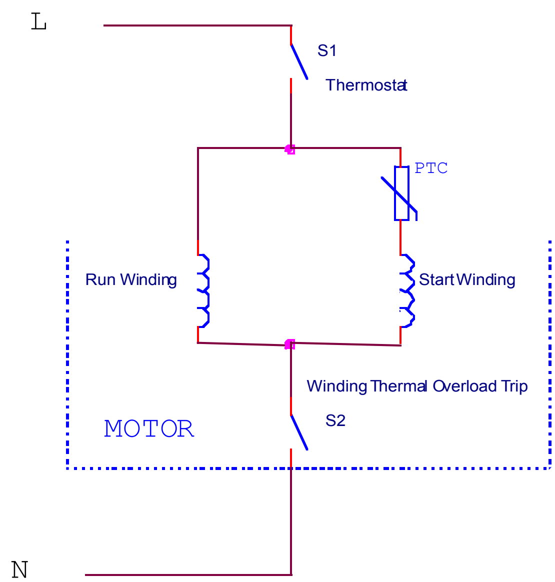
21°C to 0°C. From the graph it can be seen that it takes approximately 80 minutes to reach 7°C.



3 Conventional Fridge Control

In many domestic fridge appliances, the air temperature of the fridge compartment is controlled by a bi-metallic thermostat connected in series with a single-phase induction motor. The motor has two windings, a run winding and a start winding and a current limiting Positive Temperature Co-efficient Thermistor (PTC) in series with the start winding. It is also common practice to embed a thermal overload in the motor windings for protection in the event of overheating.For example, the thermal overload contact will open and remove power from thc motor in the event of overheating caused by a motor stall condition. The contact will then automatically reset to the closed condition when the windings return to their normal safe operating temperature.

The following diagram is a typical configuration for a domestic fridge appliance using a single-phase induction motor.

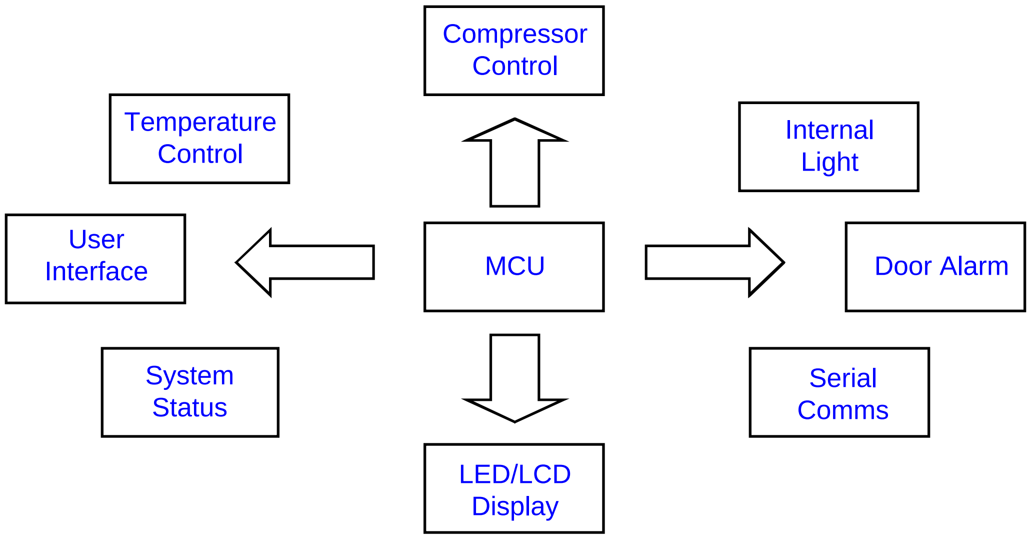


3.1 Operation

When the temperature of the fridge compartment rises above the pre-selected thermostat setting, the bi-metallic contact closes and line voltage is applied to both the start and run windings simultaneously. The start winding has a lower resistance than the run winding and provides the initial current surge required to start the motor. This inrush of current subsequently raises the temperature of the PTC and increases its resistive property, which in turn reduces the current flow to the start winding. At this point in time, the current through the start winding has been minimised by the PTC, the current through the run winding is stable and the motor continues to run. When the fridge compartment reaches the desired temperature the thermostat contact opens, removing power from the motor. When the compartment air temperature again rises, the temperature control cycle repeats.

4 The Microcontroller Solution

The HC908KX8 Microcontroller forms the heart of the refrigeration system by providing an adaptable platform for the required functionality for low-end through mid range appliances. Using the microcontroller in a refrigeration appliance can provide a system with various possibilities for developing improved efficiency and functionality. As the system is under software control, there is better scope for improving system efficiency with more accurate electronic temperature measurement and compressor control. This is complimented by additional functionality provided from the device feature set. A typical implementation follows.



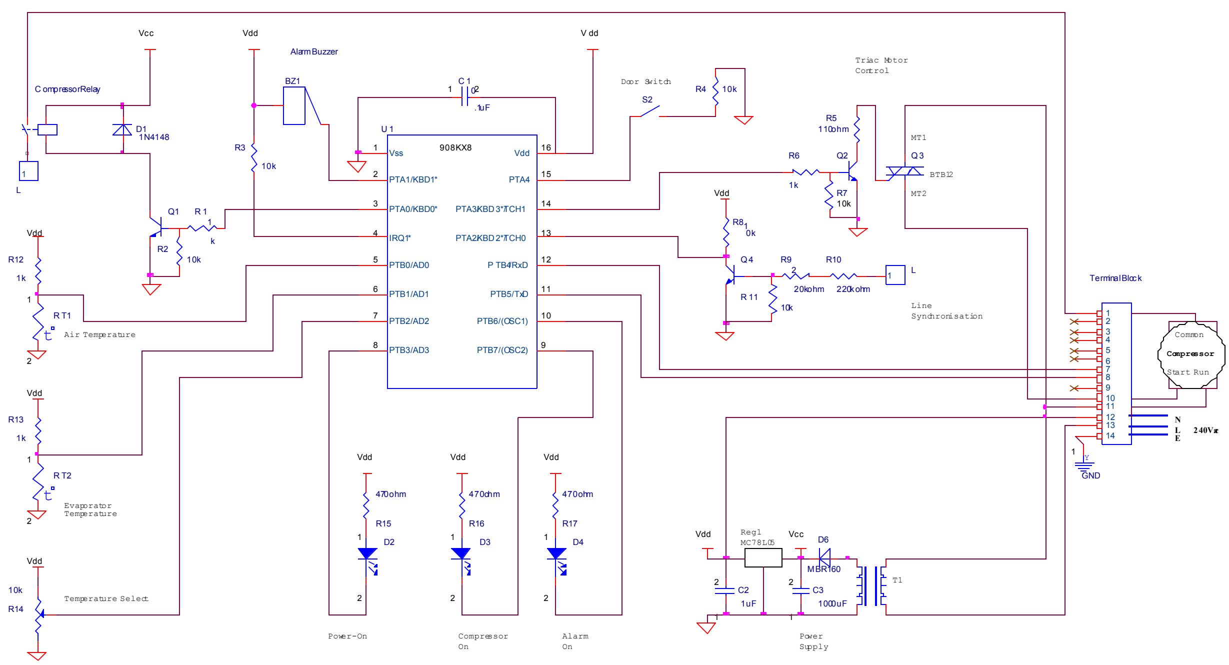
The main focus in this design is to implement a system solution that will control a domestic fridge compressor based on temperature measurement, with some additional functionality.

