DRM001/D

Passive Infrared (PIR) Intruder Detection Using the MC68HC908JK1/3, Incorporating Remote Control Adjustment Using the MC68HC908GP32

Designer Reference Manual

Digital DNA™ from Motorola

Passive Infrared (PIR) Intruder Detection Using the MC68HC908JK1/3, Incorporating Remote Control Adjustment Using the MC68HC908GP32

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 693921
Email: enquiries@ateecc.com
Web: www.ateecc.com

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

List of Sections

List of Sections

Table of Contents

Section 1. General Description

Section 2. Passive Infrared (PIR) Sensor Unit

Section 3. Infrared Communications Protocol

Section 4. REMOTE Control Unit

Section 5. Phase-Locked Loop (PLL) Initialization

Section 6. Cosmic M68HC08 Complier

Section 7. Windows® 95/98/NT Program (pir_plot.exe)

Appendix A. Fresnel Lens Mounting

Appendix B. PIR Schematics

Appendix C. Development Boards

Appendix D. MC68HC908GP32 Programmer Circuit

Appendix E. PIR Source Code Files

Appendix F. REMOTE Source Code Files

List of Figures and Tables

List of Figures and Tables

Section 1. General Description

1.1 Contents

1.2 Introduction

This document details the hardware and software required for a fully functional passive infrared (PIR) sensor with an associated REMOTE control unit. The REMOTE control unit adjusts key algorithm detection parameters which are stored in the MC68HC908JK1/3 FLASH memory area.

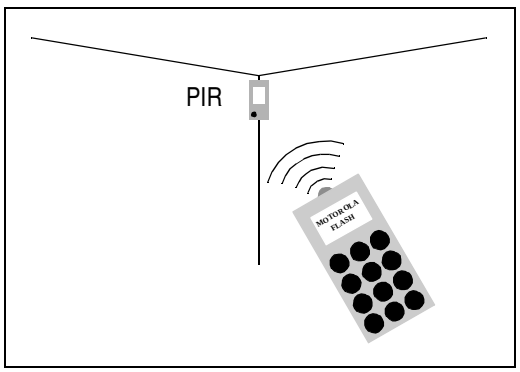


Figure 1-1. PIR Sensor with REMOTE Control Unit

The main purpose of this document is to demonstrate the ability of the MC68HC908JK1/3 to program its own FLASH memory, effectively using this FLASH memory as a nonvolatile data store.

The PIR is analyzed first with respect to its main features, followed by the two intruder detection algorithms. Motorola FLASH programming is then studied, with particular reference to self programming. The REMOTE control unit is analyzed in a manner similar to the PIR unit. Finally, the accompanying Windows® program is analyzed.

Throughout the document, references are made to source code files which can be found in **Appendix E. PIR Source Code Files** and **Appendix F. REMOTE Source Code Files**. For those viewing this document in .pdf format, these files can be accessed by clicking on the appropriate hyperlink reference. Some text areas have in-line source code extracts to highlight a particular point.

Included in this reference design are all C source code files and circuit schematics, and a Windows® 95/98/NT program is available from both Motorola and ATEECC Web sites. A development board is also available from ATEECC, which utilizes the hardware and software detailed in this document. In addition, the development board provides hardware and software for MC68HC908GP32CP device programming.

1.3 Design Overview

As previously mentioned, a key point of this document is to demonstrate the ability of the MC68HC908JK1/3 to program a FLASH row while in normal operation (user mode). This feature negates the requirement for external EEPROM storage and, consequently, can help reduce system costs.

Windows is a registered trademark of Mircrosoft in the U.S. and other countries.

To maximize design flexibility, two intruder detect event algorithms are incorporated into this application. These algorithms are jumper selectable on the development board at startup.

- The first method uses the 8-bit analog convertor to read the amplified sensor output, which is stored into a buffer for pattern analysis.
- The second method uses a modified Delta Sigma approach, allowing adjustable resolution (varying acquisition times).

The system block diagram is shown in **Figure 1-2**.

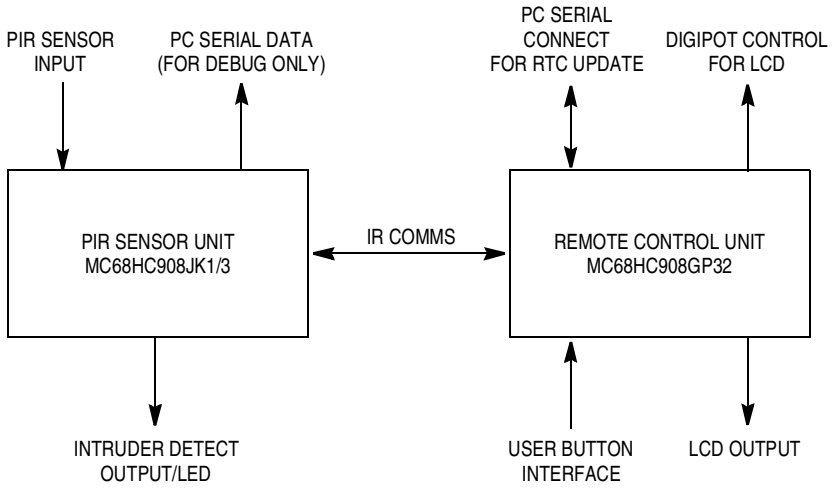


Figure 1-2. System Block Diagram

The PIR sensor is mounted behind a Fresnel lens. The output signal from the sensor is amplified and conditioned by two elements of an operational amplifier, before being connected directly to an analog-to-digital (A/D) channel of the microcontroller (MCU) which is the conventional analog approach. Alternatively, it may be AC coupled to the input of the microcontroller via an R/C network which forms the basis for the alternative Delta Sigma method of detection. The intruder detect output is a signal that is used to indicate to the PIR units parent system that a valid intruder event has been detected. In this application, a light-emitting diode (LED) is used to indicate an alarm condition. Normally, this is an alarm trigger device, such as a relay, transistor, etc.

The dotted line representing the PC serial data stream used for debugging purposes, in the final product this code would not be included. The serial data contains the real-time sensor value and parameter information. Both methods (analog/Delta Sigma) output their appropriate sensor value, giving vital feedback to the run time behavior of the sensor and allowing immediate feedback on algorithm parameter adjustment.

The REMOTE unit allows a user to adjust key detection parameters, allowing the user to quickly adjust and test. Five parameters can be adjusted; three pertain to the analog detection method and the remaining two to the Delta Sigma detection method.

1.4 Black Body Principles

The radiation emitted from a black body at a temperature of 300 K is predominantly in the region of 7 μm to 14 μm, peaking at around 9.5 μm. Research has shown that this value is modified to around 8.5 μm when the black body is moving against a different background temperature.

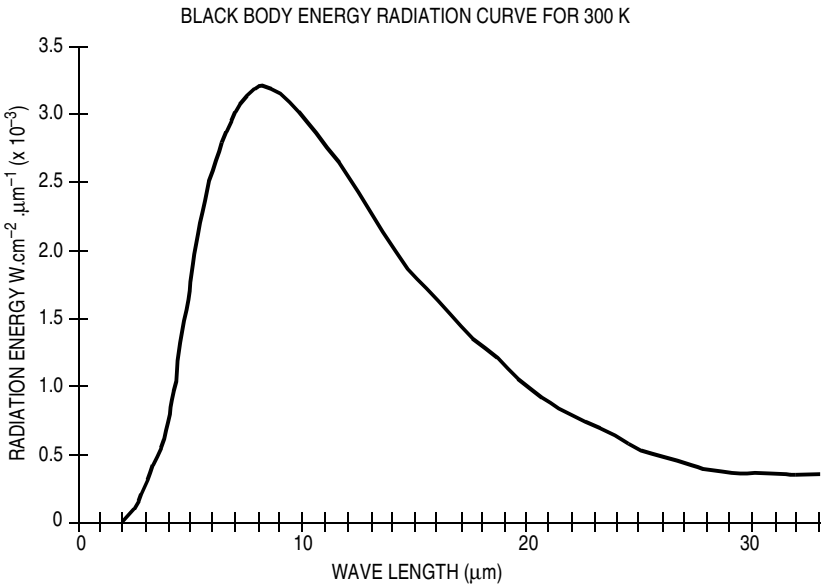


Figure 1-3. Black Body Radiation Curve

Section 2. Passive Infrared (PIR) Sensor Unit

2.1 Contents

2.2 Introduction

Pyrolytic sensors respond to movement due to a *change* in the radiation incident upon them. They are designed to be most sensitive to the wavelength described in **Figure 1-3. Black Body Radiation Curve**. A key component of the passive infrared (PIR) sensor unit is the Fresnel lens. This gives the PIR the ability to respond to radiation from a wider angle of positions as the lens effectively focuses the incident radiation to produce a series of “peaks” as an emitting body moves across the path of the lens.

Using a low-cost microcontroller (MCU) like the MC68HC908JK1/3 has many advantages compared to an analog sensor circuit since the MCU can apply real-time intelligence to the sensor data it is receiving. This intelligence forms the heart of the intruder detection algorithm; the advantage is increased by the ability of the user to modify key algorithm parameters, which are stored in FLASH memory. The FLASH memory parameters are adjusted by 2-way infrared communications using a REMOTE unit. The use of an MCU also provides the designer with an alternative method of sensor amplification, which employs considerably fewer components than the op-amp approach.

2.3 Features

Features of the PIR include:

- Infrared (IR) communications with 38 kHz tx being bit bashed and rx via the timer capture interrupt
- RS232 tx communications, bit bashed at 38,400 bit rate
- FLASH self-erase/program/verification using Motorola monitor routines
- Analog initialization/read
- Intruder detect using analog buffer scan or Delta Sigma algorithms

Figure 2-1 illustrates the top level program flow for the PIR unit, the major decision to be made initially is the required method of analysis. This may be either an 8-bit analog read or a Delta Sigma analysis. The following text describes in detail the processes involved in each method.

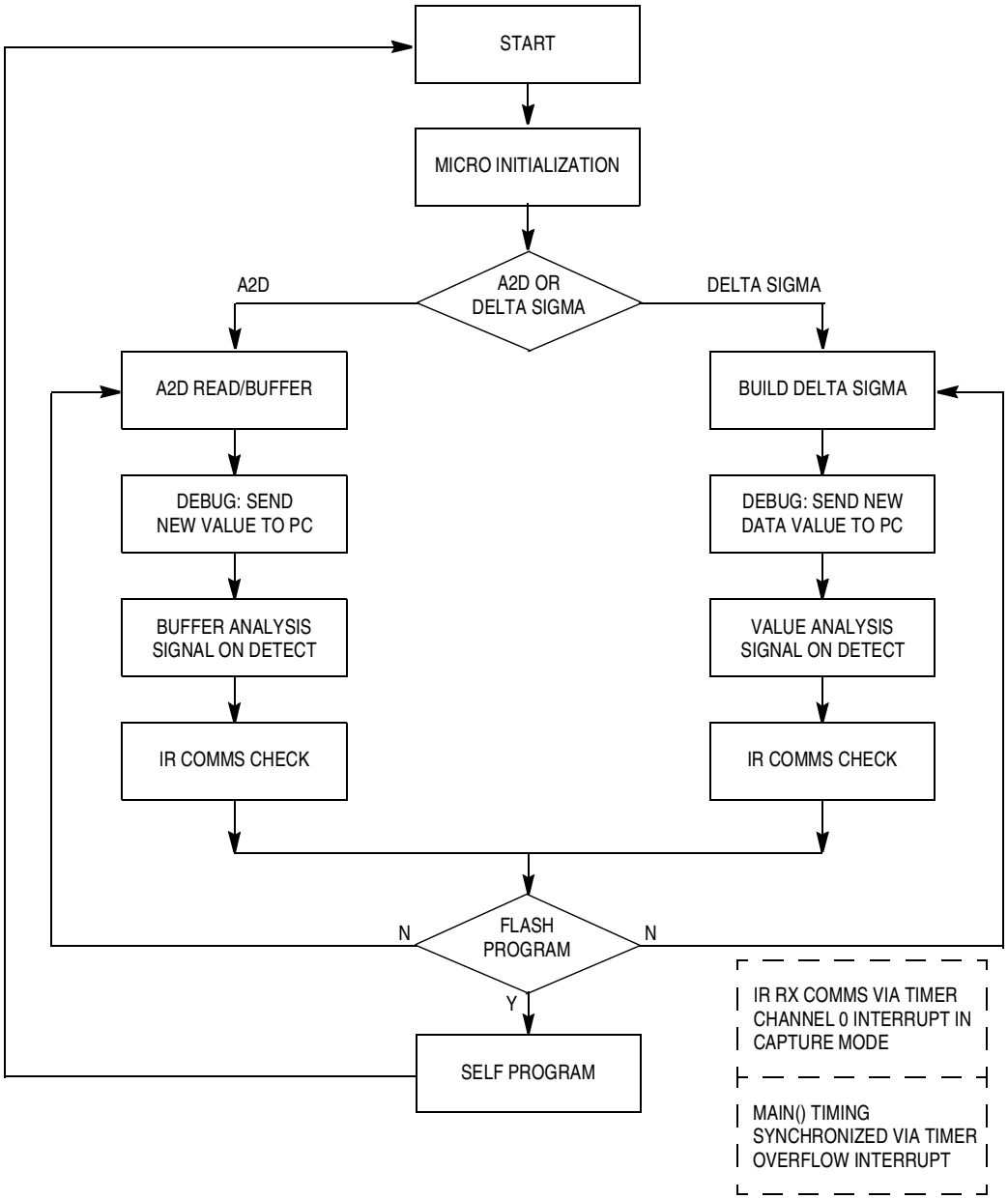


Figure 2-1. PIR Unit main( ) Flowchart

2.4 8-Bit Analog

The analog intruder detection method and the analog circuit are described in the following subsections.

2.4.1 Analog Intruder Detection Method

One of the most important factors to consider when designing intruder alarm systems is that they should offer good sensitivity combined with a high immunity to false alarms. Pyrolytic sensors used in PIR alarm systems deliver a very low amplitude output, which is proportional to *changes* in incident infrared radiation falling on them. Traditionally, a multi-stage amplification has been used to condition the sensor output to provide a usable output signal. A typical example is shown in **Figure 2-2**.

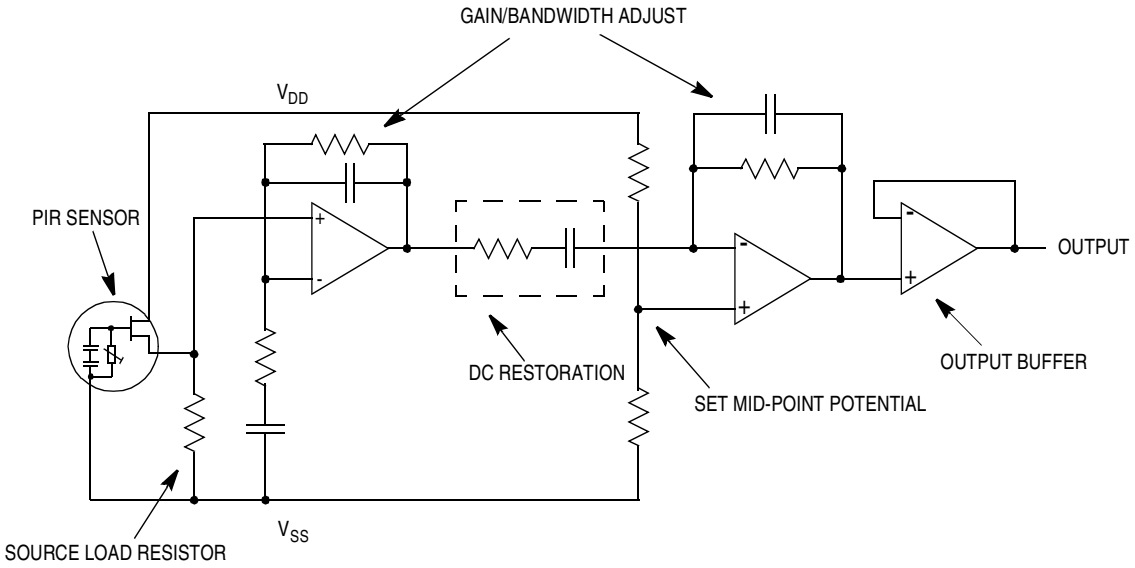


Figure 2-2. PIR Analog Circuit

2.4.2 Analog Circuit Description

The sensor output is connected in a source follower configuration and directly coupled to the non-inverting input of the first stage of amplification. The combined frequency response of the amplifiers provides a bandwidth of around 4.7 Hz, centered at approximately 0.5 Hz. This allows good detection rates to be achieved for human targets moving at speeds between 0.2 ms to 0.6 ms while attenuating the sources of noise, likely to cause false alarms. The potential divider chain connected to the non-inverting input of the second stage of amplification sets the quiescent point to 0.5 * V_{DD}. This allows maximum sensitivity to positive and negative swings from the sensor. The second stage is AC coupled, to allow slow changes of background IR radiation to be ignored. Such changes may occur when central heating radiators warm up or sunshine heats the room.

A Fresnel lens is used to collect the IR radiation emitted by the “target” and to focus it onto the sensitive quartz window of the sensor. The lens has the dual function of concentrating the very low levels of radiation, thus producing a greater output from the sensor, and it also produces velocity information by giving a series of peaks as the “target” moves through the multiple zones. The choice of lens depends on the particular application. Some lenses have multiple zones focused at different angles, which produce different waveforms depending on the height of the target, in addition to its velocity. This information can be used to discriminate between human targets and animals, which could otherwise cause false alarms. The software algorithms in this application have been optimized for a single-plane “curtain” lens.

The conditioned sensor output voltage is connected to PTB.4 (PTB bit 4) and the analog read occurs at a rate which is a multiple of the 10 ms *main()* loop. This multiple is adjustable and is one of the programmable FLASH analog method parameters. With every *main()* loop iteration *pir\a2d.c\A2DCheck()* is executed; if the number of *main()* loop scans matches *pir_parameters.main_loop_count*, then the analog pin PTB.4 is read with *pir\a2d.c\ReadA2D()* (see **[PIR:a2d.c]**). This function performs a thirty two times read and returns the average result which is then stored in *pir_buffer[]* at the appropriate location using:

```
*pir_buffer_ptr = ReadA2D(CHANNEL4);        // from “a2d.c”
```

After every analog read/store operation a magnitude difference test is performed with the previous data value. If this difference is greater than or equal to *pir_param.difference_band*, then *pir_buffer* is cleared and the current and previous values are stored at locations [0] and [1] respectively. Subsequent values are stored and when *pir_buffer* is full a call to *pir\analyse.c->Analyse_PIR_Buffer()* (see **[PIR:analyse.c]**) is performed and a detect event is scanned for.

A detect event has two parameters:

1. *pir_param.difference_band* — This is the difference between the buffer nearest neighbor data that will be accepted as an *intruder trigger*.
2. *pir_params.trigger_count* — This is the number of intruder triggers contained in the same buffer which must occur before an *intruder event is accepted.*

The “difference band” value is analogous to the rate of change of the analog signal. If a signal were changing rapidly, then the buffer contents would contain values that were increasing/decreasing by large amounts. If these changes were happening on adjacent buffer cells, this would cause a trigger event. If this change occurred in a single buffer capture and if the number of trigger events was greater than the *trigger_count* variable, then an intruder event would be signalled.

2.5 Delta Sigma

The Delta Sigma intruder detection method and operation are described in the following subsections.

2.5.1 Delta Sigma Intruder Detection Method

This method of signal detection uses considerably fewer components than the previously described method, giving benefits of cost reduction and reliability.

The principle of operation is based on a modified version of the Delta Sigma analog-to-digital (A/D) converter. The hardware overhead is just

three resistors and two capacitors. The microcontroller is then used to control the charge/discharge of the integration capacitor. This method of A/D conversion is well known, but normally requires either an external comparator or a microcontroller with an on-board comparator. In this application, one of the on-board 8-bit A/D converters is used as a comparator, with the *trip* level being specified in software. The absolute conversion accuracy is dependant on a number of factors, including the input leakage current of the analog sense pin and the fast charge pin in its quiescent state. Leakage current in the integration capacitor will also cause errors in accuracy due to asymmetric charge/discharge conditions. In this application, however, it is the *difference* in consecutive A/D values which will cause an event trigger. As a consequence small changes in absolute accuracy will not affect the overall result, making this method a good choice for this application.

The effective amplification which can be achieved is dependent on the ratio of two resistors. In this application, the output of the pyrolytic sensor is capacitively coupled to the integrator by a 33-µF capacitor. This value was selected to produce similar characteristics to the method using an operational amplifier. As the series capacitor provides DC isolation, a high value may be selected for the charge/discharge resistor without causing the comparator voltage to be “loaded” by the source resistance of the sensor. An optional resistor, supplied from a spare port pin on the microcontroller, has also been added to this circuit. Its function is to provide a fast charge path for the coupling capacitor, allowing the circuit to stabilize quickly after the initial application of power to the circuit. See **Figure 2-3**.

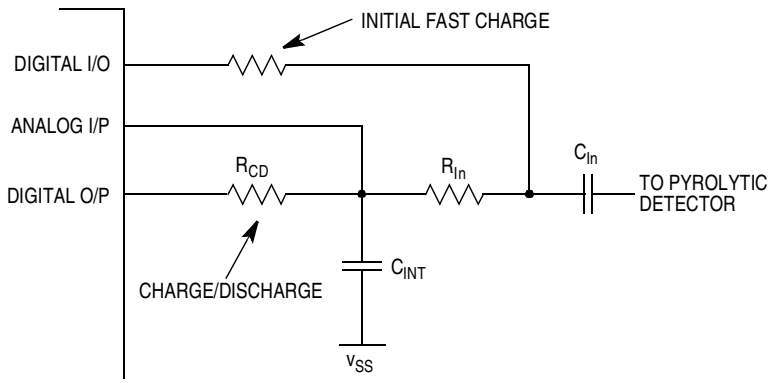


Figure 2-3. Delta Sigma Circuit

2.5.2 Delta Sigma Operation

Assuming a steady quiescent state with no sensor activity, the integration capacitor C_{INT} is charged from a digital output pin on the microcontroller via resistor R_{CD}. The voltage level across C_{INT} is monitored by the A/D input. Two software counters are used. One is a loop counter which determines the number of 'bits' to be converted. The second serves as a data value for the conversion. At the start of a conversion, they are both set to 0. The loop time multiplied by the number of bits required determines the time for each full conversion. During each software loop of the conversion, a decision is made to increment the data counter or not.

If the A/D value is equal to or greater than the *trip* value, the counter is incremented and the output port is made a logic 0 (C_{INT} discharge). Conversely, if it is less than the *trip* value, then the data counter is not incremented and the output port is made logic 1 (C_{INT} charge). With no input from the sensor, therefore, C_{INT} will be repeatedly charged and discharged, and the potential across it will be maintained at the A/D trip level.

In this application, an A/D value of 128 is used (corresponding to 0.5 * V_{REF}). The final binary output of the converter will also correspond to 0.5 of the maximum converter value. If the sensor voltage now increases due to a *target* detection, then C_{INT} will be charged by $R_{In,}$ in addition to R_{CD}. The potential across C_{INT} will rise causing the A/D *trip* value to be exceeded and, therefore, the data counter will be incremented on successive loops of the converter until it is discharged below the trip level by $R_{CD.}$ If the reverse condition occurs, and C_{INT} is discharged by the sensor output falling below the quiescent level, then the data counter will not be incremented, and the final converted number will be greater than the quiescent value.

The effective voltage amplification of the circuit is proportional to the ratio of R_{CD} to R_{INT}. Prototype testing has indicated that reliable operation can be achieved with values of 10 MΩ for R_{CD} and 10 kΩ for R_{In}, giving possible voltage gains of up to 60 dB.

With every *main()* loop iteration executed *pir\deltasig.c\DeltaSigma()*, this calls *pir\deltasig.c\BuildDeltaSigma()* to produce the Delta Sigma value (see **[PIR:deltasig.c]**). This value is then compared to the previous value and an intruder event is signalled if the difference is greater than *delta_sig_event*. With the Delta sigma detection method there are two FLASH based adjustable parameters:

1. *delta_sig_event*, this is the difference from previous reading to signal an intruder event
2. *delta_sig_bit*, this is the Delta Sigma resolution applied to the incoming PIR sensor voltage

The infrared communications are still active with this method but are slightly less responsive, the principle of the Delta Sigma method requires symmetrical capacitor charge/discharge times requiring interrupts to be disabled during *pir\deltasig.c\BuildDeltaSigma()* (see **[PIR:deltasig.c]**). After *pir\deltasig.c\DeltaSigma()* has completed interrupts are re-enabled to service any pending infrared communications.

2.6 PIR Software Files

This software has been written using the Cosmic ‘C’ Cross Compiler. All files for the PIR unit are listed here.

- Assembler:
  [PIR:crts.s], **[PIR:ireg.s]**, **[PIR:lreg.s]**
- C Source:
  [PIR:a2d.c], **[PIR:analyse.c]**, **[PIR:data.c]**, **[PIR:datasort.c]**, **[PIR:delay.c]**, **[PIR:deltasig.c]**, **[PIR:flashprg.c]**, **[PIR:interrup.c]**, **[PIR:mon_data.c]**, **[PIR:serial.c]**, **[PIR:startup.c]**, and **[PIR:vectors.c]**
- Include Files (in addition to the C source matching header):
  [PIR:declared.h], **[PIR:define.h]**, **[REMOTE:extern.h]**, and **[PIR:jk13&jl3.h]**

- Compile/Link/Make:
  [PIR:cc.bat], **[PIR:link08.bat]**, **[PIR:make08.bat]**, **[PIR:config.dat]**, and **[PIR:jk.lkf]**

2.6.1 On-Board MC68HC908JK1/MC68HC908JK3 20-Pin DIL Programmer

The programming hardware is compatible with the ICS08JLZ software from P&E Microcomputer Systems, Inc. The software used is *ics08jlz_version_1_33.exe*, this software is available from their web site at:

http://www.pemicro.com

The power supply unit uses two fixed-voltage 3-terminal regulators, which allows a wide range of input voltages to be used. Referring to **Figure 2-4**, an LM7805 regulator provides the stabilized +5 Vdc for the microcontroller and peripheral devices. An LM7808 and is used in conjunction with a series diode D2 in the common leg, to provide the necessary +8.6 V high voltage for the programmer, and also provides the power supply to the satellite main board for development purposes. A series diode in the input supply line D1, provides protection against accidental reverse polarity of the unregulated input supply. C1 provides decoupling and smoothing of the unregulated DC supply. The parallel combination of C2 and C3 provide high and low frequency decoupling to the +5-V supply. C4 and C5 provide a similar function for the +8.6-V supply.

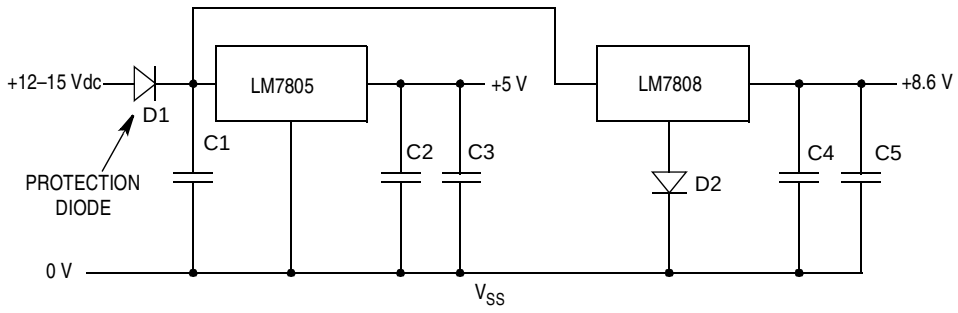


Figure 2-4. Power Supply

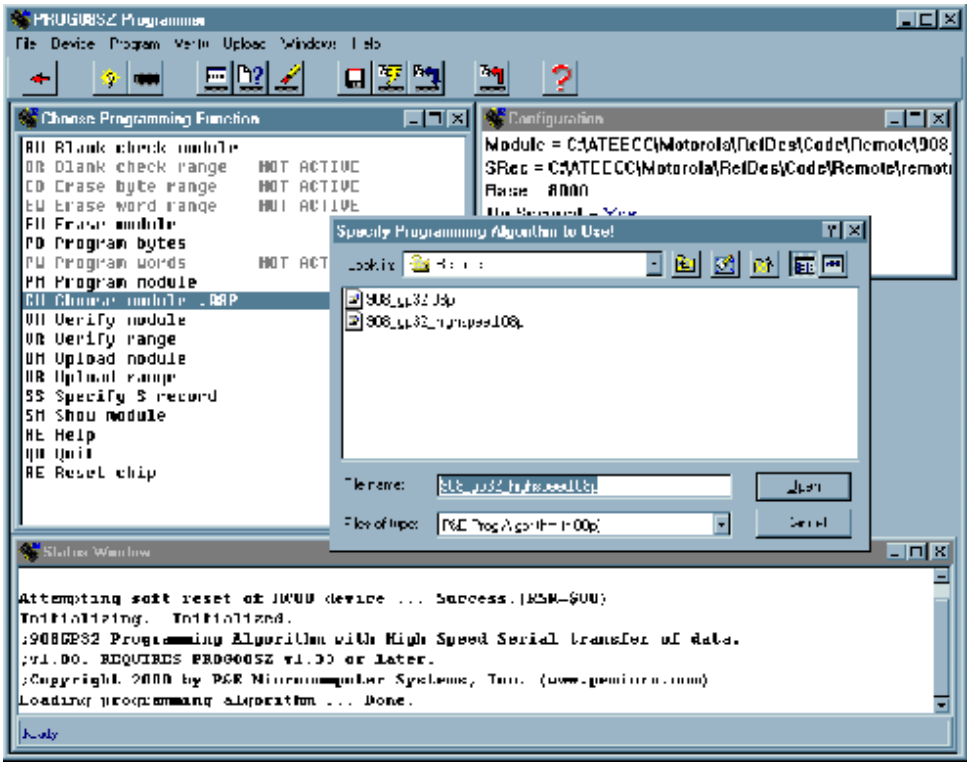
Within the P&E development environment is **prog08sz.exe**, it is this software that interfaces to the programming socket.

Programming procedure:

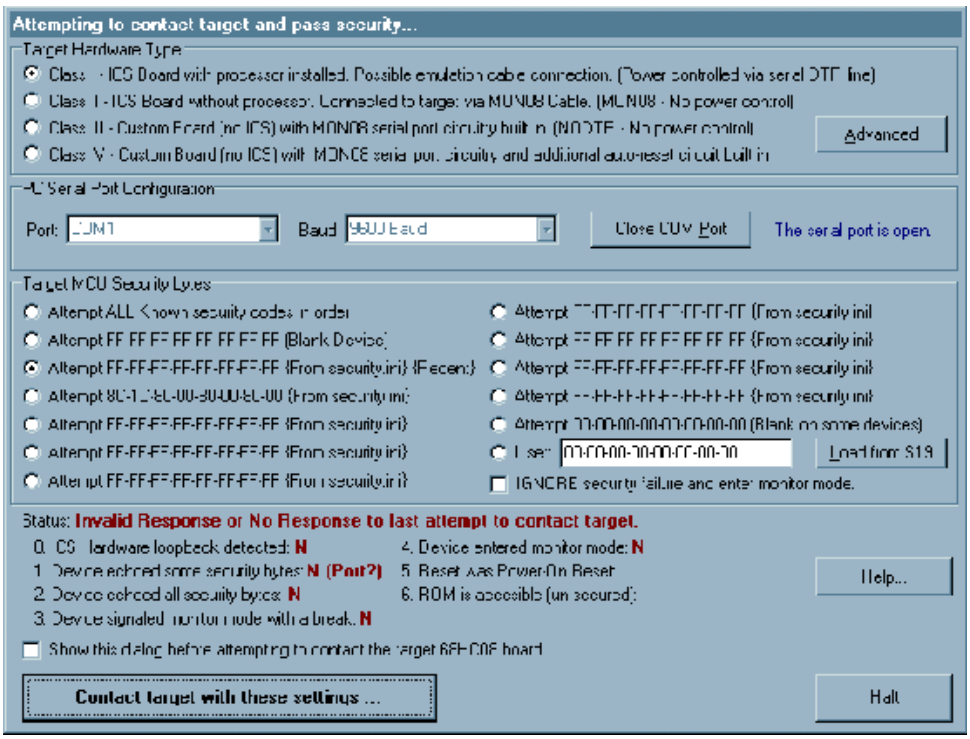
NOTE: *The PIR MC68HC908JK1/3 programming is in-circuit, there is no seperate programming socket.*

1. Ensure that the $\mathbf{V_{DD}}$ switch is **Off**.
2. Ensure that the 20-pin socket is occupied.
3. Ensure that a standard 9-way RS232 cable is connected from the PC to the development board's **Programmer** RS232 connector.
4. Set the **Osc Select** switch to **Program**.
5. Set $\mathbf{V_{DD}}$ switch to **On**.
6. Invoke c:\pemicro\ics08jlz**prog08sz.exe** (assuming default installation directory).
7. After programming is complete, set the $\mathbf{V_{DD}}$ switch to **Off** and move the **Osc Select** switch from **Program** to **Run**.

If the socketed MC68HC908JK1/3 passes the security test and the RS232 comms link is working then you will see the following screen. It is asking for a programming algorithm to be entered.



If there is a problem you will see:



The above screen typically occurs if the hardware RS232/power connections are wrong or if the socketed MC68HC908JK1/3 fails the security test.

2.6.2 Security Failure

The security check is a mechanism to prevent unauthorized access to the MC68HC908JK1/3 FLASH array. The security check centres around the interrupt vector address values at $fff6–$fffd. Before access is granted the PC program must transmit eight bytes that need to agree with those resident in the microcontroller.

If the 8-byte comparison fails then FLASH access is prevented, even though monitor mode can still be entered, before you can reprogram the MC68HC908JK1/3 or view its contents you will need to completely erase it. The program will remember the last S19 file programmed into a MC68HC908JK1/3 and use that file to pass the security test on next invocation.

Please note, if the MC68HC908JK1/3 fails the security test, the device *must* be powered down before a retry can be attempted. This power cycle will take the form:

1. $\mathbf{V_{DD}}$ switch to **Off**
2. Wait for at least two seconds.
3. $\mathbf{V_{DD}}$ switch to **On**

The program c:\pemicro\ics08jlz**prog08sz.exe** can now be retried.

2.6.3 Motorola FLASH Read-Only Memory (ROM)

The PIR unit is based upon the MC68HC908JK1/3, these are 1536-bytes/4096-byte FLASH microcontrollers, the ‘9’ in the part number denotes the part as being a FLASH device. The minimum size FLASH memory that can be erased at one time is 64 bytes and the maximum size FLASH memory that can be programmed at one time is 32 bytes (row). This reference design uses the last 64-byte block of the user code space as a 32-byte nonvolatile data store. This feature alleviates the need for an external memory IC such as an 8-pin 2-wire I^2C type.

The actual FLASH row programming differs to that of standard Motorola microcontroller electrically erasable programmable read-only memory (EEPROM) programming due to the row program requirement. With standard EEPROM it is necessary to write code that will perform the write/erase on a particular byte by using a call such as *WriteEEprom(address, data)*. This programming sequence may require an erase cycle before the program cycle. Standard Motorola microcontroller EEPROM will require up to 20 ms for an erase/program operation.

Using the Motorola FLASH cell, programming takes place in terms of a row. A row is 32 bytes of contiguous memory starting at a $XX00, $XX20, $XX40, $XX60, $XX80, $XXA0, $XXC0 or $XXE0 address. Presently if it is required to program one byte in a row then *all* bytes in that row must be reprogrammed. The programming time is markedly faster for this FLASH technology compared to standard Motorola EEPROM. The MC68HC908JK1/3 data book specifies a page (64 byte) erase time of 1 ms and a maximum FLASH byte program time of 40 µs. Motorola quotes a 2 ms program time for 64 bytes, this is a considerable improvement on the Motorola EEPROM timings.

The next consideration is the statement from the *MC68HC908JK1, MC68HRC908JK1, MC68HC908JK3, MC68HC908JL3, MC68HRC908JL3 Technical Data*, Motorola document order number MC68HC908JL3/H Rev. 1.0 which states:

> “Programming and erasing of FLASH locations cannot be performed by code being executed from FLASH memory.”

To program a FLASH row, this means software cannot be executing from FLASH ROM, and random-access memory (RAM) and monitor ROM would be acceptable. Due to the limited RAM (128 bytes) space in these devices, it would be difficult to have erase, program, and verification code in a RAM routine. To assist us, Motorola has provided monitor ROM areas which contain functions which perform the FLASH erasing, programming, and verification of supplied data.

These monitor ROM functions use three RAM variables and one RAM data array. These variables are expected to be at a fixed, known memory address. *Using MC68HC908 On-Chip FLASH Programming Routines*, Motorola document order number AN1831/D, details how details to use these monitor ROM functions. Further detail regarding the usage of these functions and variables is given in **2.6.4 PIR Parameter FLASH Programming**.

When FLASH programming is to take place, the data to be programmed is organized and *pir\flashprg.c->ProgramFlash()* (see **`[PIR:flashprg.c]`**) is called. This performs the monitor ROM variable initialization and calls the Motorola monitor ROM functions to take care of the FLASH erasing, programming, and verification. If the programming is successful, the PIR detect light-emitting diode (LED) is lit for 250 ms.

2.6.4 PIR Parameter FLASH Programming

If a decoded IR command requires a FLASH parameter programming operation, then **all** (row) FLASH parameter data must be reprogrammed, since single byte programming cannot (presently) be performed. *MONITOR_DATA[]* is used to store the PIR parameter data. The maximum number of bytes that can be programmed at one time is 32 (*a row*).

NOTE: *The address of MONITOR_DATA[0]'is at the address expected by the Motorola monitor functions ($008c). The data space occupied by MONITOR_DATA[] will overwrite run time variables. In normal operation this would be a critical error condition.*

In this application, after programming is completed, an endless loop is entered until the internal watchdog times out. Therefore, the overwriting of previous RAM space is not important. If your application requires a FLASH program operation *without* a reset, then you will need to ensure that there is enough RAM space for all variables. You might have to reduce the number of FLASH programmable variables by reducing the size of *'MONITOR_DATA[]'* which is 32 in this design. It would still be required that the variables used by the Motorola monitor functions *remain* at their fixed addresses. All other user program variables will need to *fit* around these monitor ones. You would then need to remove the '*-v*' (see the code example) switch from the linker command file which instructs the compiler not to perform overlap checks.

Once the data loading is complete within *pir\datasort.c->Decode_IR_Data()* (see **[PIR:flashprg.c]**), the next task is to initialize the variables used by the monitor ROM functions. The actual C code that will perform the FLASH PIR parameter programming resides in *pir\flashprg.c->ProgramFlash()* (see **[PIR:flashprg.c]**). This function is called from *pir\datasort->IRCommsCheck()* (see **[PIR:datasort.c]**). An extract is shown here:

```
if ( Decode_IR_Data() )   // is FLASH programming required?
{
//////////////////////////////////////////////////////
// interrupts off and reset the stack pointer as we   //
// are NOT returning from this function and we will   //
// be performing calls to the monitor functions       //
//////////////////////////////////////////////////////
SEI();
RSP();

ProgramFlash();   // RESET vector fetched at the end of this function
}
```

pir\flashprg.c->ProgramFlash() (see **[PIR:flashprg.c]**) is shown here.

```
void ProgramFlash( void )
{
unsigned char  ii;
                                  ////////////////////////////////////////
ServiceWatchDog();                // defensive measure                    //
FLBPR.reg       = 0xff;           // no FLASH protection                  //
MONITOR_CPUSPD  = SPDSET;         // 1(MHz) * 4 == 4                      //
MONITOR_CTRLBYT = 0x00;           // page erase                           //
MONITOR_LADDR   = FLASH_DATA_END; // data stored @ $FBC0/DF (32 bytes)    //
LED             = 0;              // led off...                           //
LED_DDR         = 1;              // ...and an output                     //
                                  //                                      //
_asm("ldhx  #$fbc8");             // any address in the range $fbc0 - $fbff //
ERARNGE();                        // to erase FLASH page, Motorola monitor  //
                                  // rom call                             //
_asm("ldhx  #$fbc0");             // first address in H:X to write to     //
PRGRNGE();                        // program FLASH row, Motorola monitor  //
                                  // rom call                             //
_asm("lda   #$ff");               // force ACC to non zero to ensure that   //
                                  // newly read data is placed back in the  //
                                  // data array and not to the monitor mode //
                                  // comm port.                           //
_asm("ldhx  #$fbc0");             // first address in H:X to verify FLASH   //
RDVRRNG();                        // programming, Motorola monitor rom call //
                                  //                                      //
if ( carry() )                    // carry bit set if verify is successful  //
   {                              // if so light led for 0.25s            //
   ii = 125;                      // load 0.25s counter                   //
   do {                           //                                      //
      ServiceWatchDog();          //                                      //
                                  //                                      //
      LED = 1;                    // led on                               //
                                  //                                      //
      _asm("lda   #4"  );         // Fop*4 (1MHz)                         //
      _asm("ldx   #167");         // 2000/12                              //
      DELNUS();                   // 2ms delay...Motorola monitor rom call  //
      }  while ( --ii );          // repeat                               //
   }                              //                                      //
                                  //                                      //
LED = 0;                          // led off                              //
while (1);                        // all done! wait for watchdog reset...   //
                                  ////////////////////////////////////////
}  // ProgramFlash()
```

The PIR parameters are stored at the beginning of the last 64-byte block which is $FBC0.

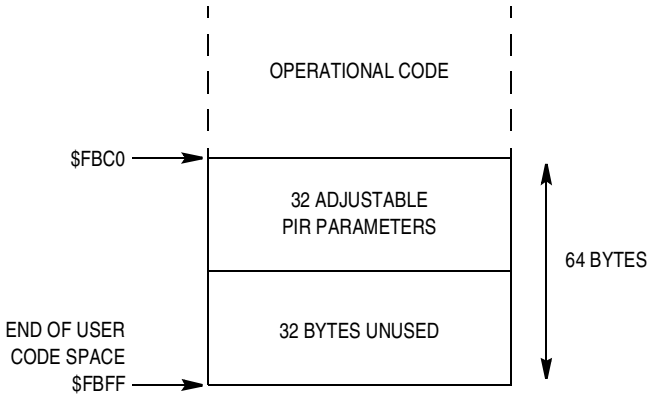


Figure 2-5. Last 64-Byte Block

The variables:

MONITOR_CTRLBYT, MONITOR_CPUSPD,
MONITOR_LADDR, and MONITOR_DATA[32]

are specific to the Motorola monitor ROM calls and are fixed addresses $0088, $0089, $008A, and $008B (16-bit variable), respectively. The data used for programming is declared as a 32-byte buffer (for instance, `MONITOR_DATA[32]` is fixed at address $008c). These variables are declared in **`[PIR:mon_data.c]`** and are fully documented in AN1831/D. Their addresses are fixed by the following entry in the linker command file **`[PIR:jk.lkf]`**:

+seg .ubsct -b 0x88 -v -n MONITOR_RAM

mon_data.o

NOTE: *The -v switch, instructs the linker not to report overlap errors for this segment.*

Section 3. Infrared Communications Protocol

3.1 Contents

3.2 Introduction

The passive infrared (PIR) detector uses an infrared (IR) protocol to communicate and allow calibration for sensitivity parameters. The IR communications is 2-way half duplex for example, the PIR detector can receive and transmit messages to the REMOTE control. The REMOTE unit is the master device as it initiates all communications. The infrared communications is based on a pulse-coded modulation (PCM) 38-kHz signal with a 50 percent duty cycle. The square wave shown in **Figure 3-1** needs to be generated.

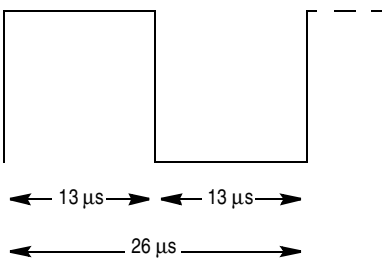


Figure 3-1. 38-kHz Timing

The 38 kHz is produced on PTD.7, the timing is achieved using in-line ‘nop’ delays, the C routines that produce the digital 1 and 0 levels are *pir\serial.c->Send_0( )* and *pir\serial.c->Send_1()* (see **[PIR:serial.c]**). **Figure 3-1** shows the 38-kHz timing to generate a logic 0/1 also illustrated is the infrared sensor output on receipt of the generated bit value. Since there are infrared transmission and receive features, the infrared sensor will *receive* what it is transmitting via PTD.4. To prevent decoding of this data, the capture interrupt is disabled during an infrared transmission.

NOTE: ***Figure 3-1*** *assumes no timing delay.*

The incoming infrared sensor output is fed into the PIR unit’s timer channel 0 pin (PTD.4), and the bit logic level determination is done in the timer channel 0 interrupt routine, *pir\interrupt.c->TIMERCHANNEL0()* (see **[PIR:interrup.c]**). From **Figure 3-1** a bit value is determined from the time between a rising edge and the corresponding falling edge. The pulse width of a logic 0 is approximately 700 µs whereas that of a logic 1 is three times that at approximately 2.1 ms. A flowchart depicting the IR interrupt code sequence is shown in **Figure 3-2**.

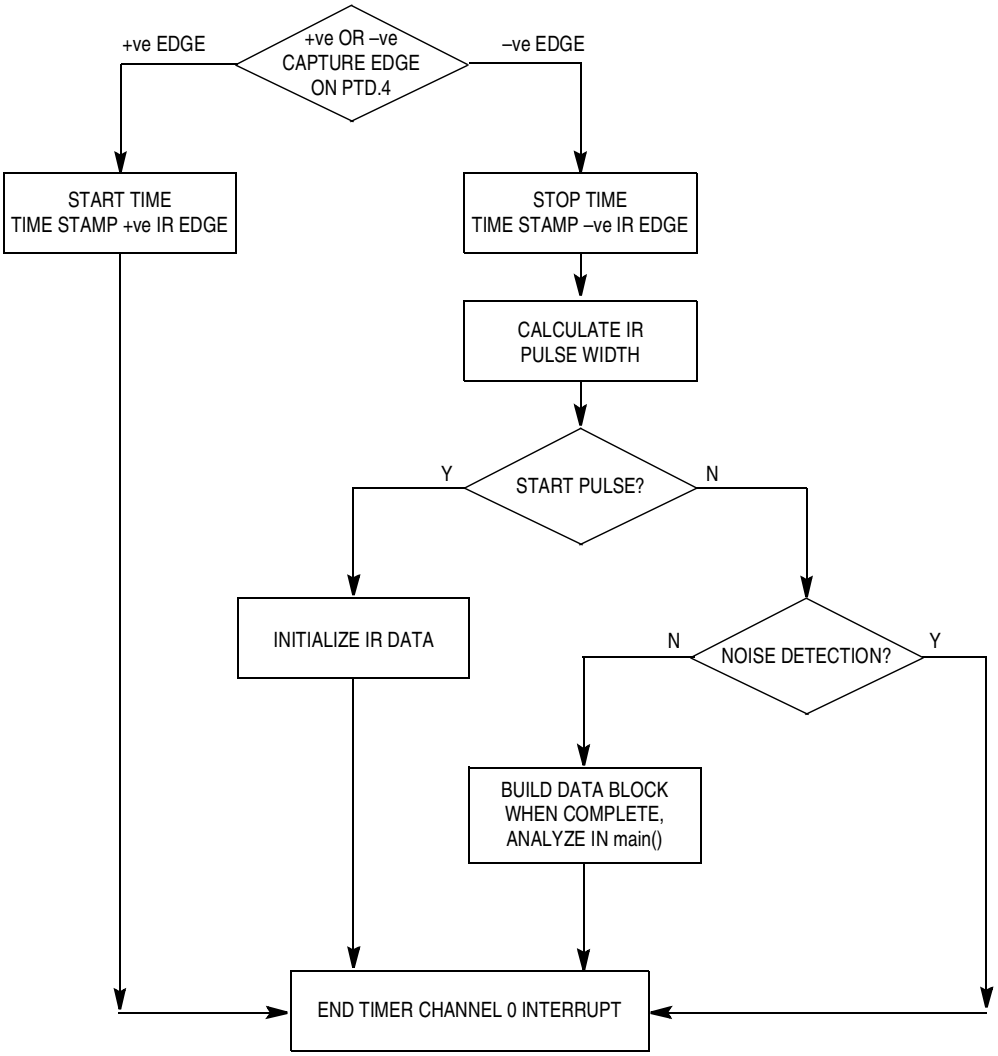


Figure 3-2. Timer Channel 0 Capture Interrupt for PIR Unit Flowchart

The infrared data is sent in a packet structure similar to that for the PC RS232 communications. The packet structure consists of:

START PULSE	BLOCK LENGTH	BLOCK TITLE	DATA BYTE 1	DATA BYTE 2	DATA BYTE n	CHECK-SUM HI	CHECK-SUM LO	STOP PULSE

START PULSE	A 4-ms synchronizing waveform
BLOCK LENGTH	Number of bytes in the packet, excluding the checksum
BLOCK TITLE	Byte representing what type of data packet
DATA	Data bytes
CHECKSUM HI/LO	Bytes refer to the 16-bit sum of: BLOCK LENGTH + BLOCK TITLE + DATA BYTE1 + DATA BYTE2 + ...+ DATA BYTE n.
STOP PULSE	Final negative edge for pulse width calculations

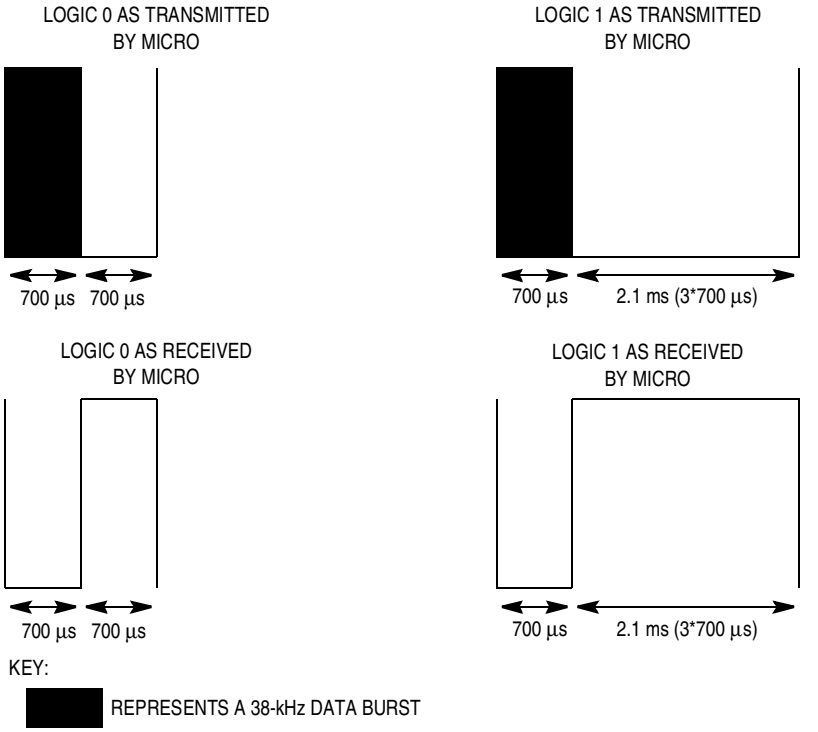


Figure 3-3. Infrared Communications Timing

3.3 Infrared Hardware Description

The IR data is transmitted via a power transmission diode with a transmissive wavelength, which matches the receiver. The receiver used in this reference design has a spectral response which peaks at 1000 nm. The operating range of the transmitter is proportional to the current used to drive the diode. The current is set by the value of R_{CL}, which is 10R on the development board, giving a peak operating current in the order of 350 mA and a transmissive distance of approximately 3 metres. Transistors Q1 and Q2 provide the high current gain necessary to drive the infrared transmitter light-emitting diode (LED). Q2 must have a suitable peak collector rating for the current set by R_{CL}. The data is produced and modulated by the microcontroller (MCU) at a frequency of 38 kHz. This frequency was selected to enable the use of industry standard, low-cost receivers, commonly used in video recorders, TVs, etc.

