MOTOROLA
intelligence everywhere™

digital dna™

Digitally Addressable Lighting Interface (DALI) Unit Using the MC68HC908KX8

Designer Reference Manual

M68HC08
Microcontrollers

DRM004/D
Rev. 3.0, 3/2002

WWW.MOTOROLA.COM/SEMICONDUCTORS

Digitally Addressable Lighting Interface (DALI) Unit Using the MC68HC908KX8

By: Magnus Grampp
Grampp R & D HB
Skimmelvagen 14
SE-252 86 Helsingborg
Sweden

Telephone: +46 42 913 95
Fax: +46 70 614 96 39
Email: info@grampp.se
Web: http://www.grampp.se

Grampp R & D

Revision History

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://www.motorola.com/semiconductors

The following revision history table summarizes changes contained in this document. For your convenience, the page number designators have been linked to the appropriate location.

Revision History

Date	Revision Level	Description	Page Number(s)
November, 2001	1	**5.5.3 DALI Module** — Description reworded for clarity	57
		Figure 3-11. `cpu-Init( )` Flowchart — Clock frequency changed from 9.8304 MHz to19.6608 MHz	40
		F.4 Master: cpu.c — Code replaced	89
		F.6 Master: dali.c — Code replaced	93
		F.15 Master: rs232.c — Code replaced	113
		G.5 Slave: cpu.c — Code replaced	122
		G.7 Slave: dali.c — Code replaced	125
		G.11 Slave: lamp.c — Code replaced	135
		G.14 Slave: rs232.c — Code replaced	159
November, 2001	2	**Appendix B. DALI Master Unit Schematic and Layout** — Replaced master schematic and master layout for readability.	71
		Appendix D. DALI Slave Unit Schematic and Layout — Replaced slave schematic and slave layout for readability.	79
March, 2002	3	**G.11 Slave: lamp.c** — Code changed to buffered PWM	135

List of Sections

List of Sections

Table of Contents

Section 1. General Description

Section 2. DALI Demonstration Board

Section 3. DALI Master Unit

Section 4. DALI Protocol Standard

Section 5. DALI Slave Unit

Section 6. PC Software

Appendix A. DALI Instruction Set

Appendix B. DALI Master Unit Schematic and Layout

Appendix C. DALI Master Unit Bill of Materials

Appendix D. DALI Slave Unit Schematic and Layout

Appendix E. DALI Slave Bill of Materials

Appendix F. DALI Master Unit Source Code Files

Appendix G. DALI Slave Source Code Files

Table of Contents

List of Figures

List of Figures

List of Tables

List of Tables

Section 1. General Description

1.1 Contents

1.2 Introduction

This document defines the reference design for a low-cost DALI (**D**igital **A**ddressable **L**ighting **I**nterface). It shows how to use the Motorola MC68HC908KX8/KX2 in a master-slave configuration where the units are communicating with a simple protocol. The communication protocol, the hardware, and the software are described in this document and although the application is intended for illumination control, the structure is also applicable for similar purposes.

1.3 Design Overview

The design is divided into two separate units a DALI master unit and a DALI slave unit. The DALI slave unit can set the luminosity of a connected lamp. The desired brightness level is set by the DALI master unit, which has a simple human interface consisting of a four-digit liquid crystal display (LCD), two shaft encoders, and a push button. The DALI master unit sends the wanted values to the connected DALI slave units. The communication is based around the DALI standard, which is an asynchronous, half-duplex, serial protocol over a two-wire differential bus. See **Figure 1-1**.

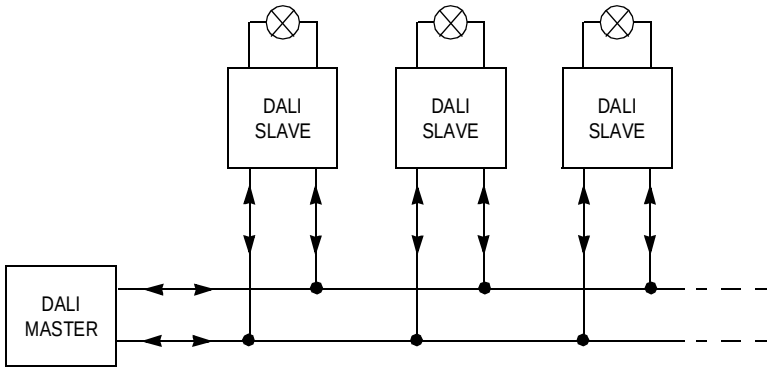

Figure 1-1. The DALI Bus System

The software is re-programmable by using the built-in monitor mode interface. Both the DALI master unit and the DALI slave unit can be controlled through an RS232 interface.

A DALI demonstration board consisting of a DALI master unit and four DALI slave units is obtainable through Motorola.

1.4 Introduction to DALI

The lighting industry has developed a new standard for communication with electronic ballast (ECG). It is an interface standard for communication between a controller and the ECGs. This standard goes by the name of DALI (**D**igital **A**ddressable **L**ighting **I**nterface) and it will be included as an appendix to ECG standard IEC 929.

DALI is designed for the use of standard components and for simple wiring which means low costs.

Fields of application may be:

- Day lighting and night lighting
- Different lighting for varying activities in multi-purpose rooms such as hotels, conference rooms, etc.
- Possibilities of dimming the lights and increasing the lighting according to requirements

- Properly adjusted light settings depending on the direction of the daylight (e.g., in the morning, at noon, in the afternoon)
- Energy savings

The DALI installation is based on the master-slave principle:

- The user operates the system through the controller (master).
- The controller sends messages to all the ECGs (slaves) containing an address and a command.
- The address determines whether the ECG should listen.
- Each ECG is digitally addressed and therefore is insensitive to electromagnetic noise (compare with an analogue 1–10-volt system).

For instance, the command could be current brightness value or lamp on or lamp off. Lighting values and group assignment will be stored in the ECGs.

The ECGs may be assigned up to sixteen freely definable groups and each group can store up to sixteen different lighting values for the lighting scenes. Each ECG can be assigned to more than one group. There may be up to 64 ECGs per each controller.

General Description

Section 2. DALI Demonstration Board

2.1 Contents

2.2 Introduction

This section describes how to connect and use a DALI (**D**igital **A**ddressable **L**ighting **I**nterface) demonstration board. A DALI demonstration board can be obtained through Motorola. It is always delivered with the software already loaded so only a power supply and four lamps have to be connected to get the system started.

NOTE: *It is necessary to read this section to get an understanding of the system even if you do not have a demonstration board.*

2.3 Contents of the System

Five printed circuit cards and the cables connecting them are mounted on the DALI demonstration board.

The system also consists of:

- One 4-wire cable for communication with a PC, see **Section 6. PC Software**
- One 6-wire cable for replacing the software, see **3.6 Upgrading the Software** (Master) and **5.6 Upgrading the Software** (Slave)
- CD-ROM containing all software and documentation

2.4 System Overview

The DALI demonstration board consists of a DALI master unit and four DALI slave units. They are connected through a 2-wire DALI network. The master unit controls all the slave units and each slave unit can control a lamp. See **Figure 2-1**.

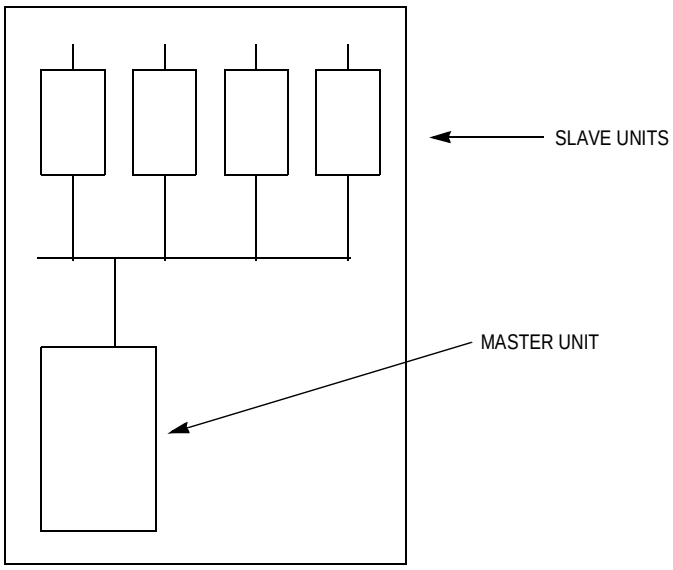

Figure 2-1. The DALI Demonstration Board

2.5 System Installation

The system has to be powered from an external source. The requirement for this supply is 12 V (10–14 V) DC. The power consumption is dependent on the number and size of the connected lamps. The master and slave boards without the lamps connected consume less then 500 mA.

Use any 12-V lamps with a power consumption equal to or less than 10 W. A 12-V/10-W halogen light bulb would be a good choice. The power supply has to be dimensioned according to this. Mascot 9120/12 is sufficient for this purpose but several other power supplies that meet the above criteria can be used.

Connect the lamps to the 2-way Phoenix screw terminal for each slave unit and connect the power supply to the terminal socket (12 V to red cable and 0 V to black cable).

2.6 System Operation

After connecting the power supply the system starts in its default mode. Every slave unit should light its connected lamp and the LED should be lit showing that the slave unit is running.

The LCD on the master unit should show 00:00 (see **Figure 2-2**). The system is now ready to accept commands according to the DALI standard.

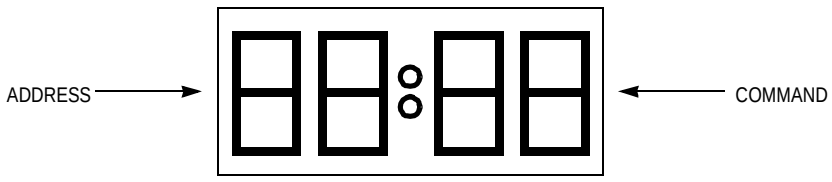

Figure 2-2. The LCD Display

The LCD on the master shows data to be sent to the slave as two hexadecimal numbers. Normally this is an address followed by a command. By turning the shaft encoders the address and the command

can be changed. Pushing the button on the right side of the shaft encoders sends the data to the slave units. The data sent may result in an answer from the slave. This is shown on the LCD as “Axx” where xx stands for a hexadecimal number.

For more information regarding addressing modes and commands see **Section 4. DALI Protocol Standard**. Address 0xFF and command 0xFF have a special meaning in this system. Selecting this initializes a demo sequence, where the master automatically sends a command to change the scene, with 3 second's interval. It starts with scene 0 and continues until scene 15 and after that it starts with scene 0 again. This will continue until new data is selected.

If an individual lamp is to be addressed the factory settings for the short addresses of the slave units are 0, 1, 2, and 3.

2.7 Examples

Here are some examples to test the system. Enter the address and the command according to **Table 2-1** using the shaft encoders and push button.

Table 2-1. Examples of DALI Commands

Address	Command	Action
FE	E6	Set the light level to E6 (approximately 50%) on all lamps
02	FE	Set lamp 3 to 100% light level
FF	01	Dim up all lamps for 200 ms
01	00	Turn off lamp 1
07	92	Check if lamp 4 is working (No answer means YES, A FF means NO)

Section 3. DALI Master Unit

3.1 Contents

3.2 Introduction

This section describes the design, hardware, and software of the DALI (**D**igital **A**ddressable **L**ighting **I**nterface) master unit. The unit has a human interface consisting of:

- 4-digit liquid crystal display (LCD)
- Two shaft encoders
- A push button

With these, the operator could select and address a DALI command to be sent over the DALI interface. The master unit also has an RS232 interface for communicating with a PC and a monitor mode interface to enable download of new software.

3.3 Features

The master unit is designed as a multi-purpose communication controller with these features:

- A MC68HC908KX8
- A display composed of a 4-digit LCD
- A keyboard composed of two shaft encoders and a push button
- A DALI interface
- A RS232 interface
- A monitor mode interface
- A small wire and wrap area for custom extensions

Refer to **Section 2. DALI Demonstration Board** for a description of how to use the DALI master unit.

3.4 Hardware

The DALI master unit is built around the MC68HC908KX8. Connected to the central processor unit (CPU) are:

- DALI interface
- Keyboard interface
- LCD interface
- RS232 interface
- Monitor mode interface

The RS232 and the monitor mode interfaces are electrically isolated from the rest of the unit. This is done to enable the DALI slave unit to use the same circuitry. Therefore, to use these interfaces separate cabling has to be connected. See **Figure 3-1**.

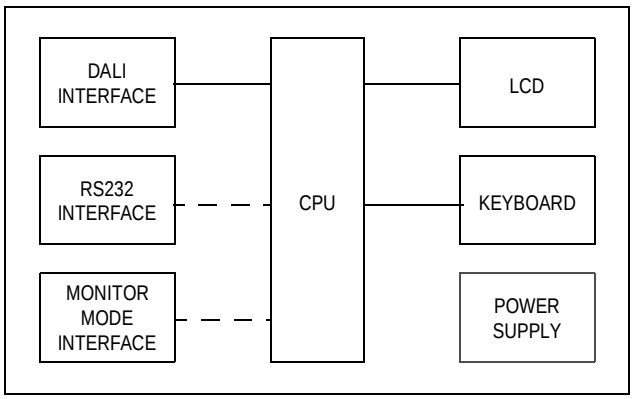

Figure 3-1. DALI Master Unit

Schematics, layout, and parts list can be found in **Appendix B. DALI Master Unit Schematic and Layout** and in **Appendix C. DALI Master Unit Bill of Materials**.

The hardware design is intended for demonstration purposes only. It is functional in a normal office environment. There are many possibilities to improve the design for increased functionality and reliability. Costs can be reduced for high volume production.

3.4.1 Microcontroller Unit (MCU)

The MCU used in this design is the MC68HC908KX8, that has these advantages:

- Low pin count
- Internal clock oscillator
- Embedded FLASH memory (8 Kbytes) which can be used for both code and data
- Embedded random-access memory (RAM) (192 bytes)
- In-circuit programmable through the monitor mode interface
- M68HC08 compatible code

As all members of the M68HC08 Family, it has compact and fast executing code as well as several development and support tools available from both Motorola and third parties.

The possibility of writing data into the FLASH memory during program execution is used in this application. However, the advantage of having an internal oscillator is not fully realized in this application. The most common reason for adjusting the clock frequency is to save power. Another reason for using an internal oscillator is to save space. If saving space is not necessary, than devices like the MC68HC908JK3 with external clock generation can be used to achieve the same design. This processor has a slightly bigger package (20 pins instead of 16 pins with one extra input/output (I/O)) but is a good alternative for this application.

Since the size of the code is less than 2 Kbytes, the MC68HC908KX2 could also be an alternative.

More information regarding the CPU can be found in the *MC68HC908KX8, MC68HC908KX2 HCMOS Microcontroller Unit Technical Data*, Motorola order number MC68HC908KX8/D.

3.4.2 DALI Interface

The DALI interface has one circuit for transmitting (see **Figure 3-2**) and one circuit for receiving (see **Figure 3-3**).

The transmission consists mainly of a power transistor (T1) for switching the power on and off. The power transistor is regulated by two transistors (T2 and T3).

One of the two transistors (T3) is controlled from the processor (PTB7) through an inverter (IC4). A low signal from the processor gives a high signal after the inverter. This high signal opens the transistor (T3) which switches off the power transistor (T1). A high signal from the processor gives a low signal after the inverter, closing the transistor (T3) and switching on the power transistor (T1).

The other transistor (T2) is controlled by the current flowing through the resistor (R1). If a DALI slave unit is connected, this current will be the same as that flowing through the power transistor. The value of the resistor is chosen in such a way that when the current exceeds 250 mA the voltage level across the resistor will open the transistor (T2) which in its turn closes the power transistor (T1). In this way the current is maximized to 250 mA.

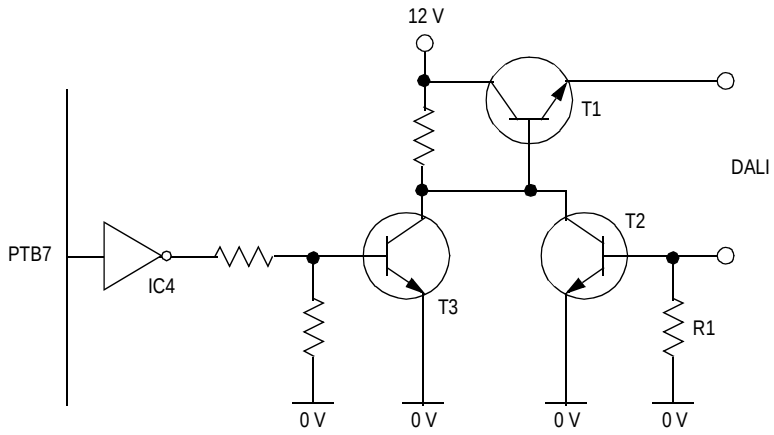

Figure 3-2. DALI Transmit Circuit

The comparator (IC3) handles the reception and sends the received signal to the processor (IRQ1 and PTB6). If very accurate voltage levels are to be achieved the surrounding resistors should be trimmed. The inverter (IC4) converts the signals to the correct logical level.

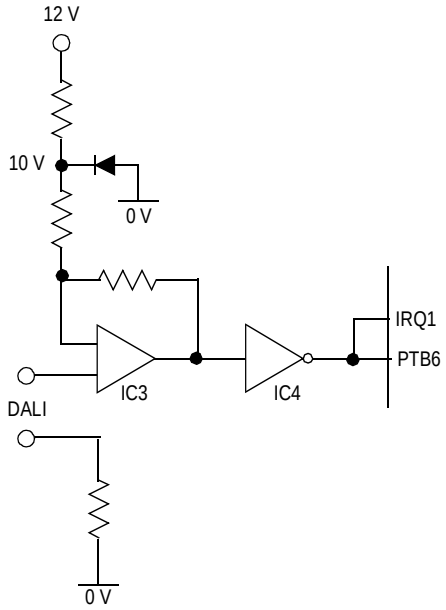

Figure 3-3. DALI Receive Circuit

NOTE: *The interface has no protection against over voltage and no electromagnetic interference (EMI) suppression.*

3.4.3 Keyboard Interface

The master unit can receive user command through a push button (S1) and two shaft encoders (S2 and S3). They are connected to the processor (PTA0–PTA4). See **Figure 3-4**.