4.1 Hardware

The HC908KX8 Microcontroller feature set provides a number of dual and multifunction pins that provide convenient application adaptability. A number of the pins can be configured as general Input /Output, Analogue Inputs, Timer Input Capture and Output Compare, Keyboard Inputs and Serial Communications. There is also an external Oscillator configuration available, and on Port 'A' some 15mA-sink/source high current pins with software programmable pull-up resistors. A demonstration configuration used for the fridge application is shown on the following schematic diagram.

4.2 Refrigeration System Schematic Diagram

The feature set of the device accommodates the required functionality of a typical refrigeration system.

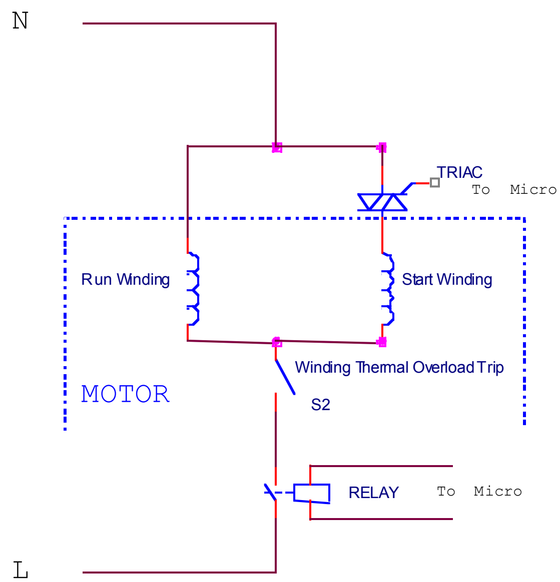


4.3 Refrigeration System Schematic Diagram Functional Overview

The temperature of the system is primarily dependent on three inputs. Two Negative Temperature Co-efficient Thermistors (NTCs), and a Potentiometer. One NTC is used to detect the fridge ambient air temperature and the other detects the Evaporator temperature. The potentiometer is used to select the desired ambient air temperature of the fridge. These inputs are connected to the microcontroller Analogue to Digital Converter Module. Other application features include an audible alarm, used to alert the user of a ‘door open’ condition or over and under temperature conditions, a door switch to determine the status of the door as either open or closed, and three system status LED’s – one to indicate power-on, another which indicates when the compressor is powered, and a third LED which is a visual indication of a ‘door open’, ‘over temperature’ or ‘under temperature’ alarm condition.

4.4 Compressor Control

In contrast to the previously described conventional compressor control case, a more efficient and long-term cost-effective solution can be implemented using the microcontroller to control the compressor motor using a triac and a relay. The PTC that was previously connected in series with the start winding is no longer required, as shown in the following diagram.



The sequence of events required to start the Induction motor is now under software control. The relay is energised and closes the contact to apply line voltage to the start and run windings simultaneously. The triac is fired on the first zero crossing point of the line voltage and every successive zero cross-detected for a period of 40mS.

In normal operation, after this time the compressor should have started and the Triac will be in the non-conducting state until another start sequence is invoked. The compressor remains powered and running whilst the relay remains energised. With the PTC out of circuit some cost and a few Watts of power can be saved.

4.5 Line Voltage Zero Crossover Detection

Ideally, the motor start winding should be energised at the zero crossing points of the applied sinusoidal line voltage waveform. The zero crossover detection circuitry is required to enable the microcontroller to detect these points so that the Triac can be fired at the appropriate time, since at zero current, the triac naturally switches off. This technique has the added advantage of minimising switching transient generation and electromagnetic radiation.

The zero crossover detection function is implemented using one of the timer input capture features of the device. This input can generate a CPU interrupt when a rising or falling edge is detected on the input capture pin. The line voltage is connected to a potential divider and drives the base of a Transistor which switches rising and falling edges between 0V and Vdd in synchronization with the line voltage zero crossover points. When a rising or falling edge is detected, the triac is controlled from the Input Capture Interrupt Service routine.

4.6 Triac Drive Circuitry

The triac in this application is used to apply power to the start winding of the motor during the start-up phase, and to remove power from this winding once the motor has started. Interfacing the microcontroller to a triac can be achieved in various ways. The main objective is to connect the microcontroller to the AC line voltage in order to provide a common electrical reference point between the AC line and the microcontroller for the application of the triac gate pulses. In this application, a positive ground system is used, which requires that the microcontroller Vdd terminal be connected to the AC Neutral Terminal

4.7 Compressor Power Relay

The purpose of this relay is to apply power to the compressor motor start and run windings at power-on, and maintain power to the motor run winding after the start-up phase has expired. Additionally, as the relay contact has a low resistance, it does not dissipate power unnecessarily under normal running conditions.

4.8 Power Supply

This is a conventional arrangement. The AC supply is rectified smoothed and then passed through a linear regulator to provide the DC power supply for the system. The 12V power supply for the relay is unregulated and is shown as Vcc in the schematic diagram.

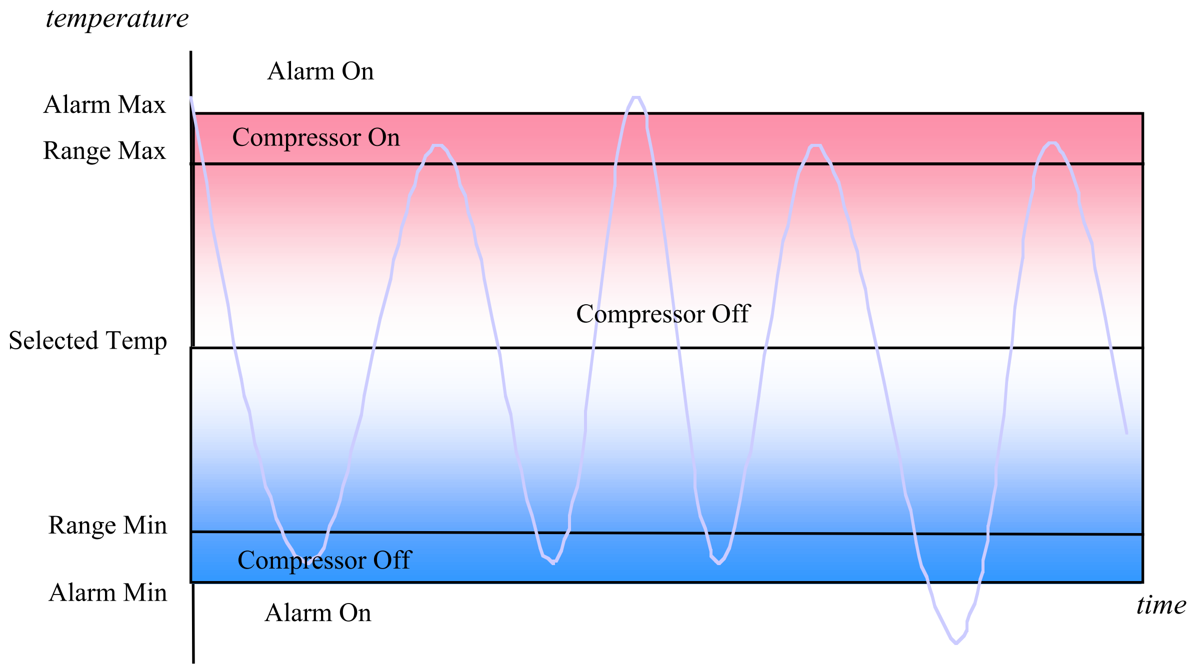
4.9 Software

This section discusses a simple temperature control algorithm accompanied by flowcharts and a ‘C’ code implementation.