Adequate decoupling of the supply lines is essential if the IR data transmission circuitry is employed, as the peak current through the IR diode is high when data is being transmitted. Therefore, it is important to keep the PCB traces as short as possible between the supply pin of the regulator and the IR diode. This is also true of the return 0-V line; otherwise, “ground lift” may occur, causing spurious data loss, reset or other problems. It is suggested that the power supply and return traces to the IR transmitter are separated from the traces supplying the microcontroller and other peripherals. The circuit diagram described here is shown in **Figure 3-4**.

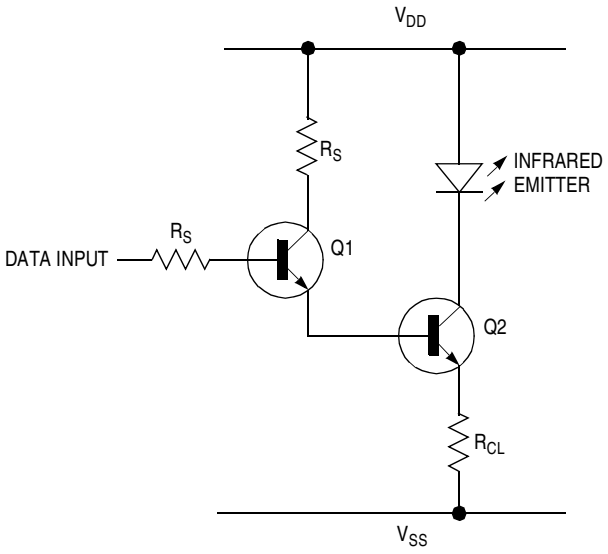


Figure 3-4. IR Transmitter Circuit

3.4 IR Receiver

The 38-kHz modulated IR data transmitted by the main PIR board is received and demodulated by IR1. This is a self-contained IR detector, amplifier, and demodulator unit, which recovers the original data in a form compatible with the microcontroller input. The device used in development is a GP1U28Q and only requires the provision of +5 V and 0 V to operate.

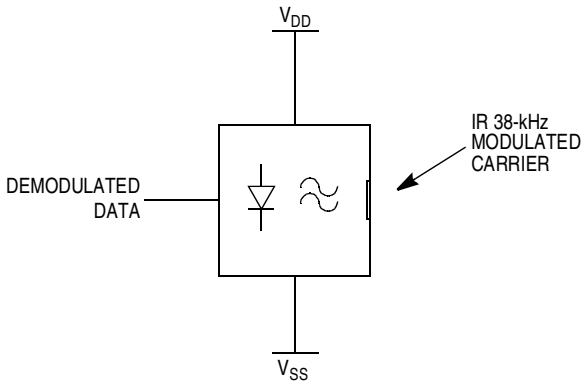


Figure 3-5. IR Receiver Circuit

Section 4. REMOTE Control Unit

4.1 Contents

4.2 Introduction

The REMOTE control unit enables the passive infrared (PIR) unit to be programmed with parameter adjustments via half-duplex infrared communications.

The REMOTE being the master by virtue that it initiates all communication events. This communications allows the REMOTE to interrogate and command the PIR to reprogram its detection parameters with the REMOTE control unit's adjusted parameters.

The parameter to be adjusted is obtained by pressing the associated button (B1/B5). An infrared (IR) communications packet is sent to the PIR unit requesting its *current value* for that parameter. Once decoded, the received parameter value is displayed on the liquid crystal display (LCD) screen for adjustment via the INC/DEC buttons. When the adjustment is complete the ENTER button is pressed, sending the new required parameter (via IR communication) value back to the PIR unit to replace its current parameter value with this new value. If the reprogramming is successful, the PIR unit will light its detect light-emitting diode (LED) for 250 ms.

4.3 Password Protection

A password protection scheme prevents unauthorized use of the REMOTE. This comprises a 5-digit decimal number with 0 to 9 being the range of entries. The 5-digit decimal number provides 99,999 possible passwords. The password is entered using the double function keys giving 0...9. If the password has been correctly entered, the buttons lose their numeric assignments.

4.4 Hardware Description

Although the hardware functionality of this board is biased toward the support of the PIR sensor board, it has been designed to be as generic as possible so that the software may be modified to perform many other functions requiring the transmission and reception of data via a remote infrared link.

The circuit diagram may be conveniently divided into functional blocks, most of which may be included or omitted as required for a particular design. This gives designers flexibility to include only the features required for the application.

The blocks are:

- Power supply unit (PSU)
- Microcontroller, crystal, and phase-locked loop (PLL)
- Keyboard
- Liquid crystal display (LCD)
- IR data transmit (DTX)
- IR data receive (DRX)
- Real-time clock (RTC)
- Digital potentiometer for LCD contrast adjustment
- Serial communications to PC (RS232)
- Stand-alone MC68HC908GP32 programmer

4.5 Button Designations

There are provisions for 15 buttons, although not all buttons are used, to implement any additional button functionality. Code will need inserting into the unused *case* statements in *pir\button.c->DecodeButtons()->StandardButtons()* (see **[REMOTE:button.c]**). The layout of the buttons with respect to the printed circuit board (PCB) is shown in **Figure 4-1**.

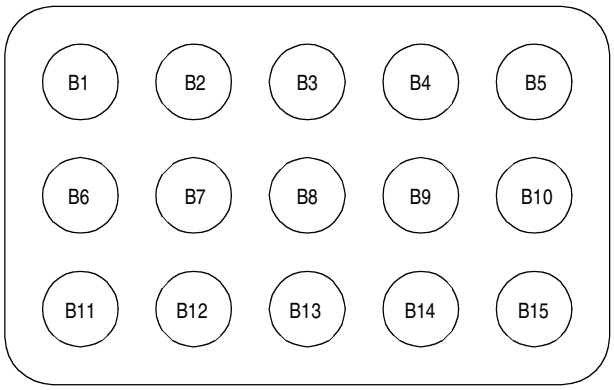


Figure 4-1. Button Assignments on PCB

The buttons during **password** entry are assigned as:

B1 Insert 1 at the current LCD cursor position

B2 Insert 2 at the current LCD cursor position

B3 Insert 3 at the current LCD cursor position

B4 Insert 4 at the current LCD cursor position

B5 Insert 5 at the current LCD cursor position

B6 Insert 6 at the current LCD cursor position

B7 Insert 7 at the current LCD cursor position

B8 Insert 8 at the current LCD cursor position

B9 Insert 9 at the current LCD cursor position

B10 Insert 0 at the current LCD cursor position

B11 Not used

B12 Not used

B13 Not used

B14 Not used

B15 ENTER, accept current password for verification

After the password has been successfully entered, the buttons then change functionality to:

B1 IR command to PIR unit for Delta Sigma event difference value

B2 IR command to PIR unit for Delta Sigma bit resolution value

B3 IR command to PIR unit for A2D 10-ms loop time value

B4 IR command to PIR unit for A2D buffer difference value

B5 IR command to PIR unit for A2D trigger count value

B6 Force real-time clock (RTC) to **Mon 01 Jan 2001 at 00:00:00**

B7 Not used

B8 Not used

B9 Not used

B10 Not used

B11 Increment current PIR FLASH parameter/LCD contrast

B12 Decrement current PIR FLASH parameter/LCD contrast

B13 LCD contrast adjust

B14 CANCEL, abort current LCD screen and revert back to time of day (TOD)

B15 ENTER, accept current LCD value and instruct PIR to reprogram with this value.

4.6 Pin Assignments

The REMOTE software has been written for a MC68HC908GP32CP, the 32 K of user code is approximately 25 percent utilized, and the input/output (I/O) count was the main requirement. The port pin assignments for the REMOTE control unit are:

PTA	PTA0	Row 1 button input
	PTA1	Row 2 button input
	PTA2	Row 3 button input
	PTA3	Row 4 button input
	PTA4	Row 5 button input
	PTA5	Column 1 button select
	PTA6	Column 2 button select
	PTA7	Column 3 button select
PTB	PTB0	RTC I^2C clock
	PTB1	RTC I^2C data
	PTB2	Digipot chip select
	PTB3	Digipot up/down
	PTB4	Digipot INC
	PTB5	LCD RS
	PTB6	LCD RW
	PTB7	LCD E
PTC	PTC0	LCD DATA0
	PTC1	LCD DATA1
	PTC2	LCD DATA2
	PTC3	LCD DATA3
	PTC4	LCD DATA4
PTD	PTD0	LCD DATA5
	PTD1	LCD DATA6
	PTD2	LCD DATA7
	PTD3	IR Comms TX
	PTD4	IR Comms RX
	PTD5	SPARE
PTE	PTE0	RS232 TX
	PTE1	RS232 RX

4.7 Program Flow

The software on reset performs preparatory tasks, such as initializing the PTA keyboard interrupt facility, and ensures the LCD screen is off, then it enters stop mode. On recovery from stop mode via any button press, the on-board PLL is initialized for 2.4576-MHz bus operation and the LCD is initialized.

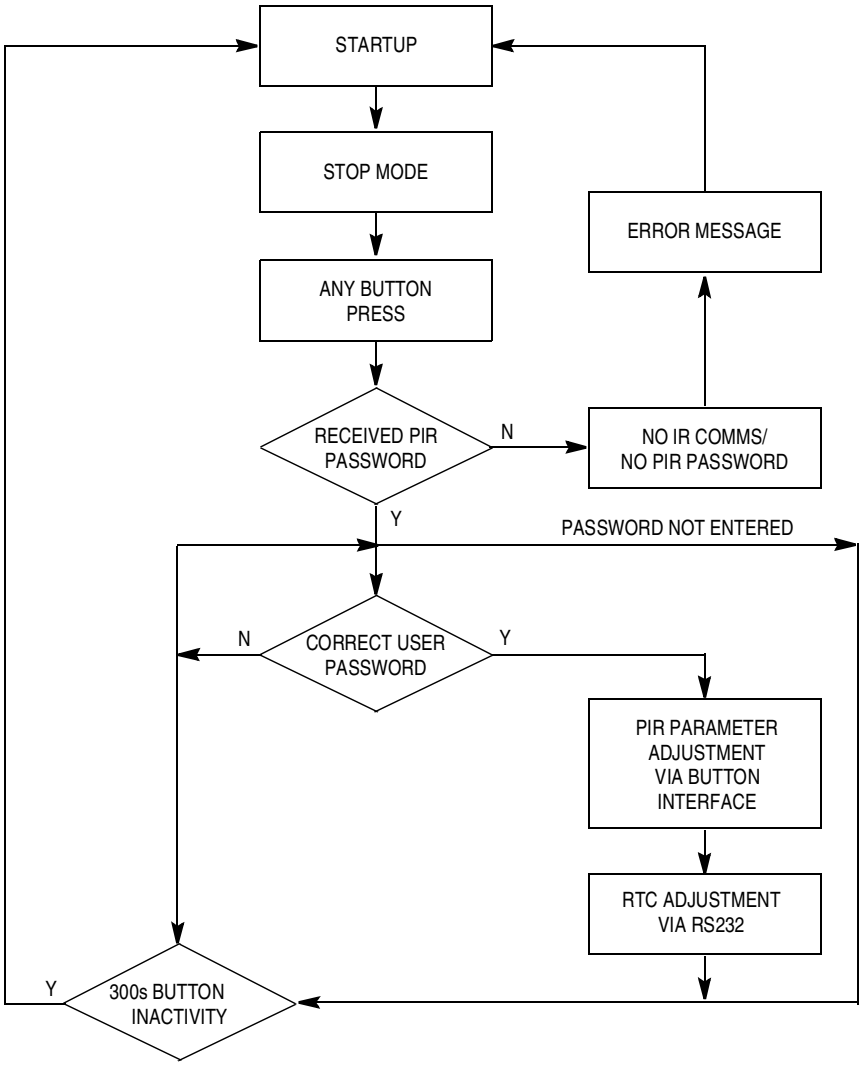


Figure 4-2. REMOTE Control Unit Top Level Functionality Flowchart

4.7.1 Run Time LCD Screen Flow

The REMOTE control unit's first task is to obtain the PIR password. It requests this from the PIR unit via the half-duplex IR communications. The LCD will show:

```
Transmitting IR
comms packets
```

The REMOTE will request this information 40 times
(300 ms * 40 = 12 s). If this fails, it can be due to one of two reasons.

1. No received IR communicationsError 1
2. No received passwordError 2

The LCD will show:

```
Error 1        5
No IR Comms [IN]
```

or

```
Error 2        5
No PIR Password
```

These errors are such that the program cannot continue, and error message screens are displayed with a 5 second count down. Upon error timeout, the REMOTE returns to stop mode and the user can retry.

If the REMOTE receives and correctly decodes the PIR password, the LCD will show:

```
Enter password:
      XXXXX
```

The user now has to enter the matching password to that received from the PIR. All fields have to be completed since the expected password is five digits. The ENTER button does not respond until all the initial **X** characters have been over written. When all **X** have been over written,

the password can be submitted with the cursor on any character. When entering a button (numeric), the LCD cursor moves to the right one position and auto wraps when the fifth password number has been entered.

When ready, press ENTER, the REMOTE will now compare the entered password value to that received from the PIR. If a match is found, then the LCD will show these screens for one second and then revert to showing the time of day:

```
Password
Accepted!
```

```
Fri 06 Oct 2000
     18:19:20
```

This is the default viewing mode, for example, time of day (TOD).

If the password attempt failed, the LCD will show:

```
Password
Rejected!
```

This text will be visible for one second before returning to the password entry screen:

```
Enter password:
      XXXXX
```

You will iterate around this loop until the password is correctly entered. With the password consisting of five decimal digits, the maximum number of individual retries will be 99,999; to reduce the security risk, the password digit count could be increased.

4.7.2 Adjustable FLASH Parameters

These five screens show a typical LCD screen content for the five adjustable PIR parameters.

Produced by pressing B1:

```
Delta Sig Event:
       350
```

Produced by pressing B2:

```
Delta Sig Res'n:
       12
```

Produced by pressing B3:

```
A2D Loop Time:
       10
```

Produced by pressing B4:

```
A2D Difference:
       6
```

Produced by pressing B5:

```
A2D Trigger:
       4
```

When the variable of choice is displayed (by pressing appropriate button **B1:B5**) it is adjusted using the INC/DEC buttons. This operation simply adjusts a local copy of the value received from the PIR. The adjustment can be discarded by pressing the CANCEL button, which will return to TOD (time of day) mode.

NOTE: *When adjusting the Delta Sigma event parameter, the min/max and step values are constrained by a const data declaration in* ***[REMOTE:data.c]****:*

```
////////////////
// const data //
////////////////
@near const struct sDELTA_SIGMA_ADJUST ds_adjust[8] =
   {
   {5  , 255  , 5}  ,      // 8  bit min, max, step
   {20 , 500  , 20} ,      // 9  bit min, max, step
   {50 , 1000 , 50} ,      // 10 bit min, max, step
   {100, 2000 , 100},      // 11 bit min, max, step
   {200, 4000 , 200},      // 12 bit min, max, step
   {400, 8000 , 400},      // 13 bit min, max, step
   {600, 16000, 600},      // 14 bit min, max, step
   {800, 32000, 800},      // 15 bit min, max, step
   };
```

Consequently, the parameters can be adjusted if required. The LCD connections are shown in **Figure 4-3**. The program flow is shown in **Figure 4-4**, and all operational paths are shown.

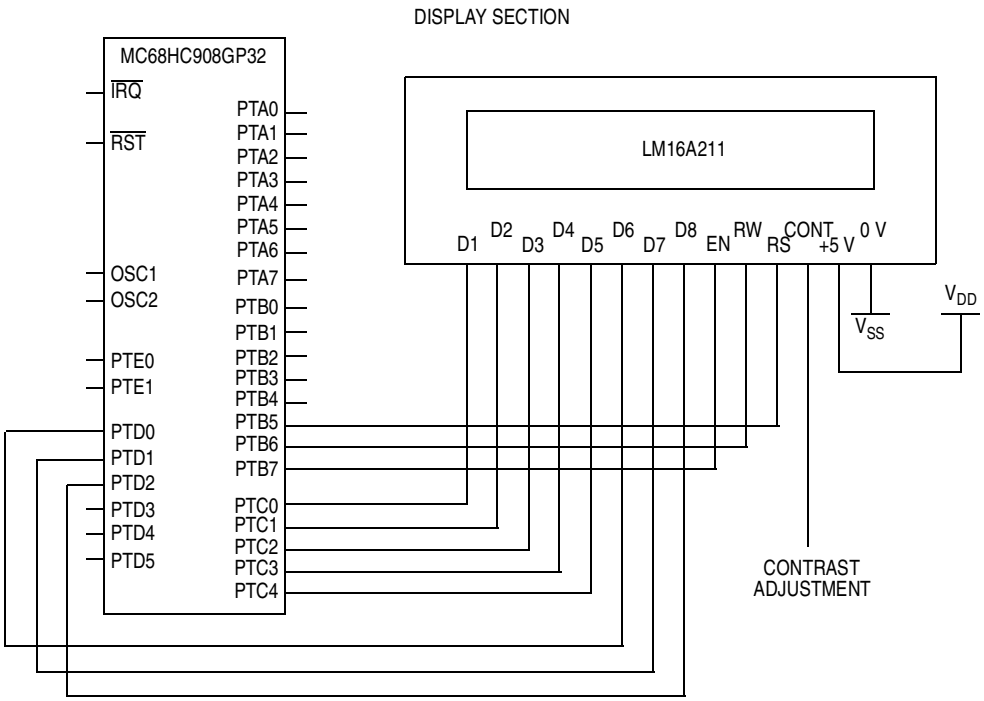


Figure 4-3. LCD Connections to MC68HC908GP32

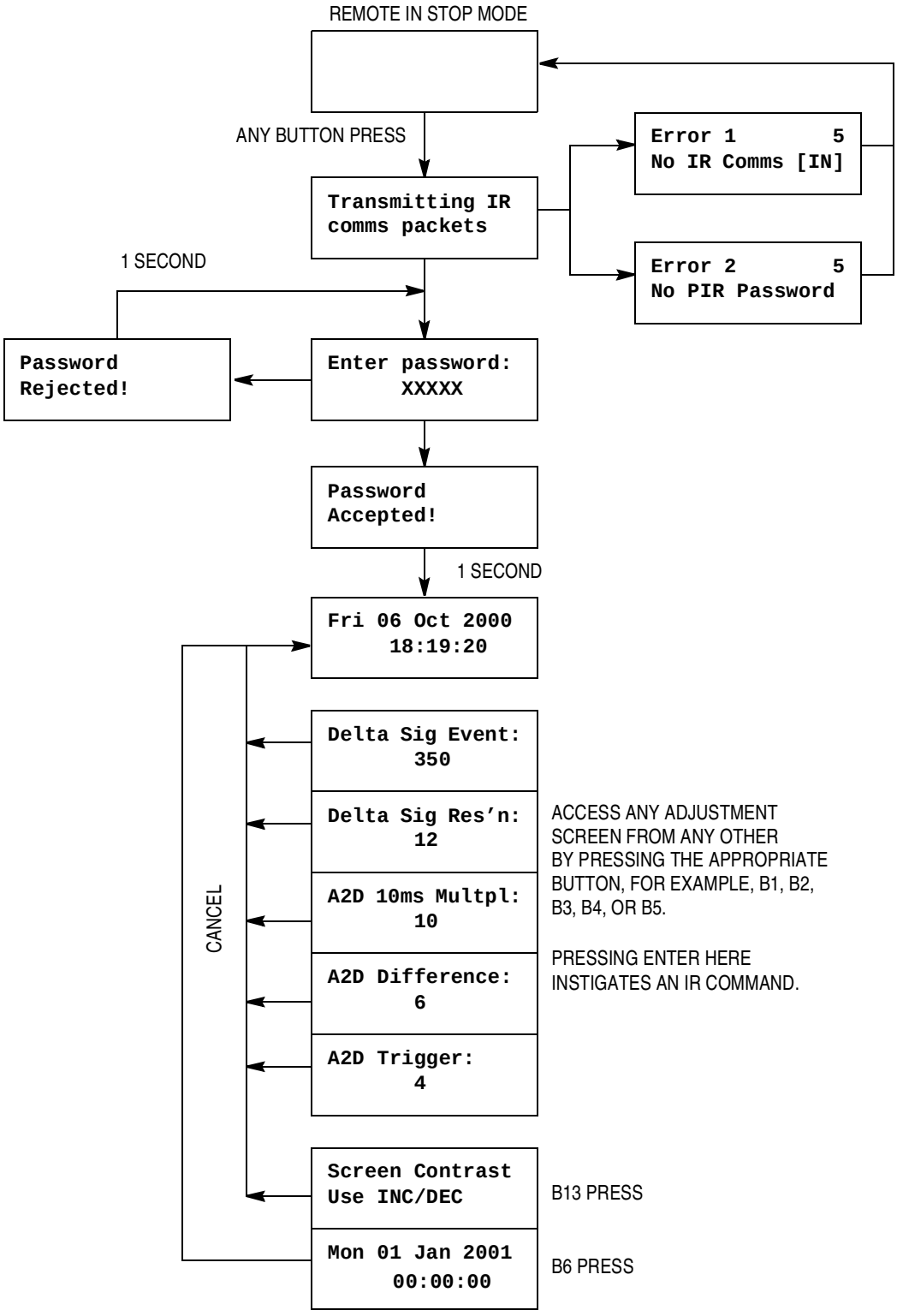


Figure 4-4. LCD Screen Functional Flowchart

4.7.3 Button Press Determination

All button activity is on PORTA data register (PTA/$0000), a matrix scan method is implemented which minimizes the number of input/output (I/O) required. The matrix used is a 5 x 3 providing up to 15 buttons using three columns and five rows. The buttons are read every *remote\main.c->main()* (see ***[REMOTE:main.c]***) loop iteration with *remote\button.c->ReadButtons()* (see **[REMOTE:button.c]**), this occurs every 10 ms.

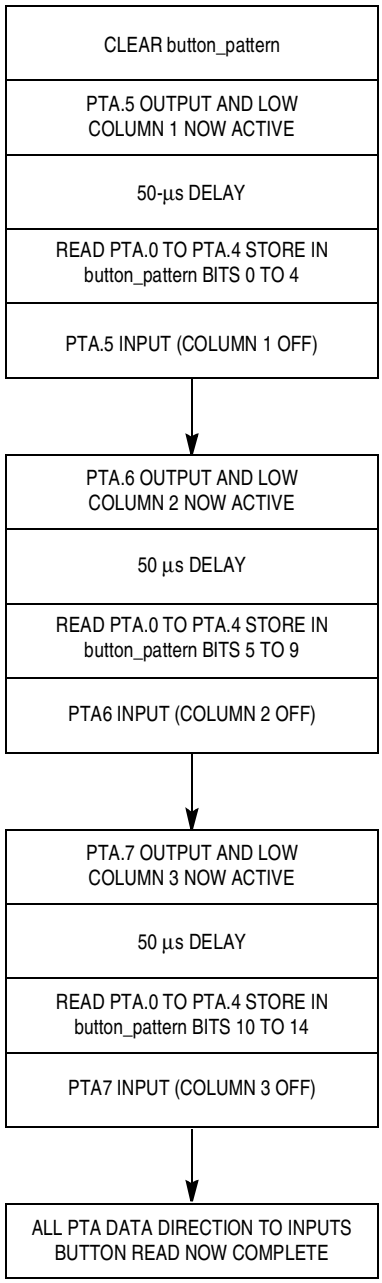
Figure 4-5 shows the linear method of activating columns and reading rows. As PORTA internal pullups are enabled, if a column is driven low, and on reading the row input lines, a row line is low, then a button is being pressed (since the other two driver columns are input).

The default (no button pressed) value of button_pattern will be 0xFFFF. For example, if button B5 is being pressed, this will force PTA.0 to be low since the column driver is being driven as an output and low, which will produce a value for button_pattern of 0xFFFE. Similarly, the value of button_pattern while B10 is being pressed will be 0xFFDF. The column driver being active determines which bit range of button_pattern is set:

Column 1: Bits 0/1/2/3/4 of button_pattern

Column 2: Bits 5/6/7/8/9 of button_pattern

Column 3: Bits 10/11/12/13/14 of button_pattern



Note: button_pattern is a 16-bit variable.

Figure 4-5. REMOTE Control Unit Button Read Flowchart

The correlation of a button press to the 16-bit variable ‘button_pattern’ is shown in **Figure 4-6**.

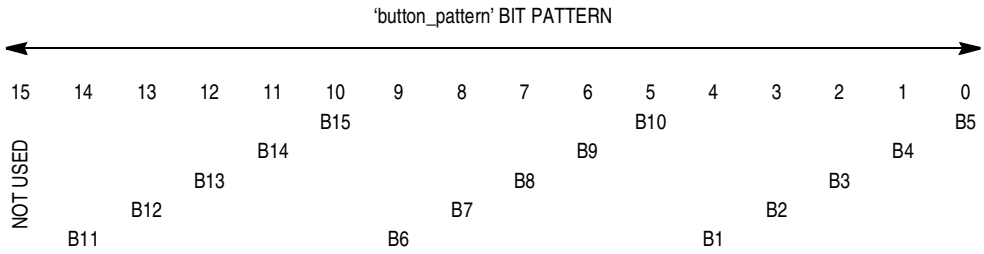


Figure 4-6. Button Press to ‘button_pattern’ Correlation

By studying the bit pattern shown in **Figure 4-6**, the software button decode mapping (shown next) can be understood. The button decode map is contained in **[REMOTE:button.h]**.

```
#define  DEFAULT_BUTTONS      0xffff
#define  BUTTON_1             0xffef
#define  BUTTON_2             0xfff7
#define  BUTTON_3             0xfffb
#define  BUTTON_4             0xfffd
#define  BUTTON_5             0xfffe
#define  BUTTON_6             0xfdff
#define  BUTTON_7             0xfeff
#define  BUTTON_8             0xff7f
#define  BUTTON_9             0xffbf
#define  BUTTON_10            0xffdf
#define  BUTTON_11            0xbfff
#define  BUTTON_12            0xdfff
#define  BUTTON_13            0xefff
#define  BUTTON_14            0xf7ff
#define  BUTTON_15            0xfbff
```

4.7.4 Button Debouncing and Functional Decode

Now that a button press can be determined, a button debounce and decode algorithm needs to be implemented.

The algorithm used incorporates a button press *and* button release debounce. The ability to have an auto scroll is included, and it occurs when a button is pressed and debounced but remains pressed. This condition will occur while performing an adjustment of a PIR parameter value, by a single press and hold of the INC/DEC button. The auto scroll feature can be enabled/disabled to any button as required. The flag that allows this feature is button_flags.bit.AUTO_SCROLL. It is set to a 1 to enable and 0 to disable this auto scroll feature.

Button connections are shown in **Figure 4-7** and a button algorithm flowchart in **Figure 4-8**.

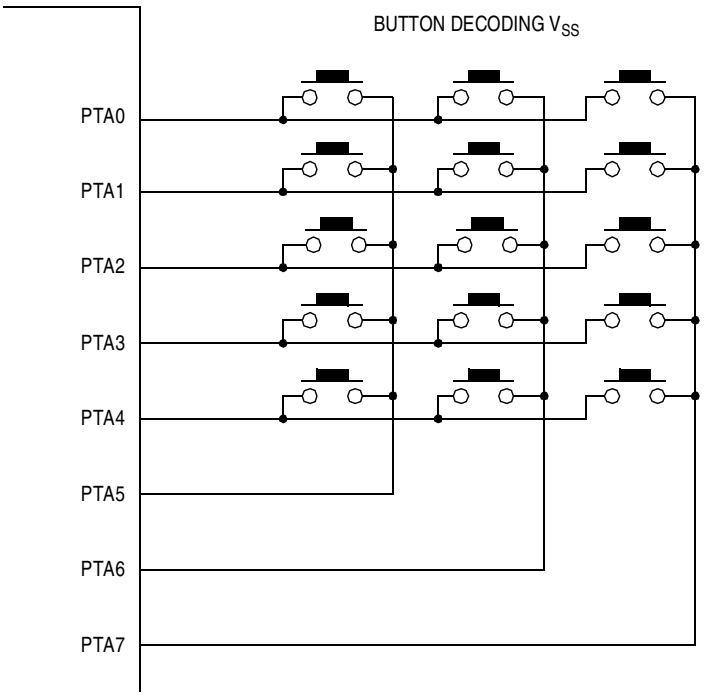


Figure 4-7. Button Connections

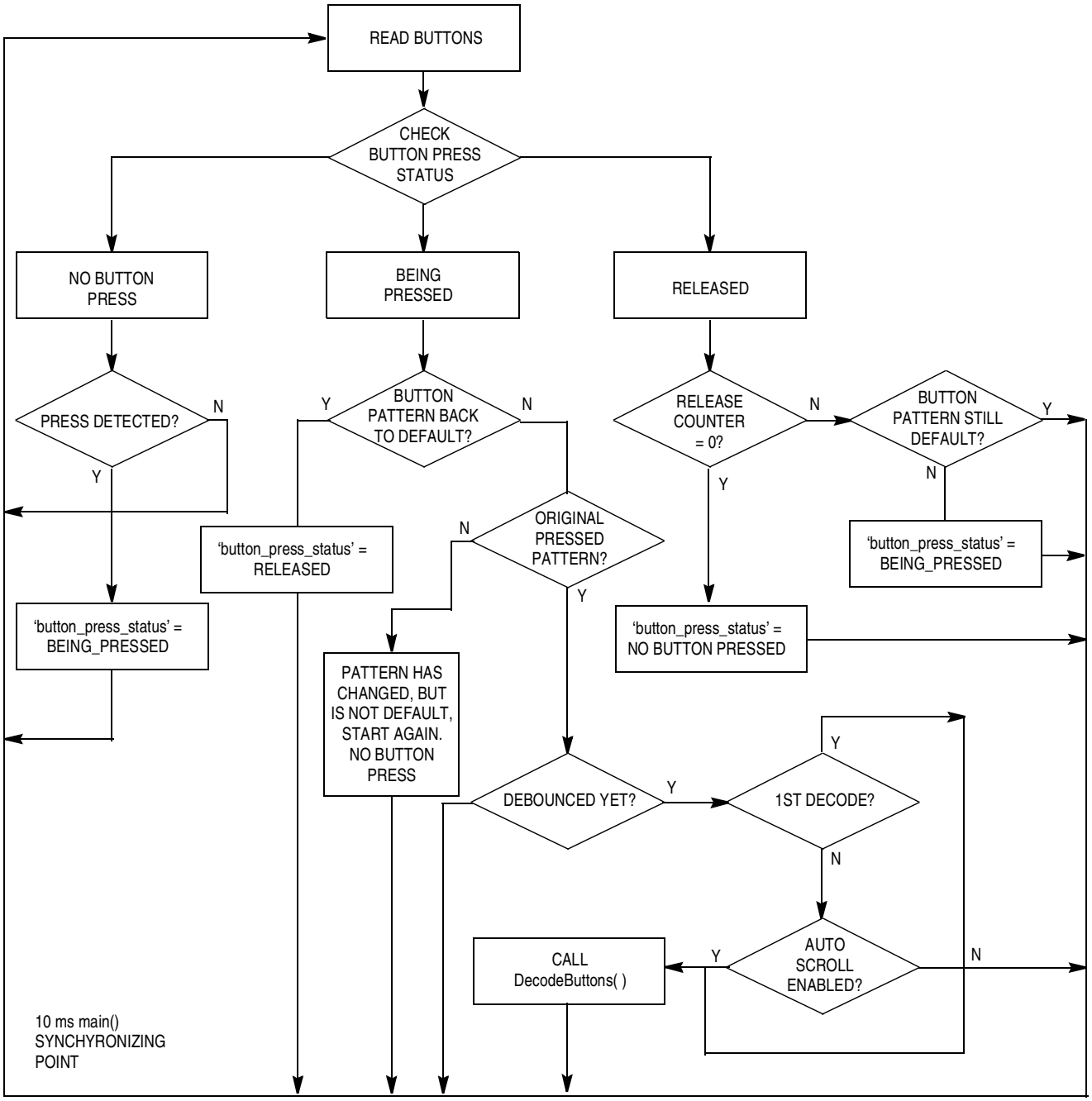


Figure 4-8. REMOTE Button Algorithm Flowchart

4.8 LCD Text Writing

The LCD used for the REMOTE is the Sharp LM16A211, is a 2 by 16 character textual display. The MC68HC908GP32 drives the display with an 8-bit data bus and three control lines. The software used for driving this display is contained in **[REMOTE:lcd.c]**.

The screen text write functions used are remote\lcd.c->*WriteText1()* and *remote\lcd.c->WriteText2()* (see **[REMOTE:lcd.c]**).

WriteText2() is used to write a supplied text string to the display. For example:

```
WriteText2( LINE1, “FLASH BASED” , NOCLEAR );
WriteText2( LINE2, “MC68HC908JK1”, NOCLEAR );
```

will produce the following on the LCD:

```
FLASH BASED
MC68HC908JK1
```

and,

```
WriteText2( Line1+4, “FLASH BASED” , NOCLEAR );
WriteText2( Line2+2, “MC68HC908JK1”, NOCLEAR );
```

will produce:

```
    FLASH BASED
MC68HC908JK1
```

The first function parameter is the desired address you want the string to start at, the second parameter is a pointer to the string, and the third parameter determines if you want the screen line you are writing to pre-cleared. This is useful if the string you are about to write is smaller than the current screen string.

WriteText1() uses *text_buffer* as its string source, which allows us to preload *text_buffer* with *formatted* data before displaying it. An example is:

```
WriteText2( LINE2, "REFERENCE DESIGN", PRECLEAR);       // this also performs a
                                                        // 'text_buffer' pre-clear
IntegerToASCII( 12345, &text_buffer[0] );               // convert arg to ASCII string
                                                        // write at 'text_buffer[0]'
WriteText1(LINE2+6);                                    // produces "      12345     "
```

text_buffer will contain:

[0x31][0x32][0x33][0x34][0x35][0x00][0x20][0x20][0x20][0x20][0x20][0x20]
[0x20][0x20][0x20][0x20][0x20] → String terminating NULL

This will produce:

```
REFERENCE DESIGN
      12345
```

The '**1**' starts on the seventh character of the second line.

4.8.1 LCD Contrast Adjust

The screen contrast can be adjusted by pressing B13, and the following screen will be shown:

```
Screen Contrast
Use INC/DEC
```

By pressing the INC button (B11), the screen contrast will increase (for instance, the display text will get darker). Conversely, pressing the DEC button (B12) will cause the screen contrast to decrease (for instance, the display text will get lighter). This control has been provided by the use of a digital potentiometer on pins PTB.2/3/4. The software for this functionality is contained in *remote\digipot.c->DigiPot()* (see **[REMOTE:digipot.c]**).

The digital potentiometer used is a Dallas Semiconductor device, DS1804Z. It is controlled by three input lines:

- Chip select (**CS**)
- Up/down (**U/D**)
- Count and increment (**INC**)

The “wiper” element is adjusted by applying a series of pulses to the **INC** input. The direction of travel is controlled by the logical status of the **U/D** pin. Once adjusted, the position of the “wiper” is stored in its internal nonvolatile memory.

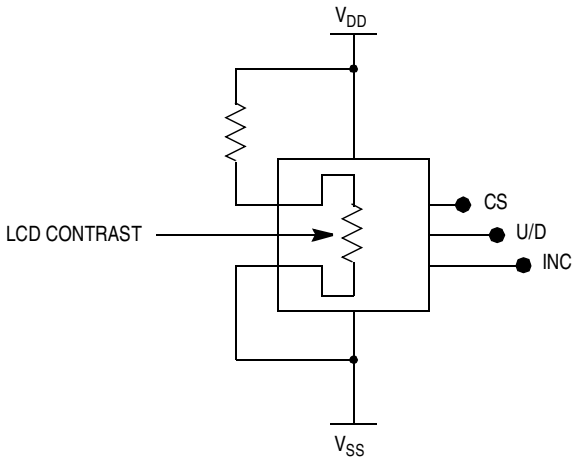


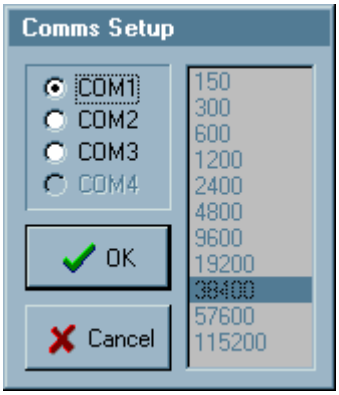
Figure 4-9. LCD Contrast Adjust Using Digital Potentiometer

4.8.2 Real-Time Clock (RTC)

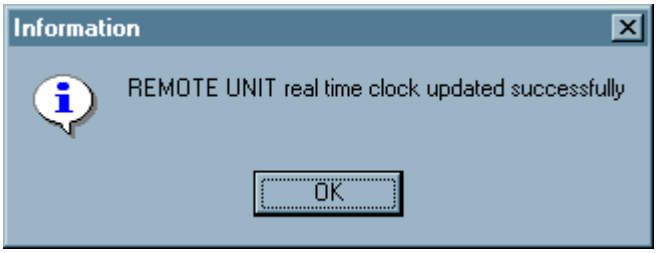
The on-board RTC (Dallas Semiconductor DS1307) is displayed on the LCD, and it can be updated from a connected PC via a RS232 connection. To reprogram, simply connect the RS232 cable, run the pir_plot program, and access:

Update->Remote RTC

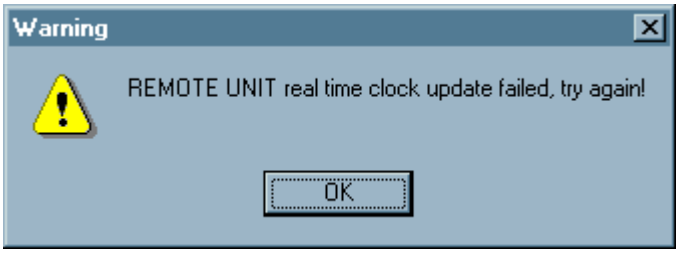
If you cannot access this menu item (for example, its greyed out), the communications port has not been selected. Exit the program and try again, this time selecting a valid communications port.



The PC will generate the appropriate data stream. On receipt of this data, the REMOTE control unit will decode and reprogram. If the update was successful, the REMOTE control unit will send an ACKNOWLEDGE to the PC and the PC will respond by displaying:



If the update failed, a warning message will be shown instead, and the reprogramming can then be retried.



4.8.3 I²C for the Real-Time Clock

The MC68HC908GP32 communicates with the Dallas Semiconductor DS1307 using the I²C protocol. If the REMOTE is in the TOD mode (time of day), then every *main()* loop iteration (10 ms) the MC68HC908GP32 interrogates the DS1307 with the function *remote\main.c->main()->ModeCheck()->UpdateTime()* (see **[REMOTE:main.c]**) to see if the seconds have changed. On the 100th interrogation (after the last change 100 * 10 ms = 1s), the display will require updating. Every time the DS1307 is interrogated, a pointer is loaded with the DS1307 current data values. Only when a second has elapsed is the new data used. This data is used to build a string using *text_buffer*; when the string is complete, it is then displayed to the user. The current time is read from the DS1307 with *remote\rtc.c->RTC_Read(SECONDS, &current_time)* (see **[REMOTE:rtc.c]**). This function is shown here.

```
void RTC_Read( unsigned char register_pointer, struct RTC *ptr )
{
////////////////////////////////////////////
// first set the internal RTC address pointer //
// to the register that you require with a    //
// WRITE command                              //
////////////////////////////////////////////
StartBit();
SendI2CByte( RTC_WRITE );
WaitForI2CAcknowledge();
SendI2CByte( register_pointer );
WaitForI2CAcknowledge();
StopBit();
```

```
////////////////////////////////////
// Then read the contents of the RTC //
// registers, with a READ command    //
////////////////////////////////////
StartBit();
SendI2CByte( RTC_READ );              // RTC_READ == 0xd1
WaitForI2CAcknowledge();
ptr->seconds = GetI2CByte();
SendI2CAcknowledge();
ptr->minutes = GetI2CByte();
SendI2CAcknowledge();
ptr->hours   = GetI2CByte();
SendI2CAcknowledge();
ptr->day     = GetI2CByte();
SendI2CAcknowledge();
ptr->date    = GetI2CByte();
SendI2CAcknowledge();
ptr->month   = GetI2CByte();
SendI2CAcknowledge();
ptr->year._8bit.lobyte = GetI2CByte();
ptr->year._8bit.hibyte = 0x20;        // century hi byte...fixed for the
                                      // next 99 years!
SET_DATA_TO_OUTPUT;                   // master sending a NOT ACK
SET_SDA;
OutClock();       // no acknowledge expected here, we generate a clock pulse
StopBit();
}  // RTC_Read()
```

As described before, the on-board real-time clock (RTC) can be updated from a PC using the ‘pir_plot.exe’ Windows® program. This sends the current PC time to the MC68HC908GP32, and the MC68HC908GP32 receives the data via the universal asynchronous receiver/transmitter (UART) receive interrupt at *remote\interrupt.c->SCI_RECEIVE()* (see **[REMOTE:interrup.c]**). When all data has been received, a flag is set to cause full checksum analysis from *remote\datasort()->RS232CommsCheck()* (see **[REMOTE:datasort.c]**). *RS232CommsCheck()* performs the RS232 receive data acceptance processing and text string formatting. If data checksum matching occurs, the real-time clock is rewritten with the new data. This takes place in the call to *remote\rtc.c->SetRTC(&new_time)* (see **[REMOTE:rtc.c]**) from *remote\main.c->main()->RS232CommsCheck()*.

```
unsigned char SetRTC( struct RTC *ptr )
{
struct RTC     compare;
unsigned char  error_count;

StartBit();                                 ////////////////////////////////
SendI2CByte( RTC_WRITE );                   // RTC_WRITE == 0xd0          //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( SECONDS );                     // point to seconds register  //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->seconds );                // seconds                    //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->minutes );                // minutes                    //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->hours );                  // hours                      //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->day );                    // day                        //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->date );                   // date                       //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->month );                  // month                      //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->year._8bit.lobyte );      // year                       //
WaitForI2CAcknowledge();                    //                            //
StopBit();                                  ////////////////////////////////

////////////////////////////////////
// now to read what's been written //
////////////////////////////////////
RTC_Read( SECONDS, &compare );

error_count = 0;

if ( compare.year._8bit.lobyte != ptr->year._8bit.lobyte )  error_count++;
if ( compare.month             != ptr->month             )  error_count++;
if ( compare.date              != ptr->date              )  error_count++;
if ( compare.day               != ptr->day               )  error_count++;
if ( compare.hours             != ptr->hours             )  error_count++;
if ( compare.minutes           != ptr->minutes           )  error_count++;
if ( compare.seconds           != ptr->seconds           )  error_count++;

if ( !error_count )
   {
   return 1;   // success
   }

return 0;      // failed
}  // SetRTC()
```

Notice the *read* after the *write*. Full agreement is checked for before a successful function return. All the RTC reading/writing operations are built with the lower level I²C routines contained in **[REMOTE:i2c.c]**.

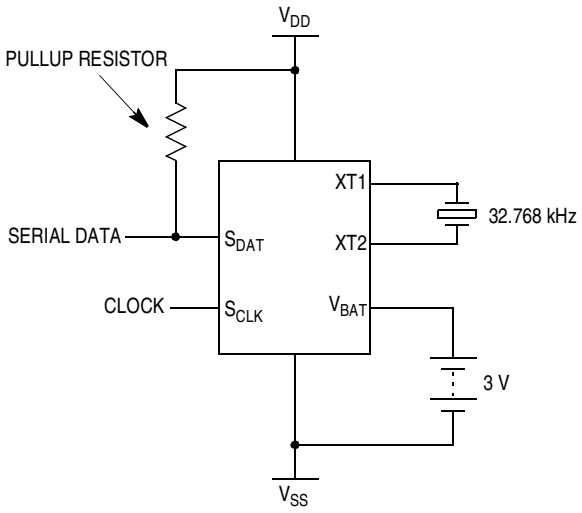


Figure 4-10. Real-Time Clock, Dallas Semiconductor DS1307 Connections

4.8.4 Forcing the Real-Time Clock (RTC) to a Known State

The RTC can be forced to a known state by pressing B6. If the programming is successful, then this screen will be shown:

```
Mon 01 Jan 2001
    00:00:00
```

The RTC will begin operation from this reset value.The decoding of button B6 calls *remote\rtc.c->ForceRTC()* (see **[REMOTE:rtc.c]**).

4.9 REMOTE Software Files

This software has been written using the Cosmic C Cross Compiler. All files for the REMOTE control unit are listed here.

- Assembler:
  [REMOTE:crtsi.s]
- C Source:
  [REMOTE:button.c], **[REMOTE:convert.c]**, **[REMOTE:data.c]**, **[REMOTE:datasort.c]**, **[REMOTE:delay.c]**, **[REMOTE:digipot.c]**, **[REMOTE:error.c]**, **[REMOTE:i2c.c]**, **[REMOTE:interrup.c]**, **[REMOTE:ir_comms.c]**, **[REMOTE:lcd.c]**, **[REMOTE:main.c]**, **[REMOTE:mode.c]**, **[REMOTE:rs_comms.c]**, **[REMOTE:rtc.c]**, **[REMOTE:startup.c]**, and **[REMOTE:vectors.c]**
- Include Files (in addition to the C source matching header file):
  [REMOTE:declared.h], **[REMOTE:define.h]**, **[REMOTE:extern.h]**, and **[REMOTE:gp32.h]**
- Compile/Link/Make:
  [REMOTE:cc.bat], **[REMOTE:link08.bat]**, **[REMOTE:make08.bat]**, **[REMOTE:config.dat]**, and **[REMOTE:gp32.lkf]**

4.9.1 On-Board MC68HC908GP32 40-Pin Dual in-Line Programmer

The programming hardware is compatible with the ICS08JLZ software from P&E Microcomputer Systems, Inc. The program used is *ics08gpz_version_1_32A.exe*, available from their Web site at:

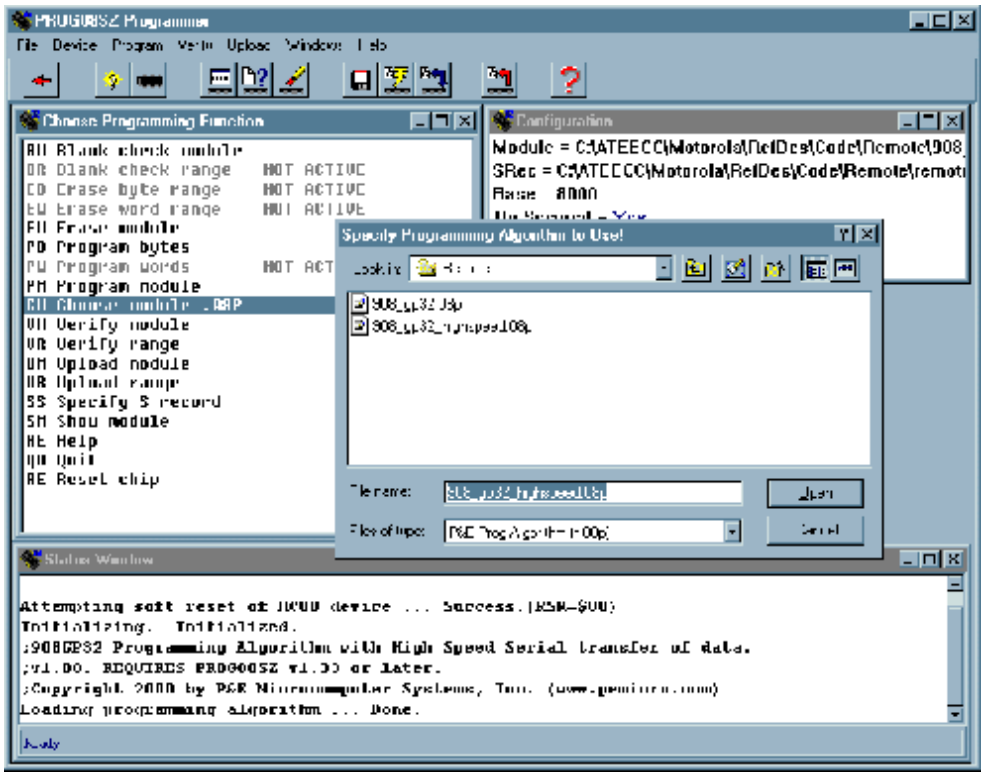
http://www.pemicro.com

Within the P&E development environment is **prog08sz.exe**, the software that interfaces to the programming socket.