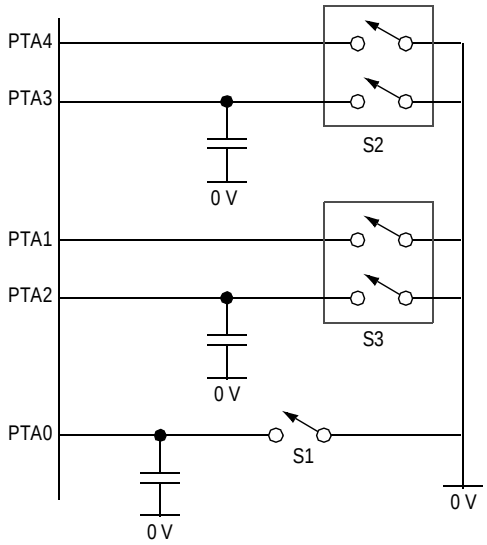

Figure 3-4. Keyboard Interface

3.4.4 The LCD Interface

The LCD (DIS1) is controlled by the processor (PTB0–PTB2) through the LCD driver circuit (IC5). See **Figure 3-5**.

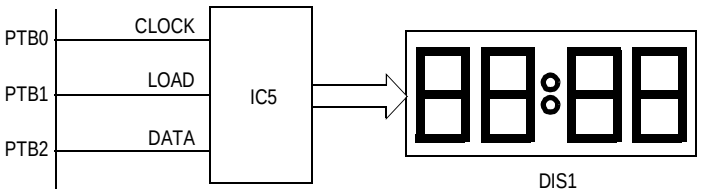

Figure 3-5. LCD Interface

3.4.5 The RS232 Interface

The RS232 interface is connected to the processor (PTB4 and PTB5) and a pin header (JP4). With a cable it can be connected to the RS232 driver circuit (IC6) through the pin header JP5. The external connection is by way of a D-sub connector (X3). The RS232 interface is not protected against over voltages, different ground potential or EMI interference. Connect the cable when the system is not powered. See **Figure 3-6**.

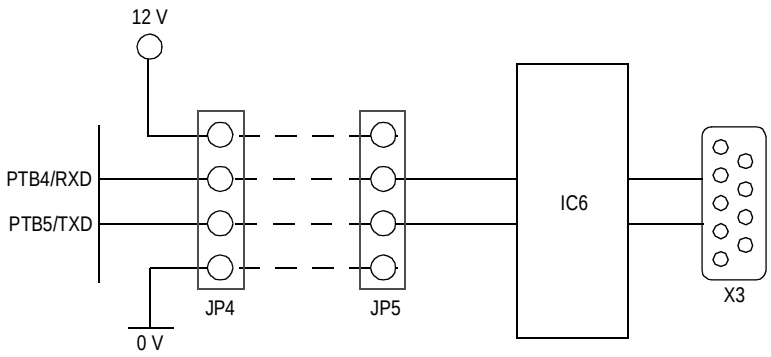

Figure 3-6. RS232 Interface

3.4.6 Monitor Mode Interface

The monitor mode interface is connected to the processor (PTA0, PTA1, PTB6, and IRQ1) if all four switches on the DIP-switch are off and a cable is connected between the pin header JP1 and pin header JP6. Connect the cable when the system is not powered. When power is returned, the processor is in monitor mode (see *MC68HC908KX8, MC68HC908KX2 HCMOS Microcontroller Unit Technical Data*, Motorola order number MC68HC908KX8/D for more information). The communication with the processor during the monitor mode goes through the D-sub (X2), the RS driver circuit (IC6), and the buffer (IC7). The crystal oscillator (QG1) ensures that the RS232 communication speed is 9600 baud. See **Figure 3-7**.

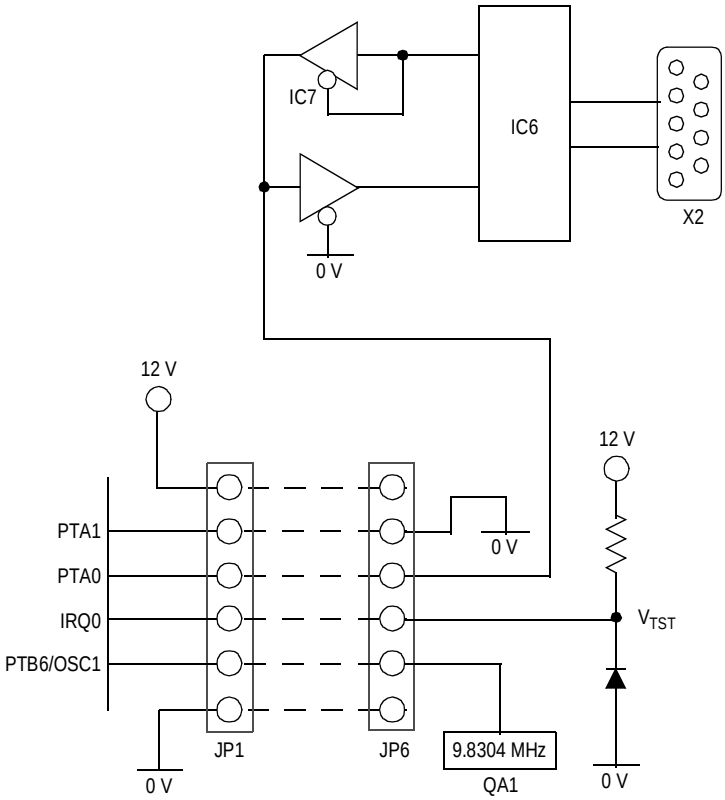

Figure 3-7. Monitor Mode Interface

3.4.7 Power Supply

The linear regulator (IC2 and IC8) makes the 5-volt conversion. There is no external protection against over voltages or EMI interference but the diode D1 serves as protection against wrong polarity. The input voltage can vary between 7 and 30 volts for the 5-volt conversion but the DALI interface requires that the voltage be kept within 10 to 14 volts. See **Figure 3-8**.

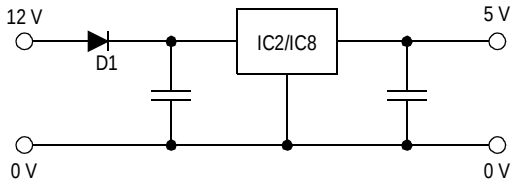

Figure 3-8. Power Supply

3.5 The Software

The software handles the DALI, the RS232 communication tasks, and the human interface. It is based around a main module that first initializes common variables and the utility modules then goes into an endless loop. The different utility modules are updated through interrupts. If an event occurs in one utility module which leads to the need to act in another utility module, a flag is set. The main module scans these flags and takes action accordingly. No call is made directly from one utility module to another. Since the structure is based on what action the different software modules will do, several processor dependent functions (TIM, ADC, etc.) may be used in one module. See **Figure 3-9**.

The source code can be viewed in **Appendix F. DALI Master Unit Source Code Files**.

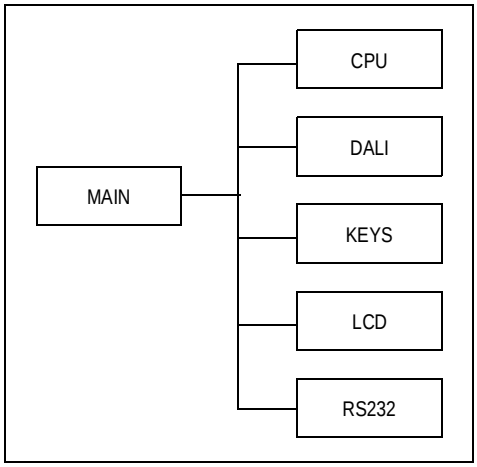

Figure 3-9. DALI Master Software

All code is written in C language except for the startup routine. The assembler, compiler, and linker used are from COSMIC Software which can be found at:

http://www.cosmic-software.com

NOTE: There is also the possibility of using software from Metrowerks which can be found at:

http://www.metrowerks.com

3.5.1 Main Module

The main module consists of initializing global variables, calls to initializing routines in other modules, and an eternal loop. In the eternal loop different flags are checked to see if an action has to be taken. The computer operating properly (COP) watchdog counter is also cleared in the loop. See **Figure 3-10**.

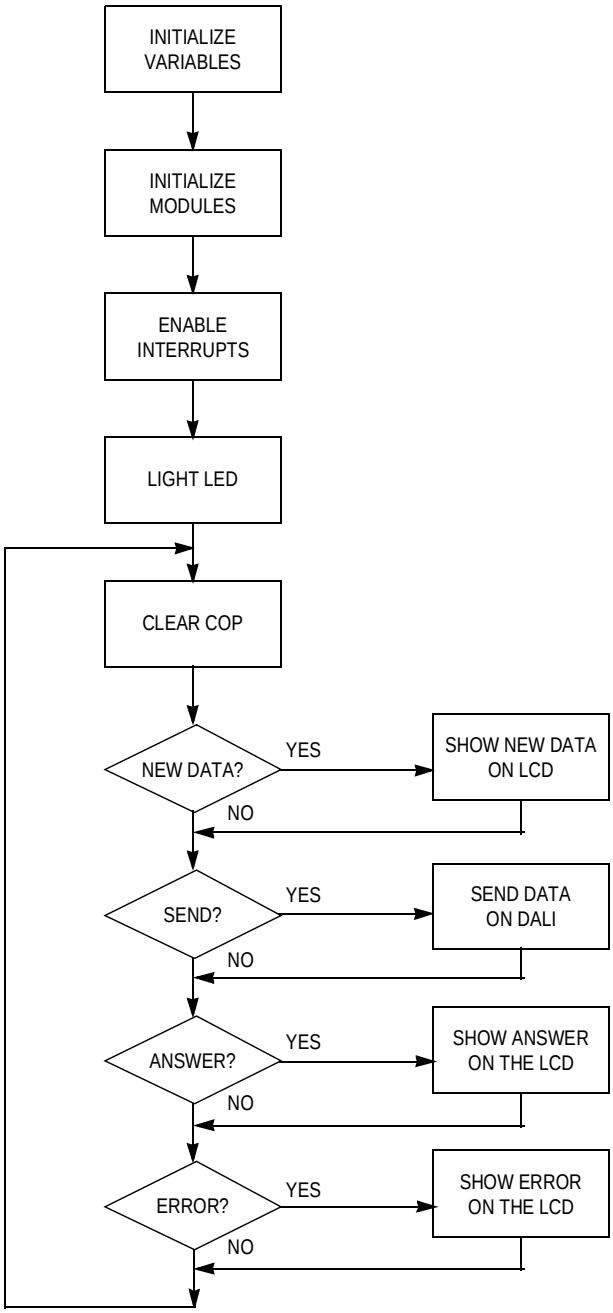

Figure 3-10. `main( )` Flowchart

3.5.2 Central Processor Unit (CPU) Module

The CPU module is responsible for setting up the processor correctly. This includes:

- LVI (power supervision)
- Port direction
- Port data
- Built-in pullup resistor
- IRQ1 interrupt
- Internal clock speed

If the clock has not previously been trimmed this is handled in the module. The clock trim value is stored in the FLASH by using the on-chip FLASH programming routines. The call is made by calling `clock_write( )` which is defined as follows:

```
#define clock_write() _asm("ldhx #$FDFF\njsr $1009\n")
```

See **Figure 3-11**.

For more information regarding the on-chip routines, refer to the Motorola Application Note entitled *Using MC68HC908 On-Chip FLASH Programming Routines*, Motorola order number AN1831/D.

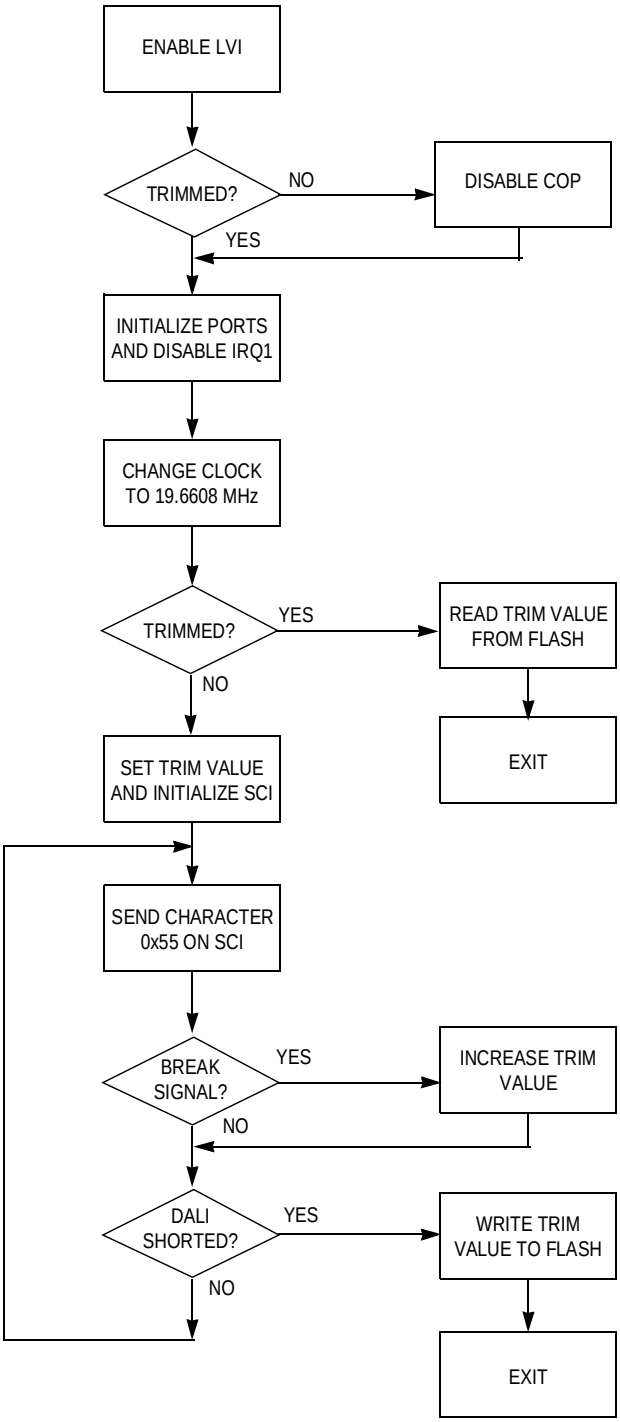

Figure 3-11. `cpu-Init( )` Flowchart

3.5.3 DALI Module

The DALI communication is handled in the DALI module. Several parts of the processor are used. Incoming data is detected by an IRQ interrupt. Then this data is sampled with an interval (4800 times per second) set by the timebase module (TBM). The timebase module also controls the transmission. If data is to be repeated this is handled in the DALI module.

3.5.4 Keys Module

The keys module detects changes to the shaft encoders or the use of the push button. The push button is simply handled by the keyboard interrupt module (KBI). This is not possible with the shaft encoders since it is impossible to determine which of the shaft encoders were rotated using only one common interrupt. The need for more interrupt sources is solved by enabling the input capture capability of the timer interface module (TIM).

3.5.5 Liquid Crystal Display (LCD) Module

The LCD is controlled through the driver circuit by the LCD module. The data is clocked into the shift register of the driver circuit by toggling the PTB output pins.

3.5.6 RS232 Module

The RS232 module handles the serial communication over RS232. It uses the serial communication interface module (SCI). The baud rate is preset to 9600 baud.

3.6 Upgrading the Software

To be able to reprogram the software, the processor has to be set into a monitor mode. The monitor mode is a state of the processor where simple debugging and downloading of new code is possible. During the monitor mode, the processor is running a built-in program that communicates through a single pin. The monitor mode is described in

more detail in the document *MC68HC908KX8, MC68HC908KX2 HCMOS Microcontroller Unit Technical Data*, Motorola order number MC68HC908KX8/D.

3.6.1 Board Configuration

To configure the board in monitor mode:

- Disconnect the power supply and connect a free serial communication port of the PC (or the computer that communicates with the processor) to the D-sub on the DALI master unit marked “Monitor” using a serial communication cable.
- Connect a cable between JP1 and JP6 (see **Appendix B. DALI Master Unit Schematic and Layout**).
- Set the DIP switch to “OFF”.

When the power supply is reconnected the processor is in the monitor mode.

After exiting monitor mode, the cable between the pin headers has to be removed and the DIP switch has to be set to “ON” to return to normal mode.

3.6.2 Monitor Mode Software

There is freely available software for the PC environment that cares about the most common tasks during the monitor mode. The program “PROG08SZ” from P&E Microcomputer Systems, Inc. (http://www.pemicro.com) will work fine if the main task is to download new software to the DALI modules. With that software you define the DALI unit as a Class III board. The monitor mode interface is prepared for a communication speed at 9600.

3.6.3 Replacing the Code

By using monitor mode software, the FLASH memory of the DALI master unit can be erased and new software downloaded. After a successful replacement of the code the internal clock has to be trimmed.

The internal clock generator of the processor has a tolerance of ±25%. This can be trimmed down to ±2%. In the process of loading new software the trim values are corrupted. Since the system is dependent on correct communication speed, a trimming has to take place before the code can be executed properly. This is done through the RS232 interface, the DALI interface, and by using an oscilloscope.

- Connect the computer to the D-sub on the DALI master unit marked “Serial”.
- Connect a cable between JP4 and JP5 (see **Appendix B. DALI Master Unit Schematic and Layout**).
- Connect the oscilloscope between ground and PTB5/TXD on the DALI unit.

A PC program that can generate a break signal on the serial communication port is also needed. When the system is powered a square-wave signal, which should be slightly larger than 2400 Hz, is visible on the oscilloscope. The frequency should be changed until it is between 2385 and 2415 Hz. Sending a break signal does this. For each break signal the frequency is lowered by about 5 Hz. When the correct frequency is achieved the trim value can be stored by making a temporary short-circuit across the DALI signal wires. The next time the unit is restarted the clock is automatically trimmed and the execution of the program is normal.