4.10 Temperature Control Algorithm

Consider the temperature control profile of the fridge as being divided into three operating bands. These are the normal operating ranges, over which the air temperature of the fridge is deemed as being acceptable, that is between ‘range max’ and ‘range min’. The upper and lower alarm levels, ‘alarm max’ and ‘alarm min’ are used to constrain over and under temperature conditions. This is shown in the following temperature profile illustration.

Air Temperature Profile



4.11 Normal Operating Temperature Range

From initial power-on, the ambient air temperature of the fridge is typically around +21°C. The user selected temperature is typically set around +5°C, and may cycle between +8 and +2°C. At power-on, the temperature is within the upper alarm region, however, this is not an alarm condition therefore, the audible alarm will be disabled until the temperature is driven below the 'range max' value of +8°C for the first time. The Compressor is powered and over a period of time drives the air temperature down through the 'range max' value and through the 'selected temp' to the 'range min' value of +2°C. When the air temperature reaches this value, the compressor is powered-down. From this point, the air temperature will gradually rise up through the selected temperature and eventually reach the 'range max' value again. At this point in time, the compressor will again be powered until the air temperature is driven to the 'range min' value and the normal temperature control cycle repeats. The cycle repetition rate is predominately dependant on thermal insulation quality and the frequency of door opening. Typically there is a 50% compressor On/Off work rate.

4.12 Alarm Conditions

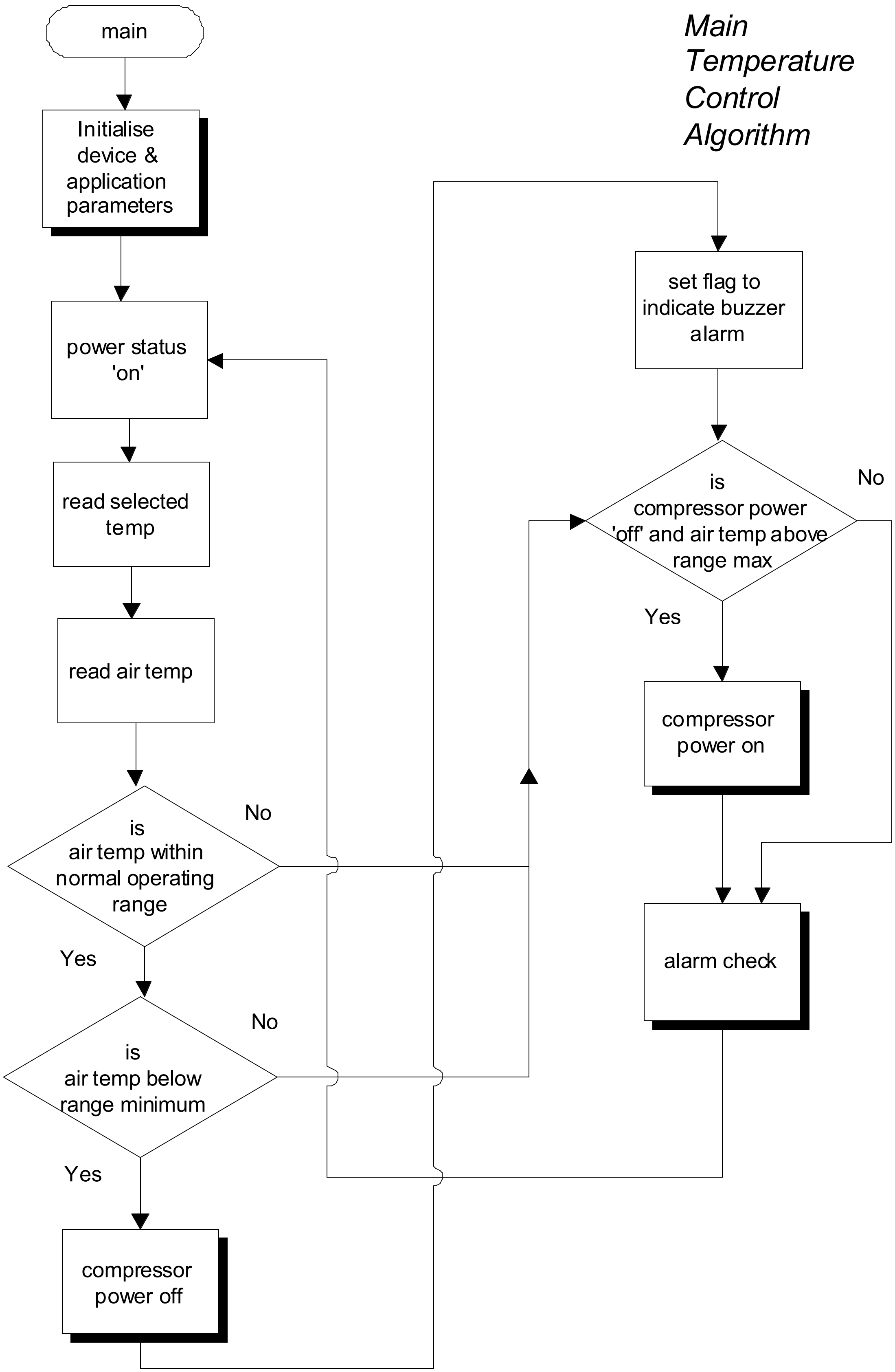
There are three possible alarm situations. These are, door open, over, or under temperature conditions. If the door has inadvertently been opened for a pre-determined time, for example, one minute, an audible alarm will be sounded.

The over temperature situation can occur if the fridge door is not closed properly or there is compressor failure or a refrigerant pressure problem. The under temperature case may occur if the compressor is permanently powered-on. In all of these situations the objective is to alert the user. After initial power-on, and when the fridge compartment temperature has stabilised, an audible alarm will be sounded if the air temperature reaches the ‘alarm max’ or ‘alarm min’ value. An LED also provides a visual indication of a door open, over, or under temperature condition.