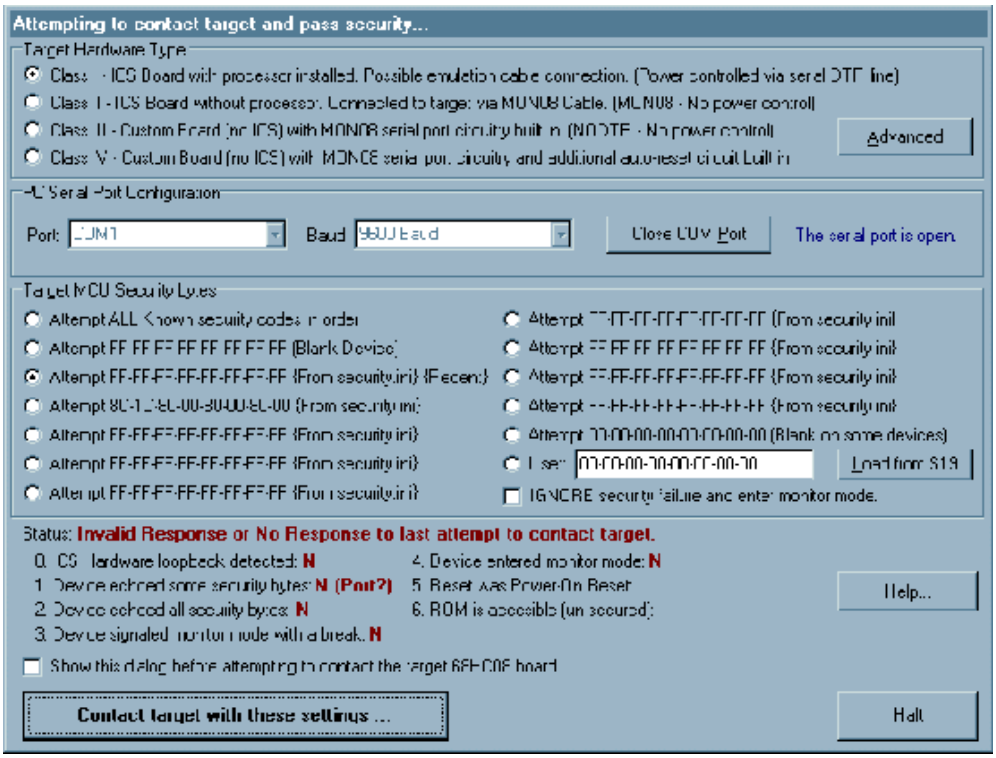
Programming procedure:

1. Ensure that the **V_{DD}** switch is **Off**.
2. Ensure that the 40-pin **Programmer** socket is occupied.
3. Ensure that a standard 9-way RS232 cable is connected from the PC to the development board's **Programmer** RS232 connector.
4. Set the **V_{DD}** switch to **On**.
5. Invoke c:\pemicro\ics08gpz**prog08sz.exe** (assuming default installation directory).
6. After programming is complete, set the **V_{DD}** switch to **Off**.
7. Remove the programmed device.

If the socketed MC68HC908GP32 passes the security test and the RS232 comms link is working, then you will see the following screen. It is asking for the programming algorithm to be entered.



If there is a problem you will see:



The above screen typically occurs if the hardware RS232/power connections are wrong or if the socketed MC68HC908GP32 fails the security test.

4.9.2 Security Failure

The security check is a mechanism to prevent unauthorized access to the MC68HC908GP32 FLASH array. The security check centers around the interrupt vector address values at $FFF6-$FFFD. Before access is granted the PC program must transmit eight bytes that need to agree with those resident in the microcontroller.

If the 8-byte comparison fails, then FLASH access is prevented. Even though monitor mode can still be entered, before you can reprogram the MC68HC908GP32 or view its contents, you will need to completely erase it. The program will remember the last S19 file programmed into a MC68HC908GP32 and use that file to pass the security test on next invocation.

NOTE: *If the MC68HC908GP32 fails the security test, the device must be powered down before a retry can be attempted. This power cycle will take the form:*

1. **V_{DD}** switch to **Off**
2. Wait for at least two seconds.
3. **V_{DD}** switch to **On**

The program c:\pemicro\ics08gpz**prog08sz.exe** can now be retried.

4.9.3 Programming Circuit

The MC68HC908GP32 programmer is configured in a modified form to that recommended in the *MC68HC908GP32 Technical Data*, Motorola document order number MC68HC908GP32/H REV. 4. It uses two sections of a three-state buffer to control the direction of data to and from the device being programmed. Both V_{DD} and V_{PP} supplies are applied via PCB mounted switches. The $\overline{\text{RESET}}$ pin of the MC68HC908GP32 is driven directly by the DTR line (pin 4:COM port) of the PC, via an inverter and level shifting circuit. When the programming supply is not present, data isolation is achieved using a digital transistor. This is used to detect the presence of the programming V_{DD} supply and controls the output of a third section of the inverting buffer. The data input terminal of the MC68HC908GP32 is thus isolated when the V_{DD} supply is removed.

See **Figure 4-11** for the MC68HC908GP32 monitor mode connections.

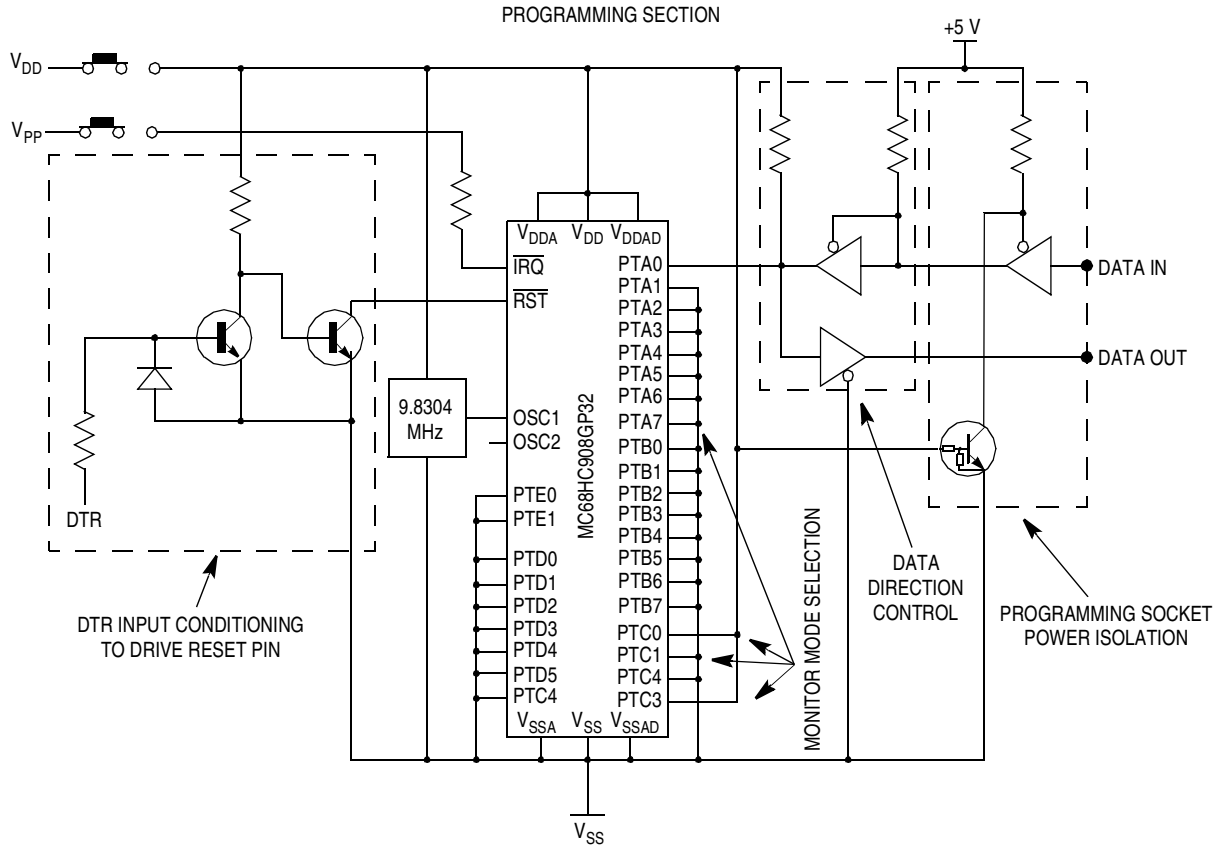


Figure 4-11. MC68HC908GP32 Monitor Mode Connections

Section 5. Phase-Locked Loop (PLL) Initialization

5.1 Contents

5.2 Introduction

The phase-locked loop (PLL) feature of the MC68HC908GP32 enables a 32.768-kHz low-cost crystal to be used to obtain a bus frequency of 2.4576 MHz. The main reason for using this bus speed is to provide the 38,400 bit rate for the hardware universal asynchronous receiver/transmitter (UART), which is used for communicating with the PC for updating the REMOTE real-time clock (RTC).

The internal 2.4576 MHz is obtained by using the values as recommended in **Table 5-1** taken from the *MC68HC908GP32 Technical Data*, Motorola document order number MC68HC908GP32/H REV. 4. **Table 5-1** provides numeric examples (numbers are in hexadecimal notation).

Table 5-1. Numeric Example

f_{BUS}	f_{RCLK}	R	N	P	E	L
2.0 MHz	32.768 kHz	1	F5	0	0	D1
2.4576 MHz	32.768 kHz	1	12C	0	1	80
2.5 MHz	32.768 kHz	1	132	0	1	83
4.0 MHz	32.768 kHz	1	1E9	0	1	D1
4.9152 MHz	32.768 kHz	1	258	0	2	80
5.0 MHz	32.768 kHz	1	263	0	2	82
7.3728 MHz	32.768 kHz	1	384	0	2	C0
8.0 MHz	32.768 kHz	1	3D1	0	2	D0

The following code will initialize the PLL unit at the desired frequency. It is from *remote\startup.c->InitialisePLL()* (see **[REMOTE:startup.c]**.

```
                        ////////////////////////////////////////////
PBWC.reg       = 0x80;     // auto mode                             //
PCTL.reg       = 0x02;     // settings here...                      //
PMS            = 0x012C;   // as described in...                    //
PMRS.reg       = 0x80;     // the MC68HC908GP32/H                   //
PMDS.reg       = 0x01;     // Rev2.0 data book section 7.4.6 page 120 //
PCTL.bit.PLLON = 1;        // turn pll on after settings 'set'      //
                        ////////////////////////////////////////////

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

PCTL.bit.BCS = 1;          // pll clock ready, drives CGMOUT
```

The *MC68HC908GP32 Technical Data* details the equations used to generate the values inserted in **Table 5-1**.

Reference clock divider, R, is equal to 1 as the PLL crystal f_{RCLK} is 32.768 kHz.

Range multiplier, N

$N = (R \times f_{VCLKDES}) / f_{RCLK}$
where $f_{VCLKDES} = 4 \times f_{BUSDES} = 9.830400E6$

N = $(1x9.830400E6)/32.768E3 = 300_{10} = \mathbf{12C_{16}}$

VCO Linear range multiplier, L

$L = F_{VCLK}/(2^E \times f_{NOM})$
where $f_{NOM} = 38.4kHz$, $F_{VCLK} = 9.830400E6$ and $E = 1$

L = $9.830400E6/(2 \times 38.4E3) = 128_{10} = \mathbf{80_{16}}$

NOTE: *E = 1 from frequency range table in the MC68HC908GP32 Technical Data.*

5.3 Clock Generator Module/PLL Hardware Description

The CGMC generates the crystal clock signal CGMXCLK, which operates at the clock frequency (32.768 kHz in this design). An internal phase-locked loop (PLL) generates the programmable VCO frequency clock and determines the bus frequency. A Pierce oscillator configuration is used (**Figure 5-1**) which uses five external components, with the crystal directly connected between the crystal amplifier input pin (OSC1) and the crystal amplifier output pin (OSC2).

R_B = feedback resistor	10 M
R_S = series resistor	330 k
X1 = crystal	32.768 kHz
C1 = tuning capacitor	2 x C_L 15 pF(1)
C2 = tuning capacitor	2 x C_L 15 pF(1)

1. Consult manufacturer's data

The PLL analog power and ground pins V_{DDA} and V_{SSA} 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 5-1**.

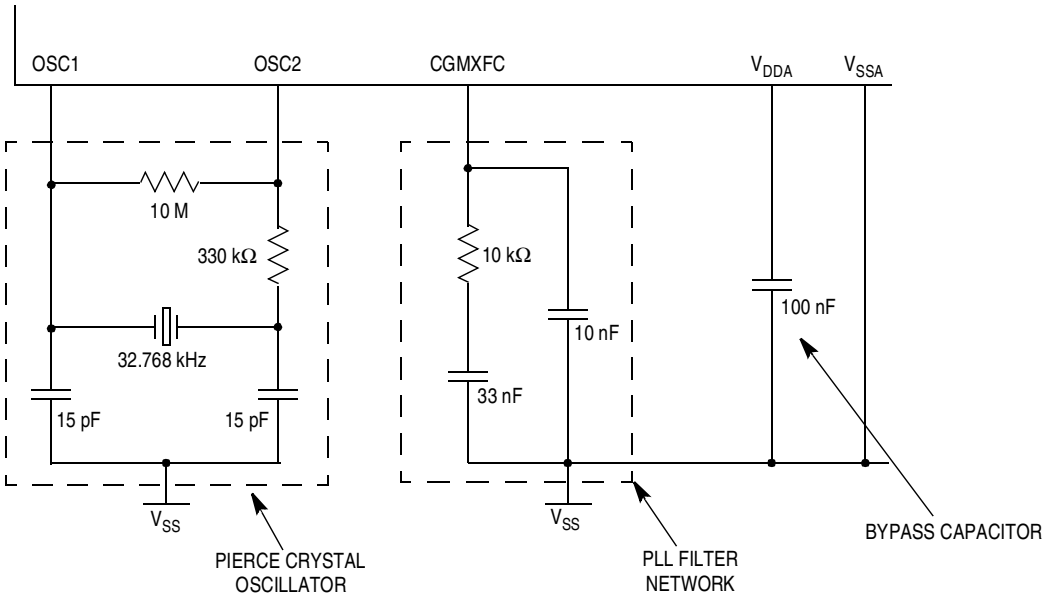


Figure 5-1. MC68HC908GP32 PLL Connections

Section 6. Cosmic M68HC08 Complier

6.1 Contents

6.2 Introduction

For this design, the Cosmic C Compiler for the M68HC08 v4.2i was used. The documentation you receive with the compiler is comprehensive. The purpose of this section is to illustrate the techniques used in this design. **6.7 IDEA Integrated Environment** will briefly discuss IDEA, a Windows® program which provides a graphical method as opposed to a command line based method for using the Cosmic tool set.

The compiler usage invokes many DOS exe files. When compiling a source file these programs will be called:

- c**p**6808.exe
- c**g**6808.exe
- c**o**6808.exe
- c**a**6808.exe

The bold letter indicate what each program run is doing. For example:

- *c**p**6808.exe* is the code **p**arser
- *c**g**6808.exe* is the code **g**enerator
- *c**o**6808.exe* is the code **o**ptimizer
- *c**a**6808.exe* is the code **a**ssembler

6.3 Compiling

The compiler is run via DOS batch files from a text editor. To run in a DOS shell is fine. For example, to compile a file called main.c, use:

cc main <ENTER>

cc.bat would contain:

```
@echo off
rem ================================================================
rem = 'f' include 'config.dat' for further compiler options
rem ================================================================
c:\cosmic\cx08\cx6808 -f config.dat %1.c
```

6.4 Configuration File

Notice the "f" switch, which allows the use of a separate file to hold all the compiler switches that are required. This keeps the batch file simple and ensures that the DOS command line limit is not exceeded.

The content of the *config.dat* file used for the PIR code is shown here. Note the use of the # for a comment.

```
##############################################
#  CONFIGURATION FILE FOR 68HC08 COMPILER  #
#  ATEECC                                  #
##############################################

############
# COMPILER #
############
#-no                         # don't use optimiser
-e                           # create error file
-l                           # create C/assembly listing file
-i c:\cosmic\cx08\h6808       # include ...

##########
# PARSER #
##########
-pic:\cosmic\cx08\h6808       # include ...
-pp                          # prototype checking
-pl                          # output line number info for listing & debug
-pck                         # extra type checking
-pnw                         # don't widen args
-px                          # produce debug info for Zap

#############
# GENERATOR #
#############
-gf                          # all lines in listing
#-gck                        # enable stack overflow checking
#-gv                         # show function being processed

#############
# ASSEMBLER #
#############
-al                          # assembler file listing
-at                          # list instruction cycles

#############
# OPTIMISER #
#############
#-ov                         # show number of removed/modified instructions
```

OK, now that a file has been compiled, what's next. Due to the compilation of *main.c,* two more *main* files have been produced, *main.ls* and *main.o*. The object file *main.o* is a relocatable object module and *main.ls* contains the M68HC08 assembler generated from the C source statements by the compiler, with only relative address reference. The absolute address listing is produced after the linking process.

6.5 Make File

Most projects will consist of many source files, which aids in keeping the code modular and more manageable in a text editor. You can recompile all source files and then link to produce the Motorola S-record (S19) file. Another batch file called *make08* does just that and is shown here.

```
[MAKE08.BAT]

rem//////////////////////////
rem// assemble Cosmic files //
rem//////////////////////////
c:\cosmic\cx08\ca6808 crts.s

rem////////////////////////////
rem// compile all source files //
rem////////////////////////////
call cc a2d
call cc analyse
call cc data
call cc datasort
call cc delay
call cc deltasig
call cc flashprg
call cc interrup
call cc main
call cc serial
call cc startup
call cc vectors

rem//////////////////////////
rem// link the object files //
rem//////////////////////////
call link08

rem////////////////////////////////
rem// deleting relative listings //
rem////////////////////////////////
del *.ls

rem//////////////////////////
rem// list any error files //
rem//////////////////////////
dir *.err
```

First, the Cosmic-supplied assembler startup file (producing crts.o) is assembled directly using ca6808; then, in turn, each of the C source files is compiled. This results in several object files that now need linking to produce the final S19 file.

6.6 Linking

The best way to understand the linking process is to do it. That means going through a cycle of linking and studying the S-record/absolute listing files. At the heart of the linker process is the linker command file, *jk.lkf*, which basically tells the linker what to put where in address terms.

In a straightforward project (if one exists,) user software would use read-only memory (ROM) space for the opcodes and random-access memory (RAM) space for the variables. If some of your variables are of type *const* (for instance, stored in ROM) then that will add another linker requirement.

The use of *segments* is used to create these fixed areas of storage. For example, the MC68HC908JK1/3 RAM area could be defined using:

```
+seg .ubsct -b 0x0080 -n TinyRam -m 128
```

where:

ubsct = non-initialized data in the zero page

b = start address of segment

n = name of segment used in linker output file

m = maximum size of segment

The MC68HC908JK1/3 RAM occupies $80 to $FF inclusive (resides in page0 entirely).

The segment where the code will reside for the MC68HC908JK3 will be:

```
+seg .text -b 0xec00 -n UserFLASH -m 4096
```

The compiler needs a *const* area if certain libraries are used (for example, switch jump tables):

```
+seg .const -a UserFLASH # ’-a’ append section to previous
```

Finally, the interrupt vectors are required:

```
+seg .const -b 0xffde -n Vectors -m 34
```

This will give the bare bones linker file for a MC68HC908JK3. To assign the object file to the relevant declared segment, simply list the object file after the segment declaration. For example:

Simple linker command file:

```
+seg .ubsct -b 0x0080 -n TinyRam -m 128
data.o

+seg .text -b 0xec00 -n UserFLASH -m 4096
+seg .const -a UserFLASH
crts.o
a2d.o
analyse.o
datasort.o
delay.o
deltasig.o
interrup.o
main.o
serial.o
startup.o

+seg .const -b 0xffde -n Vectors -m 34
vectors.o
```

To link, *clnk.exe* is used:

```
c:\cosmic\cx08\clnk  -v -m jl.inf -e jk.err -o pir.h08 jk.lkf
```

where:

v = verbose
m = produce map information file
e = log errors to file
o = output to file

S19 generation:

The linker output (pir.h08, executable image) can be converted to a hexadecimal interchange format (Motorola S19 format) using *chex.exe*:

c:\cosmic\cx08\chex -fm -o pir.s19 pir.h08

where:

fm = Motorola output format
o = output to file

Absolute listing:

Finally, *clabs.exe* is used to process the relative assembler listing files to produce an absolute listing:

c:\cosmic\cx08\clabs -l -v pir.h08

where:

l = restrict to current directory
v = verbose

Linker command file (*.lkf):

The linker command file used for the PIR software is shown here.

```
#####################################################
# LINKER COMMAND FILE FOR MOTOROLA HC908JK1/3/JL3 #
# PIR REFERENCE DESIGN                            #
# ATEECC July 2000                                #
#####################################################


############
# symbols #
############
+def __memory=@.bss                    # symbol used by startup
+def __stack=0x00ff                    # stack pointer value for 'crts.s'

###################################
# MC68HC908JK1/JK3 selection      #
#                                 #
# JK1:                            #
# ROM_START=0xf600, ROM_SIZE=1536 #
#                                 #
# JK3/JL3:                        #
# ROM_START=0xec00, ROM_SIZE=4096 #
###################################
+def ROM_START=0xf600
+def ROM_SIZE=1536
```

```
                ##############
                # CONST DATA #
                ##############
#           -b [b]eginning address of segment
#                    -n [n]ame of segment
#                                -m [m]ax size (bytes) of segment
+seg .const -b 0xfbc0 -n ConstFLASH -m 64       ################################
                                                # 64 bytes is min erase block  #
                                                # size                         #
                                                # WE USE THE LAST 64 BYTE BLOCK#
                                                # IN THE FLASH MEMORY AREA     #
                                                ################################
                #############
                # PAGE0 RAM #
                #############

############################
# run time data allocation #
############################                    ################################
+seg .ubsct -b 0x0080 -n TinyRam -m 128         # Occupies $080-$00ff (PAGE0). #
ireg.o                                          # This ensures that the Cosmic #
                                                # variables 'c_reg'            #
lreg.o                                          # and 'c_lreg' are positioned  #
                                                # at the beginning of ram this #
                                                # segment, ensuring that during#
                                                # any memcpy operations they do#
                                                # not get overwritten with     #
                                                # copied data.                 #
                                                #                              #
data.o                                          # NOTE: user global data here  #
                                                ################################

#########################################################
# This segment is for PIR FLASH parameter programming.#
# The variables from 'data.o' and 'mot_data.o' will   #
# overlap, that is ok since the variables occupying   #
# the same address will not be active at the same     #
# time. See 'datasort.c->AssignCurrentFLASHData()'    #
# for more information.                               #
#                                                     #
# The Motorola monitor routines expect their          #
# variables/data to be at known addresses.            #
#                                                     #
# Notice the '-v' switch, it tells the linker         #
# not to report overlap errors for this segment       #
#########################################################
+seg .ubsct -b 0x88 -v -n MONITOR_RAM -m 128-8  ########################
mon_data.o                                      # '8' since this segment#
                                                # starts at $0088 and   #
                                                # not $0080             #
                                                ########################
```

```
##############################
# FLASH memory for user code #
##############################
+seg .text -b ROM_START -n UserFLASH -m ROM_SIZE-64
                                              ##############################
                                              # MC68HC908JK/L3 user code    #
                                              # start address               #
                                              # '64' for const FLASH        #
                                              # variables, see 'ConstFLASH' #
                                              # segment                     #
                                              ##############################

####################################
# const area for switch jump tables #
####################################          ##############################
+seg .const -a UserFLASH                      # '-a' append section to      #
                                              # previous                    #
                                              ##############################
######################
# user object files #
######################
crts.o          # Cosmic supplied startup routine
a2d.o           # a2d initialise/read
analyse.o       # data buffer scan routine, buffer contains PIR a2d values
datasort.o      # data integrity and decode
delay.o         # inline accurate delay routine
deltasig.o      # alternative pir 'event' routines using delta-sigma
                # algorithm
flashprg.o      # flash programming
interrup.o      # interrupt service routines
main.o          # main()
serial.o        # RS232 debug (send) and IR comms routines
startup.o       # micro initialisation i.e. i/o, ram clear, timer initialisation

####################
# Cosmic libraries #
####################
c:/cosmic/cx08/lib/libi.h08
c:/cosmic/cx08/lib/libm.h08

############
# Vectors #
############
+seg .const -b 0xffde -n Vectors -m 34
vectors.o
```

6.7 IDEA Integrated Environment

For those who prefer to work in the Windows® environment, Cosmic provides a program to do just that.

The IDEA integrated environment provides a Windows® based graphical user interface (GUI) for building and managing projects. IDEA is fully integrated with all Cosmic tools including compilers, assemblers, linkers, utilities, and ZAP debuggers.

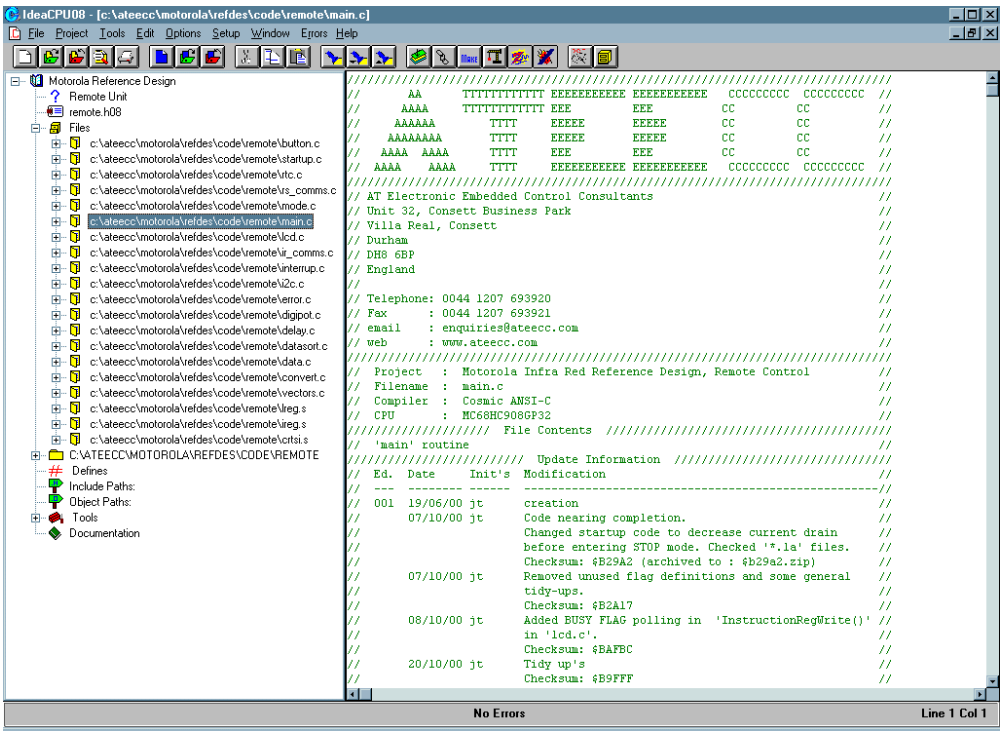


Figure 6-1. IDEA Loaded with the REMOTE Unit Project

Section 7. Windows® 95/98/NT Program (pir_plot.exe)

7.1 Contents

7.2 Introduction

This section describes the Windows® 95/98/NT program.

7.3 Program Description

The accompanying Windows® program (*pir_plot.exe*) displays the PIR infrared sensor value as seen/calculated by the PIR unit software. The sensor value is transmitted to the connected PC via the RS232 port using *1 START, 38400, NO PARITY, 8 DATA,* and *1 STOP*. The main aim of this program is to allow the user to see the real-time response of the infrared sensor.

The data is sent using this protocol:

BLOCK LENGTH	BLOCK TITLE	DATA BYTE 1	DATA BYTE 2	DATA BYTE n	CHECKSUM HI	CHECKSUM LO

BLOCK LENGTH — Number of bytes in the packet excluding the checksum

CHECKSUM HI/LO — Bytes refer to the 16-bit sum of: BLOCK LENGTH + BLOCK TITLE + DATA BYTE1 + DATA BYTE2 + ...+ DATA BYTE n.

See **Figure 7-1** and **Figure 7-2**.

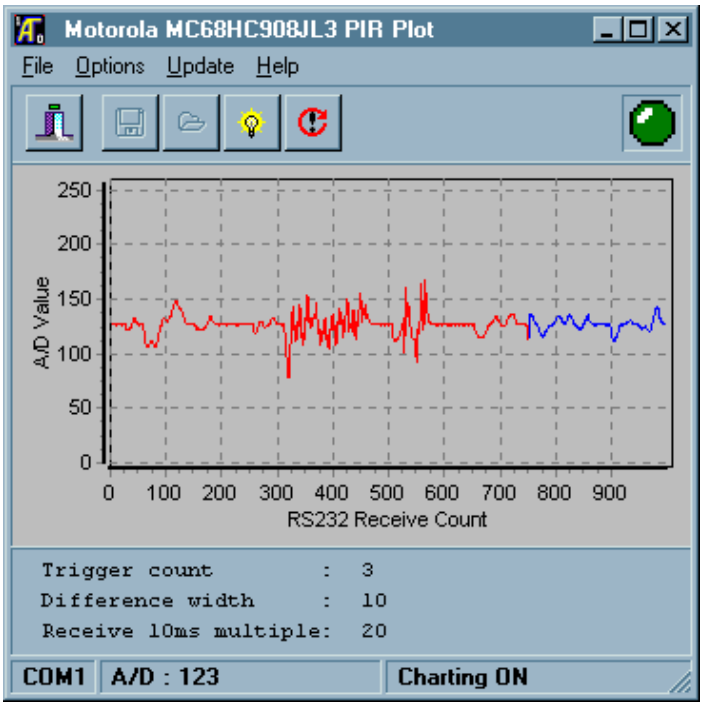


Figure 7-1. Typical Analog PIR Response

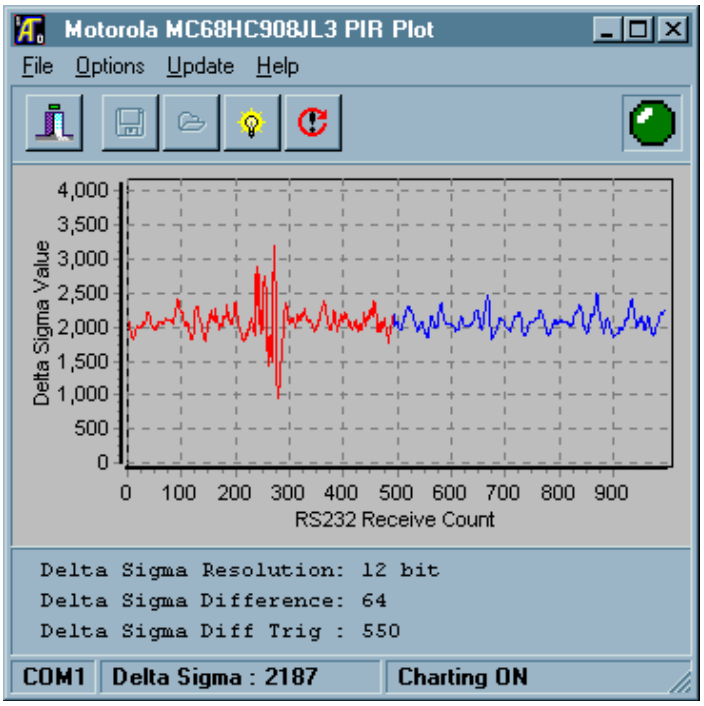


Figure 7-2. Typical Delta Sigma PIR Response

The following extract, from *pir\a2d.c->A2DCheck()* (see **[PIR:a2d.c]**), shows a *main()* loop counter being incremented, and when it is equal to a FLASH *const* value an analog read takes place. The result of the analog read is an average of *A2D_SAMPLE_COUNT* (currently 32) readings. If the debug RS232 code is included (*#ifdef __PC_DEBUG_*), then the appropriate data is assigned and checksum calculations and transmission take place in *pir\serial.c->Send_RS232_CommsPacket()* (see **[PIR:serial.c]**).

```
if ( ++a2d_count >= pir_params.main_loop_count )
   {
   a2d_count       = 0;                    // reset
   *pir_buffer_ptr = ReadA2D(CHANNEL4);    // 'A2D_SAMPLE_COUNT' average
                                           // result is returned
   .
   .
   .
   #ifdef   __PC_DEBUG_    // transmit current data to pc?
   SEI();                  // all interrupts off to ensure 38400 bit timings
   rs232_buffer[2] = *pir_buffer_ptr;
   if ( flags1.bit.ALARM_EVENT ) rs232_buffer[3] = 'Y'; // pc to 'beep'
   else                          rs232_buffer[3] = 'N'; // no pc 'beep'
   rs232_buffer[4] = pir_params.trigger_count;
   rs232_buffer[5] = pir_params.difference_band;
   rs232_buffer[6] = pir_params.main_loop_count;
   Send_RS232_CommsPacket( PIR_DATA, 5 ); //5 == above 5 data bytes
   CLI();                  // interrupt processing back on
   #endif
   .
   .
   .
  }
```

The *pir_plot.exe* program contains the usual Windows® features (for instance, traces can be saved, restored and printed). When using the program, consult the on-line help for full instructions.

NOTE: *#ifdef __PC_DEBUG_, the RS232 feature, is used during debug only. This #define ensures that the appropriate code is compiled only when required. Due to the additional bytes used, a MC68HC908JK3 will have to be programmed. Access* ***[PIR:define.h]*** *to comment/uncomment the #define declaration as required.*

The PIR unit can send a serial debug packet (9 bytes) every 10 ms if *pir_params.main_loop_count* is set to 1. There is no hardware

handshaking, and, consequently due to the 10 ms inter-packet time, it is likely that Windows® will be unable to process all incoming data.

If possible while debugging, keep *pir_params.main_loop_count* to a minimum of 5 (50 ms inter-packet time). This value was used successfully on a Pentium 133 MHz with 80 MB of random-access memory (RAM). The faster your PC the lower the value of *pir_params.main_loop_count* you can use and still receive and display all incoming data. Of course, once `#ifdef __PC_DEBUG_` is commented out, you can use any value for *pir_params.main_loop_count* required as no RS232 transmission will take place.

To reduce PC CPU processing time during an analog serial session, the graphical update occurs when 10 serial packets have been processed. That is why the screen will draw in *bursts* rather than in each data point as it is transmitted to the PC.

To ensure the 38400 RS232 bit timings, the MC68HC908JK3 disables all interrupts. This will have an impact on the IR communications which is decoded in the timer channel0 interrupt routine; consequently, the IR communications may feel slightly unresponsive. The Delta Sigma detection method also disables interrupts during the capacitor charge/discharge process. Adding the serial transmission interrupt disabling will further decrease IR communications responsiveness.

Appendix A. Fresnel Lens Mounting

The correct positioning of the Fresnel lens is critical to the operation of the PIR unit. The Fresnel lens included with this package has a 12.5-m focal length. Therefore, during debugging and close range testing, it is better to remove the Fresnel lens entirely. **Figure A-1** shows how the Fresnel lens is mounted with respect to the PCB.

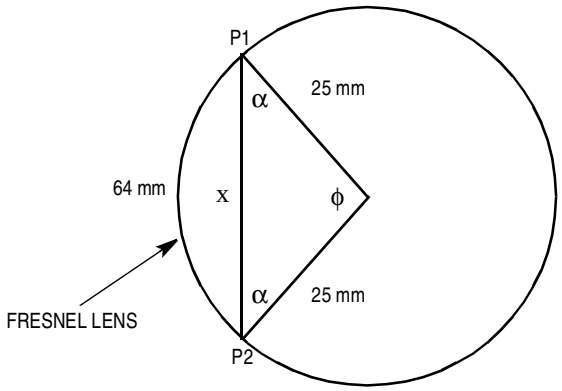


Figure A-1. Fresnel Lens Geometry

Arc distance (x), P1 to P2 is 64 mm.

$x = r\phi$

where:

s = sector length

r = radius

ϕ = angle in radians

$\phi = 148.97°$

$$\alpha = \frac{180 - \phi}{2}$$

$\alpha = 15.52°$

This produces an isosceles triangle using the *sine* rule:

$$\frac{\sin(\phi)}{x} = \frac{\sin(\alpha)}{25}$$

By substituting the known values for ϕ and α, the value of x is obtained:

$x = 48.16$ mm

Lastly, the angle of the PCB slots to hold the Fresnel lens:

$\beta = 74.48°$

See **Figure A-2**.

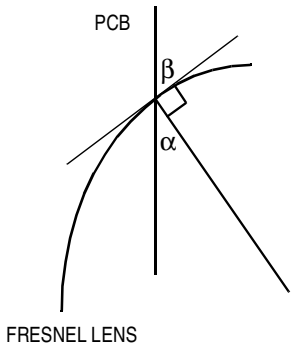


Figure A-2. PCB/Lens Connection Angle

Appendix B. PIR Schematics

This appendix provides PIR schematics. Refer to:

- **Figure B-1** for the Delta Sigma schematic
- **Figure B-2** for the analog PIR schematic

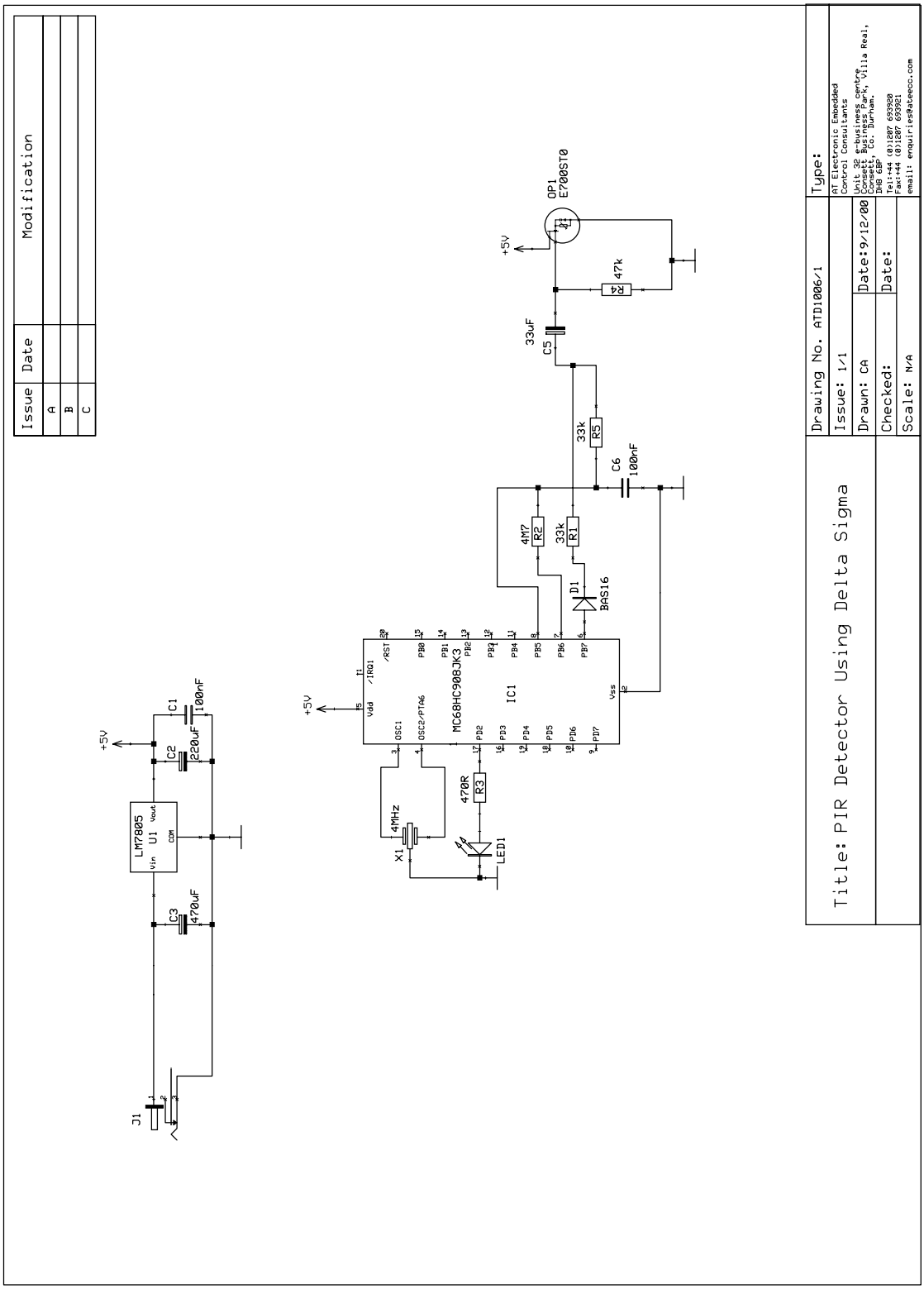


Figure B-1. Delta Sigma PIR Schematic

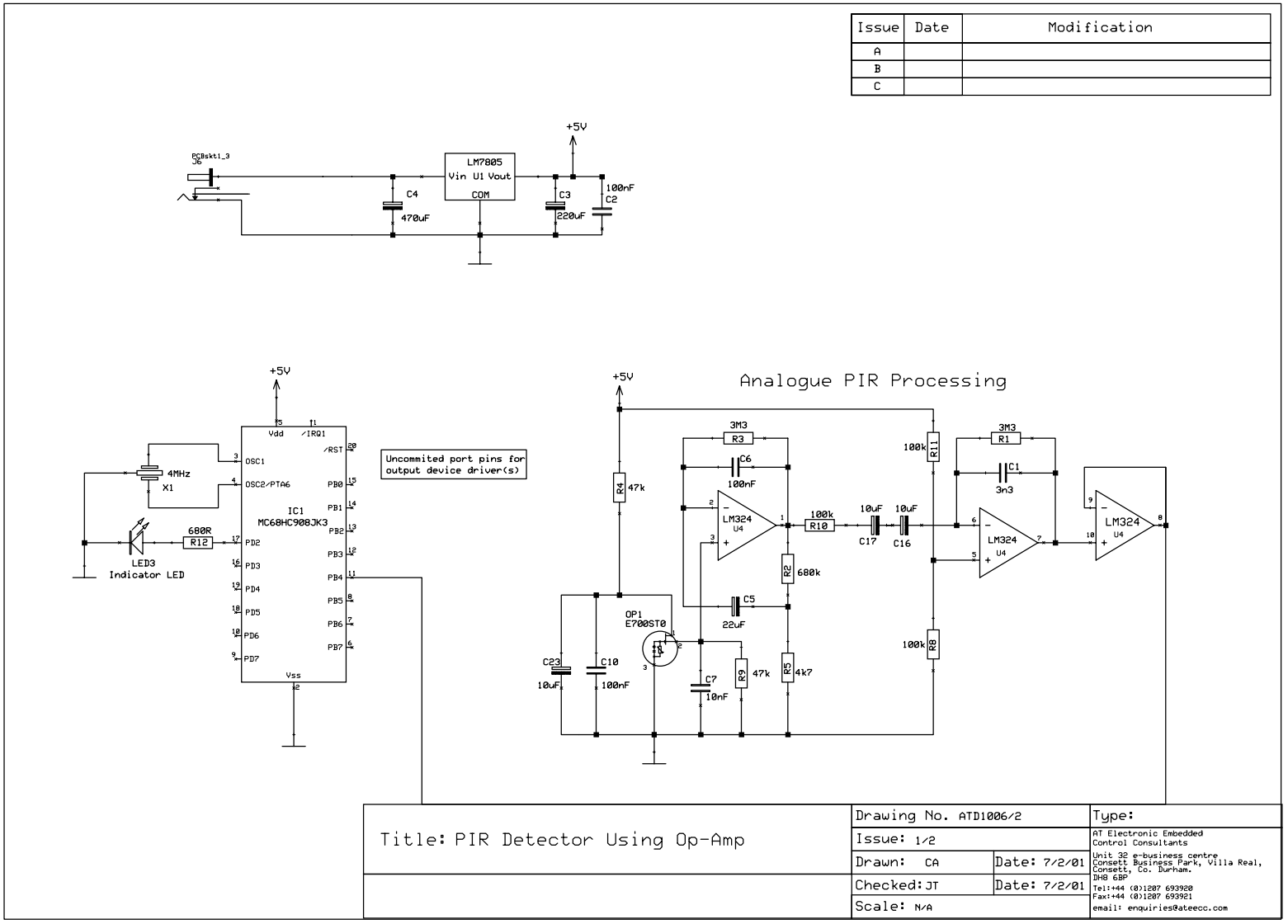


Figure B-2. Analog PIR Schematic

PIR Schematics

Appendix C. Development Boards

This appendix provides diagrams for the development boards. Refer to:

- **Figure C-1** for the PIR detector development board
- **Figure C-2** for the remote control development board

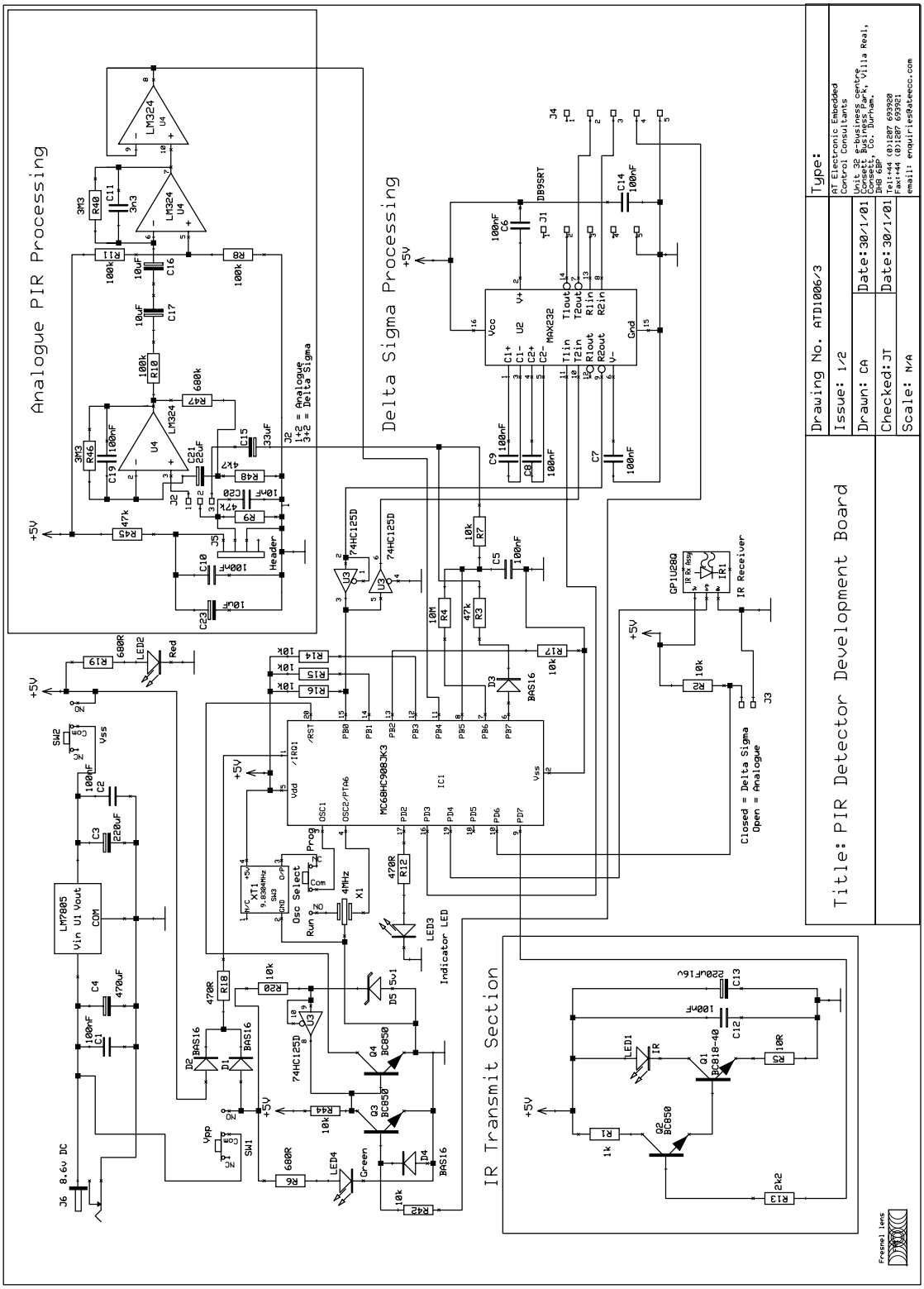


Figure C-1. PIR Detector Development Board

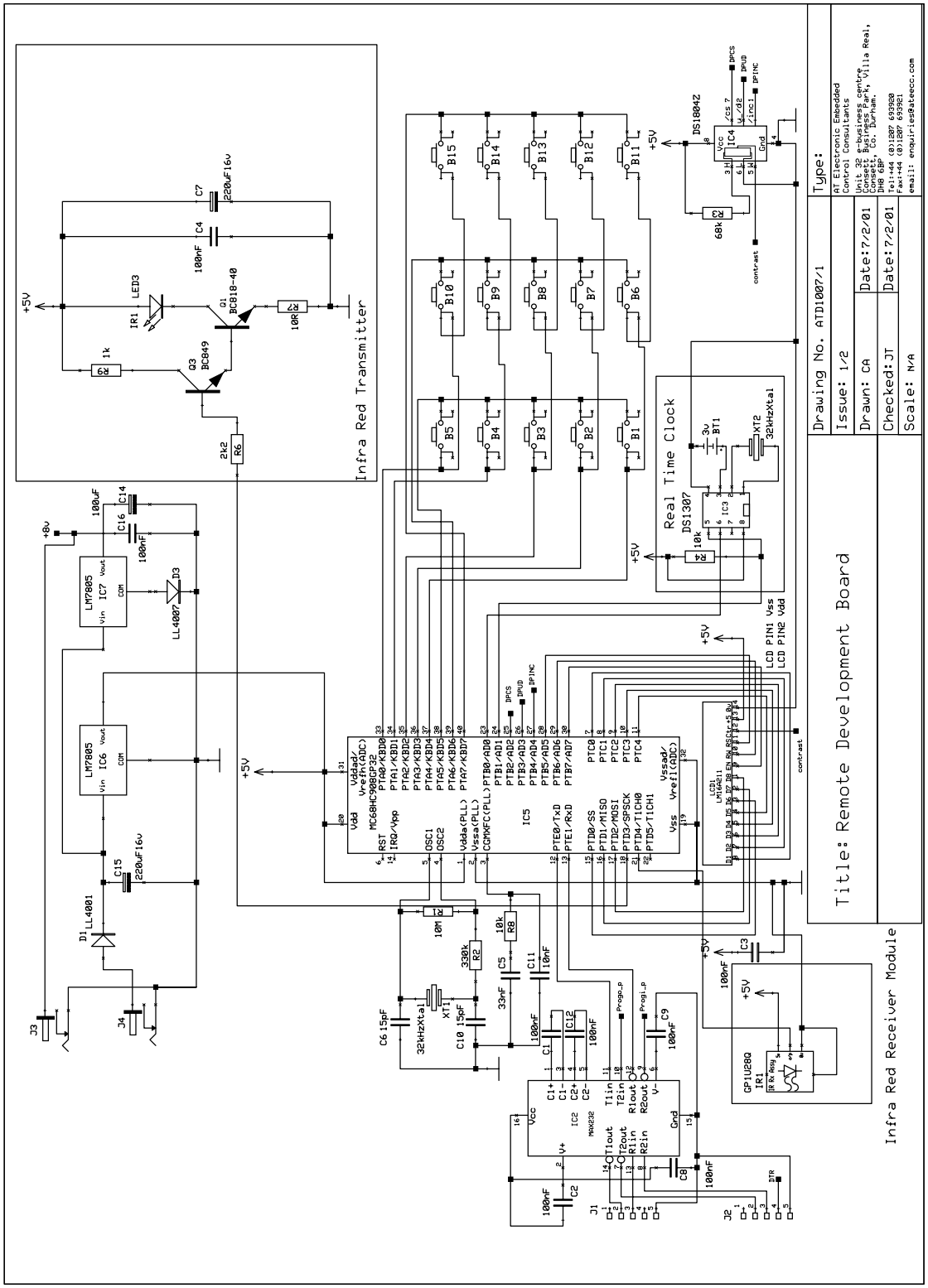


Figure C-2. REMOTE Control Development Board

Development Boards

Appendix D. MC68HC908GP32 Programmer Circuit

This appendix provides a programmer circuit diagram for the MC68HC908GP32. Refer to **Figure D-1**.