DALI Master Unit

Section 4. DALI Protocol Standard

4.1 Contents

4.2 Introduction

This section describes the main properties of the DALI (**D**igital **A**ddressable **L**ighting **I**nterface) protocol. More information regarding the DALI protocol can be found in the document *A.C.-Supplied Electronic Ballast for Tubular Fluorescent Lamps, Performance Requirement, Requirements for Controllable Ballasts* which can be obtained from IEC.

Some major characteristics of the DALI protocol are:

- Asynchronous serial protocol
- 1200 baud, bi-phase encoding, half-duplex
- Two-wire differential
- Voltage difference above 9.5 V means high level
- Voltage difference below 6.5 means low level
- The master unit controls the communication
- 64 slave units can be connected
- Each slave unit can be individually addressed
- The master unit sends 1 start bit, 16 bit data, and two stop bits
- The slave unit sends 1 start bit, 8 bit data, and two stop bits

4.3 Electrical Specification

The DALI communication is serial using two wires for communicating in both directions. The voltage difference between the wires indicates if it is a high or low level. A voltage difference above 9.5 V is a high level and a voltage difference less than 6.5 V is a low level. The master unit communicates with the slave units by setting the level high or low according to the serial protocol described in **4.4 Protocol Specification**. When no communication takes place the master unit keeps the level high.

The slave unit responds to the master unit by setting the level high or low. A high level is simply achieved by not interfering with the high level set by the master unit. A low level is obtained by forcing a short circuit across the wires. This is possible to do since the DALI standard states that the current supply for the DALI communication has to be limited to 250 mA.

Simultaneous communication is not possible since there is only one pair of wires.

4.4 Protocol Specification

There are a number of parameters stored in each slave unit (see **Table 4-1**). These parameters indicate how the lamp should behave in different situations.

Table 4-1. Description of Stored Parameters

Type of Parameter	Description
Actual dim level	The current light level of the lamp or physically the arc power level.
Power on level	The light level to use when powering up the slave
System failure level	The light level to use if the communication fails
Minimum level	The minimum allowed light level
Maximum level	The maximum allowed light level

Table 4-1. Description of Stored Parameters (Continued)

Type of Parameter	Description
Fade rate	The speed to change the light level during a certain time interval
Fade time	The time between the change of light levels
Short address	The individual address of a unit
Search address	An address used during address initialization
Random address	An address used during address initialization
Group	A value indicating which group the slave belongs to (16 bits)
Scene	Light levels for different scenes (16 bytes)
Status information	Holds the current status information regarding the slave
Version number	The currently implemented DALI protocol version number
Physical minimum level	The smallest allowed physical arc power level

Messages can be sent over the DALI communication wires to change or check the values.

The serial communication speed is 1200 baud with bi-phase encoding. All communication is controlled from a master. Every lamp is connected to a slave unit that is silent until the master requires an answer. A message from a master consists of one start bit, 16 bits data, and two stop bits. The data from the master unit consists of an 8-bit address part and an 8-bit command part. See **Figure 4-1**.

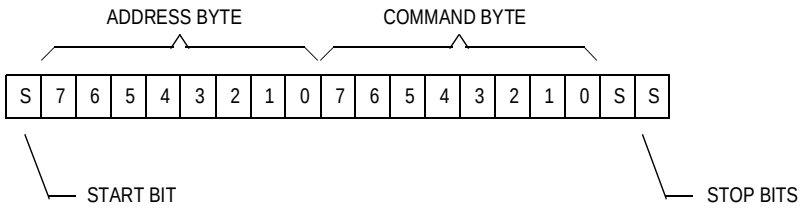

Figure 4-1. DALI Master Message

An answer from a slave unit consists of 1 start bit, 8 data bits and 2 stop bits.

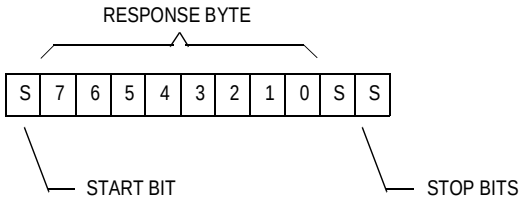

Figure 4-2. DALI Slave Message

Up to 64 slaves can be connected to the same network and each slave can receive an individual address, which is called the short address. There is also the possibility of assigning a slave unit to a group. Up to 16 groups can exist and a slave unit can belong to several groups. A message can also be broadcasted to all slave units.

Table 4-2. Type of Addresses

Type of Addresses	Byte Description
Short address	`0AAAAAAS (AAAAAA = 0 to 63, S = 0/1)`
Group address	`100AAAAS (AAAA = 0 to 15, S = 0/1)`
Broadcast address	`1111111S (S = 0/1)`
Special command	`101CCCC1 (CCCC = command number)`

The command byte is to be interpreted as an arc power level if bit S is 0. The arc power can be between 00 (off) and FE (maximum power level). A 1 in the position S means that the command byte is to be interpreted as a DALI command. Refer to **Appendix A. DALI Instruction Set** for a list of all available commands.

Section 5. DALI Slave Unit

5.1 Contents

5.2 Introduction

This section describes the design of the DALI (**D**igital **A**ddressable **L**ighting **I**nterface) slave unit hardware and the software. The DALI slave unit receives DALI commands from a DALI master unit through a DALI interface. These commands tell the DALI slave unit how to control a connected lamp.

5.3 Features

The DALI slave unit is designed to be as small and simple as possible. It has the following features:

- A MC68HC908KX8
- A polarity insensitive DALI interface
- A power switch for controlling a 12-V/10-W lamp
- A current sensor for controlling the lamp status

5.4 Hardware

The hardware consists of a power MOSFET, the CPU (MC68HC908KX8), and a DALI interface. A low-ohm resistor is used to sense the current through the connected lamp by using the analog-to-digital (A/D) converter in the central processor unit (CPU). An LED is also connected to the CPU to enable debugging even if no lamp is connected. See **Figure 5-1**

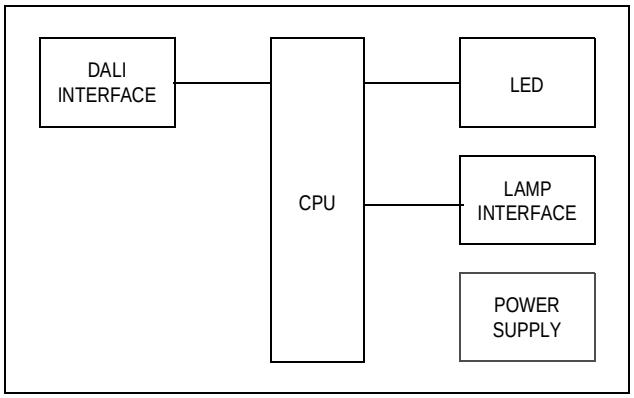

Figure 5-1. DALI Slave Unit

Schematics, layout and part list can be found in **Appendix D. DALI Slave Unit Schematic and Layout** and **Appendix E. DALI Slave Bill of Materials**.

The slave unit in a real-life application is probably part of an electronic ballast (ECG). To simplify the design a high-voltage fluorescent tube is replaced by a low-voltage halogen light bulb.

NOTE: *The current flowing through a halogen light bulb is high (approximately 1 A) so the connected cables have to be dimensioned accordingly. The construction is protected against temporary short-circuit conditions but a permanent short circuit condition will overheat some components with permanent damage as a consequence.*

There is no hardware included on the slave unit to enter the monitor mode. The slave unit can be connected to the monitor mode interface on the master unit by using a cable.

5.4.1 Microcontroller Unit (MCU)

The MCU used in this design, as in the master unit, is the MC68HC908KX8. An LED (D1) is directly connected to the processor (PTA3) to indicate if the software is running correctly. See **Figure 5-2**.

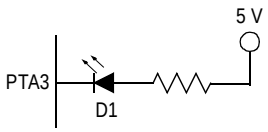

Figure 5-2. LED Interface

More information regarding the CPU can be found in the *MC68HC908KX8, MC68HC908KX2 HCMOS Microcontroller Unit Technical Data*, Motorola order number MC68HC908KX8/D.

5.4.2 DALI Interface

The DALI interface of the slave unit will not supply any current. Therefore, it can easily be optically isolated to prevent ground loops. Although making the solution polarity insensitive is not stated by the DALI standard it is suggested as an improvement. In this application, this is done by having two-way opto-diodes and an electronic relay.

The transmission uses an optically isolated electronic relay (IC5). By almost short-circuiting the DALI communication cables, it lowers the voltage since the current is limited. By using free pins on the inverter (IC2) the processor (PTB7) there is enough current for controlling. See **Figure 5-3**.

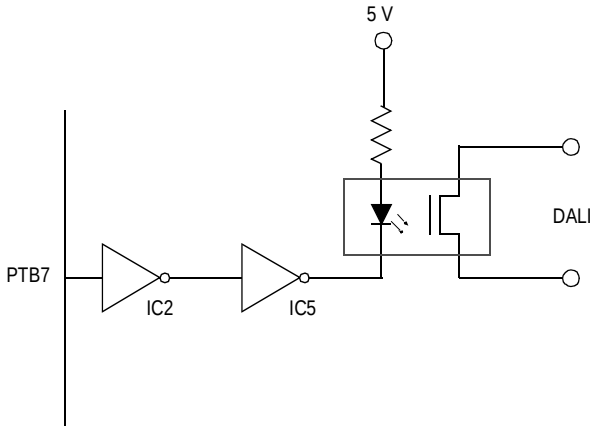

Figure 5-3. DALI Transmit Circuit

The reception is done by a two-way opto-coupler (IC3). To get a true logical signal the opto-coupler is connected to an Schmitt-trigger inverter (IC2) before the signal reaches the processor (IRQ1 and PTB6). See **Figure 5-4**.

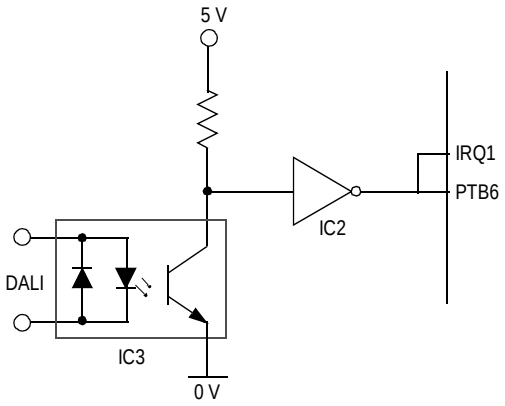

Figure 5-4. DALI Receive Circuit

5.4.3 Lamp Interface

Controlling the lamp current is done by a protected power MOSFET (T1). The processor controls the transistor directly (PTA2).

NOTE: *The current is fairly high and could cause damage if a hardware error occurs.*

To detect lamp failure a current sensing circuitry is included. This is done by letting the lamp current flow over a shunt resistance (R2). The generated voltage is measured by the built-in analog-to-digital (A/D) converter in the processor (PTB3/AD3). See **Figure 5-5**.

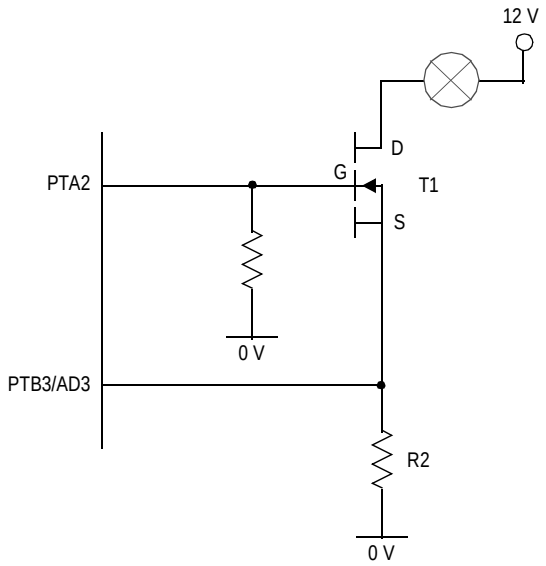

Figure 5-5. Lamp Interface

5.4.4 RS232 Interface

Only JP1 is placed on the slave unit. For communication over an RS232 interface, a cable has to be connected to the master unit. See **Figure 5-6**.

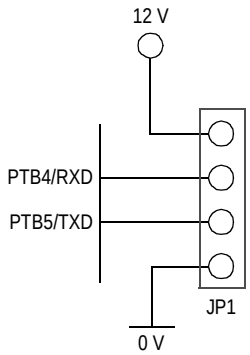

Figure 5-6. RS232 Interface

5.4.5 Monitor Mode Interface

The monitor mode interface only consists of the pin header JP2 and DIP switch S1. For simplicity the rest of the interface is located on the master unit and has to be connected with a cable. See **Figure 5-7**.

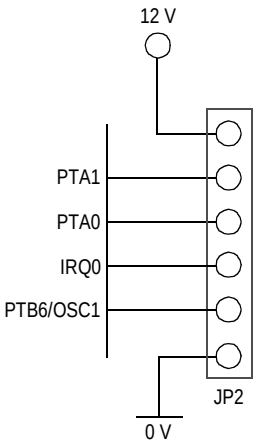

Figure 5-7. Monitor Mode Interface

5.4.6 Power Supply

The power supply solution is identical to the one used in the master unit. The regulator IC4 is responsible for the 5-volt conversion. Incorrect polarity is handled by D2. See **Figure 5-8**.

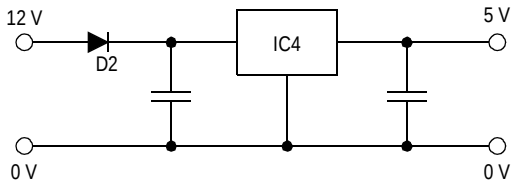

Figure 5-8. Power Supply

5.5 The Software

The software for the master unit and the slave unit is very similar but they do not share the code although some files have the same name. The structure is however identical. See **Figure 5-9**.

The source code can be viewed in **Appendix G. DALI Slave Source Code Files**.

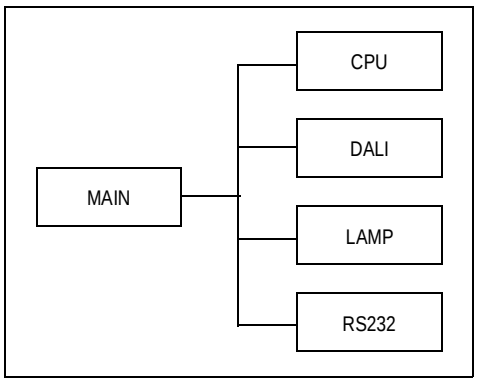

Figure 5-9. DALI Slave Software

5.5.1 Main Module

The main module in the slave unit has the same structure as the main module in the master unit. If the initialization process is successful, toggling an output pin on PTA lights the LED. The possibility to source and sink high current on PTA is used for directly connecting the LED to the processor. See **Figure 5-10**.

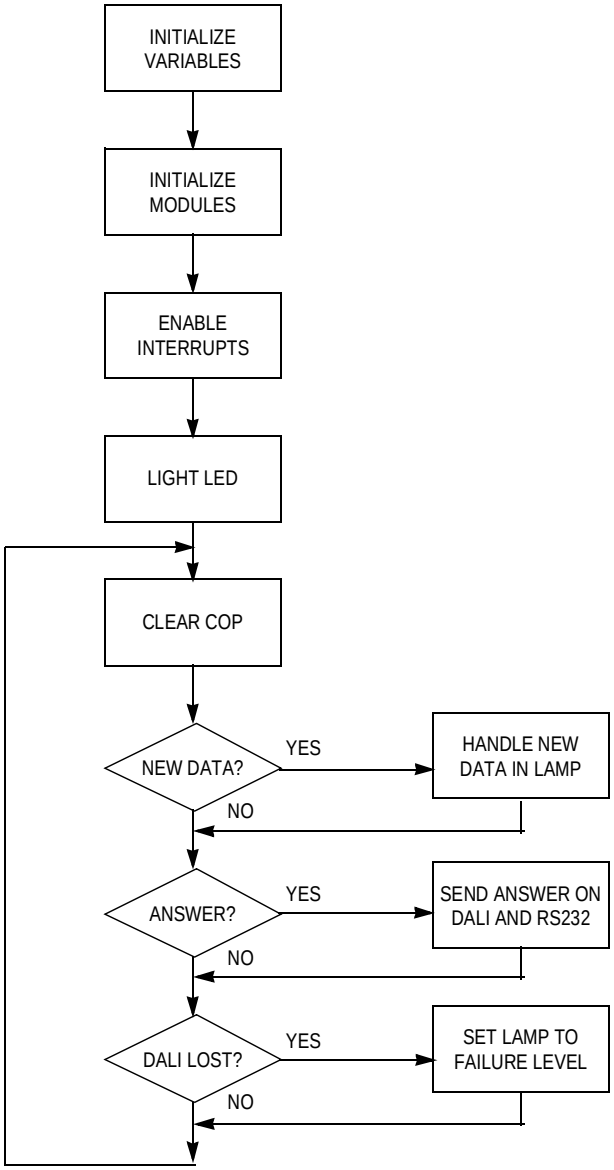

Figure 5-10. `main( )` Flowchart

5.5.2 Central Processor Unit (CPU) Module

The difference between the CPU module of the slave unit and the CPU module of the master unit is that the setup of ports is adapted to the difference in hardware.

5.5.3 DALI Module

The data received from a slave unit is 16 bits and the data sent is 8 bits. This is opposite to a master unit. To avoid disturbance from the timer interface module (TIM) this is disabled during transmission and reception of data. The DALI communication is monitored so that a disconnected slave unit sets its lamp to a failure level.

5.5.4 Lamp Module

This is the largest module. It handles all activities related to the lamp. These include:

- Regulating the light level of the lamp by a PWM-signal
- Storing parameters for the user-defined data
- Supervision of the lamp current

The lamp module also interprets all DALI commands.

The timer interface module (TIM) is used to handle all timing. The same timer is used for generating the pulse-width modulator (PWM) signal for controlling the light level. The current measurement is done by using the analog-to-digital converter (ADC).