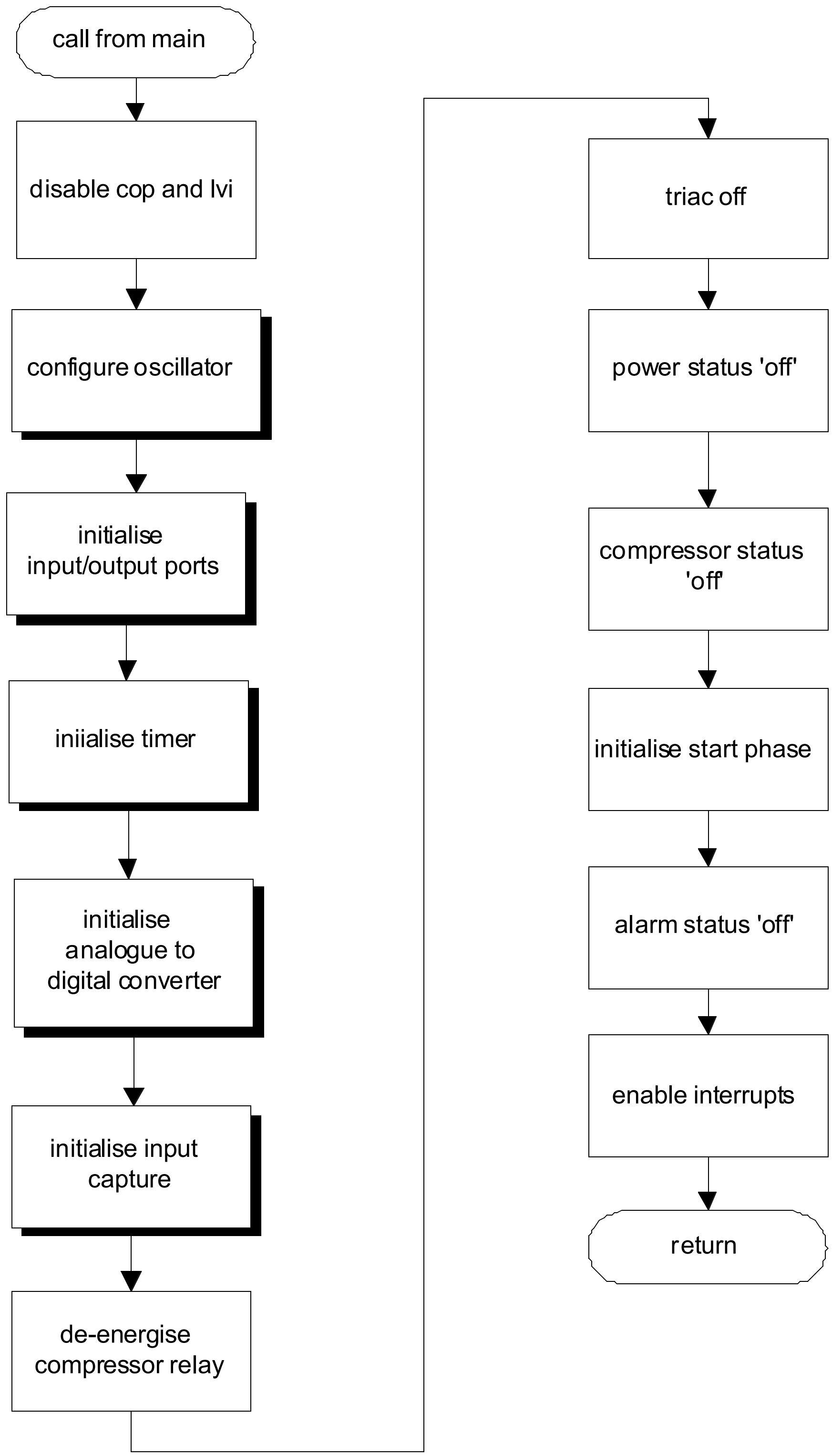
4.13 Flowcharts

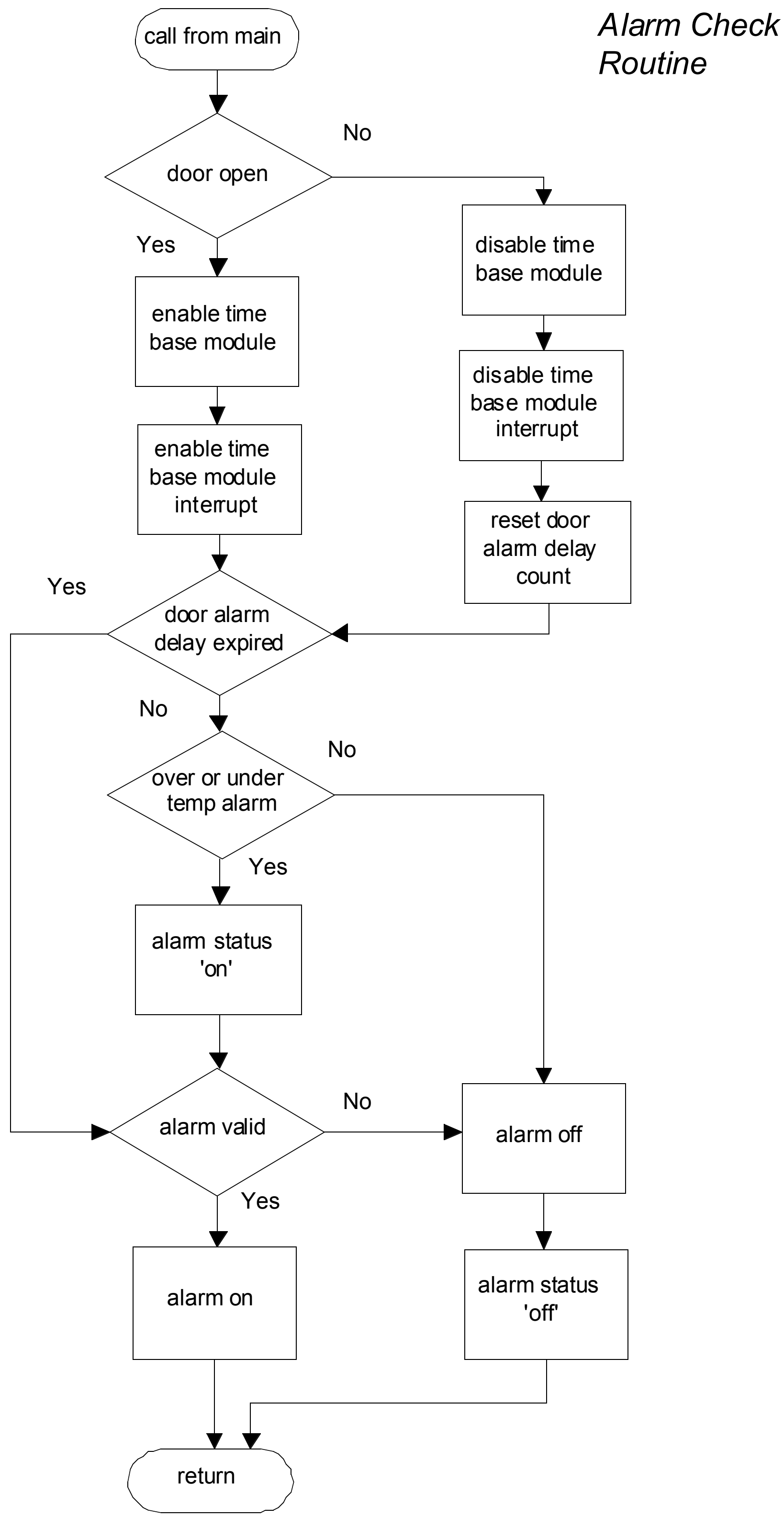
There are four main software modules used to implement the control algorithm, one is the initialisation routine which configures the microcontroller and application parameters. Secondly, the ‘main’ temperature control routine which manages the application functionality, and two interrupt service routines. There is an input capture interrupt service routine used primarily to manage zero crossover detection and a time base module interrupt routine which is used to provide a time reference for the ‘door open’ alarm delay. The following flowcharts illustrate the control flow of the application. Shadowed boxes shown in the charts indicate nested functions within the code.