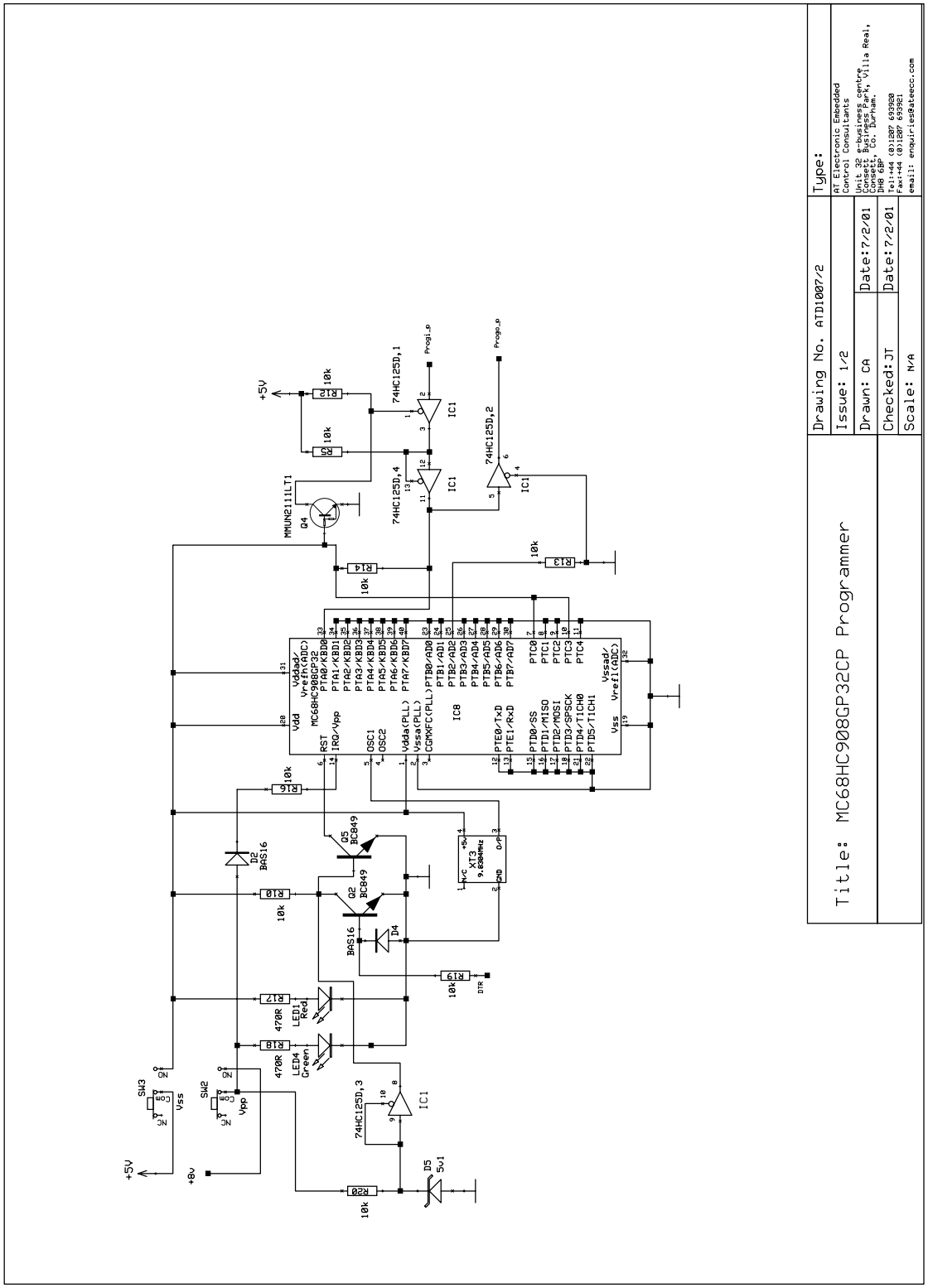


Figure D-1. MC68HC908GP32 Programmer Circuit

Appendix E. PIR Source Code Files

Throughout this document, references are made to source code files contained in this appendix. They are:

For those viewing this document in .pdf format, these files can be accessed by clicking on the appropriate hyperlink reference found in the textual portions of the document.

```
[PIR:a2d.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 Infra Red Reference Design                          //
//  Filename  :  a2d.c                                                        //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ///////////////////////////////////////
//  a2d routines                                                              //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/05/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "delay.h"
#include    "serial.h"
#include    "analyse.h"
#include    "a2d.h"


////////////////////////////////////////////////////////////////////////////
// This function returns an averaged analogue value. It's primary use it to  //
// read the amplified IR sensor output. The averaging is currently 32,        //
// determined by the value of A2D_SAMPLE_COUNT (in a2d.h)                     //
//                                                                            //
// Argument : analogue channel to read                                        //
// Returns  : averaged analogue result                                        //
////////////////////////////////////////////////////////////////////////////
unsigned char ReadA2D( unsigned char channel )
{
union uUNSIGNED_INTEGER a2d_total;
unsigned char           ii;
```

```
InitialiseA2D(channel);

a2d_total._16bit = 0;   // clear summation total

for ( ii = 0; ii < A2D_SAMPLE_COUNT; ii++ )
      {
      while ( !ADSCR.bit.COCO );       // wait for conversion to complete
      a2d_total._16bit += ADR.reg;     // update running total
      }

a2d_total._16bit /= A2D_SAMPLE_COUNT;  // average

////////////////////////////////////////////////////////////////////////////
// NOTE                                                                   //
// using the a2d continuous conversion mode this function takes approx 1ms to //
// execute @ 1MHz bus with a 32 sample count sum. Using the single conversion //
// mode the function execution time is approx 1.6ms                       //
////////////////////////////////////////////////////////////////////////////
return a2d_total._8bit.lobyte;
}  // ReadA2D()
//--------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// Motorola data book MC68HC908JL3/H Rev. 1.0 page 145, states :          //
// "the ADC clock should be set to approximately 1MHz".                   //
//                                                                        //
// A2D initialisation                                                     //
//                                                                        //
// Argument : channel to read from                                        //
// Returns  : none                                                        //
////////////////////////////////////////////////////////////////////////////
void InitialiseA2D( unsigned char channel )
{                             ////////////////////////////////////////////////
ADICLK.reg = 0x00;            // divide by 1, since we are using a 4MHz       //
                              // resonator the bus speed will be at the       //
                              // required 1MHz                                //
                              //                                              //
#ifdef   __MMDS_EMULATOR_     // recommendation from 'hc08gm32em.pdf' page 16 //
ADICLK.bit.bit4 = 1;          // NOTE: for the JK1/3 this bit is unimplemented//
#endif                        // and will cause no misoperation               //
                              //                                              //
ADSCR.reg = (unsigned char)(0x20|channel);// ints off, continuous conversion  //
                              //                                              //
Delay( A2D_STABILISATION );   // stabilisation delay, approx 50us             //
}  // InitialiseA2D()         ////////////////////////////////////////////////
//--------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// This function is called directly from 'main()'. If the required number of  //
// 'main()' loop iterations has been executed then the pir sensor is read and //
// the resulting values are stored in the global buffer 'pir_buffer' via a    //
// unsigned char pointer. A magnitude test test is performed with the previous//
// result and if greater than 'pir_params.difference_band'.                   //
//                                                                            //
// Argument : none                                                            //
// Returns  : none                                                            //
////////////////////////////////////////////////////////////////////////////
void A2DCheck( void )
{
unsigned char diff;
unsigned char previous_pir_data;



if ( ++a2d_count >= pir_params.main_loop_count )
   {
   a2d_count       = 0;                    // reset
   *pir_buffer_ptr = ReadA2D(CHANNEL4);    // 'A2D_SAMPLE_COUNT' average
                                           // result is returned
   ////////////////////////
   // trigger detected yet? //
   ////////////////////////
   if ( !flags1.bit.TRIGGER_EVENT )
      {
      ////////////////////////////////////////////////////////////////////////
      // Nearest previous neighbour test here, if magnitude is greater than   //
      // 'pir_params.difference_band' then flush buffer, insert previous and  //
      // present data into start locations of buffer and then start refilling.//
      // When full, perform analysis ie 'Analyse_PIR_Buffer()'                //
      ////////////////////////////////////////////////////////////////////////
      if ( pir_buffer_ptr > &pir_buffer[0] )
         {
         previous_pir_data = *(pir_buffer_ptr-1);
         }
      else
         {
         previous_pir_data = pir_buffer[LAST_PIR_BUFFER_ELEMENT]; // buffer wrap
         }                                                        // ocurred

      ////////////////////////////////////////////////////
      // difference a2d check on present - previous readings //
      ////////////////////////////////////////////////////
      if ( *pir_buffer_ptr > previous_pir_data )
         {
         diff = (unsigned char)(*pir_buffer_ptr - previous_pir_data);
         }
      else
         {
         diff = (unsigned char)(previous_pir_data - *pir_buffer_ptr);
         }
```

```
      ////////////////////////////////////////////////////////
      // does the difference constitute an intruder detect event? //
      ////////////////////////////////////////////////////////
      if ( diff >= pir_params.difference_band )
         {
         ////////////////////////////////////////
         // re-store present and previous value //
         ////////////////////////////////////////
         pir_buffer[0] = previous_pir_data;
         pir_buffer[1] = *pir_buffer_ptr;

         flags1.bit.TRIGGER_EVENT = 1;
         pir_buffer_ptr           = &pir_buffer[1]; // '1' due to
         }                                          // ++pir_buffer_ptr below
      }

   #ifdef   __PC_DEBUG_    // transmit current data to pc?
   SEI();                  // all interrupts off to ensure 38400 bit timings
   rs232_buffer[2] = *pir_buffer_ptr;
   if ( flags1.bit.ALARM_EVENT ) rs232_buffer[3] = 'Y'; // pc to 'beep'
   else                          rs232_buffer[3] = 'N'; // no pc 'beep'
   rs232_buffer[4] = pir_params.trigger_count;
   rs232_buffer[5] = pir_params.difference_band;
   rs232_buffer[6] = pir_params.main_loop_count;
   Send_RS232_CommsPacket( PIR_DATA, 5 ); //5 == above 5 data bytes
   CLI();                  // interrupt processing back on
   #endif

   ////////////////////////////////////////////////////////
   // increment pointer for next storage, check for buffer wrap. //
   // If buffer is full AND if we have had an event trigger,     //
   // then analyse the acquired data.                            //
   ////////////////////////////////////////////////////////
   if ( ++pir_buffer_ptr > &pir_buffer[LAST_PIR_BUFFER_ELEMENT] )
      {
      if ( flags1.bit.TRIGGER_EVENT )
         {
         flags1.bit.TRIGGER_EVENT = 0;

         if ( !flags1.bit.ALARM_EVENT )   // stop overlapping intruder detect
            {                             // events
            Analyse_PIR_Buffer();
            }
         }

      pir_buffer_ptr = &pir_buffer[0]; // reset buffer storage pointer for next
      }
   }
}  // A2DCheck()
//---------------------------------------------------------------------------
```

```
[PIR:a2d.h]
////////////////////////////////////////////////////////////////////////////
//       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 Infra Red Reference Design                          //
//  Filename  :  a2d.h                                                        //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ///////////////////////////////////////
//  a2d routines                                                              //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/05/00 jt      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  CHANNEL6               0x06
#define  CHANNEL7               0x07
#define  CHANNEL8               0x08
#define  CHANNEL9               0x09
#define  CHANNEL10              0x0a
#define  CHANNEL11              0x0b
#define  A2D_VDDA               0x1d
#define  A2D_VSSA               0x1e
#define  A2D_OFF                0x1f
#define  CLEAR_CHANNEL_SEL      0xe0
#define  A2D_SAMPLE_COUNT       32
#define  ShutDownA2D()          ADSCR.reg = A2D_OFF     // ADC power off
```

```
#define  PIR_BUFFER_SIZE         8  // need to update buttons.c->Increment()
                                    // for gp32 with this value
#define  LAST_PIR_BUFFER_ELEMENT (PIR_BUFFER_SIZE-1)

////////////////////////////////////////////////////////////////////////
// The values assume a 10ms main loop time (via timeroverflow MOD counter) //
////////////////////////////////////////////////////////////////////////
enum  {
      _10MS = 1, _20MS , _30MS , _40MS , _50MS , _60MS , _70MS , _80MS , _90MS ,
      _100MS   , _110MS, _120MS, _130MS, _140MS, _150MS, _160MS, _170MS, _180MS,
      _190MS   , _200MS, _210MS, _220MS, _230MS, _240MS, _250MS, _260MS, _270MS,
      _280MS   , _290MS, _300MS, _310MS, _320MS, _330MS, _340MS, _350MS, _360MS,
      _370MS   , _380MS, _390MS, _400MS, _410MS, _420MS, _430MS, _440MS, _450MS,
      _460MS   , _470MS, _480MS, _490MS, _500MS, _510MS, _520MS, _530MS, _540MS,
      _550MS   , _560MS, _570MS, _580MS, _590MS, _600MS, _610MS, _620MS, _630MS,
      _640MS   , _650MS, _660MS, _670MS, _680MS, _690MS, _700MS
      // etc...to _2550MS
      };

////////////////
// prototypes //
////////////////
unsigned char  ReadA2D( unsigned char );
void           InitialiseA2D( unsigned char );
void           A2DCheck( void );

#endif
```

```
[PIR:analyse.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 Infra Red Reference Design                          //
//  Filename  :  analyse.c                                                    //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ///////////////////////////////////////
//  analyse a2d buffer for movement data patterns                             //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  26/05/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "analyse.h"



void Analyse_PIR_Buffer( void )
{
unsigned char  trigger_count = 0;
unsigned char  ii;
unsigned char  pir_difference;
unsigned char  data1;
unsigned char  data2;
```

```
////////////////////////////////////////////////////////////////////////////
// 'PIR_BUFFER_SIZE' array element gives 'PIR_BUFFER_SIZE-1 'comparisons ie   //
// [0][1],[1][2],[2][3],[3][4],[4][5]...[PIR_BUFFER_SIZE-2][PIR_BUFFER_SIZE-1]//
////////////////////////////////////////////////////////////////////////////
for ( ii = 0; ii < LAST_PIR_BUFFER_ELEMENT; ii++ )
   {
   data1 = pir_buffer[ii];
   data2 = pir_buffer[ii+1];

   ///////////////////////////////////
   // determine neighbour difference //
   ///////////////////////////////////
   if ( data1 >= data2 )   pir_difference = (unsigned char)(data1 - data2);
   else                    pir_difference = (unsigned char)(data2 - data1);

   ////////////////////////////////////////////////////////////////
   // has the data changed appreciably compared to it's neighbour //
   ////////////////////////////////////////////////////////////////
   if ( pir_difference >= pir_params.difference_band )
      {
      ////////////////////////////
      // if so, how many times //
      ////////////////////////////
      if ( ++trigger_count >= pir_params.trigger_count )
         {
         ////////////////////////////////////////
         // intruder detected, start alarm process //
         ////////////////////////////////////////
         flags1.bit.ALARM_EVENT = 1;
         break;
         }
      }
   }
}  // Analyse_PIR_Buffer()
//---------------------------------------------------------------------------
```

```
[PIR:analyse.h]
////////////////////////////////////////////////////////////////////////////
//       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 Infra Red Reference Design                          //
//  Filename  :  analyse.h                                                    //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for analyse.c                                                 //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  26/05/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __ANALYSE_H_
#define     __ANALYSE_H_


////////////////
// prototypes //
////////////////
void Analyse_PIR_Buffer( void );

#endif
```

```
[PIR:cc.bat]
@echo off
rem ================================================================================
rem = 'f' include 'config.dat' for further compiler options                    =
rem = 'v' compiler verbosity ie show cp6808/cg6808/co6808/ca6808 screen output =
rem ================================================================================
rem c:\cosmic\cx08\cx6808 -v -f config.dat %1.c

rem ================================================================================
rem = 'f' include 'config.dat' for further compiler options                    =
rem ================================================================================
c:\cosmic\cx08\cx6808 -f config.dat %1.c
```

```
[PIR:config.dat]
############################################
#  CONFIGURATION FILE FOR 68HC08 COMPILER  #
#  ATEECC                                  #
############################################

#############
# COMPILER #
#############
#-no                         # don't use optimiser
-e                           # create error file
-l                           # create C/assembly listing file
-i c:\cosmic\cx08\h6808       # include ...

###########
# PARSER #
###########
-pic:\cosmic\cx08\h6808       # include ...
-pp                          # prototype checking
-pl                          # output line number info for listing & debug
-pck                         # extra type checking
-pnw                         # don't widen args
-px                          # produce debug info for Zap

#############
# GENERATOR #
#############
-gf                          # all lines in listing
#-gck                        # enable stack overflow checking
#-gv                         # show function being processed

#############
# ASSEMBLER #
#############
-al                          # assembler file listing
-at                          # list instruction cycles

#############
# OPTIMISER #
#############
#-ov                         # show number of removed/modified instructions
```

```
[PIR:crts.s]
; C STARTUP FOR MC68HC08
; Copyright (c) 1995 by COSMIC Software
;
  xref  _main, __memory, __stack
  xdef  _exit, __stext
;
  switch  .bss
__sbss:
  switch  .text
__stext:
  ldhx  #__sbss   ; start of bss
  bra loop    ; start loop
zbcl:
  clr 0,x   ; clear byte
  aix #1    ; next byte
loop:
  cphx  #__memory ; up to the end
  bne zbcl    ; and loop
prog:
   ldhx  #__stack ; initialize stack pointer

   txs
  jsr _main   ; execute main
_exit:
  bra _exit   ; and stay here
;
  end
```

```
[PIR:data.c]
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   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     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 Infra Red Reference Design                        //
//  Filename  :  data.c                                                     //
//  Author    :  jtravers                                                   //
//  Compiler  :  Cosmic ANSI-C                                              //
//  CPU       :  MC68HC908JK1/3                                             //
////////////////////  File Contents  ////////////////////////////////////////
//  global data,                                                            //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  28/03/00 jt      creation                                          //
////////////////////////////////////////////////////////////////////////////
#include    "declared.h"
#include    "define.h"
#include    "a2d.h"


//////////////////////
// Global variables //
//////////////////////
#ifdef   __PC_DEBUG_
@tiny unsigned char                rs232_buffer[12];
@tiny union uBITS                  rs232_data;
#endif

@tiny unsigned char                pir_buffer[PIR_BUFFER_SIZE];
@tiny unsigned char                * @tiny pir_buffer_ptr;
@tiny unsigned char                a2d_count;
@tiny unsigned short int           delta_sigma_result_old;
@tiny union uUNSIGNED_INTEGER      delta_sigma_result;
@tiny volatile unsigned char       ir_buffer[8];
@tiny volatile union uBITS         flags1;
@tiny volatile unsigned char       ir_byte_count;
```

```
@tiny volatile unsigned char        ir_bit_count;
@tiny volatile unsigned char        ir_block_length;
@tiny volatile unsigned short int   ir_start_time;
@tiny volatile unsigned short int   ir_stop_time;
@tiny volatile unsigned short int   detect_led_count;



////////////////////////////////////////////////////////////////////////////
// const data                                                             //
// ++++++++++                                                             //
//                                                                        //
// THIS IS THE PIR FLASH PARAMETER DATA                                   //
// The following const data is located at the start of the last 64 bytes of   //
// memory in the JK1/3/JL3 ie at $fbc0                                    //
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Analogue PIR Parameters are decalred as:                               //
//                                                                        //
// struct sPIR_FLASH_PARAMETERS                                           //
// {                                                                      //
// unsigned char  trigger_count;    // how many triggers before event bit set //
// unsigned char  difference_band;  // difference in consecutive data readings//
//                                  // to create a trigger                //
// unsigned char  main_loop_count;  // number of (10ms) main loop scans   //
//                                  // between sensor reads               //
// };                                                                     //
//                                                                        //
// At startup we have:                                                    //
// 'trigger_count'   == 4                                                 //
// 'difference_band' == 6                                                 //
// 'main_loop_count' == 10                                                //
////////////////////////////////////////////////////////////////////////////
@near const struct sPIR_FLASH_PARAMETERS  pir_params = { 4, 6, 10 };

////////////////////////////////////////////////////////////////////////
// Delta-Sigma PIR Parameters                                        //
// _12BIT == Delta Sigma build resolution                            //
// 350    == diff'ce in sequential Delta sigma values to create a trigger //
////////////////////////////////////////////////////////////////////////
@near const union uUNSIGNED_INTEGER       delta_sig_bit   = { _12BIT    };
@near const union uUNSIGNED_INTEGER       delta_sig_event = {  350      };

////////////////////////////////////////////////////////////////////////
// PIR Password                                                      //
////////////////////////////////////////////////////////////////////////
@near const union uUNSIGNED_INTEGER       password        = { 12345    };
```

```
[PIR:datasort.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 Infra Red Reference Design                          //
//  Filename  :  datasort.c                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ///////////////////////////////////////
//  IR data integrity routine(s)                                              //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/07/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    <string.h>
#include    "extern.h"
#include    "serial.h"
#include    "flashprg.h"
#include    "analyse.h"
#include    "datasort.h"

////////////////////////////////////////////////////////////////////////////
// Incoming ir data is stored in 'ir_buffer'. This data needs to be integrity //
// checked. The data format is:                                               //
//                                                                            //
// ir_buffer[0]   == block length byte                                        //
// ir_buffer[1]   == block title byte                                         //
// ir_buffer[2]   == data byte 1                                              //
// ir_buffer[n]   == data byte 'n'                                            //
// ir_buffer[n+1] == hibyte checksum                                          //
// ir_buffer[n+2] == lobyte checksum                                          //
//                                                                            //
// Block length is the number of bytes in the block, EXCLUDING the checksum.  //
// Checksum is the 16 bit total of the block, EXCLUDING the checksum.         //
////////////////////////////////////////////////////////////////////////////
```

```
unsigned char CheckSumCheck( void )
{
union uUNSIGNED_INTEGER checksum;
unsigned char           ii;
unsigned char           block_length;



block_length = ir_buffer[BLOCK_LENGTH];

if ( block_length == 0x00 )
   {
   return 0;   // bad data
   }

////////////////////////////
// calculate the checksum //
////////////////////////////
checksum._16bit = 0;
for ( ii = 0; ii < block_length; ii++ )
   {
   checksum._16bit += ir_buffer[ii];
   }

////////////////////////////////////////////
// now compare to that received in 'ir_buffer' //
////////////////////////////////////////////
if ( checksum._8bit.hibyte == ir_buffer[ block_length  ] &&
     checksum._8bit.lobyte == ir_buffer[ block_length+1] )
   {
   return 1;   // good, full 16bit checksum agreement
   }

return 0;   // checksum did not compare
}  // CheckSumCheck()
//---------------------------------------------------------------------------


//////////////////////////////////////////////////////////////////////////
// If ir data has arrived it needs to be integrity checked and actioned if the//
// received command is valid. If the decoded command is for an internal flash //
// programming operation then 'ProgramFlash()' is called.                    //
//////////////////////////////////////////////////////////////////////////
void IRCommsCheck( void )
{                                       ////////////////////////////////////
if ( ir_mode == IR_MAIN )               // have we received any IR data via //
   {                                    // TIMERCHANNEL0 capture interupt?  //
   if ( CheckSumCheck() )               // Have we received 'clean' IR      //
      {                                 // comms data?                      //
      ServiceWatchDog();                ////////////////////////////////////

      if ( Decode_IR_Data() )           // is FLASH programming required?
         {
```

```
         ////////////////////////////////////////////////
         // interrupts off and reset the stack pointer as we   //
         // are NOT returning from this function and we will   //
         // be performing calls to the monitor functions       //
         ////////////////////////////////////////////////
         SEI();
         RSP();

         ProgramFlash();   // COP reset at the end of this function
         }
      }

   ir_mode = IR_IDLE; // ready for next
   }
}  // IRCommsCheck()
//---------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// We are about to use 'MONITOR_DATA[]' as storage, we know that we have to   //
// perform a FLASH PIR parameter program sequence.                            //
// Please note that 'MONITOR_DATA[]' overlays the runtime data.               //
// This was a specific linker request since once we have reached              //
// this point in the program flow we will not be returning to the             //
// normal PIR detect mode. 'MONITOR_DATA[]' has been linked to                //
// start at address $008c, this being a requirement of using the              //
// Motorola monitor routines.                                                 //
////////////////////////////////////////////////////////////////////////////
void AssignCurrentFLASHData( void )
{
memset( &MONITOR_DATA[0], 0xff       , sizeof(MONITOR_DATA) ); // erase (0xff)
memcpy( &MONITOR_DATA[0], &pir_params, sizeof(MONITOR_DATA) ); // assign

////////////////////////////////////////////////////////////////////////////
// The above 'memcpy()' has the same effect as:-                              //
//                                                                            //
// MONITOR_DATA[0]  = pir_params.trigger_count;     // adjustable             //
// MONITOR_DATA[1]  = pir_params.difference_band;   // adjustable             //
// MONITOR_DATA[2]  = pir_params.main_loop_count;   // adjustable             //
// MONITOR_DATA[3]  = delta_sig_bit._8bit.hibyte;   // adjustable             //
// MONITOR_DATA[4]  = delta_sig_bit._8bit.lobyte;   // adjustable             //
// MONITOR_DATA[5]  = delta_sig_event._8bit.hibyte; // adjustable             //
// MONITOR_DATA[6]  = delta_sig_event._8bit.lobyte; // adjustable             //
// MONITOR_DATA[7]  = password._8bit.hibyte;        // fixed!                 //
// MONITOR_DATA[8]  = password._8bit.lobyte;        // fixed!                 //
//     ..                                           // the password does not  //
//     .. allocate as required                      // get adjusted only      //
//     .. by your design                            // re-assigned            //
//     ..                                                                    //
// MONITOR_DATA[31]                                                          //
////////////////////////////////////////////////////////////////////////////
}  // AssignCurrentFLASHData()
//---------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// The ir data in 'ir_buffer' has been deemed correct. The software response  //
// depends on the BLOCK_TITLE byte as to what action is performed.            //
////////////////////////////////////////////////////////////////////////////
unsigned char Decode_IR_Data( void )
{
unsigned char              temp[2];
union uUNSIGNED_INTEGER    password_inverse;



////////////////
// initialise //
////////////////
temp[0] = temp[1] = 0;  // assume no flash programming will be required

//////////////////////
// assign new data //
//////////////////////
switch ( ir_buffer[BLOCK_TITLE] )
   {
   case SEND_A2D_TRIGGER :
   ir_buffer[DATA_BYTE1] = pir_params.trigger_count;
   ir_buffer[DATA_BYTE2] = PIR_BUFFER_SIZE;
   Send_IR_CommsPacket( SEND_A2D_TRIGGER, 2 );           // 2 data bytes
   break;

   case SEND_A2D_DIFFERENCE :
   ir_buffer[DATA_BYTE1] = pir_params.difference_band;
   Send_IR_CommsPacket( SEND_A2D_DIFFERENCE, 1 );        // 1 data byte
   break;

   case SEND_A2D_LOOPTIME :
   ir_buffer[DATA_BYTE1] = pir_params.main_loop_count;
   Send_IR_CommsPacket( SEND_A2D_LOOPTIME, 1 );          // 1 data byte
   break;

   case SEND_DELTA_SIG_RESOLUTION :
   ir_buffer[DATA_BYTE1] = delta_sig_bit._8bit.hibyte;
   ir_buffer[DATA_BYTE2] = delta_sig_bit._8bit.lobyte;
   Send_IR_CommsPacket( SEND_DELTA_SIG_RESOLUTION, 2 );  // 2 data bytes
   break;

   case SEND_DELTA_SIG_EVENT :
   ir_buffer[DATA_BYTE1] = delta_sig_event._8bit.hibyte;
   ir_buffer[DATA_BYTE2] = delta_sig_event._8bit.lobyte;
   ir_buffer[DATA_BYTE3] = delta_sig_bit._8bit.hibyte;   // used for range
   ir_buffer[DATA_BYTE4] = delta_sig_bit._8bit.lobyte;   // checking in
   Send_IR_CommsPacket( SEND_DELTA_SIG_EVENT, 4 );       // 4 data bytes
   break;

   case SEND_PASSWORD :
   ir_buffer[DATA_BYTE1]   = password._8bit.hibyte;
   ir_buffer[DATA_BYTE2]   = password._8bit.lobyte;
```

```
   ////////////////////////////
   // additional data integrity //
   ////////////////////////////
   password_inverse._16bit = ~password._16bit;
   ir_buffer[DATA_BYTE3]   = password_inverse._8bit.hibyte;
   ir_buffer[DATA_BYTE4]   = password_inverse._8bit.lobyte;
   Send_IR_CommsPacket( SEND_PASSWORD, 4 );              // 4 data bytes
   break;

   case UPDATE_A2D_TRIGGER :
   temp[0] = ir_buffer[DATA_BYTE1];
   AssignCurrentFLASHData();
   MONITOR_DATA[0] = temp[0];
   temp[0] = 1;   // flash programming required
   break;

   case UPDATE_A2D_DIFFERENCE :
   temp[0] = ir_buffer[DATA_BYTE1];
   AssignCurrentFLASHData();
   MONITOR_DATA[1] = temp[0];
   temp[0] = 1;   // flash programming required
   break;

   case UPDATE_A2D_LOOPTIME :
   temp[0] = ir_buffer[DATA_BYTE1];
   AssignCurrentFLASHData();
   MONITOR_DATA[2] = temp[0];
   temp[0] = 1;   // flash programming required
   break;

   case UPDATE_DELTA_SIG_RESOLUTION :
   temp[0] = ir_buffer[DATA_BYTE1];
   temp[1] = ir_buffer[DATA_BYTE2];
   AssignCurrentFLASHData();
   MONITOR_DATA[3]  = temp[0];
   MONITOR_DATA[4]  = temp[1];
   temp[0] = 1;   // flash programming required
   break;

   case UPDATE_DELTA_SIG_EVENT :
   temp[0] = ir_buffer[DATA_BYTE1];
   temp[1] = ir_buffer[DATA_BYTE2];
   AssignCurrentFLASHData();
   MONITOR_DATA[5]  = temp[0];
   MONITOR_DATA[6]  = temp[1];
   temp[0] = 1;   // flash programming required
   break;
   }

return temp[0];
}  // Decode_IR_Data()
//---------------------------------------------------------------------------
```

```
[PIR:datasort.h]
////////////////////////////////////////////////////////////////////////////
//       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 Infra Red Reference Design                          //
//  Filename  :  datasort.h                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for datasort.h                                                //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/07/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DATASORT_H_
#define     __DATASORT_H_


unsigned char  CheckSumCheck( void );
void           IRCommsCheck( void );
void           AssignCurrentFLASHData( void );
unsigned char  Decode_IR_Data( void );
#endif
```

```
[PIR:declared.h]
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   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     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 Infra Red Reference Design                        //
//  Filename  :  declared.h                                                 //
//  Author    :  jtravers                                                   //
//  Compiler  :  Cosmic ANSI-C                                              //
//  CPU       :  MC68HC908JK1/3                                             //
////////////////////  File Contents  ////////////////////////////////////////
//  declared data types                                                     //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  28/03/00 jt      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DECLARED_H_
#define     __DECLARED_H_



////////////////////
// bit/byte access //
////////////////////
struct sPORT
{
unsigned char bit0 : 1;
unsigned char bit1 : 1;
unsigned char bit2 : 1;
unsigned char bit3 : 1;
unsigned char bit4 : 1;
unsigned char bit5 : 1;
unsigned char bit6 : 1;
unsigned char bit7 : 1;
};
union uBITS
{
```

```
unsigned char  byte;
unsigned char  reg;
struct sPORT   bit;
};




//////////////////////
// 16 bit data type //
//////////////////////
struct sUNSIGNED_INTEGER
{
unsigned char hibyte;   // 0x12XX
unsigned char lobyte;   // 0xXX34
};
union uUNSIGNED_INTEGER
{
unsigned short int         _16bit;
struct sUNSIGNED_INTEGER   _8bit;
};

//////////////////////
// 32 bit data type //
//////////////////////
struct sUNSIGNED_LONG
{
unsigned char byte4; // 0x12XXXXXX
unsigned char byte3; // 0xXX34XXXX
unsigned char byte2; // 0xXXXX56XX
unsigned char byte1; // 0xXXXXXX78
};
union uUNSIGNED_LONG
{
unsigned long           _32bit;
struct sUNSIGNED_LONG   _8bit;
};


////////////////
// const data //
////////////////
struct sPIR_FLASH_PARAMETERS
{
unsigned char  trigger_count;    // how many triggers before event bit set
unsigned char  difference_band;  // difference in consecutive data readings to
                                 // create a trigger
unsigned char  main_loop_count;  // number of (10ms) main loop scans between
                                 // sensor reads
};

#endif
```

```
[PIR:define.h]
////////////////////////////////////////////////////////////////////////////
//       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 Infra Red Reference Design                          //
//  Filename  :  define.h                                                     //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  MC68HC908JK1/3                                               //
////////////////////  File Contents  ///////////////////////////////////////
//  global defines                                                            //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  28/03/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef         __DEFINE_H_
#define         __DEFINE_H_



#define  __MMDS_EMULATOR_     // requirement for emulator
//#define  __PC_DEBUG_          // enables code for RS232 data send to pc for
                              // sensor a2d data display
/////////////////
// I/O defines //
/////////////////
#define  MODE_SELECT_PIN      PTD.bit.bit6
#define  DETECT_LED           PTD.bit.bit2

//////////////
// #defines //
//////////////
#define NULL                  (void *)0
#define YES                   1
#define NO                    0
#define SET                   1
#define RESET                 0
#define HI                    1
#define LO                    0
```

```
#define ERROR                0xff
#define TIMER_ROLLOVER       10000
#define _3P5MS               3500
#define _4P5MS               4500
#define _1_BITWIDTH          1500
#define MIN_PULSE_WIDTH      200
#define MAX_PULSE_WIDTH      2500
#define NOISE_LIMIT          300   // 300us
#define RAMSTART             80
#define _1MS                 82
#define  A2D_STABILISATION   4  // approx 59us 11+(12*4) cycles @1us == 59us

////////////////////
// flags1 defines //
////////////////////
#define _10MS_MAINLOOP       bit0
#define ALARM_EVENT          bit1
#define DELTA_SIGMA_HISTORY  bit2
#define TRIGGER_EVENT        bit3

//////////////////////////
// Delta Sigma defines //
//////////////////////////
#define  _8BIT               256
#define  _9BIT               512
#define  _10BIT              1024
#define  _11BIT              2048
#define  _12BIT              4096
#define  _13BIT              8192
#define  _14BIT              16384
#define  _15BIT              32768U

////////////////////
// assembler 'C' //
////////////////////

//////////////////////////////////////////////////////////////////////////////
// We clear all ram areas upto $F8. Leave the area $F8-$FF as there has been  //
// two calls already. Called from 'MicroStartUp()' in 'startup.c'             //
//////////////////////////////////////////////////////////////////////////////
#define ClearRam()    _asm("clrh\n ldx #120\nLOOP2: clr $80-1,x\n dbnzx LOOP2")

//#define  RSP()             _asm( "rsp" )   // OK for JK1/3 and JL3
#define  RSP()             _asm( "ldhx #$00ff\n txs" )
#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

enum  { IR_IDLE, IR_DATA, IR_MAIN };

#endif
```

```
[PIR:delay.c]
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   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     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 Infra Red Reference Design                        //
//  Filename  :  delay.h                                                    //
//  Author    :  jtravers                                                   //
//  Compiler  :  Cosmic ANSI-C                                              //
//  CPU       :  MC68HC908JK1/3                                             //
////////////////////  File Contents  ////////////////////////////////////////
//  delay routines                                                          //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/00 jt      creation                                          //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#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             #X          ; delay arg sent to function  [2] cycles    //
// jsr             DelayUSecs  ; branches to this function   [5] cycles    //
//loop1 deca                   ; dec acc                     [1] cycle     //
//      nop                    ; nop                         [1] cycle     //
//      nop                    ; nop                         [1] cycle     //
//      nop                    ; nop                         [1] cycle     //
//      nop                    ; nop                         [1] cycle     //
//      nop                    ; nop                         [1] cycle     //
//      nop                    ; nop                         [1] cycle     //
//      nop                    ; nop                         [1] cycle     //
//      nop                    ; nop                         [1] cycle     //
```

```
//      bne             loop1   ; loop till acc = 0           [3] cycles      //
//      rts                     ; return from sub-routine     [4] cycles      //
//                                                                            //
// This gives a total delay of 11+12*X cycles, where X is the arg sent.       //
// We are using a 4.00MHz resonator => 1us bus cycle time. For eg:            //
// we require a 1ms delay, then we have:                                      //
// 1E-3/1E-6 = 1000 bus cycles => 1000 = 11 + 12*X, => X = 82.417             //
// approx  = 82                                                               //
// 'DelayUSecs( 82 )' to get 1ms delay.                                       //
//                                                                            //
// Arguments: 'X' delay value as calculated from 'cyles = 11 + 12X'           //
// Returns  : none                                                            //
////////////////////////////////////////////////////////////////////////////////
void Delay( unsigned char uSecs )
{
#asm
LOOP1:
   deca
   nop
   nop
   nop
   nop
   nop
   nop
   nop
   nop
   bne LOOP1
#endasm
}  // Delay()
//------------------------------------------------------------------------------
```

```
[PIR:delay.h]
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   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     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 Infra Red Reference Design                         //
// Filename  :  delay.h                                                     //
// Author    :  jtravers                                                    //
// Compiler  :  Cosmic ANSI-C                                               //
// CPU       :  68HC908JK1/3                                                //
////////////////////  File Contents  /////////////////////////////////////////
//  header file for delay.c                                                 //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                       //
//  ---  -------- ------  -----------------------------------------------------//
//  001  15/05/00 jt      creation                                           //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DELAY_H_
#define     __DELAY_H_

//////////////////////////
// function prototypes //
//////////////////////////
void Delay( unsigned char );
#endif
```

```
[PIR:deltasig.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 Infra Red Reference Design                          //
//  Filename  :  deltasig.c                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ///////////////////////////////////////
//  delta sigma pir snesor routine(s)                                         //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/08/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "serial.h"
#include    "deltasig.h"



////////////////////////////////////////////////////////////////////////////
// This function is called directly from 'main()'. It controls the Delta Sigma//
// value generation. The Delta Sigma variables both global and local are      //
// initialised for use. 'BuildDeltaSigma()' is then called which generates the//
// final Delta Sigma result.                                                  //
// A difference test is performed with the previous value, if this difference //
// is >= 'delta_sig_event._16bit' then a intruder detect event is signalled.  //
//                                                                            //
// Whilst debugging ie '__PC_DEBUG_' has been defined then the Delta Sigma    //
// parameters are serially transmitted to a connected pc for analysis         //
////////////////////////////////////////////////////////////////////////////
void DeltaSigma( void )
{
unsigned short int         diff;
union uUNSIGNED_INTEGER    ds_count;
```

```
ds_count._16bit           = delta_sig_bit._16bit;  // load from FLASH
delta_sigma_result._16bit = 0;                     // reset
DS_CHARGE_LINE_DDR        = 1;                     // output to charge/discharge
DS_FAST_CHARGE_LINE_DDR   = 0;                     // re-affirmation
PIR_ANALOGUE_DDR          = 0;                     // re-affirmation
DS_ANALOGUE_DDR           = 0;                     // re-affirmation

do {
   ServiceWatchDog();
   BuildDeltaSigma();
   }  while ( --ds_count._16bit );

//////////////////////////////////////////////////////
// suspend charge/discharge process until next time round //
//////////////////////////////////////////////////////
DS_CHARGE_LINE_DDR = 0;

////////////////////
// detection code //
////////////////////
if ( flags1.bit.DELTA_SIGMA_HISTORY )
   {
   if ( delta_sigma_result._16bit > delta_sigma_result_old )
      {
      diff = delta_sigma_result._16bit - delta_sigma_result_old;
      }
   else
      {
      diff = delta_sigma_result_old - delta_sigma_result._16bit;
      }

   /////////////////
   // event check //
   /////////////////
   if ( diff >= delta_sig_event._16bit && !flags1.bit.ALARM_EVENT )
      {
      flags1.bit.ALARM_EVENT = 1;   // an intruder has been detected?
      }
   }
else
   {
   flags1.bit.DELTA_SIGMA_HISTORY = 1; // denotes data ready for comparison
   }                                   // ie old/new readings to compare too

////////////////////////////////
// update for next comparison //
////////////////////////////////
delta_sigma_result_old = delta_sigma_result._16bit;


/////////////////
// serial send //
/////////////////
#ifdef   __PC_DEBUG_
SEI();
```

```
////////////////////////////
// xmit current data to pc //
////////////////////////////
rs232_buffer[2] = delta_sigma_result._8bit.hibyte;
rs232_buffer[3] = delta_sigma_result._8bit.lobyte;

if ( flags1.bit.ALARM_EVENT ) rs232_buffer[4] = 'Y';  // pc to 'beep'
else                          rs232_buffer[4] = 'N';  // no pc 'beep'

rs232_buffer[5] = delta_sig_bit._8bit.hibyte;   // bit resolution hibyte
rs232_buffer[6] = delta_sig_bit._8bit.lobyte;   // bit resolution lobyte
rs232_buffer[7] = delta_sig_event._8bit.hibyte; // event difference trigger
rs232_buffer[8] = delta_sig_event._8bit.lobyte; // event difference trigger

Send_RS232_CommsPacket( SIGMA_DATA, 7 );  // '7' from the 7 data bytes above

CLI();
#endif
}  // DeltaSigma()
//---------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// This function performs the Delta Sigma charge/discharge process. The pir   //
// sensor output is fed into PTB.5 that has been set up as an analogue input. //
// The aim of this function is to maintain the voltage on PTB.5 to be Vdd/2   //
// ie 2.5V. This is done by attempting to maintain an analogue read result of //
// 128 from PTB.5.                                                            //
//                                                                            //
// As the signal voltage varies from the sensor this function is continually  //
// charging/discharging capcitor C5 to maintain the 2.5V. This                //
// charge/discharge process requires symmetrical times hence the use of the   //
// balancing 'nop's. To ensure this symmetry all interrups are disabled.      //
//                                                                            //
// If the measured PTB.5 voltage is >= 2.5V the capacitor is discharged and   //
// 'delta_sigma_result' is incremented.                                       //
//                                                                            //
// FOR A DETAILED EXPLANATION OF THIS PROCESS SEE THE REFERENCE               //
// DESIGN DOCUMENTATION                                                       //
////////////////////////////////////////////////////////////////////////////
void BuildDeltaSigma( void )
{
SEI();

ADSCR.reg = CHANNEL5;      // single conversion, interrupt off
while ( !ADSCR.bit.COCO );

if ( ADR.reg < 128 )
   {
   DS_CHARGE_LINE = 1;  // charge
   #asm                 // timing balance
   nop
   nop
   nop
   nop
   nop
   nop
   nop
```

```
   nop
   nop
   nop
   nop
   nop
   #endasm
   }
else                        // a2d reading >= 128
   {
   DS_CHARGE_LINE = 0;      // discharge

   Nop();                   // timing balance

   if ( ++delta_sigma_result._8bit.lobyte == 0 )  // lobyte overflow check
      {
      delta_sigma_result._8bit.hibyte++;          // increment hibyte
      }
   else
      {
      #asm                                 // timing balance
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      #endasm
      }
   }

CLI();
}  // BuildDeltaSigma()
//---------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// Due to the large RC time contstants involved with this method, we have used//
// a fast charge method to speed up the charging/discharging process.         //
////////////////////////////////////////////////////////////////////////////
void FastCharge( void )
{
ADSCR.reg = CHANNEL5;            // single conversion, no int
while ( !ADSCR.bit.COCO );       // wait for the conversion to complete
                                 // 5v @ 256 steps == 19.53mV per step
                                 // 0.8V == 0.8/19.53mV == 40.96 == 41 integer
if ( ADR.reg < (128-41) )        // charge up to 0.8V less than required.
   {                             // 128 == 2.5V
   PIR_ANALOGUE_DDR        = 0;  // PIR analogue input
   DS_ANALOGUE_DDR         = 0;  // Delta Sigma analogue input
   DS_CHARGE_LINE          = 1;  // hi to charge...
   DS_CHARGE_LINE_DDR      = 1;  // ..delta-sigma charge/discharge line
   DS_CAP_GND              = 0;  // force ground side of coupling capcitor
   DS_CAP_GND_DDR          = 1;  // output
   DS_FAST_CHARGE_LINE     = 1;  // output and...
   DS_FAST_CHARGE_LINE_DDR = 1;  // ...hi to fast charge the capacitor

   do {
```

```
      ServiceWatchDog();
      ADSCR.reg = CHANNEL5;      // single conversion, no int
      while ( !ADSCR.bit.COCO );
      }  while ( ADR.reg < (128-41) );
   }
else                             // (ADR.reg > 128)   need to discharge
   {
   PIR_ANALOGUE_DDR        = 0;  // PIR analogue input
   DS_ANALOGUE_DDR         = 0;  // Delta Sigma analogue input
   DS_CHARGE_LINE          = 0;  // lo to discharge...
   DS_CHARGE_LINE_DDR      = 1;  // ..delta-sigma charge/discharge line
   DS_CAP_GND              = 0;  // force ground side of coupling capacitor
   DS_CAP_GND_DDR          = 1;  // output
   DS_FAST_CHARGE_LINE     = 0;  // output and...
   DS_FAST_CHARGE_LINE_DDR = 1;  // ...lo to fast discharge the capacitor

   do {
      ServiceWatchDog();
      ADSCR.reg = CHANNEL5;      // single conversion, no int
      while ( !ADSCR.bit.COCO );
      }  while ( ADR.reg > (128-41) );
   }

DS_FAST_CHARGE_LINE_DDR = 0;     // fast charge/discharge now not needed
DS_CAP_GND_DDR          = 0;     // release ground side of coupling capcitor
DS_CHARGE_LINE_DDR      = 0;     // suspend charge/discharge process until next
                                 // time round
}  // FastCharge()
//---------------------------------------------------------------------------
```

```
[PIR:deltasig.h]
////////////////////////////////////////////////////////////////////////////
//       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 Infra Red Reference Design                           //
// Filename  :  deltasig.h                                                    //
// Author    :  jtravers                                                      //
// Compiler  :  Cosmic ANSI-C                                                 //
// CPU       :  68HC908JK1/3                                                  //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for deltasig.c                                                //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/08/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DELTASIG_H_
#define     __DELTASIG_H_


#define  PIR_ANALOGUE_DDR        DDRB.bit.bit4
#define  DS_ANALOGUE_DDR         DDRB.bit.bit5
#define  DS_CHARGE_LINE_DDR      DDRB.bit.bit6
#define  DS_CHARGE_LINE          PTB.bit.bit6
#define  DS_FAST_CHARGE_LINE_DDR DDRB.bit.bit7
#define  DS_FAST_CHARGE_LINE     PTB.bit.bit7
#define  DS_CAP_GND              PTD.bit.bit5
#define  DS_CAP_GND_DDR          DDRD.bit.bit5

///////////////
// prototypes //
///////////////
void DeltaSigma( void );
void BuildDeltaSigma( void );
void FastCharge( void );
#endif
```

```
[PIR:extern.h]
////////////////////////////////////////////////////////////////////////////
//       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 Infra Red Reference Design                           //
// Filename  :  extern.h                                                      //
// Author    :  jtravers                                                      //
// Compiler  :  Cosmic ANSI-C                                                 //
// CPU       :  68HC908JK1/3                                                  //
////////////////////  File Contents  ///////////////////////////////////////
//  extern data,                                                              //
////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  28/03/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#define     __EXTERN_H_

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

#ifndef     __A2D_H_
#include    "a2d.h"
#endif

#ifndef     __JK13_JL3IO_H_
#include    "jk13&jl3.h"
#endif

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