If the memory is empty, the user-defined data will be set to the default values as shown in **Table 5-1**.

Table 5-1. Default Values for the User Defined Data

Memory Position	Type of Parameter	Default Value
FD80	Power on level	FE
FD81	System failure level	FE
FD82	Minimum level	01
FD83	Maximum level	FE
FD84	Fade rate	07
FD85	Fade time	00
FD86	Short address	FF
FD87-FD89	Random address	FF FF FF
FD8A-FD8B	Group	00 00
FD8C-FD9B	Scene	FF … FF

The storage of new user-defined data in the FLASH memory is handled by the on-chip FLASH programming routines. To erase the data area in the FLASH memory a call is made to `flash_erase( )` which is defined as:

```
#define flash_erase() _asm("ldhx #$FD80\njsr $1006\n")
```

The definition for flash_write( ) is:

```
#define flash_write() _asm("ldhx #$FD80\njsr $1009\n")
```

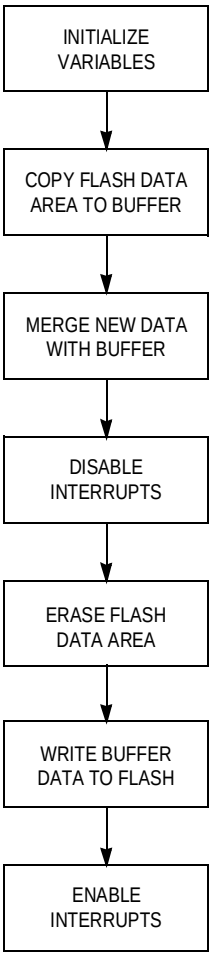

Figure 5-11. `lamp_WriteFlash( )` Flowchart

For more information regarding the on-chip routines, refer to the Motorola Application Note entitled *Using MC68HC908 On-Chip FLASH Programming Routines*, Motorola order number AN1831/D.

5.5.5 RS232 Module

The RS232 module is similar to the module for the master unit. The difference is that other flags are set. Messages received over RS232 by a master are forwarded over the DALI communication line but on a slave the messages are handled directly as if they had arrived over the DALI net.

5.6 Upgrading the Software

Reprogramming the software in the DALI slave unit is done in a similar way as in the DALI master unit. Refer to **3.6 Upgrading the Software** for more information. The main difference is that the DALI slave unit does not have a complete monitor mode interface. Therefore, to enter the monitor mode the slave unit has to use the monitor mode interface on the master unit.

5.6.1 Board Configuration

To configure the board in monitor mode:

- Disconnect the power supply and connect a free serial communication port of the PC (or the computer that communicates with the processor) to the D-sub on the DALI master unit marked “Monitor” using a serial communication cable.
- Connect a cable between JP2 on the DALI slave unit (see **Appendix D. DALI Slave Unit Schematic and Layout**) and JP6 on the DALI Master unit (see **Appendix B. DALI Master Unit Schematic and Layout**).
- Set the DIP switch on the DALI slave unit to “OFF”.

When the power supply is reconnected the processor is in the monitor mode.

After exiting the monitor mode, the cable between the pin headers has to be removed and the DIP switch has to be set to “ON” to return to the normal mode.

5.6.2 Reprogramming the Code

By using monitor mode software the FLASH memory of the DALI master unit can be erased and new software downloaded. Erasing the memory also clears all user-defined data in the slave unit including the short address. Therefore, the user-defined data (especially the short address) has to be set when downloading of new software and clock trimming are finished.

The interrupt vector is cleared when other parts of the FLASH memory are erased. Since the slave unit updates the contents of the FLASH memory during execution it is necessary to protect the interrupt vector area. After the code is replaced the memory position FF7E (FLASH Block Protect Register) has to be set to FE.

Trimming of the internal clock has to be done as described in **3.6 Upgrading the Software**. However, the cable should be connected between JP1 on the DALI slave unit (see **Appendix D. DALI Slave Unit Schematic and Layout**) and JP5 on the DALI master unit (see **Appendix D. DALI Slave Unit Schematic and Layout**).

DALI Slave Unit

Section 6. PC Software

6.1 Contents

6.2 Introduction

There is a possibility of controlling the DALI (**D**igital **A**ddressable **L**ighting **I**nterface) master through an RS232 interface (9600 baud, no parity, 8 data bits, and one stop bit). The DALI master handles data received over the RS232 interface the same way as the data is entered by the shaft encoders and the push button. This means that everything that can be done manually on the master board can also be controlled remotely. Answers received from a slave unit are also echoed on the RS232 interface. The liquid crystal display (LCD) on the master unit displays all the data communicated. To demonstrate this possibility there is a small application called “DALI demo”.

6.3 Installation

The “DALI demo” application will work on any PC running a 32-bit Microsoft® Windows® and having one free serial communication port. It is installed through a standard setup program. Follow the instructions on the screen.

6.4 User manual

To be able to communicate with the DALI master:

- A serial communication cable has to be connected between the PC and the D sub on the master unit marked “Serial”.
- The RS232 communication has to be enabled on the DALI master unit by connecting the pin header JP4 and pin header JP5 with a cable, see **Appendix B. DALI Master Unit Schematic and Layout**.

Before starting the program “DALI demo” all slave units have to be connected to the master.

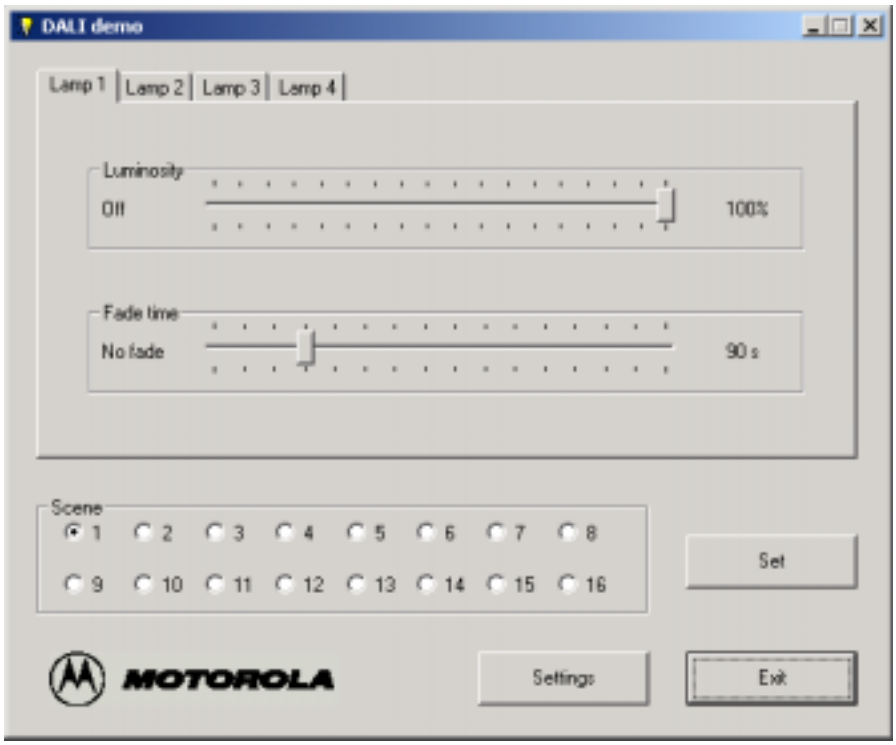

Figure 6-1. DALI Demo Application

Start “DALI demo” through the “Start” menu in Microsoft Windows. The “DALI demo” application checks all short addresses to search for connected slaves.

It displays a tab for every connected lamp. On each tab the luminosity and the fade time can be selected. By pressing the button “Set” the new setting will be communicated to the DALI slaves through the DALI master. Selecting another scene makes it possible to store 16 settings.

6.5 The Source Code

The “DALI demo” application is developed using Visual Basic 6.0. It consists of a single Visual Basic Form File “Main”. Each action by the user results in a series of DALI messages. See **Table 6-1**.

Table 6-1. DALI Messages Used in the DALI Demo Application

Action	Messages	Values	Answer
Initialization	Ask status	`XX 90 (XX=short address)`	`LL (LL=status)`

Set new	Store data in DTR
	Store DTR as fade time
	Set light level
	Move light level to DTR
	Save DTR as scene
		`A3 WW (WW=fade time)`
		`XX 2E (XX=short address)`
		`XX YY (YY=light level)`
		`FF 21`
		`FF 4Z (Z=scene)`
			-
			-
			-
			-
			-

Change scene	Select scene
	Ask fade time
	Ask light level
		`FF 1Z (Z=scene)`
		`XX A5 (XX=short address)`
		`XX A0`
			-
			`MM (MM=fade time)`
			`NN (NN=light level)`

PC Software

Appendix A. DALI Instruction Set

The DALI (**D**igital **A**ddressable **L**ighting **I**nterface) protocol specifies a number of commands that a DALI unit should recognize. They consist of two bytes where normally the first byte indicates the address of the receiver and the second byte contains the actual command. These commands are listed in **Table A-1**.

Some special commands use both bytes. Since these commands do not have an address byte, they are broadcasted to every unit. The special commands are listed in **Table A-2**.

NOTE: *In **Table A-1** and **Table A-2**:*

- *The DTR (**D**ata **T**ransfer **R**egister) is a memory location in the DALI unit used for temporary data transfer.*
- *The answer YES corresponds to the hexadecimal value FF. If no answer is received it will be interpreted as the answer NO.*

Table A-1. Standard Commands

Command Value	Description	Answer
00	Extinguish the lamp without fading	–
01	Dim up 200 ms using the selected fade rate	–
02	Dim down 200 ms using the selected fade rate	–
03	Set the actual arc power level one step higher without fading	–
04	Set the actual arc power level one step lower without fading	–
05	Set the actual arc power level to the maximum value	–
06	Set the actual arc power level to the minimum value	–
07	Set the actual arc power level one step lower without fading	–
08	Set the actual arc power level one step higher without fading	–
10+Scene	Set the light level to the value stored for the selected scene	–
20	Reset the parameters to default settings	–
21	Store the current light level in the DTR	–
2A	Store the value in the DTR as the maximum level	–
2B	Store the value in the DTR as the minimum level	–
2C	Store the value in the DTR as the system failure level	–
2D	Store the value in the DTR as the power on level	–
2E	Store the value in the DTR as the fade time	–
2F	Store the value in the DTR as the fade rate	–
40+Scene	Store the value in the DTR as the selected scene	–
50+Scene	Remove the selected scene from the slave unit	–
60+Group	Add the slave unit to the selected group	–
70+Group	Remove the slave unit from the selected group	–
80	Store the value in the DTR as a short address	–
90	Returns the status of the slave as XX	XX
91	Check if the slave is working	YES/NO
92	Check if there is a lamp failure	YES/NO
93	Check if the lamp is operating	YES/NO
94	Check if the slave has received a level out of limit	YES/NO

Table A-1. Standard Commands (Continued)

Command Value	Description	Answer
95	Check if the slave is in reset state	YES/NO
96	Check if the slave is missing a short address	YES/NO
97	Returns the version number as XX	XX
98	Returns the content of the DTR as XX	XX
99	Returns the device type as XX	XX
9A	Returns the physical minimum level as XX	XX
9B	Check if the slave is in power failure mode	YES/NO
A0	Returns the current light level as XX	XX
A1	Returns the maximum allowed light level as XX	XX
A2	Returns the minimum allowed light level as XX	XX
A3	Return the power up level as XX	XX
A4	Returns the system failure level as XX	XX
A5	Returns the fade time as X and the fade rate as Y	XY
B0+Scene	Returns the light level XX for the selected scene	XX
C0	Returns a bit pattern XX indicating which group (0-7) the slave belongs to	XX
C1	Returns a bit pattern XX indicating which group (8-15) the slave belongs to	XX
C2	Returns the high bits of the random address as HH	HH
C3	Return the middle bit of the random address as MM	MM
C4	Returns the lower bits of the random address as LL	LL

Table A-2. Special Commands

Special Command Value	Description	Answer
A1 00	All special mode processes shall be terminated	–
A3 XX	Store value XX in the DTR	–
A5 XX	Initialize addressing commands for slaves with address XX	–
A7 00	Generate a new random address	–
A9 00	Compare the random address with the search address	–
AB 00	Withdraw from the compare process	–
B1 HH	Store value HH as the high bits of the search address	–
B3 MM	Store value MM as the middle bits of the search address	–
B5 LL	Store value LL as the lower bits of the search address	–
B7 XX	Program the selected slave with short address XX	–
B9 XX	Check if the selected slave has short address XX	YES/NO
BB 00	The selected slave returns its short address XX	XX
BD 00	Go into physical selection mode	–

Appendix B. DALI Master Unit Schematic and Layout

This appendix includes:

- DALI master schematic — **Figure B-1**
- DALI master layout — **Figure B-2**

Refer to **Appendix D. DALI Slave Unit Schematic and Layout** for the mater unit schematic and layout information.

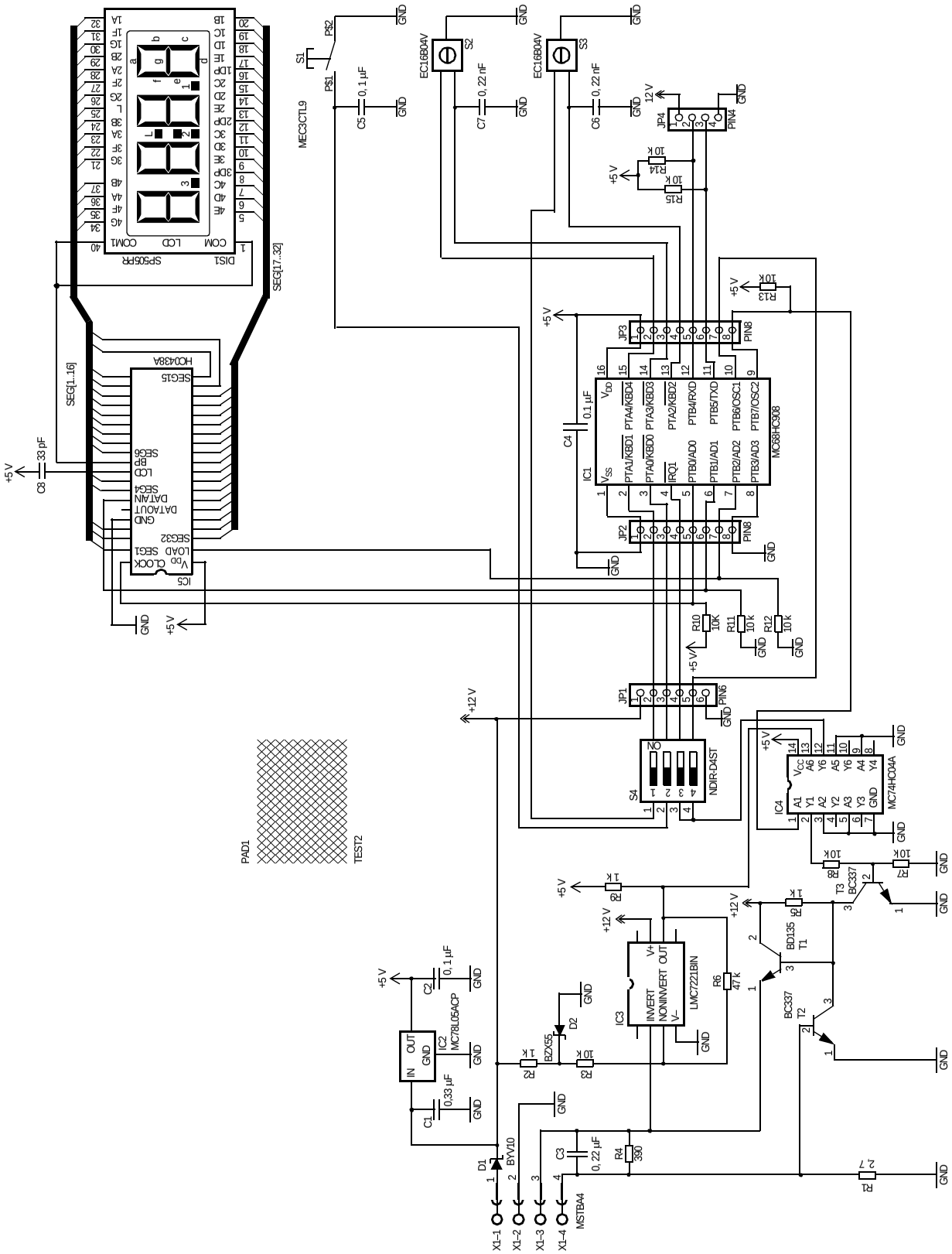

Figure B-1. DALI Master Schematic (Sheet 1 of 2)