Main Temperature Control Algorithm

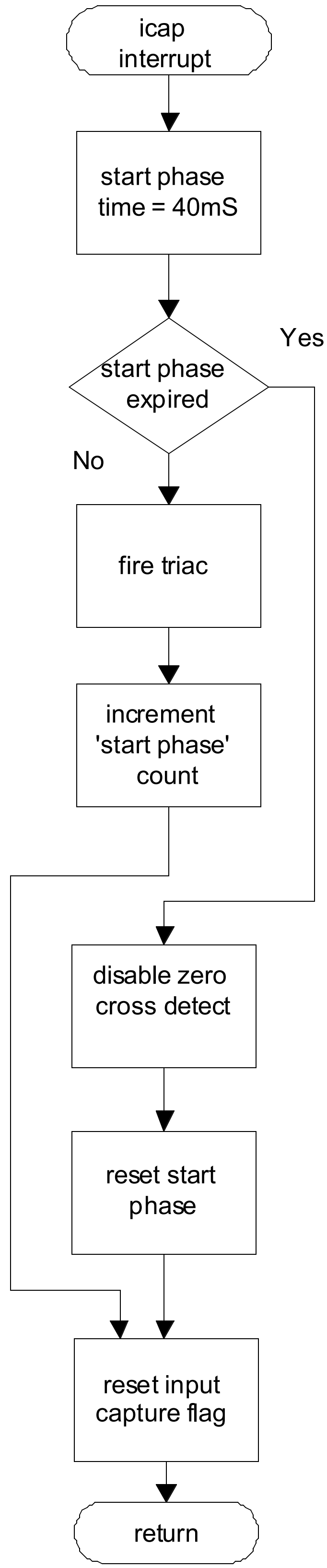


Initialisation Routine

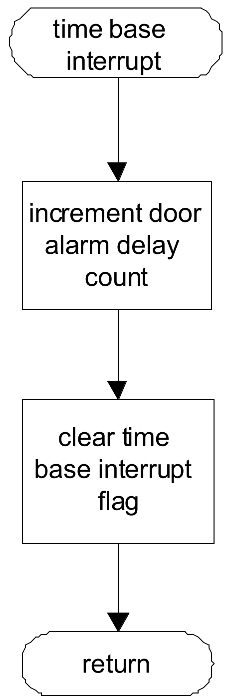



Alarm Check Routine
call from main
door open
No
Yes
disable time base module
enable time base module
disable time base module interrupt
enable time base module interrupt
reset door alarm delay count
Yes
door alarm delay expired
No
No
over or under temp alarm
Yes
alarm status 'on'
No
alarm valid
alarm off
Yes
alarm on
alarm status 'off'
return

Input Capture
Interrupt Routine



Time Base Module Interrupt Routine



4.14 Summary

Basic conventional fridge control typically uses a bi-metallic thermostat to control the fridge temperature. The thermostat simply applies power to the compressor based on a course mechanical temperature setting. This method has been used in the past for some time and is now rapidly becoming dated, due mainly to the increasing demand for more efficient appliances and for increased user functionality.

The suggested functionality shown in this application example is basic and simple to implement, however, there are many additional features that can enhance the system. For example, the evaporator temperature measurement can be included as part of the temperature control cycle. The Serial Communications Interface can be used to communicate to an LCD display, local bus based system or internet gateway for remote diagnostic or control purposes. Electronic control of the internal light can be accommodated using a Triac to ramp-up and dim the compartment illumination. These features can enhance the marketability of the appliance considerably.

An additional safety feature can be added which can detect motor rotation. This can be implemented by using both input captures and some additional software to detect and measure the phase difference between the start and run windings during normal running conditions and a motor stall condition.

The efficiency of the system can be further improved by making use of the microcontroller power saving ‘wait mode’ feature with the analogue to digital converter interrupt. Wait mode can be invoked when ‘range min’ temperature is reached, hence, taking advantage of the Fridge compartment long temperature rise time constant and optimising power conservation. Program execution can continue from the ADC interrupt routine some time later when the air temperature rises to the ‘range max’ value.

In conclusion, the HC908KX8 microcontroller has an excellent level of adaptability at very low cost, with the added advantage of having the system under software control, and the ability to program or re-program in an ‘8k flash’.

4.15 Code Implementation.

The source code to implement the functionality described in the preceding flowcharts follows, including header files

4.15.1 Main Temperature Control Routine.

```
#include "hc08kx6.h"        /* generic hc08kx6 header file*/
#include "Fridge.h“         /* application header file */

/*******************************************************************************
                                              Copyright (c)

Function Name :      main()

Engineer :           William Mackay

Location :           Freescale Microcontroller Division, East Kilbride

Date Created :       December 1999

Current Revision :   0.0

Note :               Main temperature control routine

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

```
void main(void)
{
    init();
    while(1)
    {
        POWER_STATUS = ON;                                  /* power led */
        selected_temp = single_adc(AD2);                    /* read user selected temp */
        air_temp = single_adc(AD0);                         /* read compartment air temp */
        if(((air_temp) <= (selected_temp+AIR_MAX_TEMP))&&   /* within correct temp range */
        ((air_temp) >= (selected_temp-AIR_MIN_TEMP))||
        ((air_temp) <= (selected_temp-AIR_MIN_TEMP)))       /* or below range minimum */
        {
            compressor_off();
            alarm_valid = SET;                              /* flag to indicate buzzer can be sounded...*/
                                                            /*…when an alarm condition is detected */
        }
        else
        {
            if((!COMPRESSOR_POWER)&&                        /* compressor relay de-energised */
            ((air_temp) >= (selected_temp+AIR_MAX_TEMP)))   /* temp above range max */
            {
                    compressor_on();
            }
        }
        alarm_check();
    }
}
```

4.15.2 Initialisation Routine.

```
/*********************************************************************************
                                             Copyright (c)
Function Name :      init()

Engineer :           William Mackay

Location :           Freescale Microcontroller Division, East Kilbride

Date Created :       December 1999

Current Revision :   0.0

Note :               Function configures oscillator, device modules
                     and initialises application parameters

*********************************************************************************/

void init(void)
{
        CONFIG1=0x31;                                    /* disables lvi and cop */
        Init_osc();                                      /* sets oscillator frequency */
        init_ports();                                    /* configure input/output ports */
        init_timer();                                    /* initialise timer */
        init_time_base();                                /* initialise time base module */
        init_adc();                                      /* initialise analogue to digital converter */
        init_icap();                                     /* configure input capture pin */
        COMPRESSOR_POWER = DISABLED;                     /* compressor relay de-energised */
        TRIAC_DRIVE = DISABLED;                          /* triac off */
        POWER_STATUS = OFF;                              /* red led */
        COMPRESSOR_STATUS = OFF;                         /* green led */
        ALARM_STATUS = OFF;                              /* yellow led */
        ENABLE_INTERRUPTS;                               /* enable interrupts */
}
```

4.15.3 Analogue to Digital Conversion Routine.

```
/***************************************************************************
                                             Copyright (c)
Function Name :      single_adc()

Engineer :           William Mackay

Location :           Freescale Microcontroller Division, East Kilbride

Date Created :       December 1999

Current Revision :   0.0

Notes :              Performs a single analogue to digital conversion

****************************************************************************/


unsigned char single_adc(unsigned char channel_number)
{
        START_CONVERSION = channel_number;
        delay();
        if(CONVERSION_COMPLETE)
        {
            return(ADC_VALUE);
        }
}
```

4.15.4 Compressor ‘off’ Routine.

```
/***************************************************************************
                                             Copyright (c)
Function Name :      compressor_off()

Engineer :           William Mackay

Location :           Freescale Microcontroller Division, East Kilbride

Date Created :       December 1999

Current Revision :   0.0

Note :               Function powers-down compressor

****************************************************************************/

void compressor_off(void)
{
        COMPRESSOR_POWER  = DISABLED;                    /* compressor relay de-energised */
        ZERO_CROSS_DETECT  =  DISABLED;                  /* disable icap interrupt */
        COMPRESSOR_STATUS = OFF;                         /* compressor led */
}
```

4.15.5 Compressor ‘on’ Routine.