```
//////////////////////
// Global variables //
//////////////////////
#ifdef   __PC_DEBUG_
extern @tiny unsigned char               rs232_buffer[12];
extern @tiny union uBITS                 rs232_data;
#endif

extern @tiny unsigned char               pir_buffer[PIR_BUFFER_SIZE];
extern @tiny unsigned char               * @tiny pir_buffer_ptr;
extern @tiny unsigned char               a2d_count;
extern @tiny unsigned char               MONITOR_DATA[32];
extern @tiny unsigned char               MONITOR_CPUSPD;
extern @tiny unsigned char               MONITOR_CTRLBYT;
extern @tiny unsigned short int          MONITOR_LADDR;
extern @tiny unsigned short int          delta_sigma_result_old;
extern @tiny union uUNSIGNED_INTEGER     delta_sigma_result;
extern @tiny volatile unsigned char      ir_buffer[8];
extern @tiny volatile union uBITS        flags1;
extern @tiny volatile unsigned char      ir_byte_count;
extern @tiny volatile unsigned char      ir_bit_count;
extern @tiny volatile unsigned char      ir_block_length;
extern @tiny volatile unsigned short int ir_start_time;
extern @tiny volatile unsigned short int ir_stop_time;
extern @tiny volatile unsigned short int ir_mode;
extern @tiny volatile unsigned short int detect_led_count;

//////////////////
// const section //
//////////////////
extern @near const struct sPIR_FLASH_PARAMETERS pir_params;
extern @near const union uUNSIGNED_INTEGER      delta_sig_bit;
extern @near const union uUNSIGNED_INTEGER      delta_sig_event;
extern @near const union uUNSIGNED_INTEGER      password;

#endif
```

```
[PIR:flashprg.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 Infra Red Reference Design                          //
//  Filename  :  flashmon.c                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic C                                                     //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ///////////////////////////////////////
//  Routine for FLASH programming whilst in user mode                         //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  05/07/00 jt      Internal FLASH programming whilst in user mode      //
//                        removes the requirement for an external memory ic.  //
//                        The programming is invoked via the receipt of valid //
//                        IR comms data packet in function                    //
//                        'datasort.c->IRCommsCheck()'                        //
//                                                                            //
//                        ram used :                                          //
//                        $80 - $87  [ 8]  not used in FLASH program process  //
//                        $88 -      [ 1]  control byte for monitor rom calls //
//                        $89 -      [ 1]  cpu speed byte for monitor rom     //
//                                         calls                              //
//                        $8a - $8b  [ 2]  last address for monitor rom calls //
//                        $8c - $ab  [32]  data bytes to be programmed into   //
//                                         FLASH                              //
//                        $ac - $ff  [84]  not used for FLASH programming,    //
//                                         stack usage will be required.      //
//                                   ----                                     //
//                        Total      128   bytes                              //
//                                                                            //
```

```
//                        Once the programming is complete the verify led is  //
//                        lit for 0.25s and the program then enters an endless//
//                        loop waiting for the internal watchdog to cause a   //
//                        reset and 'normal' processing ensues.               //
////////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "flashprg.h"



////////////////////////////////////////////////////////////////////////////////
// This function is invoked due to the decoding of a valid IR comms packet    //
// from the REMOTE (GP32) unit.                                               //
//                                                                            //
// Prior to this call interrupts have been turned off and the stack pointer   //
// reset (0xff). The data that requires programming into flash memory has been//
// copied into the appropriate buffer by a call to 'AssignCurrentFLASHData()'.//
//                                                                            //
// This function calls the Motorola monitor functions to perform the erase,   //
// program and verfiy of the flash memory data.                               //
//                                                                            //
// Please consult Motorola application note AN1831/D for full details of the  //
// Motorola monitor functions.                                                //
////////////////////////////////////////////////////////////////////////////////
void ProgramFlash( void )
{
unsigned char  ii;


                                  ////////////////////////////////////////////
ServiceWatchDog();                // defensive measure                        //
FLBPR.reg       = 0xff;           // no FLASH protection                      //
MONITOR_CPUSPD  = SPDSET;         // 1(MHz) * 4 == 4                          //
MONITOR_CTRLBYT = 0x00;           // page erase                               //
MONITOR_LADDR   = FLASH_DATA_END; // data stored @ $FBC0/DF (32 bytes)        //
LED             = 0;              // led off...                               //
LED_DDR         = 1;              // ...and an output                         //
                                  //                                          //
_asm("ldhx  #$fbc8");             // any address in the range $fbc0 - $fbff   //
ERARNGE();                        // to erase FLASH page, Motorola monitor    //
                                  // rom call                                 //
_asm("ldhx  #$fbc0");             // first address in H:X to write to         //
PRGRNGE();                        // program FLASH row, Motorola monitor      //
                                  // rom call                                 //
_asm("lda   #$ff");               // force ACC to non zero to ensure that     //
                                  // newly read data is placed back in the    //
                                  // data array and not to the monitor mode   //
                                  // comm port.                               //
_asm("ldhx  #$fbc0");             // first address in H:X to verify FLASH     //
RDVRRNG();                        // programming, Motorola monitor rom call   //
                                  //                                          //
if ( carry() )                    // carry bit set if verify is successful    //
   {                              // if so light led for 0.25s                //
   ii = 125;                      // load 0.25s counter                       //
                                  //                                          //
```

```
   do {                             //                                        //
      ServiceWatchDog();            //                                        //
                                    //                                        //
      LED = 1;                      // led on                                 //
                                    //                                        //
      _asm("lda   #4"  );           // Fop*4 (1MHz)                           //
      _asm("ldx   #167");           // 2000/12                                //
      DELNUS();                     // 2ms delay...Motorola monitor rom call  //
      }  while ( --ii );            // repeat                                 //
   }                                //                                        //
                                    //                                        //
LED = 0;                            // led off                                //
while (1);                          // all done! wait for watchdog reset...   //
                                    ////////////////////////////////////////////
}  // ProgramFlash()
//---------------------------------------------------------------------------
```

```
[PIR:flashprh.h]
////////////////////////////////////////////////////////////////////////////
//       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 PIR Reference Design                               //
//  Filename  :  flashprg.h                                                  //
//  Author    :  jtravers                                                    //
//  Compiler  :  Cosmic ANSI-C                                               //
//  CPU       :  68HC908JK1/3                                                //
////////////////////  File Contents  ////////////////////////////////////////
//  header file for 'flashprg.c'                                             //
////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                       //
//  ---  -------- ------  ----------------------------------------------------//
//  001  05/07/00 jt      creation                                           //
////////////////////////////////////////////////////////////////////////////
#ifndef     __FLASHPRG_H_
#define     __FLASHPRG_H_
```

```
////////////////////////////////////////////////////////////
// Fop * 4, here Fop == 1 (MHz) => 1*4 = 4, adjust as required //
////////////////////////////////////////////////////////////
#define     SPDSET            4

////////////////////////////////////////////////////////////
// currently data size for FLASH programming is 32 bytes 9/10/00 //
////////////////////////////////////////////////////////////
#define     DATA_SIZE         32

////////////////////////////////////////////////////////////////////////
// 16 bytes of data ie $fbc0...$fbdf == $fbc0+32-1, adjust as required //
////////////////////////////////////////////////////////////////////////
#define     FLASH_DATA_END    0xfbc0+DATA_SIZE-1

////////////////////////////////////////////////
// symbol table entries used purely for addressing //
////////////////////////////////////////////////
extern @near void GETBYTE( void )   @MONITOR_ROM+0 ; // Motorla monitor rom call
extern @near void RDVRRNG( void )   @MONITOR_ROM+3 ; // Motorla monitor rom call
extern @near void ERARNGE( void )   @MONITOR_ROM+6 ; // Motorla monitor rom call
extern @near void PRGRNGE( void )   @MONITOR_ROM+9 ; // Motorla monitor rom call
extern @near void DELNUS( void )    @MONITOR_ROM+12; // Motorla monitor rom call

#define     LED               PTD.bit.bit2
#define     LED_DDR           DDRD.bit.bit2

///////////////
// prototypes //
///////////////
void ProgramFlash( void );
#endif
```

[PIR:interrup.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 Infra Red Reference Design                          //
//  Filename  :  interrup.c                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ///////////////////////////////////////////
//  interrupt service routines                                                //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/08/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////////
#include    <string.h>
#include    "extern.h"
#include    "serial.h"
#include    "interrup.h"


////////////////////////////////////////////////////////////////////////////////
// The timeroverflow interrupt provides us with a 'main()' routine time base. //
// The overflow value is set at 'TIMER_ROLLOVER' which is 10000us, 10ms.      //
////////////////////////////////////////////////////////////////////////////////
@interrupt void TIMEROVERFLOW( void )
{
if ( TSC.bit.TOF && TSC.bit.TOIE )
   {
   TSC.bit.TOF = 0;                    // clear interrupt flag

   if ( flags1.bit.ALARM_EVENT )       // has an intruder been detected?
      {
      DETECT_LED = 1;                  // led on

      ////////////////////////////////////
      // led on for 6s per intruder detect //
      ////////////////////////////////////
      if ( ++detect_led_count >= 600 ) // 600*10ms == 6s
         {
         DETECT_LED             = 0;   // led off
         flags1.bit.ALARM_EVENT = 0;   // reset
         detect_led_count       = 0;   // reset
         }
      }
   else
      {
      detect_led_count       = 0;      // reset, re-affirmation
      }

   flags1.bit._10MS_MAINLOOP = 1;      // 'main()' loop synchroniser
   }
}  // TIMEROVERFLOW()
//------------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// Timer channel0 capture interrupt routine                                //
// This interrupt performs the ir comms capturing. The incoming ir comms   //
// will look like:                                                         //
//                                                                         //
// Logic 0 level as seen by receiving PTD.4:                               //
//                     ------------                                        //
//      |              |            |                                      //
//      |              |            |                                      //
//      |              |            |                                      //
//      |______________|            |                                      //
//                                                                         //
//      <-- 700us --><---700us -->                                         //
//                                                                         //
// Logic 1 level as seen by receiving PTD.4:                               //
//                     ----------------------------------------            //
//      |              |                                        |          //
//      |              |                                        |          //
//      |              |                                        |          //
//      |______________|                                        |          //
//                                                                         //
//      <-- 700us --><----------- 2100us -------------------->            //
//                                                                         //
// Leader pulse as seen by receiving PTD.4:                                //
//                     ------------                                        //
//      |              |            |                                      //
//      |              |            |                                      //
//      |              |            |                                      //
//      |______________|            |                                      //
//                                                                         //
//      <--- 4ms ---><--- 4ms --->     [NOTE: ms units here!]              //
//                                                                         //
// The micro measures the width of the +ve pulse to determine the          //
// bit value (or leader pulse).                                            //
//                                                                         //
// When all expected IR data has been received 'flags1.bit.CHECK_IR_DATA' is //
// set. This allows the decoding of this data to occur in 'main()' as part of //
// the normal program flow. When the ir buffer data has been checked then  //
// 'ir_mode' is then set to IR_IDLE.                                       //
////////////////////////////////////////////////////////////////////////////
@interrupt void TIMERCHANNEL0( void )
{
unsigned short int   time_diff;


if ( TSC0.bit.CH0F && TSC0.bit.CH0IE )
   {
   TSC0.bit.CH0F = 0;              // clear interrupt flag

   if ( TSC0.bit.ELS0A && !TSC0.bit.ELS0B ) // +ve edge event
      {
      ir_start_time  = TCH0;       // time stamp +ve edge
      TSC0.bit.ELS0A = 0;
      TSC0.bit.ELS0B = 1;          // -ve edge next
```

```
   }
else                            // -ve edge event
   {
   ir_stop_time = TCH0;         // time stamp -ve edge

   ////////////////////////////
   // pulse width calculation //
   ////////////////////////////
   if ( ir_stop_time >= ir_start_time )
      {
      time_diff = ir_stop_time - ir_start_time;
      }
   else  // rollover compensation
      {
      time_diff = (TIMER_ROLLOVER - ir_start_time) + ir_stop_time;
      }

   ////////////////////////////////////////////////////////////////////
   // Is this pulse an IR comms packet leader START pulse (approx 4ms) //
   ////////////////////////////////////////////////////////////////////
   if ( time_diff >= _3P5MS && time_diff <= _4P5MS && ir_mode == IR_IDLE )
      {
      memset( &ir_buffer[0], 0x00, sizeof(ir_buffer) );  // clear buffer...
      ir_byte_count   = 0;                               // ...and ssociated
      ir_bit_count    = 0;                               // ...IR comms build
      ir_block_length = 0;                               // ...variables
      ir_mode         = IR_DATA;
      }
   else if ( ir_mode == IR_DATA )// must be building the bit pattern...
      {
      ////////////////////////////////////////////////////
      // is this pulse in the acceptable pulse width region //
      ////////////////////////////////////////////////////
      if ( time_diff >= MIN_PULSE_WIDTH && time_diff <= MAX_PULSE_WIDTH )
         {
         ////////////////////////////////////////////////////////
         // has a '1' arrived, if so set the 'bit_count' bit ie if //
         // 'bit_count' is 3 then set bit3 of 'temp' etc           //
         ////////////////////////////////////////////////////////
         if ( time_diff >= _1_BITWIDTH )
            {
            ir_buffer[ir_byte_count] |= (unsigned char)(0x01<<ir_bit_count);
            }

         ////////////////////////////////
         // have we received a byte yet //
         ////////////////////////////////
         if ( ++ir_bit_count >= 8 )
            {
            ir_bit_count = 0;       // reset for next count of 8

            if ( !ir_byte_count )   //  == 0, first byte...block length byte
               {
```

```
                  ////////////////////////////////////////////
                  // total bytes expected is 'block_length+2' //
                  // '2' for checksum hi and lo bytes         //
                  ////////////////////////////////////////////
                  ir_block_length = (unsigned char)(ir_buffer[0] + 2);

                  ////////////////////////
                  // buffer write clamp //
                  ////////////////////////
                  if ( ir_block_length > sizeof(ir_buffer) )
                     {
                     ////////////////////////
                     // corrupt data, abort //
                     ////////////////////////
                     ir_mode        = IR_IDLE;
                     TSC0.bit.ELS0A = 1;           // +ve edge...
                     TSC0.bit.ELS0B = 0;           //           ...next
                     return;
                     }
                  }

               ////////////////////////////////////////////////////
               // have we received the expected number of data bytes //
               ////////////////////////////////////////////////////
               if ( ++ir_byte_count >= ir_block_length )
                  {                    ///////////////////////////////////
                  ir_mode = IR_MAIN;  // check data validity on        //
                                      // return to 'main()' in call to //
                                      // 'IRCommsCheck()'              //
                  }                   ///////////////////////////////////
               }     // 'if ( ++ir_bit_count >= 8 )'
            }        // 'if ( time_diff >= MIN_.. && time_diff <= MAX_.. )'
         }           // 'else if ( ir_mode == IR_DATA )'

      TSC0.bit.ELS0A = 1;           // +ve edge...
      TSC0.bit.ELS0B = 0;           //           ...next
      }              // -ve edge interupt
   }
}  // TIMERCHANNEL0()
//---------------------------------------------------------------------------
```

```
[PIR:interrup.h]
////////////////////////////////////////////////////////////////////////////
//       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 Infra Red Reference Design                          //
//  Filename  :  interrup.h                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for interrup.c                                                //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/08/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __INTERRUP_H_
#define     __INTERRUP_H_


////////////////
// prototypes //
////////////////
@interrupt void TIMERCHANNEL0( void );
@interrupt void TIMEROVERFLOW( void );

#endif
```

```
[PIR:ireg.s]
; INTEGER EXTENSION
; Copyright (c) 1995 by COSMIC Software
;
  switch  .ubsct
  xdef  c_reg
;
c_reg:
  ds.b  1
;
  end
```

```
[PIR:jk.lkf]
#####################################################
# LINKER COMMAND FILE FOR MOTOROLA HC908JK1/3/JL3 #
# PIR REFERENCE DESIGN                              #
# ATEECC July 2000                                  #
#####################################################


############
# symbols #
############
+def __memory=@.bss                    # symbol used by startup
+def __stack=0x00ff                    # stack pointer value for 'crts.s'

###################################
# MC68HC908JK1/JK3 selection      #
#                                 #
# JK1:                            #
# ROM_START=0xf600, ROM_SIZE=1536 #
#                                 #
# JK3/JL3:                        #
# ROM_START=0xec00, ROM_SIZE=4096 #
###################################
+def ROM_START=0xec00
+def ROM_SIZE=4096


               ##############
               # CONST DATA #
               ##############
#           -b [b]eginning address of segment
#                     -n [n]ame of segment
#                                  -m [m]ax size (bytes) of segment
+seg .const -b 0xfbc0 -n ConstFLASH -m 64       ###################################
                                                # 64 bytes is min erase block  #
                                                # size                         #
                                                # WE USE THE LAST 64 BYTE BLOCK#
                                                # IN THE FLASH MEMORY AREA     #
                                                ###################################
```

```
              #############
              # PAGE0 RAM #
              #############

############################
# run time data allocation #
############################                    ################################
+seg .ubsct -b 0x0080 -n TinyRam -m 128         # Occupies $080-$00ff (PAGE0). #
ireg.o                                          # This ensures that the Cosmic #
                                                # variables 'c_reg'            #
lreg.o                                          # and 'c_lreg' are positioned  #
                                                # at the beginning of ram this #
                                                # segment, ensuring that during#
                                                # any memcpy operations they do#
                                                # not get overwritten with     #
                                                # copied data.                 #
                                                #                              #
data.o                                          # NOTE: user global data here  #
                                                ################################

#########################################################
# This segment is for PIR FLASH parameter programming.#
# The variables from 'data.o' and 'mot_data.o' will   #
# overlap, that is ok since the variables occupying   #
# the same address will not be active at the same     #
# time. See 'datasort.c->AssignCurrentFLASHData()'    #
# for more information.                               #
#                                                     #
# The Motorola monitor routines expect their          #
# variables/data to be at known addresses.            #
#                                                     #
# Notice the '-v' switch, it tells the linker         #
# not to report overlap errors for this segment       #
#########################################################
+seg .ubsct -b 0x88 -v -n MONITOR_RAM -m 128-8  ########################
mon_data.o                                      # '8' since this segment#
                                                # starts at $0088 and   #
                                                # not $0080             #
                                                ########################

##############################
# FLASH memory for user code #
##############################
+seg .text -b ROM_START -n UserFLASH -m ROM_SIZE-64
                                                ################################
                                                # MC68HC908JK/L3 user code     #
                                                # start address                #
                                                # '64' for const FLASH         #
                                                # variables, see 'ConstFLASH' #
                                                # segment                      #
                                                ################################
```

```
#####################################
# const area for switch jump tables #
#####################################          ###############################
+seg .const -a UserFLASH                       # '-a' append section to      #
                                               # previous                    #
                                               ###############################

######################
# user object files #
######################
crts.o          # Cosmic supplied startup routine
a2d.o           # a2d initialise/read
analyse.o       # data buffer scan routine, buffer contains PIR a2d values
datasort.o      # data intergity and decode
delay.o         # inline accurate delay routine
deltasig.o      # alternative pir 'event' routines using using delta-sigma
                # algorithm
flashprg.o      # flash programming
interrup.o      # interrupt service routines
main.o          # main()
serial.o        # RS232 debug (send) and IR comms routines
startup.o       # micro initilisation ie i/o, ram clear, timer initialisation

####################
# Cosmic libraries #
####################
c:/cosmic/cx08/lib/libi.h08
c:/cosmic/cx08/lib/libm.h08

############
# Vectors #
############
+seg .const -b 0xffde -n Vectors -m 34
vectors.o
```

```
[PIR:jk13&jl3.h]
////////////////////////////////////////////////////////////////////////////
//       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 HC08 Infra Red Reference Design                     //
//  Filename  :  JK13&JL3.H                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ////////////////////////////////////////
//  Compiler Register Definitions For MC68HC08JK1/JK3/JL3 Motorola            //
// 8 bit microcontrollers                                                     //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------      //
//  001  28/03/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DECLARED_H_
#include    "declared.h"
#endif

#ifndef     __JK13_JL3IO_H_
#define     __JK13_JL3IO_H_


//////////////////
// CPU Registers //
//////////////////
@tiny volatile union uBITS          PTA             @0x00;
@tiny volatile union uBITS          PTB             @0x01;
@tiny volatile union uBITS          PTD             @0x03;
@tiny volatile union uBITS          DDRA            @0x04;
@tiny volatile union uBITS          DDRB            @0x05;
@tiny volatile union uBITS          DDRD            @0x07;
@tiny volatile union uBITS          PDCR            @0x0A;
@tiny volatile union uBITS          PTAPUE          @0x0D;
@tiny volatile union uBITS          KBSCR           @0x1A;
```

```
@tiny volatile union uBITS          KBIER          @0x1B;
@tiny volatile union uBITS          INTSCR         @0x1D;
@tiny volatile union uBITS          CONFIG2        @0x1E;
@tiny volatile union uBITS          CONFIG1        @0x1F;
@tiny volatile union uBITS          TSC            @0x20;
@tiny volatile union uBITS          TCNTH          @0x21;
@tiny volatile unsigned short int   TCNT           @0x21;
@tiny volatile union uBITS          TCNTL          @0x22;
@tiny volatile union uBITS          TMODH          @0x23;
@tiny volatile unsigned short int   TMOD           @0x23;
@tiny volatile union uBITS          TMODL          @0x24;
@tiny volatile union uBITS          TSC0           @0x25;
@tiny volatile union uBITS          TCH0H          @0x26;
@tiny volatile unsigned short int   TCH0           @0x26;
@tiny volatile union uBITS          TCH0L          @0x27;
@tiny volatile union uBITS          TSC1           @0x28;
@tiny volatile union uBITS          TCH1H          @0x29;
@tiny volatile unsigned short int   TCH1           @0x29;
@tiny volatile union uBITS          TCH1L          @0x2A;
@tiny volatile union uBITS          ADSCR          @0x3C;
@tiny volatile union uBITS          ADR            @0x3D;
@tiny volatile union uBITS          ADICLK         @0x3E;
@near volatile union uBITS          BSR            @0xFE00;
@near volatile union uBITS          RSR            @0xFE01;
@near volatile union uBITS          BFCR           @0xFE03;
@near volatile union uBITS          INT1           @0xFE04;
@near volatile union uBITS          INT2           @0xFE05;
@near volatile union uBITS          INT3           @0xFE06;
@near volatile union uBITS          FLCR           @0xFE08;
@near volatile union uBITS          FLBPR          @0xFE09;
@near volatile union uBITS          BRKH           @0xFE0C;
@near volatile unsigned short int   BRK            @0xFE0C;
@near volatile union uBITS          BRKL           @0xFE0D;
@near volatile union uBITS          BRKSCR         @0xFE0E;
@near volatile union uBITS          COPCTL         @0xFFFF;

//////////////////////////////////////////////////////
// access to the HC08 condition code reg : carry flag //
//////////////////////////////////////////////////////
@builtin unsigned char carry( void );

///////////
// INT1 //
///////////
#define  IF1      bit2
#define  IF3      bit4
#define  IF4      bit5
#define  IF5      bit6

///////////
// INT2 //
///////////
#define  IF14     bit7
```

```
//////////
// INT3 //
//////////
#define  IF15     bit0

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

//////////////
// TSC0 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 //
////////////////////////////
#define CH0       bit0
#define CH1       bit1
#define CH2       bit2
#define CH3       bit3
#define CH4       bit4
#define ADCO      bit5
#define AIEN      bit6
#define COCO      bit7
```

```
//////////////////////////
// A2D input clock reg //
//////////////////////////
#define ADIV0     bit5
#define ADIV1     bit6
#define ADIV2     bit7

////////////////////
// FLASH control //
////////////////////
#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

////////////////////////////////////
// Monitor ROM Code Start Address //
////////////////////////////////////
#define  MONITOR_ROM    0xFC00

#endif
```

```
[PIR:link08.bat]
@echo off
c:\cosmic\cx08\clnk  -v -m jk.inf -e jk.err -o pir.h08 jk.lkf
c:\cosmic\cx08\chex  -fm -o pir.s19 pir.h08
c:\cosmic\cx08\clabs -l -v pir.h08
```

```
[PIR:lreg.s]
; LONG/FLOAT ACCUMULATOR
; Copyright (c) 1995 by COSMIC Software
;
  switch  .ubsct
  xdef  c_lreg
;
c_lreg:
  ds.b  4
;
  end
```

```
[PIR:main.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 Infra Red Reference Design                          //
//  Filename  :  main.c                                                       //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ////////////////////////////////////////
//  'main' routine                                                            //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  28/03/00 jt      creation                                            //
//                                                                            //
//       21/10/00 jt      Improved analogue buffer detection algorithm,       //
//                        reduced buffer size to 8. Now send buffer size to   //
//                        remote unit, so trigger size check can be performed.//
//                        v1.0                                                //
//                        THIS CODE SENT TO MOTORLA ON 22/11/00, FIRST RELEASE//
//                        Checksum : $2F13E                                   //
//       23/01/01 jt      Improved ir comms robustness with changes in        //
```

```
//                        'interrup.c->TIMERCHANNEL0'. Clamped the min Delta  //
//                        Sigma resolution to 10 bits, due to system leakage .//
//                        Checksum : $2F9BA                                   //
//       07/02/01 jt      Default pir parameters changed to:                  //
//                        'trigger_count'   == 4                              //
//                        'difference_band' == 6                              //
//                        'main_loop_count' == 10                             //
//                        v1.1                                                //
//                        SECOND RELEASE TO MOTOROLA                          //
//                        Checksum : $2F20C                                   //
////////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "startup.h"
#include    "datasort.h"
#include    "deltasig.h"




////////////
// main() //
////////////
void main( void )
{
unsigned char  ii;


MicroStartUp();

while( 1 )
   {
   ////////////////////////////////////////
   // use analogue/op amp detect algorithm? //
   ////////////////////////////////////////
   if ( MODE_SELECT_PIN )
      {
      ServiceWatchDog();
      A2DCheck();          // intruder detect check

      while ( !flags1.bit._10MS_MAINLOOP );  // main() loop sync
      flags1.bit._10MS_MAINLOOP = 0;

      IRCommsCheck();   // check for received ir comms packets
      }
   else
      ////////////////////////////////////////////
      // use delta-sigma movement detect algorithm //
      ////////////////////////////////////////////
      {
      ServiceWatchDog();
      DeltaSigma();          // intruder detect check
```

```
      ////////////////////////////////////////////////////////////
      // an appreciable time here (100ms) to service any incoming IR  //
      // comms, response can be slightly sluggish due to long 'SEI()' //
      // within 'DeltaSigma()'.                                       //
      ////////////////////////////////////////////////////////////
      for ( ii = 0; ii < 10; ii++ )
         {
         ServiceWatchDog();

         while ( !flags1.bit._10MS_MAINLOOP );  // main() loop sync
         flags1.bit._10MS_MAINLOOP = 0;

         IRCommsCheck();   // check for received ir comms packets
         }
      }

   ReAffirmDDR(); // data direction re-affirmation
   }
}  // main()
//---------------------------------------------------------------------------
```

```
[PIR:make08.bat]
@echo off

rem//////////////////////////
rem// assemble Cosmic files //
rem//////////////////////////
c:\cosmic\cx08\ca6808 crts.s
c:\cosmic\cx08\ca6808 ireg.s
c:\cosmic\cx08\ca6808 lreg.s

rem////////////////////////////
rem// compile all source files //
rem////////////////////////////
call cc a2d
call cc analyse
call cc data
call cc datasort
call cc delay
call cc deltasig
call cc flashprg
call cc interrup
call cc main
call cc mon_data
call cc serial
call cc startup
call cc vectors

rem//////////////////////////
rem// link the object files //
rem//////////////////////////
call link08

rem////////////////////////////////
rem// deleting relative listings //
rem////////////////////////////////
del *.ls

rem//////////////////////////
rem// list any error files //
rem//////////////////////////
dir *.err
```

```
[PIR:mon_data.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 Infra Red Reference Design                          //
//  Filename  :  mon_data.c                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  MC68HC908JK1/3                                               //
////////////////////  File Contents  ///////////////////////////////////////////
//  monitor rom function data                                                 //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ------------------------------------------------------//
//  001  09/10/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////

//////////////////////
// Global variables //
//////////////////////
@tiny unsigned char        MONITOR_DATA[32];   // $008c   NOTE1
@tiny unsigned short int   MONITOR_LADDR;      // $008a   NOTE1
@tiny unsigned char        MONITOR_CPUSPD;     // $0089   NOTE1
@tiny unsigned char        MONITOR_CTRLBYT;    // $0088   NOTE1
////////////////////////////////////////////////////////////////////////////
// NOTE1:                                                                     //
// These variables must be assigned to previously known addresses by the      //
// linker. The Motorola monitor rom functions expect them to be at those      //
// addresses.                                                                 //
//                                                                            //
// MOTOROLA_CTRLBYT  @0x0088  when bit6 is set mass erase else page erase...  //
//                            ...used by Motorola erase routine               //
// MOTOROLA_CPUSPD   @0x0089  == 4 (Fop*4, here 4MHZ resonator =>Fop==1MHz)...//
//                            ...used by Motorola delay routine               //
// MOTOROLA_LADDR    @0x008a  $8a and $8b 'last address' for Motorola          //
//                            programming and erasing routines                //
// MOTOROLA_DATA[]   @0x008c  data to be programmed into a FLASH row          //
////////////////////////////////////////////////////////////////////////////
```

```
[PIR:serial.c]
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   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     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 Infra Red Reference Design                        //
//  Filename  :  serial.c                                                   //
//  Author    :  jtravers                                                   //
//  Compiler  :  Cosmic ANSI-C                                              //
//  CPU       :  68HC908JK1/3                                               //
////////////////////  File Contents  ////////////////////////////////////////
//  RS232 routines                                                          //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/05/00 jt      creation                                          //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "serial.h"



#ifdef   __PC_DEBUG_    // are pc comms for debug required
////////////////////////////////////////////////////////////////////////////
// This function sends it's function parameter out on the TX pin in RS232   //
// format at 38400 bit rate. The data format is :                           //
//                                                                          //
// 1 START - 8 DATA - 1 STOP - NOPARITY                                     //
//                                                                          //
// argument : data to send                                                  //
// returns  : none                                                          //
////////////////////////////////////////////////////////////////////////////
void Send_RS232_Byte( unsigned char send_data )
```

```
{                              ////////////////////////////////////////////////
rs232_data.byte = send_data;  // using global assignment as the compiled code //
                              // for the bit set/clear is smaller than using  //
                              // local 'send_data'                            //
                              ////////////////////////////////////////////////

//////////////////////////////
// re-affirm data direction //
//////////////////////////////
RS232_TX_DDR = 1;

///////////////
// START bit //
///////////////
RS232_TX = 0;
DelayBitTime();

//////////////////////////
// xmit byte, bit by bit //
//////////////////////////
if(rs232_data.bit.bit0) {RS232_TX = 1;DelayHiBitTime();}
else                    {RS232_TX = 0;DelayBitTime();  }
if(rs232_data.bit.bit1) {RS232_TX = 1;DelayHiBitTime();}
else                    {RS232_TX = 0;DelayBitTime();  }
if(rs232_data.bit.bit2) {RS232_TX = 1;DelayHiBitTime();}
else                    {RS232_TX = 0;DelayBitTime();  }
if(rs232_data.bit.bit3) {RS232_TX = 1;DelayHiBitTime();}
else                    {RS232_TX = 0;DelayBitTime();  }
if(rs232_data.bit.bit4) {RS232_TX = 1;DelayHiBitTime();}
else                    {RS232_TX = 0;DelayBitTime();  }
if(rs232_data.bit.bit5) {RS232_TX = 1;DelayHiBitTime();}
else                    {RS232_TX = 0;DelayBitTime();  }
if(rs232_data.bit.bit6) {RS232_TX = 1;DelayHiBitTime();}
else                    {RS232_TX = 0;DelayBitTime();  }
if(rs232_data.bit.bit7) {RS232_TX = 1;DelayHiBitTime();}
else                    {RS232_TX = 0;DelayBitTime();

                        ////////////////////////////////////////////////
                        // if last bit is a 0 need to pad width out here //
                        // to be the same as 'if (rs232_data.bit.bitX)'  //
                        // which is [5] bus cycles                      //
                        ////////////////////////////////////////////////
                        Nop();Nop();Nop();Nop();Nop();
                        }
//////////////
// STOP bit //
//////////////
RS232_TX = 1;
DelayBitTime();
}  // Send_RS232_Byte()
//---------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// 38400 bit rate STOP/START and LO bit time                                //
////////////////////////////////////////////////////////////////////////////
void DelayBitTime( void  )
{
Nop();Nop();Nop();Nop();Nop();Nop();Nop();Nop();Nop();
}  // DelayBitTime()
//--------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// 38400 bit rate 'HI' bit time                                             //
////////////////////////////////////////////////////////////////////////////
void DelayHiBitTime( void  )
{
Nop();Nop();Nop();Nop();Nop();Nop();
}  // DelayHiBitTime()
//--------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// This function transmits a serial buffer contents at 38400 bit rate using:- //
//             1 start - 8 data - 1 stop and no parity.                     //
//                                                                          //
// The block title is dervived from an 'enum' statement, see 'serail.h'. The  //
// 'block title' function argument serves only to identify to the recipient   //
// what type of data it is. The 'block_length' function argument denotes how  //
// many bytes of data there are, this is immediatly incremented by 2 to       //
// reflect the 'block_title' and 'block_length' bytes. Prior to transmission  //
// 'block_length' is again incremented by 2 to reflect the 16 bit checksum    //
// hi:lo bytes.                                                             //
//                                                                          //
// COMMS PACKET STRUCTURE                                                   //
// ======================                                                   //
// rs232_buffer[0]   == block length byte                                   //
// rs232_buffer[1]   == block title byte                                    //
// rs232_buffer[2]   == data byte 1                                         //
// rs232_buffer[n]   == data byte 'n'                                       //
// rs232_buffer[n+1] == hibyte checksum                                     //
// rs232_buffer[n+2] == lobyte checksum                                     //
//                                                                          //
// Block length is the number of bytes in the block, EXCLUDING the checksum.  //
// Checksum is the 16 bit total of the block, EXCLUDING the checksum.       //
//                                                                          //
// The pc program will trigger when it receives:                            //
// "ATEECC\x07\xf0" or "ATEECC\x09\xf1" these correspond to analogue and      //
// Delta Sigma data packets respectively.                                   //
// The "ATEECC" preamble was used to make the trigger string unique. It is    //
// possible that "\x07\xf0" and "\x09\xf1" could be contained in the          //
// data bytes as well as at the 'block_length' and 'block_title' locations.   //
// This could potentailly cause problems, I took no chances and used the      //
// preamble                                                                 //
//                                                                          //
// argument : block title and block length for buffer assignment            //
// returns  : none                                                          //
////////////////////////////////////////////////////////////////////////////
```

```
void Send_RS232_CommsPacket( unsigned char block_title,
                                                    unsigned char block_length )
{
union uUNSIGNED_INTEGER checksum;
unsigned char           ii;


////////////////////////////////////////////////////////
// insert the element values into the 'rs232_buffer' array //
////////////////////////////////////////////////////////
block_length              += 2;  // block_length+block_title bytes == 2
rs232_buffer[BLOCK_LENGTH] = block_length;
rs232_buffer[BLOCK_TITLE ] = block_title;

///////////////////////////////////
// calculate the packet checksum //
///////////////////////////////////
checksum._16bit = 0;
for ( ii = 0; ii < block_length; ii++ )
   {
   checksum._16bit += rs232_buffer[ii];
   }

/////////////////////////////////////////
// append checksum to 'rs232_buffer[]' //
/////////////////////////////////////////
rs232_buffer[block_length  ] = checksum._8bit.hibyte;
rs232_buffer[block_length+1] = checksum._8bit.lobyte;

//////////////
// preamble //
//////////////
Send_RS232_Byte( 'A' );
Send_RS232_Byte( 'T' );
Send_RS232_Byte( 'E' );
Send_RS232_Byte( 'E' );
Send_RS232_Byte( 'C' );
Send_RS232_Byte( 'C' );

////////////////////////////////////////////////////////////////////
// the complete block consista of:-                               //
// block length + block title + n*data + checksum hi + checksum lo  //
// The number of bytes that we have to transmit is block_length + 2 //
////////////////////////////////////////////////////////////////////
block_length += 2;

//////////////////////////
// send packet out on TX //
//////////////////////////
for ( ii = 0; ii < block_length; ii++ )
   {
   Send_RS232_Byte( rs232_buffer[ii] );
   }
}  // Send_RS232_CommsPacket()
//-------------------------------------------------------------------------
```

```
#endif


// IR comms...

////////////////////////////////////////////////////////////////////////////
// This function transmits it's function argument out on the TX pin       //
// Argument : data byte to send                                           //
// Returns  : none                                                        //
////////////////////////////////////////////////////////////////////////////
void Send_IR_Byte( unsigned char data )
{
unsigned char  ii;


///////////////
// data byte //
///////////////
for ( ii = 0; ii < 8; ii++ )
   {
   data >>= 1;

   if ( carry() ) Send_1();
   else           Send_0();
   }
}  // Send_IR_Byte()
//--------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// This functions transmits an ir data packet. The data is organised as:  //
//                                                                        //
// ir_buffer[0]   == block length byte                                    //
// ir_buffer[1]   == block title byte                                     //
// ir_buffer[2]   == data byte 1                                          //
// ir_buffer[n]   == data byte 'n'                                        //
// ir_buffer[n+1] == hibyte checksum                                      //
// ir_buffer[n+2] == lobyte checksum                                      //
//                                                                        //
// Block length is the number of bytes in                                 //
// the block, EXCLUDING the checksum.                                     //
//                                                                        //
// Checksum is the 16 bit total of the                                    //
// block, EXCLUDING the checksum.                                         //
//                                                                        //
// The checksum is calculated prior to the and the checksum bytes are     //
// appended to the transmission packet.                                   //
//                                                                        //
// arguments : block title and block length for transmission buffer       //
// returns   : none                                                       //
////////////////////////////////////////////////////////////////////////////
void Send_IR_CommsPacket( unsigned char block_title, unsigned char block_length)
{
union uUNSIGNED_INTEGER checksum;
unsigned char           ii;
```

```
                    ////////////////////////////////////////////////
TSC0.bit.CH0IE = 0;  // disable timer0 capture interrupt else we will //
                    // detect the comms we're about to transmit!     //
                    ////////////////////////////////////////////////

//////////////////////////////
// re-affirm data direction //
//////////////////////////////
IR_TX_DDR = 1;

//////////////////////////////////////////////////////
// insert the element values into the 'ir_buffer' array //
//////////////////////////////////////////////////////
block_length           += 2;              // block_length+block_title byes == 2
ir_buffer[BLOCK_LENGTH] = block_length;
ir_buffer[BLOCK_TITLE ] = block_title;

///////////////////////////////////
// calculate the packet checksum //
///////////////////////////////////
checksum._16bit = 0;
for ( ii = 0; ii < block_length; ii++ )
   {
   checksum._16bit += ir_buffer[ii];
   }

///////////////////////////////////
// append checksum to 'ir_buffer' //
///////////////////////////////////
ir_buffer[block_length  ] = checksum._8bit.hibyte;
ir_buffer[block_length+1] = checksum._8bit.lobyte;

//////////////////////////////////////////////////////////////////////
// the complete block consista of:-                                 //
// block length + block title + n*data + checksum hi + checksum lo  //
// The number of bytes that we have to transmit is block_length + 2 //
//////////////////////////////////////////////////////////////////////
block_length += 2;

//////////////////////////////
// 4ms synchronising pulse //
//////////////////////////////
StartPulse();

//////////////////////////////
// send packet out on IR TX //
//////////////////////////////
for ( ii = 0; ii < block_length; ii++ )
   {
   Send_IR_Byte( ir_buffer[ii] );
   }
```

```
StopPulse();

if ( TSC0.bit.CH0F )
   {
   TSC0.bit.CH0F = 0;   // clear interrupt flag if set whilst interrupt disabled
   }

TSC0.bit.CH0IE = 1;     // IR detect timer0 capture interrupt back on
}  // Send_IR_CommsPacket()
//---------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// Logic 0 as transmitted by the IR TX pin:                                 //
//       ------------                                                       //
//      |////////////|            |                                         //
//      |// 38kHz ///|            |                                         //
//      |////////////|            |                                         //
//      |____________|____________|                                         //
//                                                                          //
//      <-- 700us --><-- 700us -->                                          //
//                                                                          //
// Logic level as seen by receiving pin:                                    //
//                     ------------                                         //
//      |             |            |                                        //
//      |             |            |                                        //
//      |             |            |                                        //
//      |_____________|            |                                        //
//                                                                          //
//      <-- 700us --><---700us -->                                          //
//                                                                          //
// The micro measures the width of the +ve pulse to determine the bit value.  //
////////////////////////////////////////////////////////////////////////////
void Send_0( void )
{
_38KHzBurstOnTime(_700US);
_38KHzBurstOffTime(_700US);
}  // Send_0()
//---------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// Logic 1 as transmitted by the IR TX pin:                               //
//                                                                        //
//       ------------                                                     //
//      |////////////|                                      |             //
//      |// 38kHz ///|                                      |             //
//      |////////////|                                      |             //
//      |____________|______________________________________|             //
//                                                                        //
//      <-- 700us --><----------- 2100us -------------------->            //
//                                                                        //
// Logic level as seen by receiving pin:                                  //
//                     ---------------------------------------            //
//      |            |                                      |             //
//      |            |                                      |             //
//      |            |                                      |             //
//      |____________|                                      |             //
//                                                                        //
//      <-- 700us --><----------- 2100us -------------------->            //
//                                                                        //
// The micro measures the width of the +ve pulse to determine the bit value.  //
////////////////////////////////////////////////////////////////////////////
void Send_1( void )
{
_38KHzBurstOnTime(_700US);  // 27*26us approx 700us
_38KHzBurstOffTime(_2100US); // 81(3*27)*26us approx 2.1ms
}  // Send_1()
//--------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// The leader pulse as transmitted by the IR TX pin:                      //
//                                                                        //
//       ------------                                                     //
//      |////////////|            |                                        //
//      |// 38kHz ///|            |                                        //
//      |////////////|            |                                        //
//      |____________|____________|                                        //
//                                                                        //
//      <--- 4ms ---><--- 4ms --->                                        //
//                                                                        //
// Above leader pulse as seen by receiving pin:                           //
//                     ------------                                       //
//      |            |            |                                        //
//      |            |            |                                        //
//      |            |            |                                        //
//      |____________|            |                                        //
//                                                                        //
//      <--- 4ms ---><--- 4ms --->                                        //
//                                                                        //
// The receiving micro measures the width of the +ve pulse to determine the   //
// bit value.                                                             //
////////////////////////////////////////////////////////////////////////////
void StartPulse( void )
{
```

```
_38KHzBurstOnTime(_4000US);
_38KHzBurstOffTime(_4000US);
}
//---------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
void StopPulse( void )
{
_38KHzBurstOnTime(_700US);     // 27*26us approx 700us
}
//---------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// This function produces count*26us pulses with 50% mark space ratio ie      //
// 13us high and 13us low.                                                    //
//                                                                            //
// At 1MHz, 13us == 13 bus cycles                                             //
// We use 'nop' to give us the timing we require.                             //
//                                                                            //
// The number of nops is less for the low time as we include the do/while     //
// cycle count in it's timing.                                                //
//                                                                            //
// The total function cycle count is count*26 + 13 (for stack/wdg and return) //
// Note: above cycle count excludes the 'call' cycles.                        //
////////////////////////////////////////////////////////////////////////////
void _38KHzBurstOnTime( unsigned char count )
{
ServiceWatchDog();

do {
   //////////////
   // start hi //
   //////////////
   IR_TX = 1;
   Nop();Nop();Nop();Nop();Nop();Nop();Nop();Nop();Nop();

   //////////////
   // now low //
   //////////////
   IR_TX = 0;
   Nop();Nop();Nop();
   }  while ( --count );
}  // _38KHzBurstOnTime()
//---------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// This function produces a count*26us timing delay                       //
//                                                                        //
// At 1MHz bus, 26us == 26 bus cycles                                     //
// We use 'nop' to give us the timing we require.                         //
//                                                                        //
// The total function cycle count is count*26 + 13 (for stack/wdg and return) //
// Note: above cycle count excludes the 'call' cycles.                    //
////////////////////////////////////////////////////////////////////////////
void _38KHzBurstOffTime( unsigned char count )
{
ServiceWatchDog();

do {
   Nop();Nop();Nop();Nop();Nop();Nop();Nop();Nop();Nop();Nop();
   Nop();Nop();Nop();Nop();Nop();Nop();Nop();Nop();Nop();Nop();
   }  while ( --count );
}  // _38KHzBurstOffTime()
//---------------------------------------------------------------------------
```

```
[PIR:serial.h]
////////////////////////////////////////////////////////////////////////////
//       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 Infra Red Reference Design                      //
//  Filename  :  serial.h                                                 //
//  Author    :  jtravers                                                 //
//  Compiler  :  Cosmic ANSI-C                                            //
//  CPU       :  68HC908JK1/3                                             //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for serial.c                                              //
//////////////////////////  Update Information  ////////////////////////////
//  Ed.  Date     Init's  Modification                                    //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/05/00 jt      creation                                        //
////////////////////////////////////////////////////////////////////////////
```

```
#ifndef     __SERIAL_H_
#define     __SERIAL_H_

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

#define  IR_TX          PTD.bit.bit7
#define  IR_TX_DDR      DDRD.bit.bit7

enum
   {
   BLOCK_LENGTH,
   BLOCK_TITLE,
   DATA_BYTE1,
   DATA_BYTE2,
   DATA_BYTE3,
   DATA_BYTE4,
   DATA_BYTE5
   };

enum  // block title values
   {
   SEND_A2D_TRIGGER = 0x01,
   SEND_A2D_DIFFERENCE,
   SEND_A2D_LOOPTIME,
   SEND_DELTA_SIG_RESOLUTION,
   SEND_DELTA_SIG_EVENT,
   SEND_PASSWORD,
   UPDATE_A2D_TRIGGER,
   UPDATE_A2D_DIFFERENCE,
   UPDATE_A2D_LOOPTIME,
   UPDATE_DELTA_SIG_RESOLUTION,
   UPDATE_DELTA_SIG_EVENT
   };

//////////////////////
// block title values //
//////////////////////
enum
   {
   PIR_DATA = 0xf0,
   SIGMA_DATA,
   DETECT_EVENT
   };

#define  _700US              27    // 27*26us   == 702us
#define  _2100US             81    // 27*3*26us == 2106us
#define  _4000US             155   // 155*26us  == 4030us
```

```
////////////////
// prototypes //
////////////////
#ifdef   __PC_DEBUG_
#define  RS232_TX             PTD.bit.bit3
#define  RS232_TX_DDR         DDRD.bit.bit3

void Send_RS232_Byte( unsigned char );
void DelayBitTime( void  );
void DelayHiBitTime( void  );
void Send_RS232_CommsPacket( unsigned char, unsigned char );
#endif

void Send_IR_CommsPacket( unsigned char, unsigned char );
void Send_0( void );
void Send_1( void );
void StartPulse( void );
void StopPulse( void );
void _38KHzBurstOnTime( unsigned char );
void _38KHzBurstOffTime( unsigned char );
#endif
```

```
[PIR: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 Infra Red Reference Design                          //
//  Filename  :  startup.c                                                    //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1/3                                                 //
////////////////////  File Contents  ///////////////////////////////////////////
//  startup routines                                                          //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  17/05/00 jt      creation                                            //
```

```
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "deltasig.h"
#include    "startup.h"


////////////////////////////////////////////////////////////////////////////
// Micro setup and initialisation. This is the first non compiler code     //
// executed after a reset.                                                 //
////////////////////////////////////////////////////////////////////////////
void MicroStartUp( void )
{
CONFIG1.reg = 0x00;  // COP cycle == (2^18-2^4)*1/Fosc, LVI enabled,
                     // STOP mode off, watchdog enabled
CONFIG2.reg = 0x10;  // IRQ pull-up enabled, LVI enabled @4V with 5V Vdd.

ServiceWatchDog();

SEI();

INTSCR.reg  = 0x02;  // IRQ interrupts disabled

////////////////////////////////////
// data direction register setup //
////////////////////////////////////
                  ////////////////////////////////////////////////////////////
PTB.reg  = 0x00;  //                                                        //
DDRB.reg = 0x00;  // bit7:DS fast charge o/p, bit6:DS charge/discharge o/p  //
                  // bit5:DS analogue i/p   , bit4:LM324 analogue i/p       //
                  // bit3/2/1/0:for monitor mode programming                //
                  // NOTE: even though some pins are output I have made them //
                  // an input on startup, this is to stop any charge/discharge//
                  // of the Delta Sigma capacitor.                          //
PTD.reg  = 0x00;  // IR led off                                             //
DDRD.reg = 0x8f;  // bit7:IR tx, bit6:mode select i/p, bit5:cap gnd drive   //
                  // bit4:IR rx, bit3:rs232 tx       , bit2:led             //
                  ////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
// detect mode sense and charge cap if delta-sigma selected //
////////////////////////////////////////////////////////////
if ( !MODE_SELECT_PIN )
   {
   FastCharge();
   }

////////////////////////////////////////////////////////
// We clear all ram areas upto $F8. Leave the area $F8-$FF //
// as there has been two calls already.                   //
////////////////////////////////////////////////////////
ClearRam();

////////////////////////////
// variable initialisation //
////////////////////////////
pir_buffer_ptr = &pir_buffer[0];              // assign buffer pointer
```

```
/////////////////
// timer setup //
/////////////////
TSC.reg  = 0x70;           // set TOIE/TSTOP/TRST,1X prescaler ie 1us @4MHz Xtal
TMOD     = TIMER_ROLLOVER; // == 10000us == 10ms until overflow

//////////////////////////////////////////
// start timer0 capture mode for +ve edges //
//////////////////////////////////////////
TSC0.reg = 0b01000100;
//           ||||||||
//           ||||||||____  CH0MAX   100% pwm off
//           |||||||_____  TOV0     PTD4 not toggled on overflow
//           ||||||______  ELS0A    } +ve edge trigger capture
//           |||||_______  ELS0B    } -ve edge trigger capture
//           ||||________  MS0A     unbuffered compare/pwm operation on
//           |||_________  MS0B     buffered compare/pwm off
//           ||__________  CHOIE    interrupt enabled
//           |___________  CH0F     read only

TSC1.reg      = 0x00;   // timer1 off
TSC.bit.TSTOP = 0;      // start timer

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

////////////////////////////////////////////////////////////////////////////
// Defensive measure for increased robustness                               //
////////////////////////////////////////////////////////////////////////////
void ReAffirmDDR( void )
{
DDRD.reg = 0x8f;

if ( !MODE_SELECT_PIN ) DDRB.reg = 0x00;  // Delta Sigma
else                    DDRB.reg = 0xe0;  // 8 bit analogue
}  // ReAffirmDDR()
//-----------------------------------------------------------------------------
void ReAffirmDDR( void )
{
DDRD.reg = 0xaf;

if ( !MODE_SELECT_PIN ) DDRB.reg = 0x00;  // Delta Sigma
else                    DDRB.reg = 0xe0;  // 8 bit analogue
}  // ReAffirmDDR()
//-----------------------------------------------------------------------------
```

```
[PIR:startup.h]
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   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     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 Infra Red Reference Design                        //
//  Filename  :  startup.h                                                  //
//  Author    :  jtravers                                                   //
//  Compiler  :  Cosmic ANSI-C                                              //
//  CPU       :  68HC908JK1/3                                               //
////////////////////  File Contents  ////////////////////////////////////////
//  header file for startup.c                                               //
//////////////////////////  Update Information  //////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  17/05/00 jt      creation                                          //
////////////////////////////////////////////////////////////////////////////
#ifndef  __STARTUP_H_
#define  __STARTUP_H_

////////////////
// prototypes //
////////////////
void MicroStartUp( void );
void ReAffirmDDR( void );
#endif
```

```
[PIR:vectors.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 Infra Red Reference Design                          //
//  Filename  :  vectors.c                                                    //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  MC68HC908JK1/3                                               //
////////////////////  File Contents  ///////////////////////////////////////
//  vectors - an array of void pointers                                       //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  28/03/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "define.h"              // 'NULL' defined

extern void TIMEROVERFLOW( void );
extern void TIMERCHANNEL0( void );
extern void _stext();   // startup routine. defined by Cosmic in 'crts.s'
```

```
////////////////////////////////////////////////////////////////////////////
// An array of function pointers ie the addresses of the interrupt routines //
////////////////////////////////////////////////////////////////////////////
void (*const _vectab[17])() =
   {                      ///////////////////////////////////
   _stext,                // A2D CONVERSION COMPLETE $FFDE //          |
   _stext,                // KEYBOARD                $FFE0 //          |
   NULL,                  // NOT USED                $FFE2 //          |
   NULL,                  // NOT USED                $FFE4 //          |
   NULL,                  // NOT USED                $FFE6 //          |
   NULL,                  // NOT USED                $FFE8 //          |
   NULL,                  // NOT USED                $FFEA //          |
   NULL,                  // NOT USED                $FFEC //          |
   NULL,                  // NOT USED                $FFEE //          |
   NULL,                  // NOT USED                $FFF0 //          |
   TIMEROVERFLOW,         // TIMER OVERFLOW          $FFF2 //          |
   _stext,                // TIMERCHANNEL1,          $FFF4 //          |
   TIMERCHANNEL0,         // TIMERCHANNEL0,          $FFF6 //          |
   NULL,                  // NOT USED                $FFF8 //          |
   _stext,                // IRQ                     $FFFA //          |
   _stext,                // SWI                     $FFFC //          V
   _stext                 // RESET                   $FFFE //      Increasing
   };                     ///////////////////////////////////       Priority
//---------------------------------------------------------------------------
```