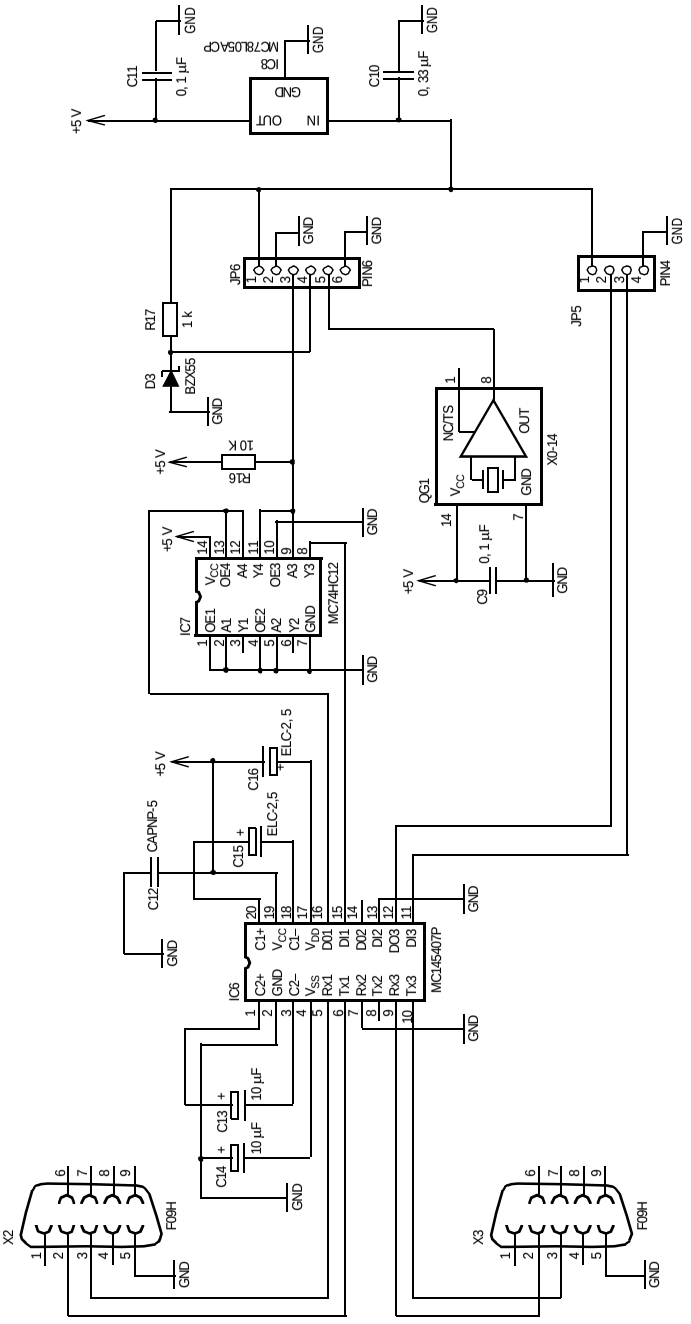

Figure B-1. DALI Master Schematic (Sheet 2 of 2)

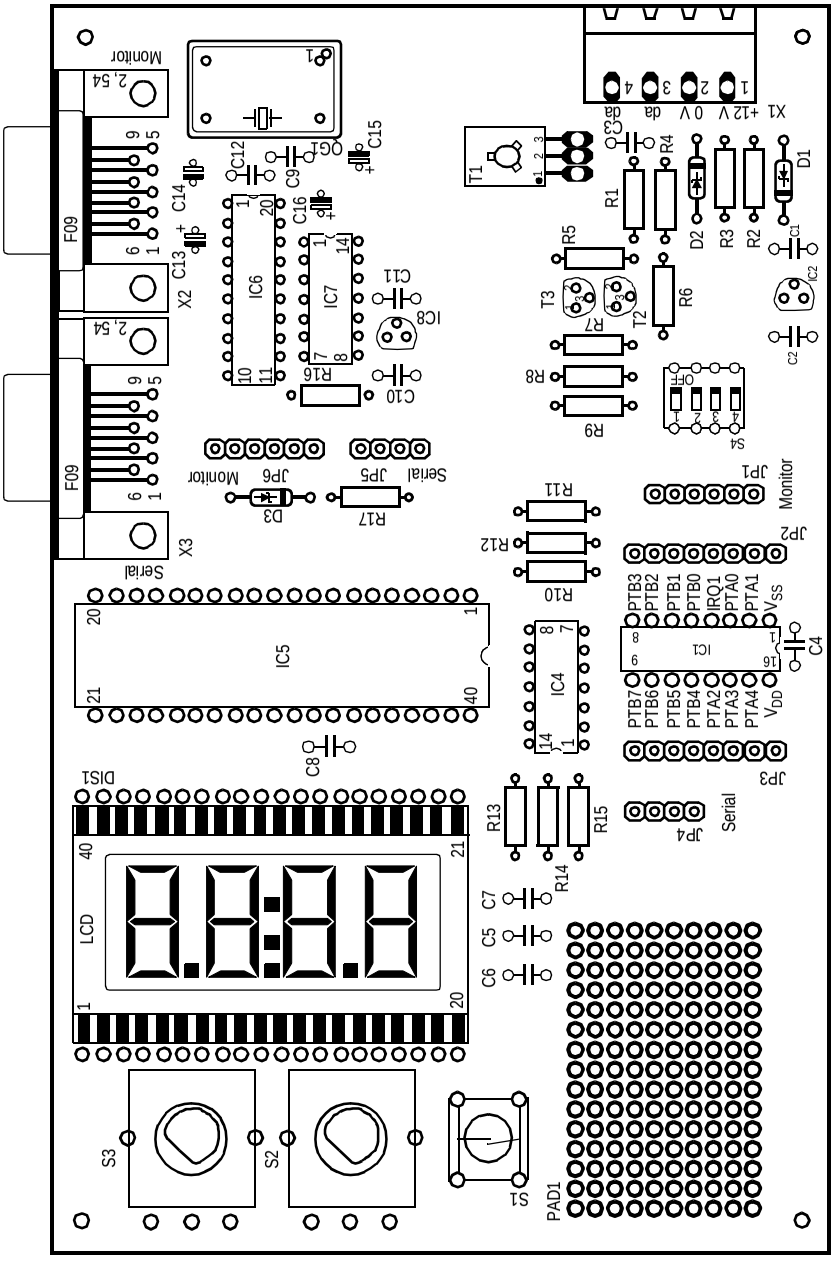

Figure B-2. DALI Master Layout

Appendix C. DALI Master Unit Bill of Materials

This appendix provides a bill of materials for the DALI (**D**igital **A**ddressable **L**ighting **I**nterface) master unit.

Refer to **Appendix E. DALI Slave Bill of Materials** for the slave unit bill of materials.

Table C-1. DALI Master Bill of Materials (Sheet 1 of 3)

Id.	Type	Description	Package	Supplier
IC1	MC68HC908KX8	Processor	DIP16	Motorola
IC2	MC78L05ACP	Regulator	TO92	Motorola/ON
IC3	LMC7221BIN	Comparator	DIP8	National
IC4	MC74HC04N	Inverter	DIP14	Motorola/ON
IC5	HC0438A	LCD driver	DIP40	Hughes
IC6	MC145407P	RS232 driver	DIP20	Motorola/ON
IC7	MC74HC125N	3-state buffer	DIP14	Motorola/ON
IC8	MC78L05ACP	Regulator	TO92	Motorola/ON
QG1	IQXO-350C, 9.8304 MHz	Clock oscillator	DIP14	IQD
D1	BYV10-30	Schottky diode	SOD81	Philips
D2	BZX55/C10	Zener diode	DO35	Temic
D3	BZX55/C8V2	Zener diode	DO35	Temic
T1	BD135	Power transistor	TO125	Motorola/ON
T2	BC337	Transistor	TO92	Philips
T3	BC337	Transistor	TO92	Philips
DIS1	SP505PR	LCD	DIP40	Seiko
R1	2.7 Ω, 0.6 W	Resistor	Spacing 10 mm	
R2	1 kΩ, 0.6 W	Resistor	Spacing 10 mm	

Table C-1. DALI Master Bill of Materials (Sheet 2 of 3)

Id.	Type	Description	Package	Supplier
R3	10 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R4	390 Ω, 0.6 W	Resistor	Spacing 10 mm	
R5	1 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R6	47 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R7	10 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R8	10 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R9	1 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R10	10 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R11	10 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R12	10 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R13	10 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R14	10 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R15	10 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R16	10 kΩ, 0.6 W	Resistor	Spacing 10 mm	
R17	1 kΩ, 0.6 W	Resistor	Spacing 10 mm	
C1	Ceramic 0.33 µF, 63 V	Capacitor	Spacing 5 mm	
C2	Ceramic 0.1 µF, 63 V	Capacitor	Spacing 5 mm	
C3	Ceramic 0.22 µF, 63 V	Capacitor	Spacing 5 mm	
C4	Ceramic 0.1 µF, 63 V	Capacitor	Spacing 5 mm	
C5	Ceramic 2200 pF, 63 V	Capacitor	Spacing 5 mm	
C6	Ceramic 2200 pF, 50 V	Capacitor	Spacing 5 mm	
C7	Ceramic 2200 pF, 50 V	Capacitor	Spacing 5 mm	
C8	Ceramic 33 pF, 50 V	Capacitor	Spacing 5 mm	
C9	Ceramic 0.1 µF, 63 V	Capacitor	Spacing 5 mm	
C10	Ceramic 0.33 µF, 63 V	Capacitor	Spacing 5 mm	
C11	Ceramic 0.1 µF, 63 V	Capacitor	Spacing 5 mm	
C12	Ceramic 0.33 µF, 63 V	Capacitor	Spacing 5 mm	
C13	Electrolytic 10 µF, 16 V	Capacitor	Spacing 2.5 mm	

Table C-1. DALI Master Bill of Materials (Sheet 3 of 3)

Id.	Type	Description	Package	Supplier
C14	Electrolytic 10 μF, 16 V	Capacitor	Spacing 2.5 mm	
C15	Electrolytic 10 μF, 16 V	Capacitor	Spacing 2.5 mm	
C16	Electrolytic 10 μF, 16 V	Capacitor	Spacing 2.5 mm	
S1	Multimec 3CTL9	Push-button	Hole	MEC
S2	EC16B24204	Shaft encoder	Hole	Alps
S3	EC16B24204	Shaft encoder	Hole	Alps
S4	NDIR-04ST, 4-p	DIP-switch	Hole	APEM
JP1	MTA100/156, 6-p	Connector	2.54 mm, straight	AMP
JP2	Not mounted			
JP3	Not mounted			
JP4	MTA100/156, 4-p	Connector	2.54 mm, straight	AMP
JP5	MTA100/156, 4-p	Connector	2.54 mm, straight	AMP
JP6	MTA100/156, 6-p	Connector	2.54 mm, straight	AMP
X1	MSTBA 2.5/4-G-5.08	Connector	5.08 mm	Phoenix
X2	D-sub, 9-p, female, 90°	Connector	Hole	
X3	D-sub, 9-p, female, 90°	Connector	Hole	
—	FI-349/30/SE15K/W	Heat sink		Alutronic

DALI Master Unit Bill of Materials

Appendix D. DALI Slave Unit Schematic and Layout

This appendix includes:

- DALI slave schematic — **Figure D-1**
- DALI slave layout — **Figure D-2**

Refer to **Appendix B. DALI Master Unit Schematic and Layout** for the mater unit schematic and layout information.

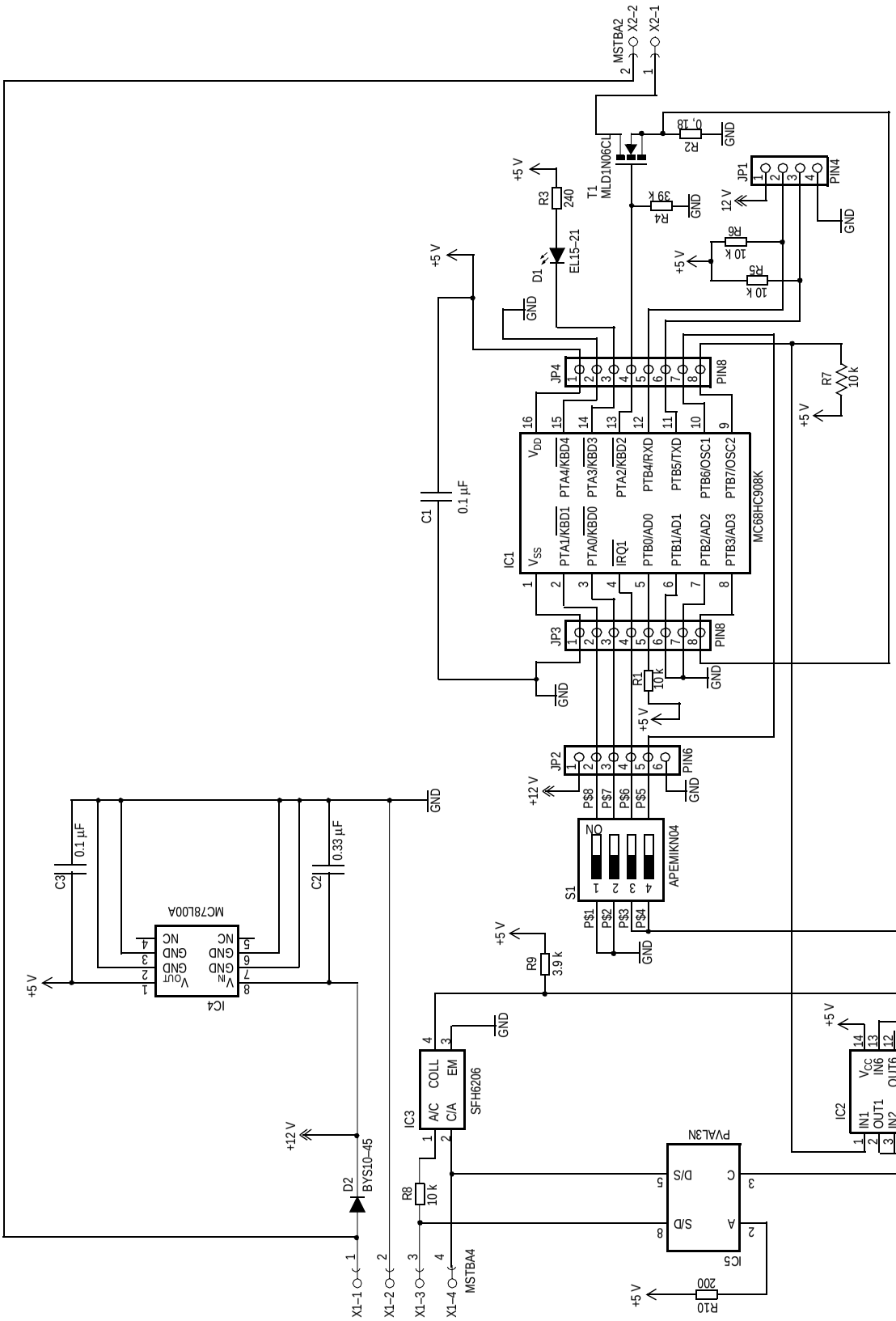

Figure D-1. DALI Slave Schematic

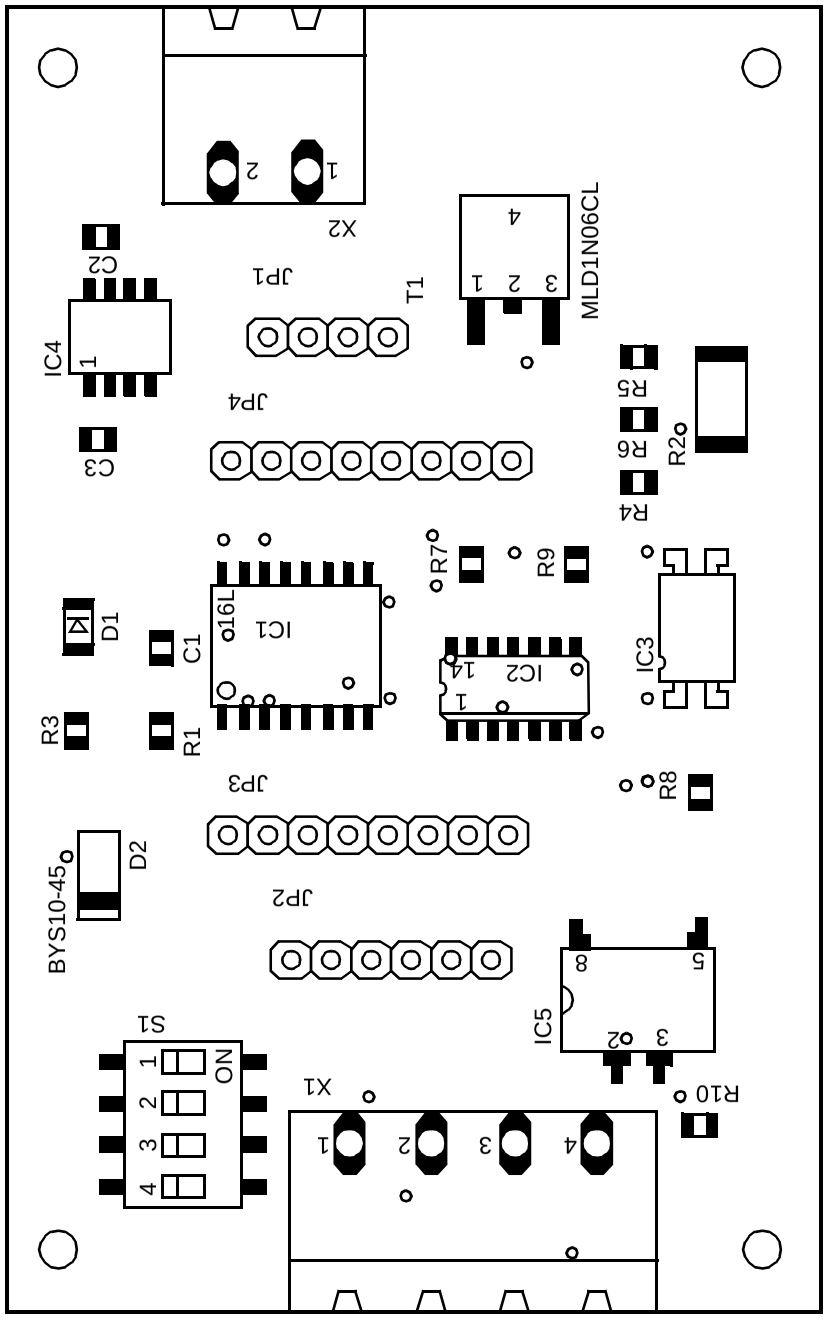

Figure D-2. DALI Slave Layout

DALI Slave Unit Schematic and Layout

Appendix E. DALI Slave Bill of Materials

This appendix provides a bill of materials for the DALI (**D**igital **A**ddressable **L**ighting **I**nterface) master unit.

Refer to **Appendix C. DALI Master Unit Bill of Materials** for the master unit bill of materials.