```
/*******************************************************************************
                                              Copyright (c)
Function Name :      compressor_on()

Engineer :           William Mackay

Location :           Freescale Microcontroller Division, East Kilbride

Date Created :       December 1999

Current Revision :   0.0

Note :               Function powers-on compressor

*******************************************************************************/

void compressor_on(void)
{
        COMPRESSOR_POWER  = ENABLED;                       /* compressor relay energised */
        ZERO_CROSS_DETECT = ENABLED;                       /* start compressor at next zero cross */
        COMPRESSOR_STATUS = ON;
}
```

4.15.6 Alarm check Routine.

```
/*******************************************************************************
                                              Copyright (c)
Function Name :      alarm_check()

Engineer :           William Mackay

Location :           Freescale Microcontroller Division, East KilbrideDate Created: December 1999

Current Revision :   0.0

Note :               Function checks for, door open, over
                     & under temperature alarm conditions

*******************************************************************************/

void alarm_check(void)
{
        if(DOOR_OPEN)
        {
            TBCR_TBON = SET;                               /* enable time base module */
            TBCR_TBIE   = ENABLED;                         /* enable time base interrupt */
    }
    else
    {
            TBCR_TBON =  RESET;                            /* disable time base and reset counter to zero
*/
            TBCR_TBIE   =  DISABLED;                       /* disable time base interrupt */
            door_alarm_delay = RESET;
        }
```

```
        if((door_alarm_delay >= ONE_MINUTE)||              /* door opened for one minute or greater */
        (air_temp >= ALARM_TEMP_MAX)||                     /* or over temp alarm  */
        (air_temp <= ALARM_TEMP_MIN))                      /* or under temp alarm */
        {
            ALARM_STATUS = ON;                             /* alarm led */
            if(alarm_valid)                                /* alarm condition is valid */
            {
                    BUZZER = ON;                           /* audible alarm */
            }
        }
        else
        {
            BUZZER = OFF;
            ALARM_STATUS = OFF;
        }
}
```

4.15.7 Input Capture Interrupt Routine.

```
/******************************************************************************
                                              Copyright (c)
Function Name :      input_capture()

Engineer :           William Mackay

Location :           Freescale Microcontroller Division, East Kilbride

Date Created :       December 1999

Current Revision :   Preliminary

Note :               This ISR pulses the triac on line voltage zero-cross
                     detection for a pre-defined motor start period

******************************************************************************/


#pragma TRAP_PROC SAVE_REGS
void input_capture()
{
    if(start_phase != START_TIME)                          /* start phase valid */
        {
            TRIAC_DRIVE  =  OFF;                            /* apply pulse to triac */
            delay();
            TRIAC_DRIVE  =  ON;
            ++start_phase;                                 /* start phase is a count of the  */
        }                                                  /* line voltage zero cross points */
        else
        {                                                  /* start time has expired */
            ZERO_CROSS_DETECT = DISABLED;                  /* disable input capture interrupt */
            start_phase = RESET;                           /* reset start phase */
        }
        read_register = TSC0                               /* reads TIM status and control register */
        ICAP_FLAG = RESET;                                 /* resets CH0F flag */
}
```

4.15.8 Time Base Module Interrupt Routine.

```
/*****************************************************************************
                                             Copyright (c)
Function Name :       time_base()

Engineer :            William Mackay

Location :            Freescale Microcontroller Division, East Kilbride

Date Created :        December 1999

Current Revision :    0.0

Note :                This ISR increments a count for the door alarm delay
                      period

******************************************************************************/


#pragma TRAP_PROC SAVE_REGS
void  time_base()
{
        ++door_alarm_delay;                          /* increment on counter rollover */
        TBCR_TACK = SET;                             /* clear time base interrupt flag */
}
```

4.15.9 Oscillator Initialisation Routine.

```
/******************************************************************************
                                             Copyright (c)
Function Name :       init_osc()

Engineer :            William Mackay

Location :            Freescale Microcontroller Division, East Kilbride

Date Created :        December 1999

Current Revision :    0.0

Note :                Function sets oscillator frequency

******************************************************************************/

void init_osc(void)
{
        ICGMR_N0 = SET;                              /* set oscillator frequency */
        ICGMR_N1 = RESET;                            /* multipier set for 29x307.2khz = 8.9Mhz */
        ICGMR_N2 = SET;                              /*  = 2.27Mhz bus freq */
        ICGMR_N3 = SET;
        ICGMR_N4 = SET;
        ICGMR_N5 = RESET;
        ICGMR_N6 = RESET;
}
```

4.15.10 Initialise input/output Ports.

```
/*******************************************************************************
                                             Copyright (c)
Function Name :     init_ports()

Engineer :          William Mackay

Location :          Freescale Microcontroller Division, East Kilbride

Date Created :      December 1999

Current Revision :  0.0

Note :              input/output port configuration

*******************************************************************************/

void init_ports(void)
{
        DDRA_BIT0 = OUTPUT;                         /* relay */
        DDRA_BIT1 = OUTPUT;                         /* buzzer */
        DDRA_BIT3 = OUTPUT;                         /* triac drive */
        DDRA_BIT4 = INPUT;                          /* door */
        PTAPUE_BIT4 = SET;                          /* enable pull-up */
        DDRB_BIT0 = INPUT;                          /* air temp adc */
        DDRB_BIT1 = INPUT;                          /* evaporator temp adc */
        DDRB_BIT2 = INPUT;                          /* temp select */
        DDRB_BIT3 = OUTPUT;                         /* power 'on' led  */
        DDRB_BIT6 = OUTPUT;                         /* alarm indicator led */
        DDRB_BIT7 = OUTPUT;                         /* compressor 'on' led */
}
```

4.15.11 Initialise Analogue to Digital Converter.

```
/*******************************************************************************
                                             Copyright (c)
Function Name :     init_adc()

Engineer :          William Mackay

Location :          Freescale Microcontroller Division, East Kilbride

Date Created :      December 1999

Current Revision :  0.0

Notes :             Function sets ADC clock source and divide ratio

*******************************************************************************/


void init_adc(void)
{
        ADCLK_ADICLK = SET;                         /* internal bus clock as ADC clock source */
        ADCLK_ADIV0  = RESET;                       /* ADC clock divide ratio = 1 */
        ADCLK_ADIV1  = RESET;
        ADCLK_ADIV2  = RESET;
```

```
}
```

4.15.12 Delay Routine.

```
/*****************************************************************************
                                             Copyright (c)
Function Name :      delay()

Engineer :           William Mackay

Location :           Freescale Microcontroller Division, East Kilbride

Date Created :       December 1999

Current Revision :   0.0

Note :               This delay accommodates the Triac pulse duration
                     and the analogue to digital conversion time delay

*****************************************************************************/

void delay(void)
{
        unsigned char i,j;
        for(i=0; i<2; i++)
        {
            for(j=0; j<2; j++);
        }
}
```

4.15.13 Initialise Timer.

```
/*****************************************************************************
                                             Copyright (c)
Function Name :      init_timer()

Engineer :           William Mackay

Location :           Freescale Microcontroller Division, East Kilbride

Date Created :       December 1999

Current Revision :   0.0

Note :               Function used to configure timer interface module.
                     Sets internal bus clock pre-scalar for timer counter

*****************************************************************************/

void init_timer(void)
{
   TSC = RESET;                                    /* internal bus clock divide by 1 */
}
```

4.15.14 Initialise Time Base Module.