Appendix F. REMOTE Source Code Files

Throughout this document, references are made to source code files contained in this appendix. They are:

For those viewing this document in .pdf format, these files can be accessed by clicking on the appropriate hyperlink reference found in the textual portions of the document.

```
[REMOTE:button.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  button.c                                                     //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  Remote control button scan routine                                        //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  19/06/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "datasort.h"
#include    "ir_comms.h"
#include    "delay.h"
#include    "lcd.h"
#include    "convert.h"
#include    "mode.h"
#include    "rs_comms.h"
#include    "digipot.h"
#include    "rtc.h"
#include    "button.h"
```

```
void DefaultButtons( void )
{
if ( button_pattern._16bit == DEFAULT_BUTTONS )
   {
   // re-affirmation
   pressed_pattern              = DEFAULT_BUTTONS;
   button_debounce_counter      = 0;
   button_flags.bit.FIRST_PASS  = 0;
   button_flags.bit.AUTO_SCROLL = 0;
   }
else
   {
   ////////////////////////////////////////////////////////
   // OK, a press detected, this is the first recognition of //
   ////////////////////////////////////////////////////////
   button_press_status = BUTTON_PRESSED;
   pressed_pattern     = button_pattern._16bit;
   }
}  // DefaultButtons()
//------------------------------------------------------------------------------


void PressedButtons( void )
{
////////////////////////////////////////////
// No longer pressed, back to pull-up values //
////////////////////////////////////////////
if ( button_pattern._16bit == 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._16bit == pressed_pattern )
      {
      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_SCROLL = 0;   // no buttons have autoscroll
                                                // presently
            DecodeButtons();                    // respond to press
            }
         else                                   // auto repeat can now occur
            {
            ////////////////////////////////////////////////////////////
            // same button as for first debounce is still being pressed, //
            // after (35-DEBOUNCE_COUNTER)*10ms allow auto repeat of IR  //
            // transmission                                              //
            ////////////////////////////////////////////////////////////
```

```
            if ( button_debounce_counter >= 35 )// (35-DEBOUNCE_COUNTER)*10ms
               {                                //  before auto repeat mode
               button_debounce_counter = 15;    // (35-15)*10ms is the effective
                                                // repeat speed == 200ms,
                                                // approx 5 times per second
               if (button_flags.bit.AUTO_SCROLL)// do you require auto scroll...
                  {
                  DecodeButtons();              // ...if so keep decoding
                  }
               }
            }
         }
      }
   else
      {
      ////////////////////////////////////////////////////////////////
      // pattern is different but something is pressed, start again... //
      // If you required dual button presses ie if one button was held //
      // and another was repeatedly pressed/released you would decode  //
      // that situation here. I have not needed this functionality but //
      // this 'else' statement would be the area to code it.           //
      ////////////////////////////////////////////////////////////////
      button_press_status          = NO_BUTTON_PRESS;
      pressed_pattern              = DEFAULT_BUTTONS;
      button_debounce_counter      = 0;
      button_flags.bit.FIRST_PASS  = 0;
      button_flags.bit.AUTO_SCROLL = 0;
      }
   }  // end of 'else'
}     // PressedButtons()
//---------------------------------------------------------------------------


void ReleasedButtons( void )
{
////////////////////////////////////////////////////
// Ok, we think all the buttons are now at their default //
////////////////////////////////////////////////////
if ( --button_release_counter == 0 )
   {
   button_press_status = NO_BUTTON_PRESS;
   }
else
   {
   // checking for noise...
   if ( button_pattern._16bit != DEFAULT_BUTTONS )
      {
      button_press_status = BUTTON_PRESSED;  // continue as pressed...
      }
   }
}  // ReleasedButtons()
//---------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// This function does the physical button decoding.                       //
//                                                                        //
// Arguments: none                                                        //
// returns  : none                                                        //
////////////////////////////////////////////////////////////////////////////
void DecodeButtons( void )
{
switch ( mode )
   {
   case MODE_USER_ENTER_PASSWORD :
   PasswordButtons();
   break;

   case MODE_EDIT_A2D_TRIGGER          :
   case MODE_EDIT_A2D_DIFFERENCE       :
   case MODE_EDIT_A2D_LOOPTIME         :
   case MODE_EDIT_DELTA_SIG_RESOLUTION :
   case MODE_EDIT_DELTA_SIG_EVENT      :
   case MODE_TIME_OF_DAY               :
   case MODE_LCD_CONTRAST_ADJUST       :
   StandardButtons();
   break;

   case MODE_SHUTTING_DOWN :
   ////////////////////////////////////////////////////////////
   // user press whilst the closing down dots '.' are beng printed //
   ////////////////////////////////////////////////////////////
   mode = mode_copy;  // restore mode before prior shut down execution

   if ( mode == MODE_USER_ENTER_PASSWORD )
      {
      PasswordEntryScreen();
      }
   if ( mode >= MODE_EDIT_A2D_TRIGGER && mode <= MODE_EDIT_DELTA_SIG_EVENT )
      {
      RedrawFlashParameterScreen();
      }

   shut_down_ii = 0;          // job done, initiliase for next
   shut_down_jj = 0;          // job done, initiliase for next
   break;
   }

stop_counter = 0; // reset the stop mode entry timeout counter
}  // DecodeButtons()
//---------------------------------------------------------------------------


void ReadButtons( void )
{
button_pattern._16bit = DEFAULT_BUTTONS;  // assume no button(s) pressed

DDRA.reg &= 0x1f;    // defensive measure, ensure column drivers are input
```

```
//////////////////////
// activate COLUMN1 //
//////////////////////
PTA.bit.bit5  = 0;   // lo...
DDRA.bit.bit5 = 1;   //      ...and an output

////////////////////////
// read B5/B4/B3/B2/B1 //
////////////////////////
Delay(_50US);  // wait...before read for pin level to settle

button_pattern._8bit.lobyte.bit.bit0 = PTA.bit.bit0;
button_pattern._8bit.lobyte.bit.bit1 = PTA.bit.bit1;
button_pattern._8bit.lobyte.bit.bit2 = PTA.bit.bit2;
button_pattern._8bit.lobyte.bit.bit3 = PTA.bit.bit3;
button_pattern._8bit.lobyte.bit.bit4 = PTA.bit.bit4;

//////////////////////
// Column1 inactive //
//////////////////////
DDRA.bit.bit5 = 0;   // ...now an input

//////////////////////
// activate COLUMN2 //
//////////////////////
PTA.bit.bit6  = 0;   // lo...
DDRA.bit.bit6 = 1;   //     ... and an output

////////////////////////
// read B10/B9/B8/B7/B6 //
////////////////////////
Delay(_50US);  // wait...before read for pin level to settle

button_pattern._8bit.lobyte.bit.bit5 = PTA.bit.bit0;
button_pattern._8bit.lobyte.bit.bit6 = PTA.bit.bit1;
button_pattern._8bit.lobyte.bit.bit7 = PTA.bit.bit2;
button_pattern._8bit.hibyte.bit.bit0 = PTA.bit.bit3;
button_pattern._8bit.hibyte.bit.bit1 = PTA.bit.bit4;

//////////////////////
// Column2 inactive //
//////////////////////
DDRA.bit.bit6 = 0;   // ...now an input

//////////////////////
// activate COLUMN3 //
//////////////////////
PTA.bit.bit7  = 0;   // lo...
DDRA.bit.bit7 = 1;   //     ... and an output
```

```
////////////////////////////
// read B15/B14/B13/B12/B11 //
////////////////////////////
Delay(_50US);  // wait...before read for pin level to settle

button_pattern._8bit.hibyte.bit.bit2 = PTA.bit.bit0;
button_pattern._8bit.hibyte.bit.bit3 = PTA.bit.bit1;
button_pattern._8bit.hibyte.bit.bit4 = PTA.bit.bit2;
button_pattern._8bit.hibyte.bit.bit5 = PTA.bit.bit3;
button_pattern._8bit.hibyte.bit.bit6 = PTA.bit.bit4;

//////////////////////
// Column3 inactive //
//////////////////////
DDRA.bit.bit7 = 0;   // ...now an input

//////////////////////
// defensive measure //
//////////////////////
DDRA.reg = 0x00;

////////////////////
// detect a press //
////////////////////
switch ( button_press_status )
   {
   case NO_BUTTON_PRESS:
   DefaultButtons();
   break;

   case BUTTON_PRESSED:
   PressedButtons();
   break;

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


////////////////////////////////////////////////////////////////////////////
// The <ENTER> button has been pressed, decide functionality wrt current mode //
////////////////////////////////////////////////////////////////////////////
void Enter( void )
{
unsigned char     temp;


switch ( mode )
   {
   case MODE_EDIT_A2D_TRIGGER :
   ir_buffer[DATA_BYTE1] = adjust_value._8bit.lobyte;
   Send_IR_CommsPacket( UPDATE_A2D_TRIGGER, 1 );
```

```
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_EDIT_A2D_DIFFERENCE :
   ir_buffer[DATA_BYTE1] = adjust_value._8bit.lobyte;
   Send_IR_CommsPacket( UPDATE_A2D_DIFFERENCE, 1 );
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_EDIT_A2D_LOOPTIME :
   ir_buffer[DATA_BYTE1] = adjust_value._8bit.lobyte;
   Send_IR_CommsPacket( UPDATE_A2D_LOOPTIME, 1 );
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_EDIT_DELTA_SIG_RESOLUTION :
   temp = adjust_value._8bit.lobyte;
   ConvertBitToResolution(temp);
   ir_buffer[DATA_BYTE1] = adjust_value._8bit.hibyte;
   ir_buffer[DATA_BYTE2] = adjust_value._8bit.lobyte;
   Send_IR_CommsPacket( UPDATE_DELTA_SIG_RESOLUTION, 2 );
   adjust_value._8bit.lobyte = temp;  // restore for next inc/dec if required
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_EDIT_DELTA_SIG_EVENT :
   ir_buffer[DATA_BYTE1] = adjust_value._8bit.hibyte;
   ir_buffer[DATA_BYTE2] = adjust_value._8bit.lobyte;
   Send_IR_CommsPacket( UPDATE_DELTA_SIG_EVENT, 2 );
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_LCD_CONTRAST_ADJUST :
   WriteText2(LINE1_2, "", PRECLEAR);  // clear whole screen prior to TOD
   mode = MODE_TIME_OF_DAY;
   break;
   }
}  // Enter()
//-------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////
// The <ESC> button has been pressed, decide functionality wrt current mode //
////////////////////////////////////////////////////////////////////////
void Esc( void )
{
if ( mode != MODE_TIME_OF_DAY )        // avoid replacing same screen
   {
   WriteText2(LINE1_2, "", PRECLEAR);  // clear whole screen prior to TOD
   }

mode = MODE_TIME_OF_DAY;
}  // Esc()
//-------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////
// The <INC> button has been pressed, decide functionality wrt current mode //
////////////////////////////////////////////////////////////////////////
void Increment( void )
{
switch ( mode )
   {
   case MODE_EDIT_A2D_TRIGGER :
   ////////////////////////////////////////////////////////////////////
   // if performing a nearest neighbour magnitude test on a buffer of size //
   // 'n' elements, there are 'n-1' comparisons to be performed.          //
   ////////////////////////////////////////////////////////////////////
   if ( adjust_value._8bit.lobyte < (pir_buffer_size-1) )
      {
      WriteText2(LINE2, "", PRECLEAR);
      IntegerToASCII( ++adjust_value._8bit.lobyte, &text_buffer[7]  );
      WriteText1(LINE2);
      }
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_EDIT_A2D_DIFFERENCE :
   case MODE_EDIT_A2D_LOOPTIME   :
   if ( adjust_value._8bit.lobyte < 255 )
      {
      WriteText2(LINE2, "", PRECLEAR);
      IntegerToASCII( ++adjust_value._8bit.lobyte, &text_buffer[7]  );
      WriteText1(LINE2);
      }
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_EDIT_DELTA_SIG_RESOLUTION :
   if ( adjust_value._8bit.lobyte < 15 )
      {
      WriteText2( LINE2, "", PRECLEAR);
      IntegerToASCII( ++adjust_value._8bit.lobyte, &text_buffer[7]  );
      WriteText1(LINE2);
      }
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_EDIT_DELTA_SIG_EVENT :
   WriteText2( LINE2, "", PRECLEAR);
   Assign_DS_Pointer(); // load min/max/step clamps

   if ( adjust_value._16bit >= ds_adjust_ptr->max )
      {
      adjust_value._16bit = ds_adjust_ptr->max;
      }
   else
      {
      adjust_value._16bit += ds_adjust_ptr->step;
      }
```

```
   IntegerToASCII( adjust_value._16bit, &text_buffer[7]  );
   WriteText1(LINE2);
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_LCD_CONTRAST_ADJUST :
   DigiPot(DP_INCREMENT);
   button_flags.bit.AUTO_SCROLL = 1;
   break;
   }
}  // Increment()
//-----------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// The <DEC> button has been pressed, decide functionality wrt current mode //
////////////////////////////////////////////////////////////////////////////
void Decrement( void )
{
switch ( mode )
   {
   case MODE_EDIT_A2D_TRIGGER :
   case MODE_EDIT_A2D_DIFFERENCE :
   case MODE_EDIT_A2D_LOOPTIME :
   if ( adjust_value._8bit.lobyte > 1 )
      {
      WriteText2( LINE2, "", PRECLEAR);
      IntegerToASCII( --adjust_value._8bit.lobyte, &text_buffer[7]  );
      WriteText1(LINE2);
      }
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_EDIT_DELTA_SIG_RESOLUTION :
   if ( adjust_value._8bit.lobyte > 10 )  // min resolution is 10 bit
      {
      WriteText2( LINE2, "", PRECLEAR);
      IntegerToASCII( --adjust_value._8bit.lobyte, &text_buffer[7]  );
      WriteText1(LINE2);
      }
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_EDIT_DELTA_SIG_EVENT :
   WriteText2( LINE2, "", PRECLEAR);
   Assign_DS_Pointer(); // load min/max/step clamps

   if ( adjust_value._16bit > ds_adjust_ptr->max )
      {
      adjust_value._16bit = ds_adjust_ptr->max;
      }
   else if ( adjust_value._16bit > ds_adjust_ptr->min )
      {
      adjust_value._16bit -= ds_adjust_ptr->step;
```

```
      }

   IntegerToASCII( adjust_value._16bit, &text_buffer[7] );
   WriteText1(LINE2);
   button_flags.bit.AUTO_SCROLL = 1;
   break;

   case MODE_LCD_CONTRAST_ADJUST :
   DigiPot(DP_DECREMENT);
   button_flags.bit.AUTO_SCROLL = 1;
   break;
   }
}  // Decrement()
//-------------------------------------------------------------------------------


/////////////////////////////////////////////////////
// A button has been decoded as being pressed, perform //
// any approriate functionality                        //
/////////////////////////////////////////////////////
void StandardButtons( void )
{
switch ( pressed_pattern )
   {
   case BUTTON_1 :
   Send_IR_CommsPacket( SEND_DELTA_SIG_EVENT, 0 );
   break;

   case BUTTON_2 :
   Send_IR_CommsPacket( SEND_DELTA_SIG_RESOLUTION, 0 );
   break;

   case BUTTON_3 :
   Send_IR_CommsPacket( SEND_A2D_LOOPTIME, 0 );
   break;

   case BUTTON_4 :
   Send_IR_CommsPacket( SEND_A2D_DIFFERENCE, 0 );
   break;

   case BUTTON_5 :
   Send_IR_CommsPacket( SEND_A2D_TRIGGER, 0 );
   break;

   case BUTTON_6 :
   ForceRTC();
   break;

   case BUTTON_7 :
   // not used...user code here
   break;

   case BUTTON_8 :
   // not used...user code here
   break;
```

```
   case BUTTON_9 :
   // not used...user code here
   break;

   case BUTTON_10 :
   // not used...user code here
   break;

   case BUTTON_11 :
   Increment();
   break;

   case BUTTON_12 :
   Decrement();
   break;

   case BUTTON_13 :
   LCD_ContrastAdjust();
   break;

   case BUTTON_14 :
   Esc();
   break;

   case BUTTON_15 :
   Enter();
   break;
   }
}  // StandardButtons()
//-------------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////
// A button has been decoded as being pressed. during password entry the //
// buttons apply numeric text in the lcd.                                //
////////////////////////////////////////////////////////////////////////
void PasswordButtons( void )
{
char  temp = 0;

switch ( pressed_pattern )
   {
   case BUTTON_1 :
   temp = '1';
   break;

   case BUTTON_2 :
   temp = '2';
   break;

   case BUTTON_3 :
   temp = '3';
   break;
```

```
   case BUTTON_4 :
   temp = '4';
   break;

   case BUTTON_5 :
   temp = '5';
   break;

   case BUTTON_6 :
   temp = '6';
   break;

   case BUTTON_7 :
   temp = '7';
   break;

   case BUTTON_8 :
   temp = '8';
   break;

   case BUTTON_9 :
   temp = '9';
   break;

   case BUTTON_10 :
   temp = '0';
   break;

   case BUTTON_15 :
   PasswordEnter();
   break;
   }

if ( temp )
   {
   text_buffer[character_count++] = temp;
   text_buffer[character_count  ] = '\0';
   WriteText1(LINE2+5);

   ////////////////////////////////
   // test for number entry wrap //
   ////////////////////////////////
   if ( character_count >= 5 )
      {
      character_count          = 0;
      flags1.bit.PASSWORD_WRAP = 1; // all the 'X' have been overwritten
      }

   ////////////////////////////////////////////////
   // make the 'blinking' cursor follow the character //
   // after each character is entered                 //
   ////////////////////////////////////////////////
   SetCursorAddress((unsigned char)(LINE2+5+character_count));

   button_flags.bit.AUTO_SCROLL = 1;
```

```
   }
}  // PasswordButtons()
//---------------------------------------------------------------------------


void PasswordEnter( void )
{
////////////////////////////////////////////////////////////////////////////
// before the password number is processed it has to contain 5 numerals, ie   //
// the user has to have overwritten the initial 'XXXXX'                       //
////////////////////////////////////////////////////////////////////////////
 if ( flags1.bit.PASSWORD_WRAP ) // correct number of digits entered?
   {
   // the -'0' converts from ASCII to decimal before the decimal place multiply
   user_password  = 10000*(text_buffer[0]-'0');
   user_password += 1000* (text_buffer[1]-'0');
   user_password += 100*  (text_buffer[2]-'0');
   user_password += 10*   (text_buffer[3]-'0');
   user_password +=       (text_buffer[4]-'0');

   InitialiseLCD(NOBLINK|NOUNDERLINE_CURSOR);   // turn 'blinking' cursor off
   WriteText2( LINE1, "Password" , NOPRECLEAR );

   if ( user_password == pir_password._16bit )
      {
      WriteText2( LINE2, "Accepted!", NOPRECLEAR );
      Delay10ms(_1S);         // show message for 1s
      mode = MODE_TIME_OF_DAY;
      WriteText2(LINE1_2, "", PRECLEAR);  // clear whole screen prior to TOD
      InitialiseRS232();      // allow RTC update
      }
   else
      {
      WriteText2( LINE2, "Rejected!", NOPRECLEAR );
      Delay10ms(_1S);         // show message for 1s
      PasswordEntryScreen();  // try again
      }
   }
}  // PasswordEnter()
//---------------------------------------------------------------------------


void LCD_ContrastAdjust( void )
{
if ( mode != MODE_LCD_CONTRAST_ADJUST )
   {
   WriteText2(LINE1_2, "", PRECLEAR);
   WriteText2(LINE1, "Screen Contrast", NOPRECLEAR);
   WriteText2(LINE2, "Use INC/DEC"    , NOPRECLEAR);
   }

mode = MODE_LCD_CONTRAST_ADJUST;
}  // LCD_ContrastAdjust()
//---------------------------------------------------------------------------
```

```
[REMOTE:button.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  button.h                                                     //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for 'button.c'                                                //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  19/06/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef  __BUTTON_H_
#define  __BUTTON_H_

////////////////////
// button decodes //
////////////////////
#define  DEFAULT_BUTTONS      0xffff
#define  BUTTON_1             0xffef
#define  BUTTON_2             0xfff7
#define  BUTTON_3             0xfffb
#define  BUTTON_4             0xfffd
#define  BUTTON_5             0xfffe
#define  BUTTON_6             0xfdff
#define  BUTTON_7             0xfeff
#define  BUTTON_8             0xff7f
#define  BUTTON_9             0xffbf
#define  BUTTON_10            0xffdf
#define  BUTTON_11            0xbfff
#define  BUTTON_12            0xdfff
#define  BUTTON_13            0xefff
#define  BUTTON_14            0xf7ff
#define  BUTTON_15            0xfbff

#define DEBOUNCE_COUNTER      3
```

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


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

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

////////////////
// prototypes //
////////////////
void ReadButtons( void );
void DefaultButtons( void );
void PressedButtons( void );
void ReleasedButtons( void );
void DecodeButtons( void );
void Enter( void );
void Esc( void );
void Increment( void );
void Decrement( void );
void StandardButtons( void );
void PasswordButtons( void );
void PasswordEnter( void );
void LCD_ContrastAdjust( void );

#endif
```

```
[REMOTE:cc.bat]
@echo off

rem verbose... c:\cosmic\cx08\cx6808 -v -f config.dat %1.c

c:\cosmic\cx08\cx6808 -f config.dat %1.c
```

```
[REMOTE:config.dat]
###############################################
#  CONFIGURATION FILE FOR 68HC08 COMPILER  #
#  Copyright (c) 1995 by COSMIC Software   #
###############################################

############
# COMPILER #
############
#-no                          # don't use optimiser
-e
-l
+debug
-i c:\cosmic\cx08\h6808       # include ...

##########
# PARSER #
##########
-pp                           # prototypes
-pl                           # output line number info for listing & debug
-pck                          # extra type checking
-pnw                          # don't widen args

#############
# GENERATOR #
#############
-gf                           # full source display
#-oc                          # leave optimised/removed instructions as comments
-gf                           # all lines in listing
#-gck                         # enable stack overflow checking
#-gv                          # verbosity
#-gst3                        # static model

#############
# ASSEMBLER #
#############
-al                           # assembler file listing
-at                           # list instruction cycles
#-av

#############
# OPTIMISER #
#############
#-ov                          # show efficiency stats

####################
# Macro Definitions #
####################
-m debug:x                    # debug: produce debug info
-m nsh:,nsh                   # nsh: static not shared
```

```
[REMOTE:convert.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  convert.c                                                    //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908GP32                                                  //
////////////////////  File Contents  ///////////////////////////////////////
//  common numeric conversion routines                                        //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  30/08/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "convert.h"



////////////////////////////////////////////////////////////////////////////
// converts hexadecimal 'value' arg to an equivalent 2 char string and inserts//
// the string at the second arg pointer address.                              //
//                                                                            //
// For example say the argument 'value' is 0x25                               //
//                                                                            //
//  (0x25>>4)    = 2                                                          //
//  (0x25&0x0f)  = 5                                                          //
//                                                                            //
//  *address_ptr++ = '2'                                                      //
//  *address_ptr   = '5'                                                      //
////////////////////////////////////////////////////////////////////////////
void HexToASCII( unsigned char value, char *address_ptr )
{
*address_ptr++ = (char)( (value>>4)   + '0' );   // upper nibble
*address_ptr   = (char)( (value&0x0f) + '0' );   // lower nibble
}  // HexToASCII()
//---------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// converts decimal 'value' arg to an ASCII string and inserts             //
// the string at the second arg pointer address.                           //
// This function takes approx 350us to execute @ 4.9152MHz bus.            //
// Not tested @ 2.4576MHz, will be approx 700us.                           //
////////////////////////////////////////////////////////////////////////////
void IntegerToASCII( unsigned short int value, char *address_ptr )
{
unsigned char        mod100;
unsigned short int   mod1000;
unsigned short int   mod10000;
char                 *ptr;
char                 temp[6];


////////////////////////////////////////////
// example, assume 'value' is 12345 (decimal) //
//                                        //
// mod10000 = 12345%10000            = 2345   //
// mod1000  = 2345%1000              = 345    //
// mod100   = 345%100                = 45     //
// temp[0]  = (12345/10000) + '0';   = '1'    //
// temp[1]  = (2345/1000)   + '0';   = '2'    //
// temp[2]  = (345/100)     + '0';   = '3'    //
// temp[3]  = (45/10)       + '0';   = '4'    //
// temp[4]  = (45%10)       + '0';   = '5'    //
// temp[5]  = '\0'          ALWAYS           //
////////////////////////////////////////////
mod10000 = value%10000;
mod1000  = mod10000%1000;
mod100   = (unsigned char)(mod1000%100);
temp[0]  = (char)( (value/10000)   + '0' );     // 10000's character
temp[1]  = (char)( (mod10000/1000) + '0' );     //  1000's character
temp[2]  = (char)( (mod1000/100)   + '0' );     //   100's character
temp[3]  = (char)( (mod100/10)     + '0' );     //    10's character
temp[4]  = (char)( (mod100%10)     + '0' );     //     1's character
temp[5]  = '\0';                                // NULL    character
ptr      = &temp[0];                            // pointer assignment
while ( *ptr == '0' )   ptr++;                  // skip leading '0' (zeros)
strcpy( address_ptr, ptr );                     // assign to calling pointer
}  // IntegerToASCII()
//--------------------------------------------------------------------------
```

```
void HexToDec( unsigned char *value )
{
////////////////////////////////////////////////////
// for example suppose we receive 0x25 as the argument: //
//                                                      //
// 0x25 >>= 4 becomes 2, *= 10 becomes 20               //
//                                                      //
// 0x25 &= 0x0f becomes 5                               //
//                                                      //
// and the argument is the sum of the two ie            //
// 20+5 = 25                                            //
//                                                      //
// [NOTE : fails if either nibble is 'a'...'f']         //
////////////////////////////////////////////////////
*value = (unsigned char)( ((*value>>4)*10) + (*value&0x0f) );
}  // HexToDec()
//---------------------------------------------------------------------------

void DecToHex( unsigned char *value )
{
////////////////////////////////////////////////////
// for example suppose we receive 25 as the argument: //
//                                                    //
// 25/10 is 2, 2<<4 = 0x20                            //
// 25%10 is 5                                         //
//                                                    //
// result :                                           //
// 0x20+5 = 0x25                                      //
////////////////////////////////////////////////////
*value = (unsigned char)( ((*value/10)<<4) + (*value%10) );
}  // DecToHex()
//---------------------------------------------------------------------------
```

```
[REMOTE:convert.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  convert.h                                                    //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for convert.c                                                 //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/09/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __CONVERT_H_
#define     __CONVERT_H_


////////////////
// prototypes //
////////////////
void HexToASCII( unsigned char, char * );
void IntegerToASCII( unsigned short int, char * );
void HexToDec( unsigned char * );
void DecToHex( unsigned char * );

#endif
```

```
[REMOTE:crtsi.s]
; C STARTUP FOR MC68HC08
; WITH AUTOMATIC DATA/CODE INITIALISATION
; Copyright (c) 2000 by COSMIC Software
;
  xref  _main, __sbss, __memory, __idesc__, __stack
  xdef  _exit, __stext
;
__stext:
  ldhx  #__stack  ; initialize stack pointer
  txs
  ldhx  #__idesc__  ; descriptor address
cbcl:
  lda 1,x   ; save start
  psha      ; address of
  lda 0,x   ; prom data
  psha
ibcl:
  lda 2,x   ; test flag byte
  beq zbss    ; no more segment
  bit #$60    ; code segment
  bne dseg    ; no, copy it
  ais #2    ; remove previous start address
  aix #5    ; next descriptor
  bra cbcl    ; and restart
dseg:
  pshx      ; save
  pshh      ; pointer
  lda 6,x   ; compute length
  sub 1,x   ; of segment
  psha      ; save count MSB
  lda 5,x   ; compute LSB
  sbc 0,x
  tst 1,sp    ; if LSB nul,
  beq ok    ; keep it
  inca      ; else increment MSB
ok:
  psha      ; save count LSB
  lda 3,x   ; destination address
  psha      ; prepared in HX
  ldx 4,x
  pulh
dbcl:
  pshx      ; save destination pointer
  pshh
  ldx 7,sp    ; load source pointer
  pshx
  pulh
  ldx 8,sp
  inc 8,sp    ; increment pointer
  bne oks
  inc 7,sp
```

```
oks:
  lda 0,x   ; load byte
  pulh      ; get destiation
  pulx      ; pointer
  sta 0,x   ; store byte
  aix #1    ; next byte
  dbnz  2,sp,dbcl ; count LSB
  dbnz  1,sp,dbcl ; count MSB
  ais #2    ; cleanup stack
  pulh      ; reload pointer
  pulx
  aix #5    ; next descriptor
  bra ibcl    ; and loop
zbss:
  ais #2    ; remove pointer
  ldhx  #__sbss   ; start of bss
  bra loop    ; start loop
zbcl:
  clr 0,x   ; clear byte
  aix #1    ; next byte
loop:
  cphx  #__memory ; up to the end
  bne zbcl    ; and loop
prog:
  jsr _main   ; execute main
_exit:
  bra _exit   ; and stay here
;
  end
```

```
[REMOTE:data.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  data.c                                                       //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  global data,                                                              //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  28/03/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "declared.h"
#include    "gp32.h"


//////////////////////
// Global variables //
//////////////////////
@tiny char                          text_buffer[17];
@tiny unsigned char                 button_press_status;
@tiny unsigned char                 button_debounce_counter;
@tiny unsigned char                 button_release_counter;
@tiny unsigned char                 mode;
@tiny unsigned char                 mode_copy;
@tiny unsigned char                 character_count;
@tiny unsigned char                 shut_down_ii;
@tiny unsigned char                 shut_down_jj;
@tiny unsigned char                 pir_buffer_size;
@tiny unsigned short int            stop_counter;
@tiny unsigned short int            user_password;
@tiny unsigned short int            pressed_pattern;
@tiny union uBITS                   button_flags;
@tiny union uUNSIGNED_INTEGER       adjust_value;
@tiny union uUNSIGNED_INTEGER       pir_password;
@tiny union uUNSIGNED_INTEGER_BIT   button_pattern;
@tiny union uUNSIGNED_INTEGER       delta_sig_res;
@near struct sDELTA_SIGMA_ADJUST    * @tiny ds_adjust_ptr;
```

```
//////////////////////////////////////////
// NOTE : Above pointer declaration syntax :  //
// This pointer resides in PAGE0 and contains //
// a '@near' (2 byte) address                 //
//////////////////////////////////////////////

////////////////////////////////////
// data used in interrupt routines //
////////////////////////////////////
@tiny volatile unsigned char        ir_buffer[15];
@tiny volatile unsigned char        rs232_buffer[15];
@tiny volatile union uBITS          flags1;

////////////////
// const data //
////////////////
@near const struct sDELTA_SIGMA_ADJUST ds_adjust[8] =
   {
   {5  , 255  , 5}  ,      // 8  bit min, max, step, useage:6 bytes
   {20 , 500  , 20} ,      // 9  bit min, max, step, useage:6 bytes
   {50 , 1000 , 50} ,      // 10 bit min, max, step, useage:6 bytes
   {100, 2000 , 100},      // 11 bit min, max, step, useage:6 bytes
   {200, 4000 , 200},      // 12 bit min, max, step, useage:6 bytes
   {400, 8000 , 400},      // 13 bit min, max, step, useage:6 bytes
   {600, 16000, 600},      // 14 bit min, max, step, useage:6 bytes
   {800, 32000, 800},      // 15 bit min, max, step, useage:6 bytes
   };

//////////////////////////////////////////////////////////////
// string tables for RTC, note NULL terminated strings for use //
// by <string.h> functions                                    //
//////////////////////////////////////////////////////////////
@near const char days_of_week[9][4] =     {
                                          {"---\0"},
                                          {"Mon\0"},
                                          {"Tue\0"},
                                          {"Wed\0"},
                                          {"Thu\0"},
                                          {"Fri\0"},
                                          {"Sat\0"},
                                          {"Sun\0"},
                                          {"XXX\0"}
                                          };
@near const char months_of_year[14][4] =  {
                                          {"---\0"},
                                          {"Jan\0"},
                                          {"Feb\0"},
                                          {"Mar\0"},
                                          {"Apr\0"},
                                          {"May\0"},
                                          {"Jun\0"},
                                          {"Jul\0"},
                                          {"Aug\0"},
                                          {"Sep\0"},
                                          {"Oct\0"},
                                          {"Nov\0"},
                                          {"Dec\0"},
                                          {"XXX\0"}
                                          };
//-----------------------------------------------------------------------------
```

```
[REMOTE:datasort.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                                                      //
// Durham                                                                   //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control        //
//  Filename  :  datasort.c                                                 //
//  Author    :  jtravers                                                   //
//  Compiler  :  Cosmic ANSI-C                                              //
//  CPU       :  68HC908JK31/3                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  IR data integrity routine(s)                                            //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/07/00 jt      creation                                          //
////////////////////////////////////////////////////////////////////////////
#include    <string.h>
#include    "extern.h"
#include    "ir_comms.h"
#include    "lcd.h"
#include    "convert.h"
#include    "mode.h"
#include    "rtc.h"
#include    "rs_comms.h"
#include    "delay.h"
#include    "datasort.h"
```

```
unsigned char CheckSumCheck( unsigned char *address_ptr )
{
union uUNSIGNED_INTEGER checksum;
unsigned char           block_length;
unsigned char           ii;



block_length = *address_ptr;  // first byte of buffer is the block length byte

if ( block_length == 0x00 )
   {
   return 0;   // bad data
   }

////////////////////////////
// calculate the checksum //
////////////////////////////
checksum._16bit = 0;
for ( ii = 0; ii < block_length; ii++ )
   {
   checksum._16bit += *(address_ptr+ii);
   }

//////////////////////////////////////////////////////////////////////
// now check to that received in the buffer addressed by 'address_ptr' //
//////////////////////////////////////////////////////////////////////
if ( checksum._8bit.hibyte == *(address_ptr+block_length)  &&
     checksum._8bit.lobyte == *(address_ptr+block_length+1)  )
   {
   return 1;   // good, full 16bit checksum agreement
   }

return 0;   // checksum did not compare
}  // CheckSumCheck()
//-----------------------------------------------------------------------------


void IRCommsCheck( void )
{
union uUNSIGNED_INTEGER temp2;



if ( ir_mode == IR_MAIN )
   {
   if ( CheckSumCheck(&ir_buffer[0]) )
      {
      WriteText2( LINE1_2, "", PRECLEAR);

      switch ( ir_buffer[BLOCK_TITLE] )
         {
         case SEND_A2D_TRIGGER :
         WriteText2( LINE1, "A2D Trigger:", NOPRECLEAR );
```

```
pir_buffer_size = ir_buffer[DATA_BYTE2];
mode            = MODE_EDIT_A2D_TRIGGER;

////////////////////////////////////
// write acquired data to screen //
////////////////////////////////////
IntegerToASCII( ir_buffer[DATA_BYTE1], &text_buffer[7] );
WriteText1( LINE2 );

//////////////////////////////
// prepare editing variable //
//////////////////////////////
adjust_value._8bit.lobyte = ir_buffer[DATA_BYTE1];
adjust_value._8bit.hibyte = 0;
break;

case SEND_A2D_DIFFERENCE :
WriteText2( LINE1, "A2D Difference:", NOPRECLEAR );
mode = MODE_EDIT_A2D_DIFFERENCE;

////////////////////////////////////
// write acquired data to screen //
////////////////////////////////////
IntegerToASCII( ir_buffer[DATA_BYTE1], &text_buffer[7] );
WriteText1( LINE2 );

//////////////////////////////
// prepare editing variable //
//////////////////////////////
adjust_value._8bit.lobyte = ir_buffer[DATA_BYTE1];
adjust_value._8bit.hibyte = 0;
break;

case SEND_A2D_LOOPTIME :
WriteText2( LINE1, "A2D Loop Time:", NOPRECLEAR );
mode = MODE_EDIT_A2D_LOOPTIME;

////////////////////////////////////
// write acquired data to screen //
////////////////////////////////////
IntegerToASCII( ir_buffer[DATA_BYTE1], &text_buffer[7] );
WriteText1( LINE2 );

//////////////////////////////
// prepare editing variable //
//////////////////////////////
adjust_value._8bit.lobyte = ir_buffer[DATA_BYTE1];
adjust_value._8bit.hibyte = 0;
break;

case SEND_DELTA_SIG_RESOLUTION :
WriteText2( LINE1, "Delta Sig Res'n:", NOPRECLEAR );
mode                     = MODE_EDIT_DELTA_SIG_RESOLUTION;
```

```
////////////////////////////
// prepare editing variable //
////////////////////////////
temp2._8bit.hibyte        = ir_buffer[DATA_BYTE1];
temp2._8bit.lobyte        = ir_buffer[DATA_BYTE2];
adjust_value._8bit.lobyte = ConvertResolutionToBit( temp2._16bit );

///////////////////////////////////
// write acquired data to screen //
///////////////////////////////////
IntegerToASCII( adjust_value._8bit.lobyte, &text_buffer[7] );
WriteText1( LINE2 );
break;

case SEND_DELTA_SIG_EVENT :
WriteText2( LINE1, "Delta Sig Event:", NOPRECLEAR );
mode = MODE_EDIT_DELTA_SIG_EVENT;

////////////////////////////
// prepare editing variable //
////////////////////////////
temp2._8bit.hibyte  = ir_buffer[DATA_BYTE1];
temp2._8bit.lobyte  = ir_buffer[DATA_BYTE2];
adjust_value._16bit = temp2._16bit;

///////////////////////////////////
// write acquired data to screen //
///////////////////////////////////
IntegerToASCII( temp2._16bit, &text_buffer[7] );
WriteText1( LINE2 );

//////////////////////////////////////////////////////////////////
// we require the current delta sigma bit resolution to provide //
// the edit clamp/checks while adjusting the event value        //
//////////////////////////////////////////////////////////////////
delta_sig_res._8bit.hibyte = ir_buffer[DATA_BYTE3];
delta_sig_res._8bit.lobyte = ir_buffer[DATA_BYTE4];
break;

case SEND_PASSWORD :
pir_password._8bit.hibyte = ir_buffer[DATA_BYTE1];
pir_password._8bit.lobyte = ir_buffer[DATA_BYTE2];
temp2._8bit.hibyte        = ir_buffer[DATA_BYTE3];
temp2._8bit.lobyte        = ir_buffer[DATA_BYTE4];
temp2._16bit              = ~temp2._16bit; // 1's complement

//////////////////////////////////////////////////////////////////
// additional data integrity check, the password must be received //
// correctly                                                    //
//////////////////////////////////////////////////////////////////
if ( temp2._16bit == pir_password._16bit )
   {
   mode = MODE_USER_ENTER_PASSWORD; // GetPassword() do-while break out
   }
break;
```

```
         }
      }
   else  // checksum failed
      {
      WriteText2( LINE1, "Bad IR Checksum!", NOPRECLEAR );

      if ( mode == MODE_WAITING_FOR_PIR_PASSWORD )
         {
         WriteText2( LINE2, "Auto Retry...", NOPRECLEAR );
         }
      else
         {
         WriteText2( LINE2, "Try Again...", NOPRECLEAR );
         Delay10ms(_1S);               // show message for 1s
         RedrawFlashParameterScreen(); // redraw previous screen if an
         }
      }

   ir_mode = IR_IDLE;   // ready for next
   }
}  // IRCommsCheck()
//-----------------------------------------------------------------------------


void RS232CommsCheck( void )
{
unsigned char           ii;
union uUNSIGNED_INTEGER year;
void                    *ptr;
struct RTC              new_time;



if ( flags1.bit.CHECK_RS232_DATA )
   {
   if ( CheckSumCheck(&rs232_buffer[0]) )
      {
      ////////////////////////////////////////
      // The received data has been deemed valid //
      // and has the format :                   //
      //                                        //
      // rs232_buffer[0]  : block length        //
      // rs232_buffer[1]  : block title         //
      // rs232_buffer[2]  : year hibyte         //
      // rs232_buffer[3]  : year lobyte         //
      // rs232_buffer[4]  : month  Jan = 1 etc  //
      // rs232_buffer[5]  : day    Mon = 1 etc  //
      // rs232_buffer[6]  : date                //
      // rs232_buffer[7]  : hours               //
      // rs232_buffer[8]  : minutes             //
      // rs232_buffer[9]  : seconds             //
      // rs232_buffer[10] : checksum hibyte     //
      // rs232_buffer[11] : checksum lobyte     //
      ////////////////////////////////////////
```

```
//////////////////////////////////////////////////////
// the time info from the pc has arrived in binary form, we //
// need to program the RTC with hex data ie to program      //
// the RTC with 3 hours and 47 mins we need to supply 0x03  //
// and 0x47 for the relevant parameters                     //
//////////////////////////////////////////////////////
for ( ii = 4; ii < 10; ii++ )
   {                                ////////////////////////////////////
   DecToHex( &rs232_buffer[ii] );   // NOTE: not applying to year bytes //
   }                                // as special processing applies    //
                                    // (see below)                      //
                                    ////////////////////////////////////

//////////////////////////////////////////////////////
// assign RTC data here with hex equivalent of decimal data //
//////////////////////////////////////////////////////
new_time.month   = rs232_buffer[4];
new_time.day     = rs232_buffer[5];
new_time.date    = rs232_buffer[6];
new_time.hours   = rs232_buffer[7];
new_time.minutes = rs232_buffer[8];
new_time.seconds = rs232_buffer[9];

//////////////////////////////////////////////////////////////////////////
// For example, if the current year is 2000 (which it is) then we find  //
// rs232_buffer[2] = 0x7D and rs232_buffer[3] = 0xD0. This doesn't look //
// much like 2000. We need to convert the 0x07D0 into 0x2000 which can  //
// be sent to the RTC.                                                  //
//                                                                      //
// Firstly we'll convert the 2000 (integer) into an ASCII form ie "2000"//
// then convert this to 0x20 and 0x00.                                  //
//////////////////////////////////////////////////////////////////////////
year._8bit.hibyte = rs232_buffer[2];
year._8bit.lobyte = rs232_buffer[3];

//////////////////////////////////////////////////////////
// NOTE1: using 'rs232_buffer' as storage here since the above //
// 'system_time' assignements have been made.                  //
// NOTE2: use of 'void' pointer here, 'IntegerToASII()' expects//
// a 'char' pointer as the second arg, we're using an 'unsigned//
// char' buffer for storage.                                   //
//////////////////////////////////////////////////////////
ptr = &rs232_buffer[0];
IntegerToASCII( year._16bit, (char *)ptr );