Table E-1. DALI Slave Bill of Materials

Id.	Type	Description	Package	Supplier
IC1	MC68HC908KX8	Processor	SO16	Motorola
IC2	MC74AC14	Inverter	SO14	Motorola/ON
IC3	SFH6206-3	Opto coupler	SMD	Infineon
IC4	MC78L05AC	Regulator	SO8	Motorola/ON
IC5	PVA1354NS	Opto relay	SMD	IR (Optorelä)
D1	EL15-21SYGC	LED	1206	Everlight
D2	BYS10-45	Shottky diode	SOD106A	Temic
T1	MLD1N06CL	Power transistor	DPAK	Motorola/ON
R1	10 kΩ, 1/8 W	Resistor	0805	
R2	0.18 Ω, 1 W	Resistor	2512	Megitt
R3	240 Ω, 1/8 W	Resistor	0805	
R4	39 kΩ, 1/8 W	Resistor	0805	
R5	10 kΩ, 1/8 W	Resistor	0805	
R6	10 kΩ, 1/8 W	Resistor	0805	
R7	10 kΩ, 1/8 W	Resistor	0805	
R8	10 kΩ, 1/8 W	Resistor	0805	
R9	3.9 kΩ, 1/8 W	Resistor	0805	
R10	200 Ω, 1/8 W	Resistor	0805	

Table E-1. DALI Slave Bill of Materials (Continued)

Id.	Type	Description	Package	Supplier
C1	Ceramic 0.1 μF, 25 V	Capacitor	0805, GRM40	Murata
C2	Ceramic 0.33 μF, 16 V	Capacitor	0805, GRM40	Murata
C3	Ceramic 0.1 μF, 25 V	Capacitor	0805, GRM40	Murata
S1	IKN0403001, 4-p	DIP-switch	SMD	APEM
X1	MSTBA 2.5/4-G-5.08	Connector	5.08 mm	Phoenix
X2	MSTBA 2.5/2-G-5.08	Connector	5.08 mm	Phoenix
JP1	MTA100/156, 4-p	Connector	2.54 mm, straight	AMP
JP2	MTA100/156, 6-p	Connector	2.54 mm, straight	AMP
JP3	Not mounted			
JP4	Not mounted			

Appendix F. DALI Master Unit Source Code Files

F.1 Contents

F.2 Master: common.h

```
// common.h
//
// This header contains common definitions
//
// Prepared: Motorola AB
//
// Functional level: Common
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// P1B  010201  Added declaration for embedded ROM flash routines
// R1A  010212  Released version

#ifndef COMMON
#define COMMON

#define TRUE 1
#define FALSE 0

// Global definition

extern unsigned char address;   // The DALI address
extern unsigned char command;   // The DALI command

extern unsigned char answer;    // The DALI answer

extern unsigned char error;     // Stores the last error code

#define SRSR_ERROR  0x01        // Reset due to a hardware/software error

extern unsigned int flag;       // Keep track of all events

#define NEW_DATA     0x01       // Set when new data is set
#define SEND_DATA    0x02       // Set when data is to be sent
#define ERROR_EVENT  0x04       // Set if an error has occurred
#define ANSWER_EVENT 0x08       // Set if an answer has arrived
#define DEMO_MODE    0x10       // Set if the demo mode is active

// Macro definitions
#define ei() _asm("cli")        // Enable interrupt
#define di() _asm("sei")        // Disable interrupt
```

```
// Declarations for the embedded ROM flash routines
#define CTRLBYT (unsigned char *)0x0048
// Address to a variable containing a erase flag
#define CPUSPD  (unsigned char *)0x0049
// Address to a variable containing the cpu speed
#define LADDR   (unsigned int *)0x004A
// Address to a variable containing the last address to write
#define DATA    (unsigned char *)0x004C
// Address to the data array to be written into flash

#define clock_write() _asm("ldhx #$FDFF\njsr $1009\n")

#endif
```

F.3 Master: cpu.h

```
// cpu.h
//
// This header contains definitions for the cpu module
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// R1A  010212  Released version

#ifndef CPU
#define CPU

// Initialize
extern void cpu_Init(void);

// Interrupt handler for reset
extern @interrupt void _stext(void);

// Trap lost interupts
extern @interrupt void cpu_Trap(void);

#endif
```

F.4 Master: cpu.c

```
// cpu.c
//
// This module handles all cpu related tasks
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1B
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// P1B  000201  Added automatic clock calibration
// R1A  010212  Released version
// R1B  011016  Adapted to new clock speed

#include "common.h"
#include "iokx8.h"

#define TRIM_FACTOR_ADDR (unsigned char *)0xFDFF
#define TRIM_FACTOR *(TRIM_FACTOR_ADDR)  // Internal clock generator trim factor

// Initialize
void cpu_Init(void)
{
    unsigned char trimmed;
    unsigned char ser_rec;
    unsigned char dali_rec;

    // Initialize LVI to 5V
    if (TRIM_FACTOR!=0xFF)
    {
        // Initialize LVI to 5V
        CONFIG2 |= 0x08;
    }
    else
    {
        // Initialize LVI to 5V and disable COP
        CONFIG2 |= 0x09;
    }

    // Initialize ports
    DDRA = 0x00;    // Initialize PTA0 to PTA4 as input registers
    PTAPUE = 0x1F;  // Enable internal pullup on PTA0 to PTA4
    PTB = 0xA1;     // Set initial values for PTB
    DDRB = 0xA7;    // Initialize PTB0, PTB1, PTB2, PTB5 and PTB7 as output register
```

```
    // Mask IRQ1 interrupt
    ISCR = 0x06;

    // Change internal clock speed to simplify baud rate selection
    ICGMR = 0x40;   // Results in 19.6608 MHz

    // Trim internal clock (if a trim value is stored in flash)
    if (TRIM_FACTOR!=0xFF)
    {
        ICGTR = TRIM_FACTOR;
    }
    else
    {
        // Clock is not trimmed
        while ((PTB & 0x50)!=0x50)  // Wait until the signals are stable
            ;
        // Initialize SCI for transmission
        ICGTR = 0x40;   // Start with a low value
        SCBR = 0x06;    // Baud rate 4800
        SCC1 = 0x40;    // Enable SCI
        SCC2 = 0x08;    // Transmitter enabled
        ser_rec = 0x10;
        dali_rec = 0x40;
        trimmed = FALSE;
        while (trimmed==FALSE)  // Loop forever
        {
           while ((SCS1 & 0x80)==0x00)        // Wait until transmit buffer is empty
               ;
            SCDR = 0x55;                        // Bit pattern
            if ((PTB & 0x10)==0x00 && ser_rec!=0x00)
            {
                (ICGTR)++;
            }
            ser_rec = PTB & 0x10;
            if ((PTB & 0x40)==0x00 && dali_rec!=0x00)
            {
                  *CTRLBYT = 0x00;                          // Page erase
                  *CPUSPD = 0x4F;                           // 4 * 19.6608
                  *LADDR = (unsigned int)TRIM_FACTOR_ADDR;  // Last address in the
                                                               flash memory
                *DATA = ICGTR;
                clock_write();
                if (TRIM_FACTOR==ICGTR)
                {
                    trimmed = TRUE;
                }
            }
            dali_rec = PTB & 0x40;
        }
    }
}
```

```
// Trap lost interupts
@interrupt void cpu_Trap(void)
{
    while (1==1)
        ;
}
```

F.5 Master: dali.h

```
// dali.h
//
// This header contains definitions for the dali module
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001110  Initial version
// R1A  010212  Released version

#ifndef DALI
#define DALI

// Initialize
extern void dali_Init(void);

// Sends the DALI address and command on the dali port
extern void dali_SendData(void);

// Interrupt handler for incoming data on the dali port
extern @interrupt void dali_Start(void);

// Interrupt handler for timebase module
extern @interrupt void dali_Tick(void);

#endif
```

F.6 Master: dali.c

```
// dali.c
//
// This module handles all tasks related to the dali port
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1B
//
// Rev  Date    Reason/description
// P1A  001110  Initial version
// R1A  010212  Released version
// R1B  011010  Adapted to new clock speed, Reception improved to be more tolerant
//
// The timebase module (TBM) is used

#include "common.h"
#include "iokx8.h"

// Address
#define BROADCAST 0xFF                  // Broadcast address for the DALI network

// Normal commands
#define GO_TO_SCENE 0x10                // Command for changing scene
#define DEMO 0xFF                       // Command for starting the demo mode (not
included in the DALI standard)

// Special commands
#define INITIALISE 0xA5                 // Command for starting initialization mode
#define RANDOMISE 0xA7                  // Command for generating a random address

static unsigned char send_position;     // Position in the data to transfer
static unsigned char send_active;       // True if transfer has started
static unsigned char send_value;        // Holds the logic level to transfer

static unsigned char rec_position;      // Position in the data to receive
static unsigned char rec_active;        // True if reception has started
static unsigned char rec_value;         // Holds the received logic level
static unsigned char rec_bit;           // Number of received bits

static unsigned char demo_scene;        // Holds the current scene
static unsigned int demo_tick;          // Number of ticks since last scene change

static unsigned char repeat_active;     // True if a command shall be repeated
```

```
// Initialize
void dali_Init(void)
{
    send_active = FALSE;
    rec_active = FALSE;
    repeat_active = FALSE;
    TBCR = 0x2E;    // Interrupt 9600 times/second
    ISCR = 0x04;    // Enable IRQ1 interrupt
}

// Sends the DALI address and command on the dali port
void dali_SendData(void)
{
    // Check if it is the special command for starting the demo mode
    if (address==BROADCAST && command==DEMO)
    {
        demo_tick = 0;
        demo_scene = 0;
        address = BROADCAST;
        command = GO_TO_SCENE;
        flag |= DEMO_MODE | NEW_DATA;
    }

    // Check if some commands shall be repeated
    if (repeat_active==TRUE)
    {
        repeat_active = FALSE;
    }
    else
    {
        if (address & 0x01)
        {
            // Command
            if ((address & 0xE0)==0xA0 || (address & 0xE0)==0xC0)
            {
                // Special command
                if (address==INITIALISE || address==RANDOMISE)
                {
                    // Command shall be repeated within 100 ms
                    repeat_active = TRUE;
                }
            }
            else
            {
                // Normal command
                if (command>=0x20 && command<=0x80)
                {
                    // Command shall be repeated within 100 ms
                    repeat_active = TRUE;
                }
            }
        }
    }
```

```
// Wait until dali port is idle
    while (send_active || rec_active)
        ;
    ISCR = 0x06;        // Disable IRQ1 interrupt
    send_value = 0x00;
    send_position = 0;
    send_active = TRUE; // Activate the timer module to transfer
}

// Interrupt handler for incoming data on the dali port
@interrupt void dali_Start(void)
{
    ISCR = 0x06;        // Disable IRQ1 interrupt
    answer = 0x00;      // Clear answer
    rec_bit = 0;        // No bit has been received
    rec_value = 0x00;   // Value is low when starting to receive
    rec_position = 0;
    rec_active = TRUE;  // Activate the timer module to receive
}

// Interrupt handler for timebase module
@interrupt void dali_Tick(void)
{
    unsigned char temp_value;

    TBCR |= 0x08;   // Acknowledge the interrupt
    if (rec_active==TRUE)
    {
        temp_value = PTB & 0x40;
        rec_position++;
        if (temp_value!=rec_value)
        {
            // An edge has been detected
            switch (rec_bit)
            {
            case 0:
                // Start bit
                rec_bit++;
                rec_position = 0;
                break;
            case 9:
                // First stop bit
                if (rec_position>6)
                {
                    // Stop bit error, no edge should exist, stop receiving
                    rec_active = FALSE;
                    ISCR = 0x04;    // Enable IRQ1 interrupt
                }
                break;
            case 10:
                // Second stop bit
                // Stop bit error, no edge should exist, stop receiving
                rec_active = FALSE;
```

```
            ISCR = 0x04;    // Enable IRQ1 interrupt
            break;
        default:
            // The address and command bits
            if (rec_position>6)
            {
                // Store the values
                if (temp_value)
                {
                    answer |= (1<<(8-rec_bit));
                }
                rec_bit++;
                rec_position = 0;
            }
            break;
        }
        rec_value = temp_value;
    }
    else
    {
        // Signal level stable
        switch (rec_bit)
        {
        case 0:
            // Start bit
            if (rec_position==8)
            {
                // Start bit error, too long delay before edge, stop receiving
                rec_active = FALSE;
                ISCR = 0x04;    // Enable IRQ1 interrupt
            }
            break;
        case 9:
            // First stop bit
            if (rec_position==8)
            {
                if (temp_value==0)
                {
                    // Stop bit error, wrong level, stop receiving
                    rec_active = FALSE;
                    ISCR = 0x04;    // Enable IRQ1 interrupt
                }
                else
                {
                    rec_bit++;
                    rec_position = 0;
                }
            }
            break;
        case 10:
            // Second stop bit
            if (rec_position==8)
            {
```

```
                // Receive ready
                flag |= ANSWER_EVENT;
                rec_active = FALSE;
                ISCR = 0x04;    // Enable IRQ1 interrupt
            }
            break;
        default:
            // The address and command bits
            if (rec_position==10)
            {
                // Data bit error, too long delay before edge, stop receiving
                rec_active = FALSE;
                ISCR = 0x04;    // Enable IRQ1 interrupt
            }
            break;
        }
    }
}
if (send_active==TRUE)
{
    if ((send_position & 0x03)==0)
    {
        PTB = (PTB & ~0x80) | send_value;
        if (send_position==0)
        {
            send_value = 0x80;  // Second half of start bit
        }
        if (send_position>=4 && send_position<=128)
        {
            // Extract bit level
            // Check if address or command
            if (send_position<68)
            {
                // Address
                temp_value = (address>>((64-send_position)/8)) & 0x01;
            }
            else
            {
                // Command
                temp_value = (command>>((128-send_position)/8)) & 0x01;
            }
            // Check if first or second half of data bit
            if (send_position & 0x04)
            {
                // First half
                if (temp_value==0x00)
                {
                    send_value = 0x80;
                }
                else
                {
                    send_value = 0x00;
                }
```

```
                }
                else
                {
                    // Second half
                    if (temp_value==0x00)
                    {
                        send_value = 0x00;
                    }
                    else
                    {
                        send_value = 0x80;
                    }
                }
            }
            if (send_position==132)
            {
                send_value = 0x80; // Start of stop bit and settling time
            }
            if (send_position==160)
            {
                ISCR = 0x04;    // Enable IRQ1 interrupt
            }
            if (send_position==236)
            {
                send_active = FALSE;
                if (repeat_active==TRUE)
                {
                    flag |= SEND_DATA;
                }
            }
        }
        send_position++;
    }
    if (flag & DEMO_MODE)
    {
        demo_tick++;
        // Change scene after 3 seconds
        if (demo_tick==28800)
        {
            demo_tick = 0;
            demo_scene++;
            if (demo_scene>15)
            {
                demo_scene = 0;
            }
            address = BROADCAST;
            command = GO_TO_SCENE+demo_scene;
            flag |= NEW_DATA | SEND_DATA;
        }
    }
}
```

F.7 Master: dali.lkf

```
#   LINK COMMAND FILE Example for 68HC908KX8
#   Copyright (c) 2000 by Motorola AB

+seg .text -b 0xE000 -m0x1E00 -nCODE -sROM  # program start address
+seg .const -a CODE -it -sROM               # constants and strings
+seg .bsct -b 0x40 -m0xC0 -nZPAGE -sRAM     # zero page start address
+seg .ubsct -a ZPAGE -nUZPAGE -sRAM
+seg .data -a UZPAGE -nIDATA -sRAM          # data start address
+seg .bss -a IDATA -nUDATA -sRAM
crtsi.o                     # startup routine
main.o                      # application program
lcd.o
cpu.o
keys.o
rs232.o
dali.o
"C:\COSMIC\EVAL08\Lib\libi.h08"
"C:\COSMIC\EVAL08\Lib\libm.h08"

+seg .const -b 0xFFDC       # vectors start address
vector.o                    # interrupt vectors

+def __memory=@.bss         # symbol used by library
+def __stack=0x0FF          # stack pointer value
```