```
/****************************************************************************
                                              Copyright (c)
Function Name :     init_time_base()

Engineer :          William Mackay

Location :          Freescale Microcontroller Division, East Kilbride

Date Created :      December 1999

Current Revision :  0.0

Note :              Initialises time base module interrupt rate

****************************************************************************/

void init_time_base(void)
{
        TBCR_TBR0  = RESET;                             /* sets interrupt divider tap to 32768 */
        TBCR_TBR1  = RESET;
        TBCR_TBR2  = RESET;
}
```

4.15.15 Initialise Input Capture.

```
/****************************************************************************
                                              Copyright (c)
Function Name :     init_icap()

Engineer :          William Mackay

Location :          Freescale Microcontroller Division, East Kilbride

Date Created :      December 1999

Current Revision :  0.0

Note :              This function configures timer channel zero as input
                    capture for rising and falling edge detection

****************************************************************************/

void init_icap(void)
{
        TSC0_MS0A   =  RESET;                           /* mode select = input capture */
        TSC0_MS0B   =  RESET;
        TSC0_ELS0A  = SET;                              /* capture on rising or falling edge */
        TSC0_ELS0B  = SET;
        TSC0_CH0IE   = SET;                             /* enable interrupts */
}
```

4.15.16 HC08KX8 Generic Header File

```
/*****************************************************************************
                                             Copyright (c)
File Name :          HC08KX8.h

Org Author :         William Mackay

Location :           Freescale Microcontroller Division, East Kilbride

Date Created :       December 1999

Current Revision :   0.0

Notes :              This file maps the 68HC908KX8 Register set required
                     for the fridge application. The registers are mapped as
                     defined in the General Release Specification.

******************************************************************************/

#ifndef _HC08KX8_H
#define _HC08KX8_H


/****************************************************************************/
/* Register Mapping Structures and Macros                                  */
/****************************************************************************/

#define      REGISTER(a)  (*((volatile unsigned char *)(a)))
#define      BIT(a,b)  (((vbitfield *)(a))->bit##b)

/* assumes right to left bit order */

typedef volatile struct{
    volatile unsigned int bit0    : 1;
    volatile unsigned int bit1    : 1;
    volatile unsigned int bit2    : 1;
    volatile unsigned int bit3    : 1;
    volatile unsigned int bit4    : 1;
    volatile unsigned int bit5    : 1;
    volatile unsigned int bit6    : 1;
    volatile unsigned int bit7    : 1;
} vbitfield;
```

```
/****************************************************************************/
/* Input Output Ports                                                      */
/****************************************************************************/

/* Port A Data register */
#define      PTA            REGISTER(0x00)
#define      PTA_BIT0       BIT(0x00,0)
#define      PTA_BIT1       BIT(0x00,1)
#define      PTA_BIT2       BIT(0x00,2)
#define      PTA_BIT3       BIT(0x00,3)
#define      PTA_BIT4       BIT(0x00,4)


/* Port B Data register */
#define      PTB            REGISTER(0x01)
#define      PTB_BIT0       BIT(0x01,0)
#define      PTB_BIT1       BIT(0x01,1)
#define      PTB_BIT2       BIT(0x01,2)
#define      PTB_BIT3       BIT(0x01,3)
#define      PTB_BIT4       BIT(0x01,4)
#define      PTB_BIT5       BIT(0x01,5)
#define      PTB_BIT6       BIT(0x01,6)
#define      PTB_BIT7       BIT(0x01,7)
/* Port A Data Direction Register */
#define      DDRA           REGISTER(0x04)
#define      DDRA_BIT0      BIT(0x04,0)
#define      DDRA_BIT1      BIT(0x04,1)
#define      DDRA_BIT2      BIT(0x04,2)
#define      DDRA_BIT3      BIT(0x04,3)
#define      DDRA_BIT4      BIT(0x04,4)

/* Port A Input Pull Up Enable Register */
#define      PTAPUE         REGISTER(0x0D)
#define      PTAPUE_BIT0    BIT(0x0D,0)
#define      PTAPUE_BIT1    BIT(0x0D,1)
#define      PTAPUE_BIT2    BIT(0x0D,2)
#define      PTAPUE_BIT3    BIT(0x0D,3)
#define      PTAPUE_BIT4    BIT(0x0D,4)

/* Port B Data Direction Register */
#define      DDRB           REGISTER(0x05)
#define      DDRB_BIT0      BIT(0x05,0)
#define      DDRB_BIT1      BIT(0x05,1)
#define      DDRB_BIT2      BIT(0x05,2)
#define      DDRB_BIT3      BIT(0x05,3)
#define      DDRB_BIT4      BIT(0x05,4)
#define      DDRB_BIT5      BIT(0x05,5)
#define      DDRB_BIT6      BIT(0x05,6)
#define      DDRB_BIT7      BIT(0x05,7)
```

```
/****************************************************************************/
/* Time Base Register                                                       */
/****************************************************************************/

#define      TBCR           REGISTER(0x1C)
#define      TBCR_TBON      BIT(0x1C,1)
#define      TBCR_TBIE      BIT(0x1C,2)
#define      TBCR_TACK      BIT(0x1C,3)
#define      TBCR_TBR0      BIT(0x1C,4)
#define      TBCR_TBR1      BIT(0x1C,5)
#define      TBCR_TBR2      BIT(0x1C,6)
#define      TBCR_TBIF      BIT(0x1C,7)


/****************************************************************************/
/* Configuration Write-Once Registers                                       */
/****************************************************************************/

#define      CONFIG2        REGISTER(0x1e)
#define      CONFIG1        REGISTER(0x1F)


/****************************************************************************/
/* Timer Registers                                                          */
/****************************************************************************/

/* Timer Status and Control Register */
#define      TSC            REGISTER(0x20)
#define      TSC_PS0        BIT(0x20,0)
#define      TSC_PS1        BIT(0x20,1)
#define      TSC_PS2        BIT(0x20,2)
#define      TSC_TRST       BIT(0x20,4)
#define      TSC_TSTOP      BIT(0x20,5)
#define      TSC_TOIE       BIT(0x20,6)
#define      TSC_TOF        BIT(0x20,7)

/* Timer Counter Register */
#define      TCNTH          REGISTER(0x21)
#define      TCNTL          REGISTER(0x22)



/* Timer Modulo Register  */
#define      TMODH          REGISTER(0x23)
#define      TMODL          REGISTER(0x24)

/* Timer Status and Control Register Channel 0 */
#define      TSC0           REGISTER(0x25)
#define      TSC0_CH0MAX    BIT(0x25,0)
#define      TSC0_TOV0      BIT(0x25,1)
#define      TSC0_ELS0A     BIT(0x25,2)
#define      TSC0_ELS0B     BIT(0x25,3)
#define      TSC0_MS0A      BIT(0x25,4)
#define      TSC0_MS0B      BIT(0x25,5)
#define      TSC0_CH0IE     BIT(0x25,6)
#define      TSC0_CH0F      BIT(0x25,7)
```

```
/* Timer Channel 0 Register */
#define      TCH0H          REGISTER(0x26)
#define      TCH0L          REGISTER(0x27)

/* Timer Status and Control Register Channel 1 */
#define      TSC1           REGISTER(0x28)
#define      TSC1_CH1MAX    BIT(0x28,0)
#define      TSC1_TOV1      BIT(0x28,1)
#define      TSC1_ELS1A     BIT(0x28,2)
#define      TSC1_ELS1B     BIT(0x28,3)
#define      TSC1_MS1A      BIT(0x28,4)
#define      TSC1_CH1IE     BIT(0x28,6)
#define      TSC1_CH1F      BIT(0x28,7)