//////////////////////////////////////
// using the above example we'll have : //
//                                      //
// rs232_buffer[0] = '2'                //
// rs232_buffer[1] = '0'                //
// rs232_buffer[2] = '0'                //
// rs232_buffer[3] = '0'                //
//////////////////////////////////////
for ( ii = 0; ii < 4; ii++ )
   {
```

```
         rs232_buffer[ii] -= '0';   // converting from char to decimal
         }                          // ie from '2' -> 2, '0' -> 0 etc

      ////////////////////////////////////////////////////////
      // the data is now represented as hi:lo byte pairs ie :  //
      //                                                    //
      // rs232_buffer[0] = 2                                //
      // rs232_buffer[1] = 0                                //
      // rs232_buffer[2] = 0                                //
      // rs232_buffer[3] = 0                                //
      ////////////////////////////////////////////////////////
      rs232_buffer[0] *= 10;
      rs232_buffer[0] += rs232_buffer[1]; // == 20

      rs232_buffer[2] *= 10;
      rs232_buffer[2] += rs232_buffer[3]; // == 0

      DecToHex( &rs232_buffer[0] );       // 20 -> 0x20
      DecToHex( &rs232_buffer[2] );       //  0 -> 0x00

      new_time.year._8bit.hibyte = rs232_buffer[0];
      new_time.year._8bit.lobyte = rs232_buffer[2];

      ////////////////////////////////////////////////
      // we can now finally send the RTC the new values //
      ////////////////////////////////////////////////
      if ( SetRTC(&new_time) )
         {
         ////////////////////////////////
         // send an ACK back to the pc //
         ////////////////////////////////
         Send_RS232_CommsPacket(ACKNOWLEDGE, 0);   // '0' for no data here
         }
      else
         {
         ////////////////////////////////////////////////
         // send a NOACK back to the pc, user can try again //
         ////////////////////////////////////////////////
         Send_RS232_CommsPacket(NOACKNOWLEDGE, 0); // '0' for no data here
         }
      }

   flags1.bit.CHECK_RS232_DATA = 0; // ready for next
   SCC2.bit.SCRIE              = 1; // allow SCI receive interupts again after
   }                                // this function processing
}  // RS232CommsCheck()
//---------------------------------------------------------------------------
```

```
unsigned char ConvertResolutionToBit( unsigned short int value )
{
unsigned char  temp;


//////////////////
// range clamps //
//////////////////
if ( value < 256   )    value = 256;
if ( value > 32768 )    value = 32768;

//////////////////////////
// convert to power of 2 //
//////////////////////////
switch ( value )
   {
   case  _8BIT :
   temp = 8;
   break;

   case  _9BIT :
   temp = 9;
   break;

   case  _10BIT :
   temp = 10;
   break;

   case  _11BIT :
   temp = 11;
   break;

   case  _12BIT :
   temp = 12;
   break;

   case  _13BIT :
   temp = 13;
   break;

   case  _14BIT :
   temp = 14;
   break;

   case  _15BIT :
   temp = 15;
   break;

   default :
   temp = 12;
   }

return temp;
}  // ConvertResolutionToBit()
//---------------------------------------------------------------------------
```

```
void ConvertBitToResolution( unsigned char bit_resolution )
{
switch ( bit_resolution )
   {
   case 8 :       // 8bit resolution, 0...255
   adjust_value._16bit = _8BIT;
   break;

   case 9 :       // 9bit resolution, 0...511
   adjust_value._16bit = _9BIT;
   break;

   case 10 :       // 10bit resolution, 0...1023
   adjust_value._16bit = _10BIT;
   break;

   case 11 :       // 11bit resolution, 0...2047
   adjust_value._16bit = _11BIT;
   break;

   case 12 :       // 12bit resolution, 0...4097
   adjust_value._16bit = _12BIT;
   break;

   case 13 :       // 13bit resolution, 0...8191
   adjust_value._16bit = _13BIT;
   break;

   case 14 :       // 14bit resolution, 0...16383
   adjust_value._16bit = _14BIT;
   break;

   case 15 :       // 15bit resolution, 0...32767
   adjust_value._16bit = _15BIT;
   break;
   }
}  // ConvertBitToResolution()
//---------------------------------------------------------------------------
```

```
void Assign_DS_Pointer( void )
{
switch ( delta_sig_res._16bit )
   {
   case _8BIT :       // 8bit resolution, 0...255
   ds_adjust_ptr = &ds_adjust[0];
   break;

   case _9BIT :       // 9bit resolution, 0...511
   ds_adjust_ptr = &ds_adjust[1];
   break;

   case _10BIT :       // 10bit resolution, 0...1023
   ds_adjust_ptr = &ds_adjust[2];
   break;

   case _11BIT :       // 11bit resolution, 0...2047
   ds_adjust_ptr = &ds_adjust[3];
   break;

   case _12BIT :       // 12bit resolution, 0...4097
   ds_adjust_ptr = &ds_adjust[4];
   break;

   case _13BIT :       // 13bit resolution, 0...8191
   ds_adjust_ptr = &ds_adjust[5];
   break;

   case _14BIT :       // 14bit resolution, 0...16383
   ds_adjust_ptr = &ds_adjust[6];
   break;

   case _15BIT :       // 15bit resolution, 0...32767
   ds_adjust_ptr = &ds_adjust[7];
   break;
   }
}  // Assign_DS_Pointer()
//-----------------------------------------------------------------------------
```

```
[REMOTE:datasort.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  datasort.h                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JL3                                                   //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for 'datasort.c'                                              //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/07/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DATASORT_H_
#define     __DATASORT_H_



unsigned char  CheckSumCheck( unsigned char * );
void           IRCommsCheck( void );
void           RS232CommsCheck( void );
unsigned char  ConvertResolutionToBit( unsigned short int );
void           ConvertBitToResolution( unsigned char );
void           Assign_DS_Pointer( void );
#endif
```

```
[REMOTE:declared.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  declared.h                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908GP32                                                  //
////////////////////  File Contents  ///////////////////////////////////////
//  declared data types                                                       //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/07/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DECLARED_H_
#define     __DECLARED_H_



////////////////////
// bit/byte access //
////////////////////
struct sPORT
{
unsigned char bit0 : 1;
unsigned char bit1 : 1;
unsigned char bit2 : 1;
unsigned char bit3 : 1;
unsigned char bit4 : 1;
unsigned char bit5 : 1;
unsigned char bit6 : 1;
unsigned char bit7 : 1;
};
union uBITS
{
```

```
unsigned char  byte;
unsigned char  reg;
struct sPORT   bit;
};

//////////////////////////
// 16 bit 'bit' data type //
//////////////////////////
struct sUNSIGNED_INTEGER
{
unsigned char  hibyte;   // 0x12XX
unsigned char  lobyte;   // 0xXX34
};
union uUNSIGNED_INTEGER
{
unsigned short int         _16bit;
struct sUNSIGNED_INTEGER   _8bit;
};


struct sUNSIGNED_INTEGER_BIT
{
union uBITS    hibyte;   // 0x12XX
union uBITS    lobyte;   // 0xXX34
};
union uUNSIGNED_INTEGER_BIT
{
unsigned short int            _16bit;
struct sUNSIGNED_INTEGER_BIT  _8bit;
};

struct RTC
{
unsigned char           seconds;
unsigned char           minutes;
unsigned char           hours;
unsigned char           day;
unsigned char           date;
unsigned char           month;
union uUNSIGNED_INTEGER year;
};

struct sDELTA_SIGMA_ADJUST
{
unsigned short int   min;
unsigned short int   max;
unsigned short int   step;
};

#endif
```

```
[REMOTE:define.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  define.h                                                     //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  global defines                                                            //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  27/07/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef         __DEFINE_H_
#define         __DEFINE_H_


//////////////////////////
// conditional defines //
//////////////////////////
                              ////////////////////////////////////////////
#define  __MMDS_EMULATOR_     // see recommendation from 'hc08gm32em.pdf' //
                              // page 16                                  //
                              ////////////////////////////////////////////
/////////////////
// I/O defines //
/////////////////
#define  TIMING_PIN           PTD.bit.bit5   // for debugging
#define  TIMING_PIN_DDR       DDRD.bit.bit5  // for debugging
```

```
//////////////////////////////
// general numerical defines //
//////////////////////////////
#define _1S                   100   // 100*10ms == 1s

#define _5MINUTE              30000 // 300s/10ms
// debug only quick time-out   #define _5MINUTE              1000 // 10s/10ms

//#define _1MINUTE              6000  // 60s/10ms
//#define _1MINUTE              1000  // 10s/10ms FOR DEBUGGING
#define  TIMER_ROLLOVER       24576 // 10E-3 * 2.4576E6
#define  _3P5MS               8602  // 3.5E-3* 2.4576E6
#define  _4P5MS               11059 // 4.5E-3/2.4576E6
#define _1_BITWIDTH           3686  // == 1.5ms
#define MIN_PULSE_WIDTH       492   // == 200us
#define MAX_PULSE_WIDTH       6144  // == 2.5ms
#define NOISE_LIMIT           737   // 300E-6 * 2.4576E6
#define _50US                 16    // from delay.c, 11+(7*16)==123 bus cycles
                                    // == 123*(1/2.4576E6) = 50.0us
#define _100US                35    // from delay.c, 11+(7*35)==256 bus cycles
                                    // == 256*(1/2.4576E6) = 104.2us

////////////////////
// flags1 defines //
////////////////////
#define _10MS_LOOP            bit0
#define IR_ACTIVITY           bit1
#define PASSWORD_WRAP         bit2
#define CHECK_RS232_DATA      bit3
#define TO_BE_ASSIGNED_1      bit4  // this is free for use
#define TO_BE_ASSIGNED_2      bit5  // this is free for use
#define TO_BE_ASSIGNED_3      bit6  // this is free for use
#define TO_BE_ASSIGNED_4      bit7  // this is free for use

//////////////////
// Assembler 'C' //
//////////////////
#define  ClrPAGE0Ram() _asm("clrh\n ldx #192\nLOOP2: clr $40-1,x\n dbnzx LOOP2")
#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

enum  { IR_IDLE=0x01, IR_DATA, IR_MAIN };

#endif
```

```
[REMOTE:delay.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design                          //
//  Filename  :  delay.h                                                      //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  delay routines                                                            //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#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             #X          ; delay arg sent to function  [2] cycles      //
// jsr             DelayUSecs  ; branches to this function   [5] cycles      //
//loop1 nop                    ; nop                         [1] cycle       //
//      nop                    ; nop                         [1] cycle       //
//      nop                    ; nop                         [1] cycle       //
//      nop                    ; nop                         [1] cycle       //
//      dbnza      loop1       ; loop till acc = 0           [3] cycles      //
//      rts                    ; return from sub-routine     [4] cycles      //
//                                                                           //
```

```
// This gives a total delay of 11+(4+3)*X cycles, where X is the arg sent.    //
// We are using a 2.4576MHz internal bus via the PLL. This gives us a bus     //
// cycle time of 1/2.4576E6 = 406.9ns                                         //
//                                                                            //
// For eg, if we want to delay for 50us, then we have:                        //
// 50E-6/406.9E-9 = 122.88 bus cycles => 123 = 11 + 7*X, => X = 16            //
//                                                                            //
// 'DelayUSecs( 16 )' to get 50us delay.                                      //
//                                                                            //
// Arguments: 'X' delay value as calculated from 'cyles = 11 + 7X'            //
// Returns  : none                                                            //
////////////////////////////////////////////////////////////////////////////////
void Delay( unsigned char uSecs )
{
#asm
LOOP1:
   nop
   nop
   nop
   nop
   dbnza  LOOP1
#endasm
}  // Delay()
//------------------------------------------------------------------------------


void Delay10ms( unsigned char _10ms_multiple )
{
unsigned char  ii;

for ( ii = 0; ii < _10ms_multiple; ii++ )
   {
   ServiceWatchDog();
   while ( !flags1.bit._10MS_LOOP );
   flags1.bit._10MS_LOOP = 0;
   }
}  // Delay10ms()
//------------------------------------------------------------------------------
```

```
[REMOTE:delay.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design                          //
//  Filename  :  delay.h                                                      //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908GP32                                                  //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for delay.c                                                   //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  15/05/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __DELAY_H_
#define     __DELAY_H_

////////////////////////
// function prototypes //
////////////////////////
void Delay( unsigned char );
void Delay10ms( unsigned char );

#endif
```

```
[REMOTE:digipot.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  digipot.c                                                    //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  LCD contast control via digital pot                                       //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  08/11/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "digipot.h"



void DigiPot( unsigned char command )
{
unsigned char  ii;


DIGIPOT_CS  = 0;              // ensure CS active
DIGIPOT_INC = 1;              // signal stable, hi->lo is command transition
DDRB.reg   |= 0b00011100;     // ensure output
                              //
for ( ii = 0; ii < 5; ii++ )  // we insert this for loop to speed up the
   {                          // user perceived contrast change per button press
                              // if you decrease the value the lcd will change
                              // more slowly and conversely if you increase the
                              // the loop max value the lcd contrast will change
                              // quickly.

   if ( command == DP_INCREMENT )   DIGIPOT_UD = 0;
   else                             DIGIPOT_UD = 1;
```

```
   ////////////////////////
   // apply digipot command //
   ////////////////////////
   DIGIPOT_INC = 0;           // force wiper position change
   NOP();
   DIGIPOT_INC = 1;           //
   }

DIGIPOT_CS = 1;               // ensure CS off + write to value to eeprom
}  // DigiPot()
//////////////////////////////////////////////////////////////////////////////
```

```
[REMOTE:digipot.h]
//////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
//////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  digipot.h                                                    //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for digipot.c                                                 //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  08/11/00 jt      creation                                            //
//////////////////////////////////////////////////////////////////////////////
#ifndef     __DIGIPOT_H_
#define     __DIGIPOT_H_

#define  DIGIPOT_CS  PTB.bit.bit2
#define  DIGIPOT_UD  PTB.bit.bit3
#define  DIGIPOT_INC PTB.bit.bit4

enum     { DP_INCREMENT, DP_DECREMENT };
```

```
/////////////////
// protoype(s) //
/////////////////
void DigiPot( unsigned char );

#endif
```

```
[REMOTE:error.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  error.c                                                      //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  Error routines                                                            //
//////////////////////////  Update Information  ////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/09/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    <string.h>
#include    "extern.h"
#include    "lcd.h"
#include    "convert.h"
#include    "delay.h"
#include    "error.h"


void ErrorCondition( unsigned char value )
{
```

```
unsigned char  breakout_count;
unsigned char  ii;



WriteText2( LINE1_2, "", PRECLEAR );
strcpy( &text_buffer[0], "Error " );   // last ’ ’ char occupies ’text_buffer[5]
IntegerToASCII( value, &text_buffer[6] );  // since next free position is 6
WriteText1( LINE1 );

switch ( value )
   {
   case ERROR_NO_IR_COMMS :
   WriteText2( LINE2, "No IR Comms [IN]", NOPRECLEAR );
   break;

   case ERROR_NO_PASSWORD :
   WriteText2( LINE2, "No PIR Password ", NOPRECLEAR );
   break;
   }

//////////////////////////
// show message for 5s //
//////////////////////////
breakout_count = 5;

for ( ii = 0; ii < 50; ii++ ) // 50*100ms == 5s
   {
   Delay10ms(10);       // 100ms

   if ( ii % 10 == 0 )  // every second
      {
      IntegerToASCII( breakout_count--, &text_buffer[0] );  // show lcd counter
      WriteText1( LINE1+15 );                               // decrementing
      }
   }
}  // ErrorCondition()
//-----------------------------------------------------------------------------
```

```
[REMOTE:error.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  error.h                                                      //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for 'error.c'                                                 //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/09/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __ERROR_H_
#define     __ERROR_H_


enum  {
      ERROR_NO_IR_COMMS = 0x01,
      ERROR_NO_PASSWORD
      };

///////////////
// prototypes //
///////////////
void ErrorCondition( unsigned char  );

#endif
```

```
[REMOTE:extern.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  error.h                                                      //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ////////////////////////////////////////
//  'extern' declarations for global variables declared in 'data.c'           //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/09/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __EXTERN_H_
#define     __EXTERN_H_

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

#ifndef     __GP32_H_
#include    "gp32.h"
#endif

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

```
//////////////////////
// Global variables //
//////////////////////
extern @tiny char                          text_buffer[17];
extern @tiny unsigned char                 button_press_status;
extern @tiny unsigned char                 button_debounce_counter;
extern @tiny unsigned char                 button_release_counter;
extern @tiny unsigned char                 mode;
extern @tiny unsigned char                 mode_copy;
extern @tiny unsigned char                 character_count;
extern @tiny unsigned char                 shut_down_ii;
extern @tiny unsigned char                 shut_down_jj;
extern @tiny unsigned char                 pir_buffer_size;
extern @tiny unsigned short int            stop_counter;
extern @tiny unsigned short int            user_password;
extern @tiny unsigned short int            pressed_pattern;
extern @tiny union uBITS                   button_flags;
extern @tiny union uUNSIGNED_INTEGER       pir_password;
extern @tiny union uUNSIGNED_INTEGER_BIT   button_pattern;
extern @tiny union uUNSIGNED_INTEGER       delta_sig_res;
extern @tiny union uUNSIGNED_INTEGER       adjust_value;

////////////////////////////////////////////////////////
// NOTE : declaration syntax :                        //
// This pointer resides in PAGE0 and points to 'near' data //
////////////////////////////////////////////////////////
extern @near struct sDELTA_SIGMA_ADJUST    * @tiny ds_adjust_ptr;

/////////////////////////////////////
// data used in interrupt routines //
/////////////////////////////////////
extern @tiny volatile unsigned char        ir_buffer[15];
extern @tiny volatile unsigned char        rs232_buffer[15];
extern @tiny volatile union uBITS          flags1;

////////////////
// const data //
////////////////
extern @near const struct sDELTA_SIGMA_ADJUST   ds_adjust[8];
extern @near const char                         days_of_week[9][4];
extern @near const char                         months_of_year[14][4];

#endif
```

```
[REMOTE:gp32.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  gp32.h                                                       //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  register definitions for MC68HC908GP32                                    //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/09/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __GP32_H_
#define     __GP32_H_

#include    "declared.h"

//////////////////
// CPU Registers //
//////////////////
@tiny volatile union uBITS          PTA         @0x00;
@tiny volatile union uBITS          PTB         @0x01;
@tiny volatile union uBITS          PTC         @0x02;
@tiny volatile union uBITS          PTD         @0x03;
@tiny volatile union uBITS          DDRA        @0x04;
@tiny volatile union uBITS          DDRB        @0x05;
@tiny volatile union uBITS          DDRC        @0x06;
@tiny volatile union uBITS          DDRD        @0x07;
@tiny volatile union uBITS          PTE         @0x08;
@tiny volatile union uBITS          DDRE        @0x0C;
@tiny volatile union uBITS          PTAPUE      @0x0D;
@tiny volatile union uBITS          PTCPUE      @0x0E;
@tiny volatile union uBITS          PTDPUE      @0x0F;
@tiny volatile union uBITS          SPCR        @0x10;
@tiny volatile union uBITS          SPSCR       @0x11;
@tiny volatile union uBITS          SPDR        @0x12;
```

```
@tiny volatile union uBITS          SCC1        @0x13;
@tiny volatile union uBITS          SCC2        @0x14;
@tiny volatile union uBITS          SCC3        @0x15;
@tiny volatile union uBITS          SCS1        @0x16;
@tiny volatile union uBITS          SCS2        @0x17;
@tiny volatile union uBITS          SCDR        @0x18;
@tiny volatile union uBITS          SCBR        @0x19;
@tiny volatile union uBITS          INTKBSCR    @0x1A;
@tiny volatile union uBITS          INTKBIER    @0x1B;
@tiny volatile union uBITS          INTSCR      @0x1D;
@tiny volatile union uBITS          CONFIG2     @0x1E;
@tiny volatile union uBITS          CONFIG1     @0x1F;
@tiny volatile union uBITS          T1SC        @0x20;
@tiny volatile union uBITS          T1CNTH      @0x21;
@tiny volatile unsigned short int   T1CNT       @0x21;
@tiny volatile union uBITS          T1CNTL      @0x22;
@tiny volatile union uBITS          T1MODH      @0x23;
@tiny volatile unsigned short int   T1MOD       @0x23;
@tiny volatile union uBITS          T1MODL      @0x24;
@tiny volatile union uBITS          T1SC0       @0x25;
@tiny volatile union uBITS          T1CH0H      @0x26;
@tiny volatile unsigned short int   T1CH0       @0x26;
@tiny volatile union uBITS          T1CH0L      @0x27;
@tiny volatile union uBITS          T1SC1       @0x28;
@tiny volatile union uBITS          T1CH1H      @0x29;
@tiny volatile unsigned short int   T1CH1       @0x29;
@tiny volatile union uBITS          T1CH1L      @0x2A;
@tiny volatile union uBITS          T2SC        @0x2B;
@tiny volatile union uBITS          T2CNTH      @0x2C;
@tiny volatile unsigned short int   T2CNT       @0x2C;
@tiny volatile union uBITS          T2CNTL      @0x2D;
@tiny volatile union uBITS          T2MODH      @0x2E;
@tiny volatile unsigned short int   T2MOD       @0x2E;
@tiny volatile union uBITS          T2MODL      @0x2F;
@tiny volatile union uBITS          T2SC0       @0x30;
@tiny volatile union uBITS          T2CH0H      @0x31;
@tiny volatile unsigned short int   T2CH0       @0x31;
@tiny volatile union uBITS          T2CH0L      @0x32;
@tiny volatile union uBITS          T2SC1       @0x33;
@tiny volatile union uBITS          T2CH1H      @0x34;
@tiny volatile unsigned short int   T2CH1       @0x34;
@tiny volatile union uBITS          T2CH1L      @0x35;
@tiny volatile union uBITS          PCTL        @0x36;
@tiny volatile union uBITS          PBWC        @0x37;
@tiny volatile union uBITS          PMSH        @0x38;
@tiny volatile unsigned short int   PMS         @0x38;
@tiny volatile union uBITS          PMSL        @0x39;
@tiny volatile union uBITS          PMRS        @0x3A;
@tiny volatile union uBITS          PMDS        @0x3B;
@tiny volatile union uBITS          ADSCR       @0x3C;
@tiny volatile union uBITS          ADR         @0x3D;
@tiny volatile union uBITS          ADICLK      @0x3E;
@near volatile union uBITS          SBSR        @0xFE00;
@near volatile union uBITS          SRSR        @0xFE01;
@near volatile union uBITS          SUBAR       @0xFE02;
@near volatile union uBITS          SBFCR       @0xFE03;
@near volatile union uBITS          INT1        @0xFE04;
@near volatile union uBITS          INT2        @0xFE05;
```

```
@near volatile union uBITS          INT3        @0xFE06;
@near volatile union uBITS          FLCR        @0xFE08;
@near volatile union uBITS          BRKH        @0xFE09;
@near volatile unsigned short int   BRK         @0xFE09;
@near volatile union uBITS          BRKL        @0xFE0A;
@near volatile union uBITS          BRKSCR      @0xFE0B;
@near volatile union uBITS          LVISR       @0xFE0C;
@near volatile union uBITS          FLBPR       @0xFF7E;
@near volatile union uBITS          COPCTL      @0xFFFF;

////////////////////////////////////////////////////
// access to the HC08 condition code reg : carry flag //
////////////////////////////////////////////////////
@builtin unsigned char carry( void );

//////////
// INT1 //
//////////
#define  IF1      bit2
#define  IF3      bit4
#define  IF4      bit5
#define  IF5      bit6

//////////
// INT2 //
//////////
#define  IF14     bit7

//////////
// INT3 //
//////////
#define  IF15     bit0

//////////////
// 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 //
////////////////////////////
#define CH0       bit0
#define CH1       bit1
#define CH2       bit2
#define CH3       bit3
#define CH4       bit4
#define ADCO      bit5
#define AIEN      bit6
#define COCO      bit7

//////////////////////
// A2D input clock reg //
//////////////////////
#define ADIV0     bit5
#define ADIV1     bit6
#define ADIV2     bit7

//////////////////
// FLASH control //
//////////////////
#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
```

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

#endif
```

```
[REMOTE:gp32.lkf]
#############################################################################
# COSMIC HC08 LINKER COMMAND FILE FOR MOTOROLA HC908GP32 PIR REMOTE UNIT #
# ATEECC July 2000                                                        #
#############################################################################


####################
# declared symbols #
####################
+def __memory=@.bss     # symbol used by startup
+def __stack=0x023f     # NOTE: stack pointer relocation,
                        # instructions occur in 'crtsi.s'
                        # 0x023f is the last ram byte in the 908GP32
+def __sbss=0x00f0      # for static initialised data 'bsct' see below


#############
# PAGE0 RAM #
#############
+seg .ubsct -b 0x0040 -n TinyRam    -m 192-16  # PAGE0 RAM, 16 for below
+seg .bsct  -b 0x00f0 -n StaticInit -m 16      # initialised PAGE0 static data

##############################
# GP32 additional RAM block #
##############################
+seg .data  -b 0x100 -n NearRAM -m 320          # total ram = 192+320 = 512

##############
# CONST DATA #
##############
+seg .const -b 0xfd8b -n ConstData -m 117       # for const data declared
                                                # in 'data.c', occupies the
                                                # last 117 bytes of FLASH
                                                # memory
############################
# variables data for PAGE0 #
############################
ireg.o                                          # ensuring that the Cosmic
                                                # variables 'c_reg'
lreg.o                                          # and 'c_lreg' are positioned
                                                # at the beginning of ram this
                                                # ensures that during memcpy
                                                # operations they do not get
                                                # overwritten with copied data
data.o                                          # user declared data

##############################
# FLASH memory for user code #
##############################
+seg .text -b 0x8000 -n UserFLASH -m 32256-117  # MC68HC908GP32 user code start address
                                                # 117 for const data see above
```

```
####################################
# const area for switch jump tables #
####################################
+seg .const -a UserFLASH -n ConstFLASH          # '-a' append section to previous

#############################
# FLASH memory object files #
#############################
crtsi.o         # Cosmic startup routine
                # user code from here...
button.o        # button debounce/decode interface
convert.o       # conversion routines
datasort.o      # ir received data integrity
delay.o         # inline delay
digipot.o       # lcd contrast control
error.o         # IR comms error condition routines
interrup.o      # interrupt handling routines
ir_comms.o      # IR comms routines
i2c.o           # low level i2c routines for the RTC
lcd.o           # lcd routines
main.o          # main() and interrupt vectors
mode.o          # user interface routines
rs_comms.o      # rs232 routines
rtc.o           # real time clock read/write
startup.o       # micro initialise, i/o and timer


####################
# Cosmic libraries #
####################
c:/cosmic/cx08/lib/libi.h08
c:/cosmic/cx08/lib/libm.h08

###########
# Vectors #
###########
+seg .const -b 0xffdc -n Vectors -m 36
vectors.o
```

```
[REMOTE:i2c.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  i2c.c                                                        //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908GP32                                                  //
////////////////////  File Contents  ///////////////////////////////////////
//  i2c routines for accessing real time clock ic                             //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  17/06/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#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 == 1 )
   {                          ////////////////////////////
   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()
//---------------------------------------------------------------------------
```

```
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 )      // now defined in 'define.h' as assembler C
{
////////////////////////////////
// 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 ( carry() ) 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/(1/2.4576E6) == 12.3 == 13 bus cycles                                  //
//                                                                            //
// The call (jsr) is [5] cycles, the return (rts) is [4] cycles leaving the   //
// function body to occupy 13-(5+4) == 4 cycles                               //
////////////////////////////////////////////////////////////////////////////
void SetUpAndHoldTimingDelay( void )
{
NOP();NOP();NOP();NOP();
}  // SetUpAndHoldTimingDelay()
//---------------------------------------------------------------------------
```

```
[REMOTE:i2c.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  i2c.h                                                        //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908GP32                                                  //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for 'i2c.c'                                                   //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  17/06/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __I2C_H_
#define     __I2C_H_
```

```
/////////////////
// I2C defines //
/////////////////

// data
#define SET_SDA               PTB.bit.bit1   = 1
#define RESET_SDA             PTB.bit.bit1   = 0
#define SET_DATA_TO_OUTPUT    DDRB.bit.bit1  = 1
#define SET_DATA_TO_INPUT     DDRB.bit.bit1  = 0
#define READ_SDA              PTB.bit.bit1

// clock
#define SET_SCL               PTB.bit.bit0   = 1
#define RESET_SCL             PTB.bit.bit0   = 0
#define SET_CLOCK_TO_OUTPUT   DDRB.bit.bit0  = 1


unsigned char  InClock( void );
void           OutClock( void );
void           StartBit( void );
void           StopBit( void );
void           SendI2CByte( unsigned char );
unsigned char  GetI2CByte( void );
unsigned char  WaitForI2CAcknowledge( void );
void           SendI2CAcknowledge( void );
void           SetUpAndHoldTimingDelay( void );
#endif
```

```
[REMOTE:interrup.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  interrup.c                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  Interrupt routines                                                        //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/09/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    <string.h>
#include    "extern.h"
#include    "ir_comms.h"
#include    "interrup.h"



@interrupt void TIMER1OVERFLOW( void )
{
if ( T1SC.bit.TOF && T1SC.bit.TOIE )
   {
   T1SC.bit.TOF          = 0;    // clear interrupt flag
   flags1.bit._10MS_LOOP = 1;    // main() sequencer
   }
}  // TIMER1OVERFLOW()
//---------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// timer1 channel0 interrupt routine                                      //
////////////////////////////////////////////////////////////////////////////
@interrupt void TIMER1CHANNEL0( void )
{
static @tiny unsigned char       ir_byte_count   = 0;
static @tiny unsigned char       ir_bit_count    = 0;
static @tiny unsigned char       ir_block_length = 0;
static @tiny unsigned short int  ir_start_time   = 0;
static @tiny unsigned short int  ir_stop_time    = 0;
unsigned short int               time_diff;



////////////////////////////////
// Infra-red decoding routine //
////////////////////////////////
if ( T1SC0.bit.CH0F && T1SC0.bit.CH0IE )
   {
   T1SC0.bit.CH0F = 0;              // clear interrupt flag

   if ( T1SC0.bit.ELS0A && !T1SC0.bit.ELS0B ) // +ve edge event
      {
      ir_start_time   = T1CH0;      // time stamp +ve edge
      T1SC0.bit.ELS0A = 0;
      T1SC0.bit.ELS0B = 1;          // -ve edge next
      }
   else                             // -ve edge event
      {
      ir_stop_time = T1CH0;         // time stamp -ve edge

      ////////////////////////////
      // pulse width calculation //
      ////////////////////////////
      if ( ir_stop_time >= ir_start_time )    // timer rollover!
         {
         time_diff = ir_stop_time - ir_start_time; // standard
         }
      else  // rollover compensation
         {
         time_diff = (TIMER_ROLLOVER-ir_start_time) + ir_stop_time;
         }

      ////////////////////////////////////////////////////////////////
      // Is this pulse an IR comms packet leader START pulse (approx 4ms) //
      ////////////////////////////////////////////////////////////////
      if ( time_diff > _3P5MS && time_diff < _4P5MS && ir_mode == IR_IDLE )
         {
         ////////////////////////////////////////////
         // clear variables for incoming data stream //
         ////////////////////////////////////////////
         memset( &ir_buffer[0], 0x00, sizeof(ir_buffer) );  // clear buffer...
         ir_byte_count            = 0;
         ir_bit_count             = 0;
         ir_block_length          = 0;
```

```
   ir_mode                  = IR_DATA;
   flags1.bit.IR_ACTIVITY   = 1;   // denote a valid START to show there
                                   // has been some comms activity
                                   // never cleared
   }
else if ( ir_mode == IR_DATA )   // must be building a bit pattern
   {
   ////////////////////////////////////////////////////
   // is this pulse in the acceptable pulse width region //
   ////////////////////////////////////////////////////
   if ( time_diff >= MIN_PULSE_WIDTH && time_diff <= MAX_PULSE_WIDTH )
      {
      ////////////////////////////////////////////////////
      // has a '1' arrived, if so set the 'bit_count' bit ie //
      // if 'bit_count' is 3 then set bit3 of 'temp' etc     //
      ////////////////////////////////////////////////////
      if ( time_diff >= _1_BITWIDTH )
         {
         ir_buffer[ir_byte_count] |= (unsigned char)(0x01<<ir_bit_count);
         }

      ////////////////////////////////
      // have we received a byte yet //
      ////////////////////////////////
      if ( ++ir_bit_count >= 8 )
         {
         ir_bit_count = 0;

         if ( !ir_byte_count )  // == 0, first byte...block length byte
            {
            ////////////////////////////////////////////
            // total bytes expected is 'block_length+2' //
            // ('2' for checksum hi and lo bytes)       //
            ////////////////////////////////////////////
            ir_block_length = (unsigned char)(ir_buffer[0] + 2);

            ////////////////////////
            // buffer write clamp //
            ////////////////////////
            if ( ir_block_length > sizeof(ir_buffer) )
               {
               ////////////////////////////////////
               // corrupt data has arrived, abort. //
               ////////////////////////////////////
               ir_mode         = IR_IDLE;
               T1SC0.bit.ELS0A = 1;          // +ve edge...
               T1SC0.bit.ELS0B = 0;          //           ...next
               return;
               }
            }
```

```
               if ( ++ir_byte_count >= ir_block_length )
                  {                    ////////////////////////////////
                  ir_mode = IR_MAIN;   // check data validity on     //
                                       // this packet has been       //
                                       // processed in               //
                                       // 'main()->IRCommsCheck()'   //
                  }                    ////////////////////////////////
               }     // 'if ( ++ir_bit_count >= 8 )'
            }        // 'if ( time_diff >= MIN_.. && time_diff <= MAX_.. )'
         }           // 'else if ( ir_mode == IR_DATA )

      T1SC0.bit.ELS0A = 1;
      T1SC0.bit.ELS0B = 0;           // +ve edge next
      }              // -ve edge
   }
}  // TIMERCHANNEL0()
//---------------------------------------------------------------------------


@interrupt void KEYBOARD( void )
{
INTKBSCR.bit.IMASKK = 1;      // prevent further interrupts until STOP mode
INTKBSCR.bit.ACKK   = 1;      // clear this interrupt request
}  // KEYBOARD()
//---------------------------------------------------------------------------


@interrupt void SCI_RECEIVE( void )
{
unsigned char                 rx_data;
static @tiny unsigned char    rx_count          = 0;
static @tiny unsigned char    * @tiny rs232_ptr = 0;
////////////////////////////////////////////
// NOTE : Above pointer declaration syntax : //
// This pointer resides in PAGE0 and holds a //
// PAGE0 (1 byte) address                    //
////////////////////////////////////////////


if ( SCS1.bit.SCRF )
   {
   ////////////////////////////
   // store latest data byte //
   ////////////////////////////
   rx_data = SCDR.reg;

   ////////////////////////////////////////////
   // is this the first data byte of a packet? //
   ////////////////////////////////////////////
   if ( !rx_count )
      {
      rs232_ptr  = &rs232_buffer[0];
      *rs232_ptr = rx_data;            // should be the incoming block length
      rx_count   = (char)(rx_data-1+2); // block_length-1+2, bytes yet to arrive
                                       // '-1' since this byte is the first
```

```
                                    // '+2' for the additional chksum bytes
      }
   else
      {
             ////////////////////////////////////////
             // assign incoming data to 'rs232_buffer' //
             ////////////////////////////////////////

                           ////////////////////////////////////////////////
      *++rs232_ptr = rx_data; // unary operators associate right to left ie //
                           // the pointer pre increment then dereference //
                           // occurs                                     //
                           //                                            //
      if ( !--rx_count )   // have the expected number of bytes arrived? //
         {                 // (similar to above), decrement before the   //
                           // true test.                                 //
                           //  'if ( --rx_count == 0 )' is the equivalent//
                           ////////////////////////////////////////////////

         flags1.bit.CHECK_RS232_DATA = 1; // all data received, analyse
                                          //  it in main()

         ////////////////////////////////////////////////////////////////////
         // disable receive interrupts until this packet has been processed //
         // in 'RS232CommsCheck()' called from 'main()'                     //
         ////////////////////////////////////////////////////////////////////
         SCC2.bit.SCRIE = 0;
         }
      }
   }
}  // SCI_RECEIVE()
//---------------------------------------------------------------------------
```

```
[REMOTE:interrup.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  interrup.h                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  Header file for interrup.c                                                //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/09/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __INTERRUP_H_
#define     __INTERRUP_H_


////////////////
// prototypes //
////////////////
@interrupt void TIMER1OVERFLOW( void );
@interrupt void TIMER1CHANNEL0( void );
@interrupt void KEYBOARD( void );
@interrupt void SCI_RECEIVE( void );

#endif
```

```
[REMOTE:ir_comms.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  ir_comms.c                                                   //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908GP32                                                  //
////////////////////  File Contents  ///////////////////////////////////////
//  Infrared routines                                                         //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  12/05/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "ir_comms.h"



////////////////////////////////////////////////////////////////
// This function transmits it's function argument out on the TX pin //
// Argument : data byte to send                                   //
// Returns  : none                                                //
////////////////////////////////////////////////////////////////
void Send_IR_Byte( unsigned char data )
{
unsigned char  ii;


ServiceWatchDog();
```

```
///////////////
// data byte //
///////////////
for ( ii = 0; ii < 8; ii++ )
   {
   data >>= 1;

   if ( carry() ) Send_1();
   else           Send_0();
   }
}  // Send_IR_Byte()
//-------------------------------------------------------------------------------


         /*************************/
         /* COMMS PACKET STRUCTURE */
         /*************************/
//////////////////////////////////////////
// ir_buffer[0]   == block length byte    //
// ir_buffer[1]   == block title byte     //
// ir_buffer[2]   == data byte 1          //
// ir_buffer[n]   == data byte 'n'        //
// ir_buffer[n+1] == hibyte checksum      //
// ir_buffer[n+2] == lobyte checksum      //
//                                        //
// Block length is the number of bytes in //
// the block, EXCLUDING the checksum.     //
//                                        //
// Checksum is the 16 bit total of the    //
// block, EXCLUDING the checksum.         //
//////////////////////////////////////////
void Send_IR_CommsPacket( unsigned char block_title, unsigned char block_length)
{
union uUNSIGNED_INTEGER checksum;
unsigned char           ii;


ServiceWatchDog();


////////////////////////////////////////////////
// disable timer0 capture interrupt as we'd likely //
// detect the comms we're about to transmit        //
////////////////////////////////////////////////
T1SC0.bit.CH0IE = 0;

//////////////////////////////
// re-affirm data direction //
//////////////////////////////
IR_TX_DDR = 1;

block_length += 2; // add inherent BLOCK_LENGTH/BLOCK_TITLE bytes to block size
```

```
////////////////////////////////////////////////////
// insert the element values into the 'ir_buffer' array //
////////////////////////////////////////////////////
ir_buffer[0] = block_length;
ir_buffer[1] = block_title;

//////////////////////////////////
// calculate the packet checksum //
//////////////////////////////////
checksum._16bit = 0;
for ( ii = 0; ii < block_length; ii++ )
   {
   checksum._16bit += ir_buffer[ii];
   }

////////////////////////////
// append to 'ir_buffer' //
////////////////////////////
ir_buffer[block_length  ] = checksum._8bit.hibyte;
ir_buffer[block_length+1] = checksum._8bit.lobyte;

//////////////////////////////////////////////////////////////////
// the complete block consista of:-                             //
//                                                              //
// block length + block title + n*data + checksum hi + checksum lo  //
//                                                              //
// The number of bytes that we have to transmit is block_length + 2 //
//////////////////////////////////////////////////////////////////
block_length += 2;

//////////////////
// Reader Pulse //
//////////////////
StartPulse();

/////////////////
// xmit packet //
/////////////////
for ( ii = 0; ii < block_length; ii++ )
   {
   Send_IR_Byte( ir_buffer[ii] );
   }

StopPulse();

if ( T1SC0.bit.CH0F )
   {
   T1SC0.bit.CH0F = 0; // clear interrupt flag if set whilst interrupt disabled
   }

T1SC0.bit.CH0IE = 1;              // IR detect timer0 capture interrupt back on
}  // Send_IR_CommsPacket()
//-----------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// Logic 0 as transmitted by the IR TX pin:                               //
//       ------------                                                     //
//      |/////////////|            |                                      //
//      |// 38kHz ///|            |                                      //
//      |/////////////|            |                                      //
//      |_____________|____________|                                      //
//                                                                        //
//      <-- 700us --><-- 700us -->                                        //
//                                                                        //
// Logic level as seen by receiving pin:                                  //
//                     ------------                                       //
//      |            |            |                                      //
//      |            |            |                                      //
//      |            |            |                                      //
//      |____________|            |                                      //
//                                                                        //
//      <-- 700us --><---700us -->                                        //
//                                                                        //
// The micro measures the width of the +ve pulse to determine the bit value.  //
////////////////////////////////////////////////////////////////////////////
void Send_0( void )
{
ServiceWatchDog();
_38KHzBurstOnTime(_700US);
_38KHzBurstOffTime(_700US);
}  // Send_0()
//--------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// Logic 1 as transmitted by the IR TX pin:                               //
//                                                                        //
//       ------------                                                     //
//      |/////////////|                                      |            //
//      |// 38kHz ///|                                      |            //
//      |/////////////|                                      |            //
//      |_____________|______________________________________|            //
//                                                                        //
//      <-- 700us --><----------- 2100us -------------------->            //
//                                                                        //
// Logic level as seen by receiving pin:                                  //
//                     ----------------------------------------           //
//      |            |                                      |            //
//      |            |                                      |            //
//      |            |                                      |            //
//      |____________|                                      |            //
//                                                                        //
//      <-- 700us --><----------- 2100us -------------------->            //
//                                                                        //
// The micro measures the width of the +ve pulse to determine the bit value.  //
////////////////////////////////////////////////////////////////////////////
```

```
void Send_1( void )
{
ServiceWatchDog();
_38KHzBurstOnTime(_700US);
_38KHzBurstOffTime(_2100US);
}  // Send_1()
//---------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
// The leader pulse as transmitted by the IR TX pin:                      //
//                                                                        //
//       ------------                                                     //
//      |////////////|            |                                      //
//      |// 38kHz ///|            |                                      //
//      |////////////|            |                                      //
//      |____________|____________|                                      //
//                                                                        //
//      <--  4ms  --><--- 4ms --->                                        //
//                                                                        //
// Above leader pulse as seen by micro receiving pin:                     //
//                     ------------                                       //
//      |            |            |                                      //
//      |            |            |                                      //
//      |            |            |                                      //
//      |____________|            |                                      //
//                                                                        //
//      <--- 4ms ---><--- 4ms --->                                        //
//                                                                        //
// The receiving micro measures the width of the +ve pulse to determine the   //
// bit value.                                                             //
////////////////////////////////////////////////////////////////////////////
void StartPulse( void )
{
ServiceWatchDog();
_38KHzBurstOnTime(_4000US);
_38KHzBurstOffTime(_4000US);
}  // StartPulse()
//---------------------------------------------------------------------------


void StopPulse( void )
{
ServiceWatchDog();
_38KHzBurstOnTime(_700US);
}  // StopPulse()
//---------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// This function produces count*26us pulses with 50% mark space ratio ie      //
// 13us high and 13us low.                                                    //
//                                                                            //
// At 2.4576MHz, 13us == 32 (31.95) bus cycles ie 13E-6*(1/2.4576E6)          //
// We use 'nop' to give us the timing we require.                             //
//                                                                            //
// The number of nops is less for the low time as we include the do/while     //
// cycle count in it's timing.                                                //
//                                                                            //
// The total function cycle count is count*64 + 13 (for stack/wdg and return) //
// Note: above cycle count excludes the 'call' cycles.                        //
////////////////////////////////////////////////////////////////////////////
void _38KHzBurstOnTime( unsigned char count )
{
ServiceWatchDog();

//////////////////////////
// transmit 38KHz ZERO //
//////////////////////////
do {
   //////////////
   // start hi //
   //////////////
   IR_TX = 1;

   NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();
   NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();
   NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();

   //////////////
   // now low //
   //////////////
   IR_TX = 0;

   NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();
   NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();
   }  while ( --count );
}  // _38KHzBurstOnTime()
//---------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
// This function produces count*26us timing delay                         //
//                                                                        //
// At 2.4576MHz, 26us == 64 (63.89) bus cycles ie 26E-6*(1/2.4576E6)      //
// We use 'nop' to give us the timing we require.                         //
//                                                                        //
// The total function cycle count is count*64 + 13 (for stack/wdg and return) //
// Note: above cycle count excludes the 'call' cycles.                    //
////////////////////////////////////////////////////////////////////////////
void _38KHzBurstOffTime( unsigned char count )
{
ServiceWatchDog();

do {
   NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();
   NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();
   NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();
   NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();
   NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP();
   }  while ( --count );
}  // _38KHzBurstOffTime()
//---------------------------------------------------------------------------
```

```
[REMOTE:ir_comms.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                    //
// Durham                                                                 //
// DH8 6BP                                                                //
// England                                                                //
//                                                                        //
// Telephone: 0044 1207 693920                                            //
// Fax      : 0044 1207 693921                                            //
// email    : enquiries@ateecc.com                                        //
// web      : www.ateecc.com                                              //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control      //
//  Filename  :  ir_comms.c                                               //
//  Author    :  jtravers                                                 //
//  Compiler  :  Cosmic ANSI-C                                            //
//  CPU       :  68HC908GP32                                              //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for 'ir_comms.c'                                          //
```

```
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                         //
//  ---  -------- ------  -----------------------------------------------------//
//  001  12/05/00 jt      creation                                             //
////////////////////////////////////////////////////////////////////////////////
#ifndef     __IR_COMMS_H_
#define     __IR_COMMS_H_

#define  IR_TX          PTD.bit.bit3
#define  IR_TX_DDR      DDRD.bit.bit3
#define  BLOCK_LENGTH   0
#define  BLOCK_TITLE    1
#define  DATA_BYTE1     2
#define  DATA_BYTE2     3
#define  DATA_BYTE3     4
#define  DATA_BYTE4     5
#define  DATA_BYTE5     6

enum  // block title values
   {
   SEND_A2D_TRIGGER = 0x01,
   SEND_A2D_DIFFERENCE,
   SEND_A2D_LOOPTIME,
   SEND_DELTA_SIG_RESOLUTION,
   SEND_DELTA_SIG_EVENT,
   SEND_PASSWORD,
   UPDATE_A2D_TRIGGER,
   UPDATE_A2D_DIFFERENCE,
   UPDATE_A2D_LOOPTIME,
   UPDATE_DELTA_SIG_RESOLUTION,
   UPDATE_DELTA_SIG_EVENT
   };

//////////////////////////
// Delta Sigma defines //
//////////////////////////
#define  _8BIT                256
#define  _9BIT                512
#define  _10BIT               1024
#define  _11BIT               2048
#define  _12BIT               4096
#define  _13BIT               8192
#define  _14BIT               16384
#define  _15BIT               32768U

#define  _700US               27    // 27*26us   == 702us
#define  _2100US              81    // 27*3*26us == 2106us
#define  _4000US              155   // 155*26us  == 4030us

////////////////
// prototypes //
////////////////
void Send_IR_Byte( unsigned char );
void Send_IR_CommsPacket( unsigned char, unsigned char );
void Send_0( void );
```

```
void Send_1( void );
void StartPulse( void );
void StopPulse( void );
void _38KHzBurstOnTime( unsigned char );
void _38KHzBurstOffTime( unsigned char );
#endif
```

```
[REMOTE:ireg.s]
; INTEGER EXTENSION
; Copyright (c) 1995 by COSMIC Software
;
  switch  .ubsct
  xdef  c_reg
;
c_reg:
  ds.b  1
;
  end
```

```
[REMOTE:lcd.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  lcd.c                                                        //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908GP32                                                  //
////////////////////  File Contents  ///////////////////////////////////////////
//  lcd read/write routines                                                   //
```

```
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                         //
//  ---  -------- ------  ----------------------------------------------------//
//  001  21/08/00 jt      creation                                            //
//       08/10/00 jt      Added BUSY FLAG polling in  'InstructionRegWrite()'.//
//                        During password enter mode, very occasionally the   //
//                        cursor would move to a wrong position. I haven't    //
//                        seen this event as yet with BUSY FLAG polling method//
//                        Previous, I had a 51us delay to encompass the LCD   //
//                        instruction write. The spec claims 40us.            //
////////////////////////////////////////////////////////////////////////////////
#include    <string.h>
#include    "extern.h"
#include    "delay.h"
#include    "lcd.h"



void InitialiseLCD( unsigned char options )
{
InstructionRegWrite( 0x38 ); // FUNCTION SET   : 8bit, dual line display
InstructionRegWrite( (unsigned char)(0x0c|options) ); // DISPLAY ON/OFF :
//                                                    display on,cursor off,blink off
InstructionRegWrite( 0x06 ); // ENTRY MODE     : display increment no shift
InstructionRegWrite( 0x14 ); // DISPLAY CURSOR SHIFT : move cursor right
WriteText2( LINE1_2, "", PRECLEAR);
}  //end of InitialiseTextLCD()
//------------------------------------------------------------------------------


void LcdOff( void )
{
InstructionRegWrite( 0x08 ); // DISPLAY OFF
}
//------------------------------------------------------------------------------


void WriteChar( unsigned char value )
{
DataRegWrite( value );
}  // end of WriteChar()
//------------------------------------------------------------------------------


void InstructionRegWrite( unsigned char value )
{
unsigned char  temp;
unsigned char  breakout;



temp = (unsigned char)(value>>5);   // upper three data bits for 8 bit bus
EN = 0;           // re-affirmation
RS = 0;           // access instruction reg
RW = 0;           // write
```