F.8 Master: iokx8.h

```
/*  IO DEFINITIONS FOR MC68HC908KX8
 *  Copyright (c) 2000 by Motorola AB
 *  Based on definition from Cosmic
 */

#define uint    unsigned int

/*  PORTS section
 */
@tiny volatile  char PTA      @0x00;    /* port A */
@tiny volatile  char PTB      @0x01;    /* port B */
@tiny volatile  char DDRA     @0x04;    /* data direction port A */
@tiny volatile  char DDRB     @0x05;    /* data direction port B */
@tiny volatile  char PTAPUE   @0x0d;    /* pull-up enable port A */

/*  SCI section
 */
@tiny volatile  char SCC1     @0x13;    /* SCI control register 1 */
@tiny volatile  char SCC2     @0x14;    /* SCI control register 2 */
@tiny volatile  char SCC3     @0x15;    /* SCI control register 3 */
@tiny volatile  char SCS1     @0x16;    /* SCI status register 1 */
@tiny volatile  char SCS2     @0x17;    /* SCI status register 2 */
@tiny volatile  char SCDR     @0x18;    /* SCI data register */
@tiny volatile  char SCBR     @0x19;    /* SCI baud rate */

/*  KEYBOARD section
 */
@tiny volatile  char KBSCR    @0x1a;
/* keyboard status/control register */
@tiny volatile  char KBIER    @0x1b;    /* keyboard interrupt enable */

/*  TIMEBASE section
 */
@tiny volatile  char TBCR     @0x1c;    /* timebase module control */

/*  IRQ section
 */
@tiny volatile  char ISCR     @0x1d;    /* IRQ status/control register */

/*  CONFIG section
 */
@tiny volatile  char CONFIG1  @0x1e;    /* configuration register 1 */
@tiny volatile  char CONFIG2  @0x1f;    /* configuration register 2 */

/*  TIMER section
 */
@tiny volatile  char TSC      @0x20;    /* timer status/ctrl register */
@tiny volatile  uint TCNT     @0x21;    /* timer counter register */
```

```
@tiny volatile  char TCNTH    @0x21;    /* timer counter high */
@tiny volatile  char TCNTL    @0x22;    /* timer counter low */
@tiny volatile  uint TMOD     @0x23;    /* timer modulo register */
@tiny volatile  char TMODH    @0x23;    /* timer modulo high */
@tiny volatile  char TMODL    @0x24;    /* timer modulo low */
@tiny volatile  char TSC0     @0x25;    /* timer chan 0 status/ctrl */
@tiny volatile  uint TCH0     @0x26;    /* timer chan 0 register */
@tiny volatile  char TCH0H    @0x26;    /* timer chan 0 high */
@tiny volatile  char TCH0L    @0x27;    /* timer chan 0 low */
@tiny volatile  char TSC1     @0x28;    /* timer chan 1 status/ctrl */
@tiny volatile  uint TCH1     @0x29;    /* timer chan 1 register */
@tiny volatile  char TCH1H    @0x29;    /* timer chan 1 high */
@tiny volatile  char TCH1L    @0x2a;    /* timer chan 1 low */

/*  ICG section
 */
@tiny volatile  char ICGCR    @0x36;    /* ICG control register */
@tiny volatile  char ICGMR    @0x37;    /* ICG multiplier register */
@tiny volatile  char ICGTR    @0x38;    /* ICG trim register */
@tiny volatile  char ICGDVR   @0x39;    /* ICG divider control register */
@tiny volatile  char ICGDSR   @0x3a;    /* ICG stage control register */

/*  A/D section
 */
@tiny volatile  char ADSCR    @0x3c;    /* A/D status/ctrl register */
@tiny volatile  char ADR      @0x3d;    /* A/D data register */
@tiny volatile  char ADICLK   @0x3e;    /* A/D input clock register */

/*  SIM section
 */
@near volatile  char SBSR     @0xfe00;  /* SIM break status register */
@near volatile  char SRSR     @0xfe01;  /* SIM reset status register */
@near volatile  char SBFCR    @0xfe03;  /* SIM break control register */

/*  INTERUPT section
 */
@near volatile  char INT1     @0xfe04;  /* INTERUPT status register 1 */
@near volatile  char INT2     @0xfe05;  /* INTERUPT status register 2 */
@near volatile  char INT3     @0xfe06;  /* INTERUPT status register 3 */

/*  FLASH section
 */
@near volatile  char FLTCR    @0xfe07;  /* FLASH test control register */
@near volatile  char FLCR     @0xfe08;  /* FLASH control register */
@near volatile  char FLBPR    @0xff7e;  /* FLASH block protect register */

/*  BREAK section
 */
@near volatile  char BRKAR    @0xfe02;  /* BREAK auxiliary register */
@near volatile  uint BRK      @0xfe09;  /* BREAK address register */
@near volatile  char BRKH     @0xfe09;  /* BREAK address register low */
```

```
@near volatile  char BRKL     @0xfe0a;  /* BREAK address register high */
@near volatile  char BRKSCR   @0xfe0b;  /* BREAK status/ctrl register */

/*  LVI section
 */
@near volatile  char LVISR    @0xfe0c;  /* LVI status register */

/*  COP section
 */
@near volatile  char COPCTL   @0xffff;  /* COP control register */

#undef uint
```

F.9 Master: keys.h

```
// keys.h
//
// This header contains definitions for the keys module
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001025  Initial version
// R1A  010212  Released version

#ifndef KEYS
#define KEYS

// Initialize
extern void keys_Init(void);

// Interrupt handler for send button
extern @interrupt void keys_SendButton(void);

// Interrupt handler for command pulse sender
extern @interrupt void keys_CommandPulse(void);

// Interrupt handler for address pulse sender
extern @interrupt void keys_AddressPulse(void);

#endif
```

F.10 Master: keys.c

```
// keys.c
//
// This module handles all tasks related to the keys
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001025  Initial version
// R1A  010212  Released version
//
// The keyboard interrupt module (KBI) and the timer interface module (TIM)
   are used

#include "common.h"
#include "iokx8.h"

// Initialize
void keys_Init(void)
{
    // Keyboard Interrupt Module
    KBIER = 0x01;   // Interrupt enabled on PTA0
    KBSCR = 0x04;
    // Interrupt request not masked, Clear interrupt request, Falling edges
       sensitivity
    // Timer Interface Module
    TSC = 0x00;
    // Start timer (only interrupts are used, not the captured values)
    TSC0 = 0x48;    // Interrupt enabled, Input capture on falling edge
    TSC1 = 0x48;    // Interrupt enabled, Input capture on falling edge
}

// Interrupt handler for send button
@interrupt void keys_SendButton(void)
{
    flag |= SEND_DATA;
    flag &= ~DEMO_MODE; // Clear demo mode
}

// Interrupt handler for command pulse sender
@interrupt void keys_CommandPulse(void)
{
```

```
    TSC1 &= ~0x80;   // Clear flag bit
    // Check in which direction the pulse sender is rotated
    if (PTA & 0x10)
    {
        command++;
    }
    else
    {
        command--;
    }
    flag |= NEW_DATA;
}

// Interrupt handler for address pulse sender
@interrupt void keys_AddressPulse(void)
{
    TSC0 &= ~0x80;   // Clear flag bit
    // Check in which direction the pulse sender is rotated
    if (PTA & 0x02)
    {
        address++;
    }
    else
    {
        address--;
    }
    flag |= NEW_DATA;
}
```

F.11 Master: lcd.h

```
// lcd.h
//
// This header contains definitions for the lcd module
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// R1A  010212  Released version

#ifndef LCD
#define LCD

// Initialize
extern void lcd_Init(void);

// Print the address and the command on the LCD module
extern void lcd_ShowData(void);

// Print the answer on the LCD module
extern void lcd_ShowAnswer(void);

// Print the error code on the LCD module
extern void lcd_ShowError(void);

#endif
```

F.12 Master: lcd.c

```
// lcd.c
//
// This module handles all lcd related tasks
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// R1A  010212  Released version

#include "common.h"
#include "iokx8.h"

// Translation tables between hexadecimal digits and corresponding bits to
   set on the LCD

#define COLON 0x01000000

const unsigned long digit1[16] =
{
    0x000DE000, 0x00082000, 0x000EC000, 0x000E6000,
    0x000B2000, 0x00076000, 0x0007E000, 0x000C2000,
    0x000FE000, 0x000F2000, 0x000FA000, 0x0003E000,
    0x0005C000, 0x000AE000, 0x0007C000, 0x00078000
};

const unsigned long digit2[16] =
{
    0x00E00E00, 0x00800200, 0x00D00C00, 0x00D00600,
    0x00B00200, 0x00700600, 0x00700E00, 0x00C00200,
    0x00F00E00, 0x00F00200, 0x00F00A00, 0x00300E00,
    0x00600C00, 0x00900E00, 0x00700C00, 0x00700800
};

const unsigned long digit3[16] =
{
    0x1C0000E0, 0x10000020, 0x1A0000C0, 0x1A000060,
    0x16000020, 0x0E000060, 0x0E0000E0, 0x18000020,
    0x1E0000E0, 0x1E000020, 0x1E0000A0, 0x060000E0,
    0x0C0000C0, 0x120000E0, 0x0E0000C0, 0x0E000080
};
```

```
const unsigned long digit4[16] =
{
    0xC000000F, 0x00000003, 0xA000000D, 0xA0000007,
    0x60000003, 0xE0000006, 0xE000000E, 0x80000003,
    0xE000000F, 0xE0000003, 0xE000000B, 0x6000000E,
    0xC000000C, 0x2000000F, 0xE000000C, 0xE0000008
};

// Takes data and sends it to LCD driver
static void lcd_SendData(unsigned long data)
{
    unsigned char index;

    for (index = 0; index<32; index++)
    {
        if (data & 0x01)
        {
            PTB |= 0x04;    // Set DATA high (PTB2)
        }
        else
        {
            PTB &= ~0x04;   // Set DATA low (PTB2)
        }
        data >>= 1;
        PTB &= ~0x01;       // Toggle CLOCK (PTB0)
        PTB |= 0x01;
    }
}

// Initialization
void lcd_Init(void)
{
    PTB |= 0x02;    // Set LOAD high (PTB1)
    lcd_SendData(digit1[0x00] | digit2[0x00] | digit3[0x00] | digit4[0x00] |
                 COLON);
}

// Print the address and the command on the LCD module
void lcd_ShowData(void)
{
    unsigned long result;

    result = digit1[(address & 0xF0) >> 4] | digit2[address & 0x0F];
    // Add digit 1 & 2
    result |= digit3[(command & 0xF0) >> 4] | digit4[command & 0x0F];
    // Add digit 3 & 4
    result |= COLON;
    // Add :
    lcd_SendData(result);
}
```

```
// Print the answer on the LCD module
void lcd_ShowAnswer(void)
{
    unsigned long result;

    result = digit2[0x0A] | digit3[(answer & 0xF0) >> 4] | digit4[answer &
                    0x0F];
    lcd_SendData(result);
}

// Print the error code on the LCD module
void lcd_ShowError(void)
{
    unsigned long result;

    result = digit2[0x0E] | digit3[(error & 0xF0) >> 4] | digit4[error &
                    0x0F];
    lcd_SendData(result);
}
```

F.13 Master: main.c

```
// main.c
//
// This module contains the main function
//
// Prepared: Motorola AB
//
// Functional level: Application
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// R1A  010212  Released version

#include "common.h"
#include "iokx8.h"
#include "cpu.h"
#include "lcd.h"
#include "keys.h"
#include "rs232.h"
#include "dali.h"

// Global variables declaration

unsigned char address;              // The DALI address
unsigned char command;              // The DALI command

unsigned char answer;               // The DALI answer

unsigned char error;                // Stores the last error code

unsigned int flag;                  // Keep track of all events

// Main function
void main(void)
{
    address = 0x00;                 // Initialize when starting
    command = 0x00;                 // Initialize when starting

    answer = 0x00;                  // Initialize when starting

    error = 0x00;                   // Initialize when starting

    flag = 0x00;                    // Initialize when starting

    cpu_Init();                     // Initialize the cpu module
    lcd_Init();                     // Initialize the lcd module
    keys_Init();                    // Initialize the keys module
```

```
    rs232_Init();                   // Initialize the keys module
    dali_Init();                    // Initialize the dali module

    ei();                           // Enable interrupt

    while (1==1)                    // Loop forever
    {
        COPCTL = 0x00;              // Clear COP counter

        // NEW_DATA flag can be set from the keys module,
        // the rs232 module or the dali module
        if (flag & NEW_DATA)
        {
            flag &= ~NEW_DATA;      // Clear NEW_DATA flag
            lcd_ShowData();
            // Update lcd with new address and command
        }

        // SEND_DATA flag can be set from the keys module,
        // the rs232 module or the dali module
        if (flag & SEND_DATA)
        {
            flag &= ~SEND_DATA;     // Clear SEND_DATA flag
            dali_SendData();
            // Send address and command on the dali port
        }

        // ANSWER_EVENT flag can be set from the dali module
        if (flag & ANSWER_EVENT)
        {
            flag &= ~ANSWER_EVENT;  // Clear ANSWER_EVENT flag
            lcd_ShowAnswer();       // Update the lcd with the answer
            rs232_SendAnswer();     // Send the answer on the rs232 port
        }

        // ERROR_EVENT flag can be set from any module
        if (flag & ERROR_EVENT)
        {
            flag &= ~ERROR_EVENT;   // Clear ERROR_EVENT flag
            lcd_ShowError();        // Update the lcd with the error code
        }
    }
}
```

F.14 Master: rs232.h

```
// rs232.h
//
// This header contains definitions for the rs232 module
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001027  Initial version
// R1A  010212  Released version

#ifndef RS232
#define RS232

// Initialize
extern void rs232_Init(void);

// Sends the DALI answer on the rs232 port
extern void rs232_SendAnswer(void);

// Interrupt handler for received data on the rs232 port
extern @interrupt void rs232_ReceiveData(void);

#endif
```

F.15 Master: rs232.c

```
// rs232.c
//
// This module handles all tasks related to the rs232 port
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1B
//
// Rev  Date    Reason/description
// P1A  001027  Initial version
// R1A  010212  Released version
// R1B  011016  Adapted to new clock speed
//
// The serial communication interface module (SCI) is used

#include "common.h"
#include "iokx8.h"

static unsigned char first; // Keeps track if incoming data is address or command

// Initialize
void rs232_Init(void)
{
    SCBR = 0x05;    // Baud rate 9600
    SCC1 = 0x40;    // Enable SCI
    SCC2 = 0x2C;    // Enable receive interrupt, transmitter enabled, receiver
enabled
    first = TRUE;   // Next incoming byte is an address
}

// Sends the answer on the rs232 port
void rs232_SendAnswer(void)
{
    while ((SCS1 & 0x80)==0x00) // Wait until transmit buffer is empty
        ;
    SCDR = answer;
}
```

```
// Interrupt handler for received data on the rs232 port
@interrupt void rs232_ReceiveData(void)
{
    if (SCS1 & 0x20)    // Received data available in SCDR
    {
        if (first==TRUE)
        {
            address = SCDR;
            first = FALSE;      // Next data is a command
        }
        else
        {
            command = SCDR;
            first = TRUE;       // Next data is an address
            flag |= NEW_DATA | SEND_DATA;
            flag &= ~DEMO_MODE; // Clear demo mode
        }
    }
}
```

F.16 Master: vector.c

```
// vector.c
//
// This module contains the interrupt vector
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// R1A  010212  Released version

#include "cpu.h"
#include "keys.h"
#include "rs232.h"
#include "dali.h"

void (* const _vectab[])() =
{
    dali_Tick,          // Timebase module
    cpu_Trap,           // ADC conversion complete
    keys_SendButton,    // Keyboard
    cpu_Trap,           // SCI transmit
    rs232_ReceiveData,  // SCI receive
    cpu_Trap,           // SCI receive error
    cpu_Trap,           // Reserved
    cpu_Trap,           // Reserved
    cpu_Trap,           // Reserved
    cpu_Trap,           // Reserved
    cpu_Trap,           // Reserved
    cpu_Trap,           // TIM overflow
    keys_CommandPulse,  // TIM channel 1
    keys_AddressPulse,  // TIM channel 0
    cpu_Trap,           // CMIREQ
    dali_Start,         // IRQ1
    cpu_Trap,           // SWI
    _stext              // Reset
};
```

DALI Master Unit Source Code Files

Appendix G. DALI Slave Source Code Files

G.1 Contents

G.2 Slave: command.h

```
// command.h
//
// This header contains DALI command definitions
//
// Prepared: Motorola AB
//
// Functional level: Application
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001216  Initial version
// R1A  010212  Released version

#ifndef COMMAND
#define COMMAND

// Indirect arc power control commands
#define OFF                                     0x00
#define UP                                      0x01
#define DOWN                                    0x02
#define STEP_UP                                 0x03
#define STEP_DOWN                               0x04
#define RECALL_MAX_LEVEL                        0x05
#define RECALL_MIN_LEVEL                        0x06
#define STEP_DOWN_AND_OFF                       0x07
#define ON_AND_STEP_UP                          0x08
#define GO_TO_SCENE                             0x10

// General configuration commands
#define RESET                                   0x20
#define STORE_ACTUAL_LEVEL_IN_THE_DTR           0x21

// Arc power parameters settings
#define STORE_THE_DTR_AS_MAX_LEVEL              0x2A
#define STORE_THE_DTR_AS_MIN_LEVEL              0x2B
#define STORE_THE_DTR_AS_SYSTEM_FAILURE_LEVEL   0x2C
#define STORE_THE_DTR_AS_POWER_ON_LEVEL         0x2D
#define STORE_THE_DTR_AS_FADE_TIME              0x2E
#define STORE_THE_DTR_AS_FADE_RATE              0x2F
#define STORE_THE_DTR_AS_SCENE                  0x40

// System parameters settings
#define REMOVE_FROM_SCENE                       0x50
#define ADD_TO_GROUP                            0x60
#define REMOVE_FROM_GROUP                       0x70
#define STORE_DTR_AS_SHORT_ADDRESS              0x80
```

```
// Queries related to status information
#define QUERY_STATUS                            0x90
#define QUERY_BALLAST                           0x91
#define QUERY_LAMP_FAILURE                      0x92
#define QUERY_LAMP_POWER_ON                     0x93
#define QUERY_LIMIT_ERROR                       0x94
#define QUERY_RESET_STATE                       0x95
#define QUERY_MISSING_SHORT_ADDRESS             0x96
#define QUERY_VERSION_NUMBER                    0x97
#define QUERY_CONTENT_DTR                       0x98
#define QUERY_DEVICE_TYPE                       0x99
#define QUERY_PHYSICAL_MINIMUM_LEVEL            0x9A
#define QUERY_POWER_FAILURE                     0x9B

// Queries related to arc power parameters settings
#define QUERY_ACTUAL_LEVEL                      0xA0
#define QUERY_MAX_LEVEL                         0xA1
#define QUERY_MIN_LEVEL                         0xA2
#define QUERY_POWER_ON_LEVEL                    0xA3
#define QUERY_SYSTEM_FAILURE_LEVEL              0xA4
#define QUERY_FADE                              0xA5

// Queries related to system parameters settings
#define QUERY_SCENE_LEVEL                       0xB0
#define QUERY_GROUPS_0_7                        0xC0
#define QUERY_GROUPS_8_15                       0xC1
#define QUERY_RANDOM_ADDRESS_H                  0xC2
#define QUERY_RANDOM_ADDRESS_M                  0xC3
#define QUERY_RANDOM_ADDRESS_L                  0xC4

// Terminate special processes
#define TERMINATE                               0xA1

// Download information to the dtr
#define DATA_TRANSFER_REGISTER                  0xA3

// Addressing commands
#define INITIALISE                              0xA5
#define RANDOMISE                               0xA7
#define COMPARE                                 0xA9
#define WITHDRAW                                0xAB
#define SEARCHADDRH                             0xB1
#define SEARCHADDRM                             0xB3
#define SEARCHADDRL                             0xB5
#define PROGRAM_SHORT_ADDRESS                   0xB7
#define VERIFY_SHORT_ADDRESS                    0xB9
#define QUERY_SHORT_ADDRESS                     0xBB
#define PHYSICAL_SELECTION                      0xBD

#endif
```