/* Timer Channel 1 Register */
#define      TCH1H          REGISTER(0x29)
#define      TCH1L          REGISTER(0x2a)


/****************************************************************************/
/* ICG Registers                                                          */
/****************************************************************************/

/* ICG Control Register */
#define      ICGCR          REGISTER(0x36)
#define      ICGCR_ECGS     BIT(0x36,0)
#define      ICGCR_ECGON    BIT(0x36,1)
#define      ICGCR_ICGS     BIT(0x36,2)
#define      ICGCR_ICGON    BIT(0x36,3)
#define      ICGCR_CS       BIT(0x36,4)
#define      ICGCR_CMON     BIT(0x36,5)
#define      ICGCR_CMF      BIT(0x36,6)
#define      ICGCR_CMIE     BIT(0x36,7)

/* ICG Multiply Register */
#define      ICGMR          REGISTER(0x37)
#define      ICGMR_N0       BIT(0x37,0)
#define      ICGMR_N1       BIT(0x37,1)
#define      ICGMR_N2       BIT(0x37,2)
#define      ICGMR_N3       BIT(0x37,3)
#define      ICGMR_N4       BIT(0x37,4)
#define      ICGMR_N5       BIT(0x37,5)
#define      ICGMR_N6       BIT(0x37,6)

/* ICG Trim Register */
#define      ICGTR          REGISTER(0x38)
#define      ICGTR_TRIM0    BIT(0x38,0)
#define      ICGTR_TRIM1    BIT(0x38,1)
#define      ICGTR_TRIM2    BIT(0x38,2)
#define      ICGTR_TRIM3    BIT(0x38,3)
#define      ICGTR_TRIM4    BIT(0x38,4)
#define      ICGTR_TRIM5    BIT(0x38,5)
#define      ICGTR_TRIM6    BIT(0x38,6)
#define      ICGTR_TRIM7    BIT(0x38,7)
```

```
/***************************************************************************/
/* Analogue To Digital Converter Registers                                 */
/***************************************************************************/

/* A/D Status and Control Register */
#define      ADSCR          REGISTER(0x3c)
#define      ADSCR_ADCH0    BIT(0x3c,0)
#define      ADSCR_ADCH1    BIT(0x3c,1)
#define      ADSCR_ADCH2    BIT(0x3c,2)
#define      ADSCR_ADCH3    BIT(0x3c,3)
#define      ADSCR_ADCH4    BIT(0x3c,4)
#define      ADSCR_ADCO     BIT(0x3c,5)
#define      ADSCR_AIEN     BIT(0x3c,6)
#define      ADSCR_COCO     BIT(0x3c,7)

/* A/D-Data Register */
#define      ADR            REGISTER(0x3d)

/* A/D Input Clock Register */
#define      ADCLK          REGISTER(0x3e)
#define      ADCLK_ADICLK   BIT(0x3e,4)
#define      ADCLK_ADIV0    BIT(0x3e,5)
#define      ADCLK_ADIV1    BIT(0x3e,6)
#define      ADCLK_ADIV2    BIT(0x3e,7)


/***************************************************************************/
/* Low Voltage Inhibit Register                                            */
/***************************************************************************/

/* LVI Status Register */
#define      LVISR          REGISTER(0xFE0C)
#define      LVISR_LVIOUT   BIT(0xFE0C,7)

#endif
```

4.15.17 Application Header File.

```
/*****************************************************************************
                                             Copyright (c)
File Name :          Fridge.h

Engineer :           William Mackay

Location :           Freescale Microcontroller Division, East Kilbride

Date Created :       December 1999

Current Revision :   0.0

Notes :              This file contains application specific definitions

******************************************************************************/

#ifndef _FRIDGE_H
#define _FRIDGE_H


/****************************************************************************/
/* Constant Definitions                                                     */
/****************************************************************************/

#define     ON      0
#define     OFF     1
#define     SET     1
#define     RESET   0
#define     CLEAR   0
#define     ENABLED 1
#define     DISABLED0
#define     OUTPUT  1
#define     INPUT   0

#define     START_TIME     0x28                   /* motor start-up period (40mS) */
#define     AIR_MAX_TEMP   0x33                   /* 1V maximum air temperature */
#define     AIR_MIN_TEMP   0x33                   /* 1V minimum air temperature */
#define     ALARM_TEMP_MAX 0xE8                   /* 4.5V maximum alarm temperature */
#define     ALARM_TEMP_MIN 0x1A                   /* 0.5V minimum alarm temperature */
#define     ONE_MINUTE     0x3FAB                 /* count for time base module divider tap = 32768 */
#define     TEN_SECONDS    0x01B2                 /* count for time base module divider tap = 32768 */


/****************************************************************************/
/* Input/Output Port Application Definitions                                */
/****************************************************************************/

#define     COMPRESSOR_POWER     PTA_BIT0
#define     BUZZER               PTA_BIT1
#define     DOOR_OPEN            PTA_BIT4
#define     TRIAC_DRIVE          PTA_BIT3
#define     POWER_STATUS         PTB_BIT3
#define     COMPRESSOR_STATUS    PTB_BIT7
#define     ALARM_STATUS         PTB_BIT6
```

```
/***************************************************************************/
/* Analogue channel definitions                                            */
/***************************************************************************/

#define      AD0     0
#define      AD1     1
#define      AD2     2
#define      AD3     3



/***************************************************************************/
/* ADC Data Register                                                       */
/***************************************************************************/

#define      ADC_VALUE      ADR


/***************************************************************************/
/* ADC Status and Control Register                                         */
/***************************************************************************/

#define      START_CONVERSION     ADSCR
#define      CONVERSION_COMPLETE  ADSCR_COCO
#define      ADC_INTERRUPT        ADCSR_AIEN
#define      CONVERSION_MODE      ADCSR_ADCO


/***************************************************************************/
/* Timer                                                                   */
/***************************************************************************/

#define      ZERO_CROSS_DETECT    TSC0_CH0IE
#define      ICAP_FLAG            TSC0_CH0F

/***************************************************************************/
/* Function Prototypes                                                     */
/***************************************************************************/

void main (void);
void init(void);
void InputCapture(void);
void delay(void);
unsigned char single_adc(unsigned char);
void init_osc(void);
void compressor_on(void);
void compressor_off(void);
void alarm_check(void);
void init_adc(void);
void init_ports(void);
void init_timer(void);
void init_time_base();
void init_icap(void);
void reset_icap(void);
```

```
/****************************************************************************/
/* Global Variables                                                         */
/****************************************************************************/

/* Zero Page RAM variables */
#pragma      DATA_SEG _DATA_ZEROPAGE
unsigned     char    air_temp;                /* fridge compartment temperature */
unsigned     char    selected_temp;           /* user selected temperature */
unsigned     char    start_phase;             /* indicates status of start time interval */
unsigned     char    door_alarm_delay;        /* time base count for door open alarm delay */
unsigned     char    alarm_valid;             /* flag to indicate buzzer can be sounded ..*/
                                              /*..when alarm condition is detected */
unsigned     char    read_register            /* dummy read location for flag clearing */

/****************************************************************************/
/* Interrupt Definitions                                                    */
/****************************************************************************/

#define      ENABLE_INTERRUPTS    asm cli;
#define      DISABLE_INTERRUPTS   asm sei;

#endif
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

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

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

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

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

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

freescale™ semiconductor