```
NOP();NOP();      // ensure setup time
EN = 1;           // enable write process
PTC.reg = value;  // lcd data 0:4
PTD.reg = temp;   // lcd data 5:7
NOP();NOP();NOP();//
NOP();NOP();NOP();//
NOP();NOP();      // ensure data setup time
EN = 0;           // disable write process
RW = 1;           // write complete

//////////////////////
// busy flag polling //
//////////////////////
breakout = 0;

ServiceWatchDog();

/*

// Not using for now, reverting back to an inline delay...BUSY flag
// polling seemed to cause some problems... jt

do {
   DDRD.bit.bit2 = 0;      // input to read BF
   RS = 0;                 // access the...
   RW = 1;                 //             ...busy flag
   NOP();NOP();            // ensure setup time
   EN = 1;                 // do it
   NOP();NOP();NOP();NOP();// 2us@8MHz bus, spec requires 1us
   NOP();NOP();NOP();NOP();//
   NOP();NOP();NOP();NOP();//
   NOP();NOP();NOP();NOP();//
   EN = 0;                 // try it

   if ( ++breakout >= 200 )
      {
      break;   // lcd problems here!
      }
   }  while(PTD.bit.bit2); // wait to go lo
*/
Delay(_100US); // inline dleay to replace above

DDRD.bit.bit2 = 1;   // default
}  // InstructionRegWrite()
//-------------------------------------------------------------------------


void DataRegWrite( unsigned char value )
{
unsigned char  temp;
unsigned char  breakout;



temp    = (unsigned char)(value>>5);// upper three data bits for 8 bit bus
```

```
EN      = 0;      // re-affirmation
RS      = 1;      // access data register
RW      = 0;      // write
NOP();NOP();      // ensure setup time
EN      = 1;      // enable write process
PTC.reg = value;  // lcd data 0:4
PTD.reg = temp;   // lcd data 5:7
NOP();NOP();NOP();//
NOP();NOP();NOP();//
NOP();NOP();      // ensure data setup time
EN      = 0;      // disable write
RW      = 1;      // write complete

Delay(_100US);
}  // DataRegWrite()
//---------------------------------------------------------------------------


void SetCursorAddress( unsigned char value )
{
InstructionRegWrite( (unsigned char)(0x80|value) );
}  // SetCursorAddress()
//---------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////
// This function 'string' ie 'text_buffer' is filled before this function   //
// call. 'text_buffer' is mainly filled using 'strcpy()', although on a few //
// occasions I perform a manual copy as the string consists of a single     //
// character.                                                               //
////////////////////////////////////////////////////////////////////////
void WriteText1( unsigned char address )
{
unsigned char  ii;
unsigned char  length;


SetCursorAddress(address);  // write start position
length = (unsigned char)strlen(&text_buffer[0]);

///////////////
// write it! //
///////////////
for ( ii = 0; ii < length; ii++ )
   {
   WriteChar(text_buffer[ii]);
   }
}  // WriteText1()
//---------------------------------------------------------------------------


void WriteText2( unsigned char address, char *ptr, unsigned char clear )
{
unsigned char  ii;
unsigned char  length;
```

```
////////////////////////////////////////////////////////////////////////////
// if there are other characters currently on the screen that need to be      //
// erased prior to this write (because this string will not overwrite them)   //
// then we need to clear (by writing ' ' (0x20) ) before writing 'ptr'        //
////////////////////////////////////////////////////////////////////////////
if ( clear == PRECLEAR )
   {
   memset( &text_buffer[0], ' ', sizeof(text_buffer) );  // clear buffer

   if ( address < 0x10 )      // first line cursor address is : $00...$0f
      {                       // from lcd spec
      SetCursorAddress(LINE1);    for ( ii = 0; ii < 16; ii++ ) WriteChar(' ');
      }
   else  // address >= 0x10
      {
      if ( address < 0x50 )   // 2nd line cursor address is 0x40...$4f
         {                    // from lcd spec
         SetCursorAddress(LINE2); for ( ii = 0; ii < 16; ii++ ) WriteChar(' ');
         }
      else                    // must be both lines
         {
         SetCursorAddress(LINE1); for ( ii = 0; ii < 16; ii++ ) WriteChar(' ');
         SetCursorAddress(LINE2); for ( ii = 0; ii < 16; ii++ ) WriteChar(' ');
         }
      }
   }

length = (unsigned char)strlen(ptr);

////////////////////////////////////////////////////////////////
// does the cursor address either LINE1 or LINE2 AND is the string //
// length non zero, if so then we have a valid string to write     //
////////////////////////////////////////////////////////////////
if ( address < 0x50 && length )
   {
   SetCursorAddress(address);  // set write start position

   ///////////////
   // write it! //
   ///////////////
   for ( ii = 0; ii < length; ii++ )
      {
      WriteChar( *ptr++ );
      }
   }
}  // WriteText2()
//---------------------------------------------------------------------------
```

```
[REMOTE:lcd.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  lcd.h                                                        //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for 'lcd.c'                                                   //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  21/08/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef         __LCD_H_
#define         __LCD_H_


////////////////////////////////////////////
// port defines for serial shifting of address //
////////////////////////////////////////////
#define RS                      PTB.bit.bit5
#define RW                      PTB.bit.bit6
#define EN                      PTB.bit.bit7

#define NOBLINK                 0x00
#define BLINK                   0x01
#define NOUNDERLINE_CURSOR      0x00
#define UNDERLINE_CURSOR        0x02

enum  { LINE1, LINE2=0x40, LINE1_2=0x80 };
enum  { NOPRECLEAR, PRECLEAR };
```

```
////////////////
// prototypes //
////////////////
void InitialiseLCD( unsigned char );
void LcdOff( void );
void WriteChar( unsigned char );
void SetCursorAddress( unsigned char );
void InstructionRegWrite( unsigned char );
void DataRegWrite( unsigned char );
void WriteText1( unsigned char );
void WriteText2( unsigned char, char *, unsigned char );

#endif
```

```
[REMOTE:link08.bat]
@echo off
c:\cosmic\cx08\clnk  -v  -m gp32.inf -e gp32.err -o remote.h08 gp32.lkf
c:\cosmic\cx08\chex  -fm -o remote.s19 remote.h08
c:\cosmic\cx08\clabs -l  -v remote.h08
```

```
[REMOTE:lreg.s]
; LONG/FLOAT ACCUMULATOR
; Copyright (c) 1995 by COSMIC Software
;
  switch  .ubsct
  xdef  c_lreg
;
c_lreg:
  ds.b  4
;
  end
```

```
[REMOTE:main.c]
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   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     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Durham                                                                   //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control        //
//  Filename  :  main.c                                                     //
//  Compiler  :  Cosmic ANSI-C                                              //
//  CPU       :  MC68HC908GP32                                              //
////////////////////  File Contents  ///////////////////////////////////////
//  'main' routine                                                          //
//////////////////////////  Update Information  ////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  19/06/00 jt      creation                                          //
//       07/10/00 jt      Code nearing completion.                          //
//                        Changed startup code to decrease current drain    //
//                        before entering STOP mode. Checked '*.la' files.  //
//                        Checksum: $B29A2 (archived to : $b29a2.zip)       //
//       07/10/00 jt      Removed unused flag definitions and some general  //
//                        tidy-ups.                                         //
//                        Checksum: $B2A17                                  //
//       08/10/00 jt      Added BUSY FLAG polling in  'InstructionRegWrite()' //
//                        in 'lcd.c'.                                       //
//                        Checksum: $BAFBC                                  //
//       20/10/00 jt      Tidy up's                                         //
//                        Checksum: $B9FFF                                  //
//       30/10/00 jt      Tidy up's                                         //
//                        Checksum: $B7ABD                                  //
//       16/11/00 jt      Button decodes now same as pdf documentation      //
//                        Checksum: $BEE4C                                  //
//       20/11/00 jt      Full cycle check on 38kHz IR comms timings        //
//                        checksum: $C3097                                  //
//       22/11/00 jt      Functional tweaks                                 //
//                        v1.0                                              //
//                        FIRST RELEASE TO MOTOROLA                         //
//                        checksum: $C15E5                                  //
//       05/12/00 jt      Adjusted 'nop' count in 'SetUpAndHoldTimingDelay()' //
```

```
//                      in 'i2c.c', the 'nop' count needed reducing due to  //
//                      the reduced 2.4576MHZ bus, (was 4.9152MHz).         //
//                      Dugald Campbell of Motorola spotted some lcd string //
//                      anomalies, fixed them.                              //
//                      checksum: $C3FED                                    //
//       23/01/01 jt    Improved ir comms robustness to noise with changes  //
//                      in 'interrup.c->TIMER1CHANNEL0'.                    //
//                      checksum: $DCF93                                    //
//       07/02/01 jt                                                        //
//                      v1.1                                                //
//                      SECOND RELEASE TO MOTOROLA                          //
//                      checksum: $DCE17                                    //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "startup.h"
#include    "button.h"
#include    "datasort.h"
#include    "mode.h"



////////////
// main() //
////////////
void main( void )
{
MicroStartUp();

while( 1 )
   {
   STOP();
   ServiceWatchDog();

   ////////////////////////////////////////////////////
   // by virtue of the fact that we have got to this part of //
   // the software a KEYBOARD interrupt must have 'fired'.   //
   ////////////////////////////////////////////////////
   Initialise908GP32();

   if ( GetPirPassword() )
      {
      do {
         ServiceWatchDog();
         ReadButtons();
         IRCommsCheck();
         RS232CommsCheck();
         ModeCheck();

         ////////////////////////
         // 10ms do-while sync //
         ////////////////////////
         while ( !flags1.bit._10MS_LOOP );
         flags1.bit._10MS_LOOP = 0;
         }  while ( ++stop_counter < _5MINUTE );
      }
```

```
   ////////////////////////////////////////////////////////////////////////
   // there must have been 5mins of no button activity OR a password problem //
   ////////////////////////////////////////////////////////////////////////
   PrepareForSTOP();
   }
}  // main()
//---------------------------------------------------------------------------
```

```
[REMOTE:make08.bat]
@echo off

rem/////////////////////////////////
rem// rebuilding crtsi.s startup file //
rem/////////////////////////////////
c:\cosmic\cx08\ca6808 crtsi.s
c:\cosmic\cx08\ca6808 ireg.s
c:\cosmic\cx08\ca6808 lreg.s

rem//////////////////////////////
rem// compile all source files //
rem//////////////////////////////
call cc button
call cc convert
call cc data
call cc delay
call cc datasort
call cc digipot
call cc error
call cc interrup
call cc ir_comms
call cc i2c
call cc lcd
call cc main
call cc mode
call cc rs_comms
call cc rtc
call cc startup
call cc vectors

rem//////////////////////////
rem// link the object files //
rem//////////////////////////
call link08

rem////////////////////////////////
rem// deleting relative listings //
rem////////////////////////////////
del *.ls

dir *.err
```

```
[REMOTE:mode.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                                                      //
// Durham                                                                   //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control        //
//  Filename  :  mode.c                                                     //
//  Author    :  jtravers                                                   //
//  Compiler  :  Cosmic HC08                                                //
//  CPU       :  MC68HC908GP32                                              //
////////////////////  File Contents  ////////////////////////////////////////
//  lcd screen mode functionality                                           //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/09/00 jt      creation                                          //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "ir_comms.h"
#include    "datasort.h"
#include    "lcd.h"
#include    "error.h"
#include    "delay.h"
#include    "startup.h"
#include    "rtc.h"
#include    "convert.h"
#include    "mode.h"



unsigned char GetPirPassword( void )
{
unsigned char  retry_count = 0;


mode = MODE_WAITING_FOR_PIR_PASSWORD;
```

```
/////////////////
// prepare LCD //
/////////////////
InitialiseLCD(NOBLINK|NOUNDERLINE_CURSOR);
WriteText2( LINE1, "Transmitting IR ", NOPRECLEAR );
WriteText2( LINE2, "comms packets   "  , NOPRECLEAR );

do {
   ServiceWatchDog();
   Send_IR_CommsPacket(SEND_PASSWORD, 0);    // xmit ir comms packet
   Delay10ms(30);                            // 300ms  between transmissions
   IRCommsCheck();                           // check for appropriate reply
   UpdateDots(retry_count);                  // write the progress '.' dots

   if ( ++retry_count > 40 )                 // 40*0.3s == 12s of attempts
      {
      if ( !flags1.bit.IR_ACTIVITY )
         {
         ErrorCondition(ERROR_NO_IR_COMMS);  // error message
         }
      else
         {
         ErrorCondition(ERROR_NO_PASSWORD);  // error message
         }

      return 0;                              // back to STOP mode
      }
   }  while ( mode == MODE_WAITING_FOR_PIR_PASSWORD );

//////////////////////////////////////////////////
// OK, password value received from PIR unit. Now the //
// user needs to type in a matching one               //
//////////////////////////////////////////////////
PasswordEntryScreen();

return 1;
}  // GetPirPassword()
//---------------------------------------------------------------------------


void UpdateDots( unsigned char value )
{
////////////////////////////////////////////////////////////
// using instead of 'strcpy(&text_buffer[0], ".")' since there is //
// only 1 character to this string                          //
////////////////////////////////////////////////////////////
text_buffer[0] = '.';text_buffer[1] = '\0';

switch ( value )
   {
   case 12 :
   WriteText1( LINE2+13 );
   break;

   case 24 :
```

```
   WriteText1( LINE2+14 );
   break;

   case 36 :
   WriteText1( LINE2+15 );
   break;
   }
}  // UpdateDots()
//-------------------------------------------------------------------------


void PasswordEntryScreen( void )
{
InitialiseLCD(BLINK|NOUNDERLINE_CURSOR);
WriteText2( LINE1, "Enter password: " , NOPRECLEAR );
WriteText2( LINE2, "     XXXXX      " , NOPRECLEAR );
SetCursorAddress(LINE2+5);
character_count          = 0;
flags1.bit.PASSWORD_WRAP = 0;
mode                     = MODE_USER_ENTER_PASSWORD;
}  // PasswordEntryScreen()
//-------------------------------------------------------------------------


void PrepareForSTOP( void )
{
InitialiseKeyboardInt();
InitialiseLCD(NOBLINK|NOUNDERLINE_CURSOR);
WriteText2(LINE1_2, "", PRECLEAR);
LcdOff();
DDRB.reg    = 0x00;        // all input
DDRC.reg    = 0x00;        //          to minimise
DDRD.reg    = 0x00;        //                      current drain
ServiceWatchDog();
}  // PrepareForSTOP()
//-------------------------------------------------------------------------


void ShuttingDown( void )
{
if ( shut_down_ii == 0 && shut_down_jj == 0 )   //
   {
   InitialiseLCD(NOBLINK|NOUNDERLINE_CURSOR);
   WriteText2( LINE1, "Shutting down..." , NOPRECLEAR );
   }

////////////////////////////////////
// show progress dots increasing... //
////////////////////////////////////
if ( ++shut_down_jj == 10 ) // 10*10ms == 100ms between dot writes
   {
   shut_down_jj = 0;
```

```
   ////////////////////////////////////////////////
   // show the 16 dots moving increasing along display //
   ////////////////////////////////////////////////
   if ( shut_down_ii < 16 )
      {
      WriteText2( (unsigned char)(LINE2+shut_down_ii++), ".", NOPRECLEAR );
      }
   }
}  // ShuttingDown()
//-------------------------------------------------------------------------------


void ModeCheck( void )
{
switch ( mode )
   {
   case MODE_TIME_OF_DAY :
   UpdateTime();
   break;

   case MODE_SHUTTING_DOWN :           ////////////////////////////////////////
   ShuttingDown();                     // The time between increasing dots is //
   break;                              // 100ms. We draw 16 of them in total. //
   }                                   ////////////////////////////////////////
                                       // 180 == 180*10ms == 1.8s             //
                                       // the 16 dot draws will take 100ms*16 //
// start shutting down @ 58.2s         // == 160ms.                           //
if ( stop_counter == (_5MINUTE-180) )  // '180' not '160' for 100ms to show   //
   {                                   // the last printed dot                //
   WriteText2( LINE1_2, "", PRECLEAR); ////////////////////////////////////////
   mode_copy = mode; // store mode before shut down execution
   mode      = MODE_SHUTTING_DOWN;
   }
}  // ModeCheck()
//-------------------------------------------------------------------------------


void RedrawFlashParameterScreen( void )
{
switch ( mode )
   {
   case MODE_EDIT_A2D_TRIGGER :
   WriteText2( LINE1, "A2D Trigger:    " , NOPRECLEAR );
   WriteText2( LINE2, "", PRECLEAR);
   IntegerToASCII( adjust_value._8bit.lobyte, &text_buffer[7]  );
   WriteText1(LINE2);
   break;

   case MODE_EDIT_A2D_DIFFERENCE :
   WriteText2( LINE1, "A2D Difference: " , NOPRECLEAR );
   WriteText2( LINE2, "", PRECLEAR);
   IntegerToASCII( adjust_value._8bit.lobyte, &text_buffer[7]  );
   WriteText1(LINE2);
   break;
```

```
   case MODE_EDIT_A2D_LOOPTIME :
   WriteText2( LINE1, "A2D Loop Time:  " , NOPRECLEAR );
   WriteText2( LINE2, "", PRECLEAR);
   IntegerToASCII( adjust_value._8bit.lobyte, &text_buffer[7]  );
   WriteText1(LINE2);
   break;

   case MODE_EDIT_DELTA_SIG_RESOLUTION :
   WriteText2( LINE1, "Delta Sig Res'n:" , NOPRECLEAR );
   WriteText2( LINE2, "", PRECLEAR);
   IntegerToASCII( adjust_value._8bit.lobyte, &text_buffer[7]  );
   WriteText1(LINE2);
   break;

   case MODE_EDIT_DELTA_SIG_EVENT :
   WriteText2( LINE1, "Delta Sig Event:" , NOPRECLEAR );
   WriteText2( LINE2, "", PRECLEAR);
   IntegerToASCII( adjust_value._16bit, &text_buffer[7]  );
   WriteText1(LINE2);
   break;

   case MODE_LCD_CONTRAST_ADJUST :
   WriteText2(LINE1_2, "", PRECLEAR);
   WriteText2(LINE1, "Screen Contrast", NOPRECLEAR);
   WriteText2(LINE2, "Use INC/DEC"    , NOPRECLEAR);
   break;

   case MODE_TIME_OF_DAY :
   WriteText2(LINE1_2, "", PRECLEAR);  // clear whole screen prior to TOD
   break;
   }
}  // RedrawFlashParameterScreen();
//---------------------------------------------------------------------------
```

```
[REMOTE:mode.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  mode.h                                                       //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for 'mode.c'                                                  //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/09/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef     __MODE_H_
#define     __MODE_H_



enum  {
      MODE_WAITING_FOR_PIR_PASSWORD = 0x01,
      MODE_USER_ENTER_PASSWORD,
      MODE_EDIT_A2D_TRIGGER,
      MODE_EDIT_A2D_DIFFERENCE,
      MODE_EDIT_A2D_LOOPTIME,
      MODE_EDIT_DELTA_SIG_RESOLUTION,
      MODE_EDIT_DELTA_SIG_EVENT,
      MODE_TIME_OF_DAY,
      MODE_SHUTTING_DOWN,
      MODE_LCD_CONTRAST_ADJUST
      };
```

```
///////////////
// prototypes //
///////////////
unsigned char GetPirPassword( void );
void UpdateDots( unsigned char );
void PasswordEntryScreen( void );
void PrepareForSTOP( void );
void ShuttingDown( void );
void ModeCheck( void );
void RedrawFlashParameterScreen( void );
#endif
```

```
[REMOTE:rs_comms.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                                                      //
// Durham                                                                   //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control        //
//  Filename  :  rs_comms.c                                                 //
//  Author    :  jtravers                                                   //
//  Compiler  :  Cosmic ANSI-C                                              //
//  CPU       :  68HC908GP32                                                //
////////////////////  File Contents  ////////////////////////////////////////
//  RS232 routines                                                          //
//////////////////////////  Update Information  ///////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/09/00 jt      creation                                          //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "rs_comms.h"
```

```
void InitialiseRS232( void )
{
SCC1.reg = 0x40;     // ENSCI set, 8 data, 1 start, 1 stop
SCC2.reg = 0x2c;     // rx interrupts and receiver/transmitter enabled
SCC3.reg = 0x00;     //
SCBR.reg = _38400;   //
}  // InitialiseRS232()
//-------------------------------------------------------------------------------


void Send_RS232_Byte( unsigned char data )
{
/*-----------------02/09/00 22:44-------------------
 * there are subtle differences between the two methods shown below.
 *
 * The 'SCTE' flag is set when the data has been transferred to the
 * transmit shift register (NOTE: it has not necessarily been sent)
 *
 * The 'TC' is set after 'SCTE' has been set and after the
 * data has been transmitted.
 * --------------------------------------------------*/

/*
unsigned char  temp;

////////////////////
// using SCTE flag //
////////////////////
temp = SCS1.reg;  // force clear of SCTE
SCDR.reg = data;
while ( !SCS1.bit.SCTE );  // wait for data to be transferred
*/
//////////////////
// using TC flag //
//////////////////
SCDR.reg = data;           // 'TC' automatically cleared by this
while ( !SCS1.bit.TC );    // wait while transmission in progress
}  // Send_RS2232_Byte()
//-------------------------------------------------------------------------------


unsigned char Get_RS232_Byte( void  )
{
while ( !SCS1.bit.SCRF );
return SCDR.reg;
}  // Get_RS232_Byte()
//-------------------------------------------------------------------------------
```

```
        ////////////////////////////
        // COMMS PACKET STRUCTURE //
        ////////////////////////////
//////////////////////////////////////////////
// rs232_buffer[0]   == block length byte   //
// rs232_buffer[1]   == block title byte    //
// rs232_buffer[2]   == data byte 1         //
// rs232_buffer[n]   == data byte 'n'       //
// rs232_buffer[n+1] == hibyte checksum     //
// rs232_buffer[n+2] == lobyte checksum     //
//                                          //
// Block length is the number of bytes in   //
// the block, EXCLUDING the checksum.       //
//                                          //
// Checksum is the 16 bit total of the      //
// block, EXCLUDING the checksum.           //
//////////////////////////////////////////////
void Send_RS232_CommsPacket( unsigned char block_title,
                                                    unsigned char block_length )
{
union uUNSIGNED_INTEGER checksum;
unsigned char           ii;

ServiceWatchDog();

block_length += 2; // add inherent BLOCK_LENGTH/BLOCK_TITLE bytes to block size

//////////////////////////////////////////////////////////
// insert the element values into the 'rs232_buffer' array //
//////////////////////////////////////////////////////////
rs232_buffer[0] = block_length;
rs232_buffer[1] = block_title;

///////////////////////////////////
// calculate the packet checksum //
///////////////////////////////////
checksum._16bit = 0;
for ( ii = 0; ii < block_length; ii++ )
   {
   checksum._16bit += rs232_buffer[ii];
   }

//////////////////////
// append checksum //
//////////////////////
rs232_buffer[block_length  ] = checksum._8bit.hibyte;
rs232_buffer[block_length+1] = checksum._8bit.lobyte;

//////////////////////////////////////////////////////////////////////
// the complete block consista of:-                                 //
//                                                                  //
// block length + block title + n*data + checksum hi + checksum lo  //
//                                                                  //
// The number of bytes that we have to transmit is block_length + 2 //
//////////////////////////////////////////////////////////////////////
```

```
block_length += 2;

//////////////
// preamble //
//////////////
Send_RS232_Byte( 'A' );
Send_RS232_Byte( 'T' );
Send_RS232_Byte( 'E' );
Send_RS232_Byte( 'E' );
Send_RS232_Byte( 'C' );
Send_RS232_Byte( 'C' );

/////////////////
// xmit packet //
/////////////////
for ( ii = 0; ii < block_length; ii++ )
   {
   Send_RS232_Byte( rs232_buffer[ii] );
   }
}  // Send_RS232_CommsPacket()
//---------------------------------------------------------------------------
```

```
[REMOTE:rs_comms_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                                                      //
// Durham                                                                   //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control        //
//  Filename  :  rs_comms.h                                                 //
//  Author    :  jtravers                                                   //
//  Compiler  :  Cosmic ANSI-C                                              //
//  CPU       :  68HC908GP32                                                //
////////////////////  File Contents  ////////////////////////////////////////
//  header file for 'rs_comms.c'                                            //
```

```
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                         //
//  ---  -------- ------  -----------------------------------------------------//
//  001  01/09/00 jt      creation                                             //
////////////////////////////////////////////////////////////////////////////////
#ifndef     __RS_COMMS_H_
#define     __RS_COMMS_H_


#define     _38400   0x00


enum  // RS232 block title values
   {
   UPDATE_RTC,
   ACKNOWLEDGE = 0x55,
   NOACKNOWLEDGE
   };

void InitialiseRS232( void );
void Send_RS232_Byte( unsigned char );
unsigned char Get_RS232_Byte( void  );
void Send_RS232_CommsPacket( unsigned char, unsigned char );

#endif
```

```
[REMOTE:rtc.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  rtc.c                                                        //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic HC08                                                  //
//  CPU       :  MC68HC908GP32                                                //
```

```
////////////////////  File Contents  ////////////////////////////////////////
//  Real Time Clock routines                                                //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/09/00 jt      creation                                          //
////////////////////////////////////////////////////////////////////////////////
#include    <string.h>
#include    "extern.h"
#include    "lcd.h"
#include    "i2c.h"
#include    "convert.h"
#include    "mode.h"
#include    "delay.h"
#include    "rtc.h"




void UpdateTime( void )
{
static @tiny unsigned char    seconds_compare = 0;
struct RTC                    current_time;


if ( mode == MODE_TIME_OF_DAY )
   {
   ServiceWatchDog();

   ////////////////////////
   // get the current time //
   ////////////////////////
   RTC_Read(SECONDS, &current_time);

   ////////////////////////////////////////////////////
   // write it to the screen, only if the time has changed //
   ////////////////////////////////////////////////////
   if ( seconds_compare != current_time.seconds )
      {
      memset( &text_buffer[0], ' ', sizeof(text_buffer) );

      ////////////////////////////////////////////
      // what day is it, occupies text_buffer[0][1][2] //
      ////////////////////////////////////////////
      if ( current_time.day >= 1 && current_time.day <= 7 )
         {
         strcpy( &text_buffer[0], &days_of_week[current_time.day][0] );
         }
      else
         {
         strcpy( &text_buffer[0], &days_of_week[8][0] ); // "XXX" error read
         }
```

```
   ////////////////////
   // what date is it //
   ////////////////////
   text_buffer[3] = ' ';
   HexToASCII( current_time.date, &text_buffer[4] );  // uses elements [4][5]
   text_buffer[6] = ' ';

   ////////////////////////////////////////////////
   // what month is it, occupies text_buffer[7][8][9] //
   ////////////////////////////////////////////////
   HexToDec( &current_time.month );

   if ( current_time.month >= 1 && current_time.month <= 12 )
      {
      strcpy( &text_buffer[7], &months_of_year[current_time.month][0] );
      }
   else
      {
      strcpy( &text_buffer[7], &months_of_year[13][0] ); // "XXX" error read
      }

   ////////////////////////////////////////////////////////
   // century hi:lo bytes occupt text_buffer[11][12][13][14] //
   ////////////////////////////////////////////////////////
   text_buffer[10] = ' ';
   HexToASCII( current_time.year._8bit.hibyte, &text_buffer[11]  );  // fixed
   HexToASCII( current_time.year._8bit.lobyte, &text_buffer[13]  );
   text_buffer[15] = '\0';

   ////////////////////////////////////////////
   // all formatting complete, write first line //
   ////////////////////////////////////////////
   WriteText1( LINE1 );

   /////////////////
   // second line //
   /////////////////
   HexToASCII( current_time.hours,   &text_buffer[0] );// text_buffer[0][1]
   text_buffer[2] = ':';
   HexToASCII( current_time.minutes, &text_buffer[3] );// text_buffer[3][4]
   text_buffer[5] = ':';
   HexToASCII( current_time.seconds, &text_buffer[6] );// text_buffer[6][7]
   text_buffer[8] = '\0';

   //////////////
   // write it //
   //////////////
   WriteText1( LINE2 + 4 );

   ////////////////////////////////
   // update for next comparison //
   ////////////////////////////////
   seconds_compare = current_time.seconds;
   }
}
```

```
}  // UpdateTime()
//---------------------------------------------------------------------------


unsigned char SetRTC( struct RTC *ptr )
{
struct RTC     compare;
unsigned char  error_count;

StartBit();                                 ////////////////////////////////
SendI2CByte( RTC_WRITE );                   // RTC_WRITE == 0xd0          //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( SECONDS );                     // point to seconds register  //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->seconds );                // seconds                    //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->minutes );                // minutes                    //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->hours );                  // hours                      //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->day );                    // day                        //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->date );                   // date                       //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->month );                  // month                      //
WaitForI2CAcknowledge();                    //                            //
SendI2CByte( ptr->year._8bit.lobyte );      // year                       //
WaitForI2CAcknowledge();                    //                            //
StopBit();                                  ////////////////////////////////

////////////////////////////////////
// now to read what's been written //
////////////////////////////////////
RTC_Read( SECONDS, &compare );

error_count = 0;

if ( compare.year._8bit.lobyte != ptr->year._8bit.lobyte )  error_count++;
if ( compare.month             != ptr->month             )  error_count++;
if ( compare.date              != ptr->date              )  error_count++;
if ( compare.day               != ptr->day               )  error_count++;
if ( compare.hours             != ptr->hours             )  error_count++;
if ( compare.minutes           != ptr->minutes           )  error_count++;
if ( compare.seconds           != ptr->seconds           )  error_count++;

if ( !error_count )
   {
   return 1;   // success
   }

return 0;      // failed
}  // SetRTC()
//---------------------------------------------------------------------------
```

```
void RTC_Read( unsigned char register_pointer, struct RTC *ptr )
{
////////////////////////////////////////////
// first set the internal RTC address pointer //
// to the register that you require with a    //
// WRITE command                              //
////////////////////////////////////////////
StartBit();
SendI2CByte( RTC_WRITE );
WaitForI2CAcknowledge();
SendI2CByte( register_pointer );
WaitForI2CAcknowledge();
StopBit();

////////////////////////////////////
// Then read the contents of the RTC //
// registers, with a READ command    //
////////////////////////////////////
StartBit();
SendI2CByte( RTC_READ );              // RTC_READ == 0xd1
WaitForI2CAcknowledge();
ptr->seconds = GetI2CByte();
SendI2CAcknowledge();
ptr->minutes = GetI2CByte();
SendI2CAcknowledge();
ptr->hours   = GetI2CByte();
SendI2CAcknowledge();
ptr->day     = GetI2CByte();
SendI2CAcknowledge();
ptr->date    = GetI2CByte();
SendI2CAcknowledge();
ptr->month   = GetI2CByte();
SendI2CAcknowledge();
ptr->year._8bit.lobyte = GetI2CByte();
ptr->year._8bit.hibyte = 0x20;        // century hi byte...fixed for the
                                      // next 99 years!

SET_DATA_TO_OUTPUT;                   // master sending a NOT ACK
SET_SDA;
OutClock();       // no acknowledge expected here, we generate a clock pulse
StopBit();
}  // RTC_Read()
//-----------------------------------------------------------------------------


////////////////////////////
// load RTC with known data //
////////////////////////////
void ForceRTC( void )
{
struct RTC  force_rtc;


if ( mode == MODE_TIME_OF_DAY )
   {
```

```
   force_rtc.seconds           = 0x00;
   force_rtc.minutes           = 0x00;
   force_rtc.hours             = 0x00;
   force_rtc.day               = 0x01;
   force_rtc.date              = 0x01;
   force_rtc.month             = 0x01;
   force_rtc.year._8bit.hibyte = 0x20;
   force_rtc.year._8bit.lobyte = 0x01;

   WriteText2( LINE1, "  RTC Override  ", NOPRECLEAR );

   if ( SetRTC( &force_rtc ) )   // write it!
      {
      WriteText2( LINE2, "  Successful!   ", NOPRECLEAR );
      }
   else
      {
      WriteText2( LINE2, "Failed,Try Again", NOPRECLEAR );
      }

   Delay10ms(_1S);                     // show message for 1s
   WriteText2(LINE1_2, "", PRECLEAR);  // clear whole screen prior to TOD
   }
}  // ForceRTC()
//---------------------------------------------------------------------------
```

```
[REMOTE:rtc.h]
////////////////////////////////////////////////////////////////////////////
//       AA      TTTTTTTTTTTT EEEEEEEEEE EEEEEEEEEE   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     EEEEEEEEEE EEEEEEEEEE   CCCCCCCCC  CCCCCCCCC  //
////////////////////////////////////////////////////////////////////////////
// AT Electronic Embedded Control Consultants                               //
// Unit 32, Consett Business Park                                           //
// Villa Real, Consett                                                      //
// Durham                                                                   //
// DH8 6BP                                                                  //
// England                                                                  //
//                                                                          //
// Telephone: 0044 1207 693920                                              //
// Fax      : 0044 1207 693921                                              //
// email    : enquiries@ateecc.com                                          //
// web      : www.ateecc.com                                                //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control        //
//  Filename  :  rtc.h                                                      //
//  Author    :  jtravers                                                   //
//  Compiler  :  Cosmic HC08                                                //
//  CPU       :  MC68HC908GP32                                              //
```

```
////////////////////  File Contents  ////////////////////////////////////////
//  header file for rtc.c                                                   //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                      //
//  ---  -------- ------  ----------------------------------------------------//
//  001  01/09/00 jt      creation                                          //
////////////////////////////////////////////////////////////////////////////////
#ifndef     __RTC_H_
#define     __RTC_H_

#include    "declared.h"


enum
   {
   RTC_WRITE = 0xd0,
   RTC_READ
   };
enum
   {
   SECONDS = 0x00,
   MINUTES,
   HOURS,
   DAY,
   DATE,
   MONTH,
   YEAR
   };

////////////////
// prototypes //
////////////////
void           UpdateTime( void );
unsigned char  SetRTC( struct RTC * );
void           RTC_Read( unsigned char, struct RTC * );
void           ForceRTC( void );

#endif
```

```
[REMOTE: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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  startup.c                                                    //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908JK1                                                   //
////////////////////  File Contents  ///////////////////////////////////////
//  startup routines                                                          //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  19/06/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "extern.h"
#include    "lcd.h"
#include    "button.h"
#include    "delay.h"
#include    "startup.h"



void MicroStartUp( void )
{                       ////////////////////////////////////////////////////
CONFIG1.reg = 0x0a;     // COP time out (2^18-2^24)*1/Fosc, LVI enabled (5V), //
                        // STOP instruction enabled, watchdog enabled        //
ServiceWatchDog();      //                                                   //
CONFIG2.reg = 0x01;     // oscillator off in STOP, bit rate from internal bus //
INTSCR.reg  = 0x02;     // IRQ interrupts disabled                           //
SEI();                  // re-affirm                                         //
                        ////////////////////////////////////////////////////
InitialisePLL();
```

```
////////////////////////////////////////////
// assign data direction for LCD activity only //
////////////////////////////////////////////
PTB.reg  = 0x00;
DDRB.reg = 0xe0;
PTC.reg  = 0x00;
DDRC.reg = 0xff;
PTD.reg  = 0x00;
DDRD.reg = 0x07;

////////////////////////////////////////////
// ensure LCD is off to minimise current drain //
////////////////////////////////////////////
InitialiseLCD(NOBLINK|NOUNDERLINE_CURSOR);
WriteText2(LINE1_2, "", PRECLEAR);
LcdOff();
                        ////////////////////////////////////////////////////
InitialiseKeyboardInt();  // porta setup, to enable us to recover from STOP  //
                        // ready for keyboard interrupt to bring micro out //
                        // of STOP mode                                    //
DDRB.reg    = 0x00;     // all input                                       //
DDRC.reg    = 0x00;     //          to minimise                            //
DDRD.reg    = 0x00;     //                      current drain              //
CLI();                  // ready for STOP mode recovery via KEYBOARD int   //
                        ////////////////////////////////////////////////////
}   // MicroStartUp()
//---------------------------------------------------------------------------


void InitialisePLL( void )
{                       ////////////////////////////////////////
PBWC.reg       = 0x80;  // auto mode                           //
PCTL.reg       = 0x02;  // settings here...                    //
PMS            = 0x012c;// as described in...                 //
PMRS.reg       = 0x80;  // the MC68HC908GP32/H                 //
PMDS.reg       = 0x01;  // Rev2.0 data book section 7.4.6 page 120 //
PCTL.bit.PLLON = 1;     // turn pll on after settings 'set'    //
                        ////////////////////////////////////////

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

PCTL.bit.BCS = 1;       // pll clock drives CGMOUT

//////////////////////////////////////////////////////////////////
// bus frequency is 2.4576MHz, this produces a watchdog timeout of: //
// (2^18-2^4)*1/2.4576E6 == (262144 - 16)/2.4576E6 => 106.66ms      //
//////////////////////////////////////////////////////////////////
}  // InitialisePLL()
//---------------------------------------------------------------------------
```

```
//////////////////////////////////////////////////////
// keyboard interrupt setup, as per page 189, section 13.5 //
// of the MC68HC908GP32 Rev2.0 Technical Data book.        //
//////////////////////////////////////////////////////
void InitialiseKeyboardInt( void )
{
ServiceWatchDog();
INTKBSCR.bit.IMASKK = 1;      // mask interrupts
INTKBIER.reg        = 0x1f;   // bits 0..4 as interrupt sources
INTKBSCR.bit.MODEK  = 0;      // falling edge active only
PTA.reg             = 0x1f;   // column drivers output...
DDRA.reg            = 0xe0;   //                        ... and low
PTAPUE.reg          = 0x1f;   // portA pullups enabled
Delay(_50US);                 // wait...before read for pin level to settle
INTKBSCR.bit.ACKK   = 1;      // clear interrupt request, if one pending
INTKBSCR.bit.IMASKK = 0;      // ready
}  // InitialiseKeyboardInt()
//---------------------------------------------------------------------------


void Initialise908GP32( void )
{
SEI();

InitialisePLL();

///////////////
// I/O Setup //
///////////////         ////////////////////////////////////////////////////////
PTB.reg     = 0x1c;     //                                                    //
DDRB.reg    = 0xff;     // bit7:lcd E, bit6:lcd RW, bit5:lcd RS,              //
                        // bit4:digipot inc, bit3:digipot up/down,            //
                        // bit2:digipot CS, bit1:RTC data, bit0:RTC Clock     //
PTC.reg     = 0x00;     //                                                    //
DDRC.reg    = 0xff;     // bit4:lcd data4, bit3:lcd data3, bit2:lcd data2     //
                        // bit1:lcd data1, bit0:lcd data0                     //
PTD.reg     = 0x00;     // ensure ir xmit off                                 //
DDRD.reg    = 0x2f;     // bit5:SPARE, bit4:IR RX, bit3:IR TX, bit2:lcd data7 //
                        // bit1:lcd data6, bit0:lcd data5                     //
                        ////////////////////////////////////////////////////////
//////////////////
// timer1 setup //
//////////////////      ////////////////////////////////////////////////////////
T1SC.reg = 0x70;        // set TOIE, stop and reset timer counter,            //
                        // timer: 1X prescaler                                //
                        // Using the PLL to provide a bus clock of 2.4576MHz  //
                        // this gives us a bus cycle period of 1/2.4576E6     //
                        // == 0.407us                                         //
                        // To obtain our 10ms timer overflow count we need a  //
                        // a timer mod value of (10E-3)/(1/2.4576E6) ==       //
                        // (10E-3)*(2.4576E6) == 24576                        //
                        // This has been defined in 'define.h'                //
T1MOD = TIMER_ROLLOVER; ////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////
// start timer1 channel0 as capture mode for +ve edges (for ir comms rx) //
////////////////////////////////////////////////////////////////////////
T1SC0.reg = 0b01000100;
//            ||||||||
//            ||||||||____  CH0MAX   100% pwm off
//            |||||||_____  TOV0     PTD4 not toggled on overflow
//            ||||||______  ELS0A    } +ve edge trigger capture
//            |||||_______  ELS0B    } -ve edge trigger capture
//            ||||________  MS0A     unbuffered compare/pwm operation on
//            |||_________  MS0B     buffered compare/pwm off
//            ||__________  CHOIE    interrupt enabled
//            |___________  CH0F     read only

T1SC1.reg = 0x00; // timer1 channel1 off
T2SC.reg  = 0x00; // mod timer2 off
T2SC0.reg = 0x00; // timer2 channel0 off
T2SC1.reg = 0x00; // timer2 channel1 off

T1SC.bit.TSTOP = 0;      // start mod timer1

////////////////////////////
// Clear all variable ram //
////////////////////////////
ClrPAGE0Ram();

//////////////////////////
// initial assignments //
//////////////////////////
button_press_status = NO_BUTTON_PRESS;
ds_adjust_ptr       = &ds_adjust[0];
ir_mode             = IR_IDLE;
flags1.byte         = 0x00;

///////////////////////////////////
// ready for interrupt processing //
///////////////////////////////////
CLI();
}  // Initialise908GP32()
//-------------------------------------------------------------------------
```

```
[REMOTE:startup.h]
////////////////////////////////////////////////////////////////////////////
//       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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  startup.h                                                    //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  68HC908GP32                                                  //
////////////////////  File Contents  ///////////////////////////////////////
//  header file for 'startup.c'                                               //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  19/06/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#ifndef  __STARTUP_H_
#define  __STARTUP_H_


////////////////
// prototypes //
////////////////
void MicroStartUp( void );
void InitialisePLL( void );
void InitialiseKeyboardInt( void );
void Initialise908GP32( void );
#endif
```

```
[REMOTE:vectors.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                                                        //
// Durham                                                                     //
// DH8 6BP                                                                    //
// England                                                                    //
//                                                                            //
// Telephone: 0044 1207 693920                                                //
// Fax      : 0044 1207 693921                                                //
// email    : enquiries@ateecc.com                                            //
// web      : www.ateecc.com                                                  //
////////////////////////////////////////////////////////////////////////////
//  Project   :  Motorola Infra Red Reference Design, Remote Control          //
//  Filename  :  vectors.c                                                    //
//  Author    :  jtravers                                                     //
//  Compiler  :  Cosmic ANSI-C                                                //
//  CPU       :  MC68HC908GP32                                                //
////////////////////  File Contents  ///////////////////////////////////////
//  vectors - an array of void pointers                                       //
//////////////////////////  Update Information  ////////////////////////////////
//  Ed.  Date     Init's  Modification                                        //
//  ---  -------- ------  ----------------------------------------------------//
//  001  19/06/00 jt      creation                                            //
////////////////////////////////////////////////////////////////////////////
#include    "define.h"              // 'NULL' defined

extern void TIMER1OVERFLOW( void );
extern void TIMER1CHANNEL0( void );
extern void KEYBOARD( void );
extern void SCI_RECEIVE( void );
extern void _stext();         // startup routine. defined by Cosmic in 'crtsi.s'


////////////////////////////////////
// an array of function pointers //
////////////////////////////////////
void (*const _vectab[18])(void) =
{
```

```
              ////////////////////////////////////
_stext,       // TIMEBASE                 $FFDC //          |
_stext,       // A2D CONVERSION COMPLETE  $FFDE //          |
KEYBOARD,     // KEYBOARD                 $FFE0 //          |
_stext,       // SCI TRANSMIT             $FFE2 //          |
SCI_RECEIVE,  // SCI RECEIVE              $FFE4 //          |
_stext,       // SCI ERROR                $FFE6 //          |
_stext,       // SPI TRANSMIT             $FFE8 //          |
_stext,       // SPI RECEIVE              $FFEA //          |
_stext,       // TIMER2 OVERFLOW          $FFEC //          |
_stext,       // TIMER2 CHANNEL1          $FFEE //          |
_stext,       // TIMER2 CHANNEL0          $FFF0 //          |
TIMER1OVERFLOW,// TIMER1 OVERFLOW         $FFF2 //          |
_stext,       // TIMER1 CHANNEL1          $FFF4 //          |
TIMER1CHANNEL0,// TIMER1 CHANNEL0         $FFF6 //          |
_stext,       // PLL                      $FFF8 //          |
_stext,       // IRQ                      $FFFA //          |
_stext,       // SWI                      $FFFC //          V
_stext        // RESET                    $FFFE //  Increasing Priority
              ////////////////////////////////////
};
//---------------------------------------------------------------------------
```

Appendix G. PIR Unit Bill of Materials

AT Electronic Embedded Control Consultants
Parts List for PIR Board
ATCD1006/3 02/01
Issue 2

Resistors

R1	1k
R2	10k
R3	47k
R4	10M
R5	10R
R6	680R
R7	10k
R8	100k
R9	47k
R10	100k
R11	100k
R12	470R
R13	2k2
R14	10k
R15	10k
R16	10k
R17	10k
R18	470R
R19	680R
R20	10k
R40	3M3
R42	10k
R44	10k
R45	47k
R46	3M3
R47	680k
R48	4k7

PIR Unit Bill of Materials

Capacitors

C1	100nF	50V DC
C2	100nF	50V DC
C3	220uF	16V DC
C4	470uF	25V DC
C5	100nF	50V DC
C6	100nF	50V DC
C7	100nF	50V DC
C8	100nF	50V DC
C9	100nF	50V DC
C10	100nF	50V DC
C11	3nF3	50V DC
C12	100nF	50V DC
C13	220µF	16V DC
C14	100nF	50V DC
C15	33µF	10V Tantalum
C16	10µF	16V DC
C17	10µF	16V DC
C19	100nF	50V DC
C20	10nF	50V DC
C21	22µF	16V DC
C23	10µF	16V DC

Semiconductors

D1	BAS16
D2	BAS16
D3	BAS16
D4	BAS16
D5	5V1
IC1	MC68HC908JK3
IC2	LM7805
IC3	MAX232
IC4	74HC125D
IC5	LM324D
Q1	BC818-40
Q2	BC850
Q3	BC850
Q4	BC850

Miscellaneous

SW1	SPCO slide switch
SW2	SPCO slide switch
SW3	SPCO slide switch
X1	4MHz resonator
XT1	9.8304MHz Oscillator Module
FR1	Curtain’ Fresnel lens
IR1	GP1U28Q
J1	9 way rt angle male ‘D’ Connector
J2	“3pin 0.1”” header”
J3	“3pin 0.1”” header”
J4	9 way rt angle male ‘D’ Connector
J5	3pin PIR Connector
J6	1.2mm
LED1	Infra Red transmitter
LED2	5mm Red
LED3	5mm Yellow
LED4	5mm Green

PIR Unit Bill of Materials

Appendix H. REMOTE Unit Bill of Materials

AT Electronic Embedded Control Consultants
Parts List for I/R Remote Board
ATCD1007/2 02/01

Resistors

R1	10M	5% 0.25W
R2	330k	5% 0.25W
R3	68k	5% 0.25W
R4	10k	5% 0.25W
R5	10k	5% 0.25W
R6	2k2	5% 0.25W
R7	10R	5% 0.25W
R8	10k	5% 0.25W
R9	1k	5% 0.25W
R10	10k	5% 0.25W
R12	10k	5% 0.25W
R13	10k	5% 0.25W
R14	10k	5% 0.25W
R16	10k	5% 0.25W
R17	470R	5% 0.25W
R18	470R	5% 0.25W
R19	10k	5% 0.25W
R20	47k	5% 0.25W

Capacitors

C1	100nF	50V DC
C2	100nF	50V DC
C3	100nF	50V DC
C4	100nF	50V DC
C5	33nF	50V DC
C6	15pF	50V DC
C7	220µF	16V DC
C8	100nF	50V DC
C9	100nF	50V DC
C10	15pF	50V DC
C11	10nF	50V DC
C12	100nF	50V DC
C14	100µF	50V DC
C15	220µF	16V DC
C16	100nF	50V DC

Semiconductors

D1	LL4001
D2	BAS16
D3	LL4007
D4	BAS16
D5	5V1 300mW
IC1	74HC125D
IC2	MAX232
IC3	DS1307
IC4	DS1804Z
IC5	MC68HC908GP32
IC6	LM7805
IC7	LM7808
Q1	BC818-40
Q2	BC849
Q3	BC849
Q4	MMUN2111LT1
Q5	BC849

Miscellaneous

IR1	GP1U28Q
J1	9 way rt angle male ‘D’ Connector
J2	9 way rt angle male ‘D’ Connector
J3	PCB skt 1.3mm
J4	PCB skt 2.1mm
LCD1	Sharp LM16A211
LED1	5mm Red
LED3	Infra Red transmitter
LED4	5mm Green
B1-B15	Tactile switches
BT1	3.0V Lithium
SW2	SPCO Slide Switch
SW3	SLSwitch
XT1	32kHz Xtal
XT2	32kHz Xtal
XT3	9.8304MHz Oscillator Module
SW2	SPCO Slide Switch
SW3	SPCO Slide Switch
XT1	32kHz Xtal
XT2	32kHz Xtal
XT3	none

REMOTE Unit Bill of Materials

How to Reach Us:

USA/EUROPE/LOCATIONS NOT LISTED:
Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
1-303-675-2140
1-800-441-2447

TECHNICAL INFORMATION CENTER:
1-800-521-6274

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

HOME PAGE:
http://www.motorola.com/semiconductors/

MOTOROLA

DRM001/D