G.3 Slave: common.h

```
// common.h
//
// This header contains common definitions
//
// Prepared: Motorola AB
//
// Functional level: Common
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// P1B  010201  Added declaration for embedded ROM flash routines
// R1A  010212  Released version

#ifndef COMMON
#define COMMON

#define TRUE 1
#define FALSE 0

// Global definition

extern unsigned char address;   // The DALI address
extern unsigned char command;   // The DALI command

extern unsigned char answer;    // The DALI answer

extern unsigned int flag;       // Keep track of all events

#define NEW_DATA     0x01       // Set when new data has arrived
#define SEND_ANSWER  0x02       // Set when an answer is to be sent
#define SIGNAL_ERROR 0x04
// Set when the DALI signal level has been low for more than 500 ms

// Macro definitions
#define ei() _asm("cli")        // Enable interrupt
#define di() _asm("sei")        // Disable interrupt

// Declarations for the embedded ROM flash routines
#define CTRLBYT (unsigned char *)0x0048
// Address to a variable containting a erase flag
#define CPUSPD  (unsigned char *)0x0049
// Address to a variable containing the cpu speed
#define LADDR   (unsigned int *)0x004A
// Address to a variable containing the last address to write
#define DATA    (unsigned char *)0x004C
// Address to the data array to be written into flash

#define flash_erase() _asm("ldhx #$FD80\njsr $1006\n")
#define flash_write() _asm("ldhx #$FD80\njsr $1009\n")

#define clock_write() _asm("ldhx #$FDFF\njsr $1009\n")

#endif
```

G.4 Slave: cpu.h

```
// cpu.h
//
// This header contains definitions for the cpu module
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// R1A  010212  Released version

#ifndef CPU
#define CPU

// Initialize
extern void cpu_Init(void);

// Interrupt handler for reset
extern @interrupt void _stext(void);

// Trap lost interupts
extern @interrupt void cpu_Trap(void);

#endif
```

G.5 Slave: cpu.c

```
// cpu.c
//
// This module handles all cpu related tasks
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1B
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// P1B  000201  Added automatic clock calibration
// R1A  010212  Released version
// R1B  011016  Adapted to new clock speed

#include "common.h"
#include "iokx8.h"

#define TRIM_FACTOR_ADDR (unsigned char *)0xFDFF
#define TRIM_FACTOR *(TRIM_FACTOR_ADDR)  // Internal clock generator trim factor

// Initialize
void cpu_Init(void)
{
    unsigned char trimmed;
    unsigned char ser_rec;
    unsigned char dali_rec;

    if (TRIM_FACTOR!=0xFF)
    {
        // Initialize LVI to 5V
        CONFIG2 |= 0x08;
    }
    else
    {
        // Initialize LVI to 5V and disable COP
        CONFIG2 |= 0x09;
    }

    // Initialize ports
    PTA = 0x08;     // Set initial values for PTA
    DDRA = 0x0C;    // Initialize PTA2 and PTA3 as output registers
    PTB = 0xA0;     // Set initial values for PTB
    DDRB = 0xA0;    // Initialize PTB5 and PTB7 as output register

    // Mask IRQ1 interrupt
    ISCR = 0x06;
// Change internal clock speed to simplify baud rate selection
    ICGMR = 0x40;   // Results in 19.6608 MHz
```

```
// Trim internal clock (if a trim value is stored in flash)
    if (TRIM_FACTOR!=0xFF)
    {
        ICGTR = TRIM_FACTOR;
    }
    else
    {
        // Clock is not trimmed
        while ((PTB & 0x50)!=0x50)  // Wait until the signals are stable
            ;
        // Initialize SCI for transmission
        ICGTR = 0x40;   // Start with a low value
        SCBR = 0x06;    // Baud rate 4800
        SCC1 = 0x40;    // Enable SCI
        SCC2 = 0x08;    // Transmitter enabled
        ser_rec = 0x10;
        dali_rec = 0x40;
        trimmed = FALSE;
        while (trimmed==FALSE)  // Loop forever
        {
            while ((SCS1 & 0x80)==0x00)         // Wait until transmit buffer is
empty
                ;
            SCDR = 0x55;                        // Bit pattern
            if ((PTB & 0x10)==0x00 && ser_rec!=0x00)
            {
                (ICGTR)++;
            }
            ser_rec = PTB & 0x10;
            if ((PTB & 0x40)==0x00 && dali_rec!=0x00)
            {
                *CTRLBYT = 0x00;                            // Page erase
                *CPUSPD = 0x4F;                             // 4 * 19.6608
                *LADDR = (unsigned int)TRIM_FACTOR_ADDR;    // Last address in
                                                               the flash memory
                *DATA = ICGTR;
                clock_write();
                if (TRIM_FACTOR==ICGTR)
                {
                    trimmed = TRUE;
                }
            }
            dali_rec = PTB & 0x40;
        }
    }
}

// Trap lost interupts
@interrupt void cpu_Trap(void)
{
    while (1==1)
        ;
}
```

G.6 Slave: dali.h

```
// dali.h
//
// This header contains definitions for the dali module
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001110  Initial version
// R1A  010212  Released version

#ifndef DALI
#define DALI

// Initialize
extern void dali_Init(void);

// Sends the DALI answer on the dali port
extern void dali_SendAnswer(void);

// Interrupt handler for incoming data on the dali port
extern @interrupt void dali_Start(void);

// Interrupt handler for timebase module
extern @interrupt void dali_Tick(void);

#endif
```

G.7 Slave: dali.c

```
// dali.c
//
// This module handles all tasks related to the dali port
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1B
//
// Rev  Date    Reason/description
// P1A  001110  Initial version
// R1A  010212  Released version
// R1B  011010  Adapted to new clock speed, Reception improved to be more tolerant
//
// The timebase module (TBM) is used

#include "common.h"
#include "iokx8.h"

static unsigned char send_position;     // Position in the data to transfer
static unsigned char send_active;       // True if transfer has started
static unsigned char send_value;        // Holds the logic level to transfer

static unsigned char rec_position;      // Position in the data to receive
static unsigned char rec_active;        // True if reception has started
static unsigned char rec_value;         // Holds the received logic level
static unsigned char rec_bit;           // Number of received bits

static unsigned int low_counter;       // Number of ticks the signal level has been
low

// Initialize
void dali_Init(void)
{
    send_active = FALSE;
    rec_active = FALSE;
    TBCR = 0x2E;    // Interrupt 9600 times/second
    ISCR = 0x04;    // Enable IRQ1 interrupt
}

// Sends the DALI address and command on the dali port
void dali_SendAnswer(void)
{
    // Check if some commands shall be repeated
    // Wait until dali port is idle
    while (send_active || rec_active)
        ;
    ISCR = 0x06;        // Disable IRQ1 interrupt
    TSC &= ~0x40;       // Disable timer overflow interrupt
```

```
    send_value = 0x80;  // Delay before answering
    send_position = 0;
    send_active = TRUE; // Activate the timer module to transfer
}

// Interrupt handler for incoming data on the dali port
@interrupt void dali_Start(void)
{
    ISCR = 0x06;        // Disable IRQ1 interrupt
    TSC &= ~0x40;       // Disable timer overflow interrupt
    address = 0x00;     // Clear address
    command = 0x00;     // Clear command
    rec_bit = 0;        // No bit has been received
    rec_value = 0x00;   // Value is low when starting to receive
    rec_position = 0;
    rec_active = TRUE;  // Activate the timer module to receive
}

// Interrupt handler for timebase module
@interrupt void dali_Tick(void)
{
    unsigned char temp_value;

    TBCR |= 0x08;   // Acknowledge the interrupt
    if (rec_active==TRUE)
    {
        temp_value = PTB & 0x40;
        rec_position++;
        if (temp_value!=rec_value)
        {
            // An edge has been detected
            switch (rec_bit)
            {
            case 0:
                // Start bit
                rec_bit++;
                rec_position = 0;
                break;
            case 17:
                // First stop bit
                if (rec_position>6)
                {
                    // Stop bit error, no edge should exist, stop receiving
                    rec_active = FALSE;
                    TSC |= 0x40;    // Enable timer overflow interrupt
                    ISCR = 0x04;    // Enable IRQ1 interrupt
                }
                break;
            case 18:
                // Second stop bit
                // Stop bit error, no edge should exist, stop receiving
                rec_active = FALSE;
                TSC |= 0x40;    // Enable timer overflow interrupt
```

```
            ISCR = 0x04;    // Enable IRQ1 interrupt
            break;
        default:
            // The address and command bits
            if (rec_position>6)
            {
                // Store the values
                if (temp_value)
                {
                    // Check if address or command
                    if (rec_bit<=8)
                    {
                        address |= (1<<(8-rec_bit));
                    }
                    else
                    {
                        command |= (1<<(16-rec_bit));
                    }
                }
                rec_bit++;
                rec_position = 0;
            }
            break;
        }
        rec_value = temp_value;
    }
    else
    {
        // Signal level stable
        switch (rec_bit)
        {
        case 0:
            // Start bit
            if (rec_position==8)
            {
                // Start bit error, too long delay before edge, stop receiving
                rec_active = FALSE;
                TSC |= 0x40;    // Enable timer overflow interrupt
                ISCR = 0x04;    // Enable IRQ1 interrupt
            }
            break;
        case 17:
            // First stop bit
            if (rec_position==8)
            {
                if (temp_value==0)
                {
                    // Stop bit error, wrong level, stop receiving
                    rec_active = FALSE;
                    TSC |= 0x40;    // Enable timer overflow interrupt
                    ISCR = 0x04;    // Enable IRQ1 interrupt
                }
                else
```

```
                {
                    rec_bit++;
                    rec_position = 0;
                }
            }
            break;
        case 18:
            // Second stop bit
            if (rec_position==8)
            {
                // Receive ready
                flag |= NEW_DATA;
                rec_active = FALSE;
                TSC |= 0x40;    // Enable timer overflow interrupt
                ISCR = 0x04;    // Enable IRQ1 interrupt
            }
            break;
        default:
            // The address and command bits
            if (rec_position==10)
            {
                // Data bit error, too long delay before edge, stop receiving
                rec_active = FALSE;
                TSC |= 0x40;    // Enable timer overflow interrupt
                ISCR = 0x04;    // Enable IRQ1 interrupt
            }
            break;
        }
    }
}
if (send_active==TRUE)
{
    if ((send_position & 0x03)==0)
    {
        PTB = (PTB & ~0x80) | send_value;
        if (send_position==24)
        {
            send_value = 0x00;  // First half of start bit
        }
        if (send_position==28)
        {
            send_value = 0x80;  // Second half of start bit
        }
        if (send_position>=32 && send_position<=92)
        {
            // Extract bit level
            temp_value = (answer>>((92-send_position)/8)) & 0x01;
            // Check if first or second half of data bit
            if ((send_position & 0x04)==0)
            {
                // First half
                if (temp_value==0x00)
                {
```

```
                        send_value = 0x80;
                    }
                    else
                    {
                        send_value = 0x00;
                    }
                }
                else
                {
                    // Second half
                    if (temp_value==0x00)
                    {
                        send_value = 0x00;
                    }
                    else
                    {
                        send_value = 0x80;
                    }
                }
            }
            if (send_position==96)
            {
                send_value = 0x80; // Start of stop bit and settling time
            }
            if (send_position==112)
            {
                send_active = FALSE;
                TSC |= 0x40;    // Enable timer overflow interrupt
                ISCR = 0x04;    // Enable IRQ1 interrupt
            }
        }
        send_position++;
    }
    if (PTB & 0x40)
    {
        low_counter = 0;
    }
    else
    {
        // If the signal level has been low for 500 ms then go to failure level
        low_counter++;
        if (low_counter>4800)
        {
            low_counter = 0;
            flag |= SIGNAL_ERROR;
        }
    }
}
```

G.8 Slave: dali.lkf

```
#   LINK COMMAND FILE Example for 68HC908KX8
#   Copyright (c) 2000 by Motorola AB

+seg .text -b 0xE000 -m0x1E00 -nCODE -sROM  # program start address
+seg .const -a CODE -it -sROM               # constants and strings
+seg .bsct -b 0x70 -m0x90 -nZPAGE -sRAM     # zero page start address
+seg .ubsct -a ZPAGE -nUZPAGE -sRAM
+seg .data -a UZPAGE -nIDATA -sRAM          # data start address
+seg .bss -a IDATA -nUDATA -sRAM
crtsi.o                          # startup routine
main.o                           # application program
cpu.o
dali.o
rs232.o
lamp.o
"C:\COSMIC\EVAL08\Lib\libi.h08"
"C:\COSMIC\EVAL08\Lib\libm.h08"

+seg .const -b 0xFFDC           # vectors start address
vector.o                         # interrupt vectors

+def __memory=@.bss             # symbol used by library
+def __stack=0x0FF              # stack pointer value
```

G.9 Slave: iokx8.h

```
/*  IO DEFINITIONS FOR MC68HC908KX8
 *  Copyright (c) 2000 by Motorola AB
 *  Based on definition from Cosmic
 */

#define uint    unsigned int

/*  PORTS section
 */
@tiny volatile  char PTA      @0x00;    /* port A */
@tiny volatile  char PTB      @0x01;    /* port B */
@tiny volatile  char DDRA     @0x04;    /* data direction port A */
@tiny volatile  char DDRB     @0x05;    /* data direction port B */
@tiny volatile  char PTAPUE   @0x0d;    /* pull-up enable port A */

/*  SCI section
 */
@tiny volatile  char SCC1     @0x13;    /* SCI control register 1 */
@tiny volatile  char SCC2     @0x14;    /* SCI control register 2 */
@tiny volatile  char SCC3     @0x15;    /* SCI control register 3 */
@tiny volatile  char SCS1     @0x16;    /* SCI status register 1 */
@tiny volatile  char SCS2     @0x17;    /* SCI status register 2 */
@tiny volatile  char SCDR     @0x18;    /* SCI data register */
@tiny volatile  char SCBR     @0x19;    /* SCI baud rate */

/*  KEYBOARD section
 */
@tiny volatile  char KBSCR    @0x1a;
/* keyboard status/control register */
@tiny volatile  char KBIER    @0x1b;    /* keyboard interrupt enable */

/*  TIMEBASE section
 */
@tiny volatile  char TBCR     @0x1c;    /* timebase module control */

/*  IRQ section
 */
@tiny volatile  char ISCR     @0x1d;    /* IRQ status/control register */

/*  CONFIG section
 */
@tiny volatile  char CONFIG1  @0x1e;    /* configuration register 1 */
@tiny volatile  char CONFIG2  @0x1f;    /* configuration register 2 */

/*  TIMER section
 */
```

```
@tiny volatile  char TSC      @0x20;    /* timer status/ctrl register */
@tiny volatile  uint TCNT     @0x21;    /* timer counter register */
@tiny volatile  char TCNTH    @0x21;    /* timer counter high */
@tiny volatile  char TCNTL    @0x22;    /* timer counter low */
@tiny volatile  uint TMOD     @0x23;    /* timer modulo register */
@tiny volatile  char TMODH    @0x23;    /* timer modulo high */
@tiny volatile  char TMODL    @0x24;    /* timer modulo low */
@tiny volatile  char TSC0     @0x25;    /* timer chan 0 status/ctrl */
@tiny volatile  uint TCH0     @0x26;    /* timer chan 0 register */
@tiny volatile  char TCH0H    @0x26;    /* timer chan 0 high */
@tiny volatile  char TCH0L    @0x27;    /* timer chan 0 low */
@tiny volatile  char TSC1     @0x28;    /* timer chan 1 status/ctrl */
@tiny volatile  uint TCH1     @0x29;    /* timer chan 1 register */
@tiny volatile  char TCH1H    @0x29;    /* timer chan 1 high */
@tiny volatile  char TCH1L    @0x2a;    /* timer chan 1 low */

/*  ICG section
 */
@tiny volatile  char ICGCR    @0x36;    /* ICG control register */
@tiny volatile  char ICGMR    @0x37;    /* ICG multiplier register */
@tiny volatile  char ICGTR    @0x38;    /* ICG trim register */
@tiny volatile  char ICGDVR   @0x39;    /* ICG divider control register */
@tiny volatile  char ICGDSR   @0x3a;    /* ICG stage control register */

/*  A/D section
 */
@tiny volatile  char ADSCR    @0x3c;    /* A/D status/ctrl register */
@tiny volatile  char ADR      @0x3d;    /* A/D data register */
@tiny volatile  char ADICLK   @0x3e;    /* A/D input clock register */

/*  SIM section
 */
@near volatile  char SBSR     @0xfe00;  /* SIM break status register */
@near volatile  char SRSR     @0xfe01;  /* SIM reset status register */
@near volatile  char SBFCR    @0xfe03;  /* SIM break control register */

/*  INTERUPT section
 */
@near volatile  char INT1     @0xfe04;  /* INTERUPT status register 1 */
@near volatile  char INT2     @0xfe05;  /* INTERUPT status register 2 */
@near volatile  char INT3     @0xfe06;  /* INTERUPT status register 3 */

/*  FLASH section
 */
@near volatile  char FLTCR    @0xfe07;  /* FLASH test control register */
@near volatile  char FLCR     @0xfe08;  /* FLASH control register */
@near volatile  char FLBPR    @0xff7e;  /* FLASH block protect register */
```

```
/*  BREAK section
 */
@near volatile  char BRKAR    @0xfe02;  /* BREAK auxiliary register */
@near volatile  uint BRK      @0xfe09;  /* BREAK address register */
@near volatile  char BRKH     @0xfe09;  /* BREAK address register low */
@near volatile  char BRKL     @0xfe0a;  /* BREAK address register high */
@near volatile  char BRKSCR   @0xfe0b;  /* BREAK status/ctrl register */

/*  LVI section
 */
@near volatile  char LVISR    @0xfe0c;  /* LVI status register */

/*  COP section
 */
@near volatile  char COPCTL   @0xffff;  /* COP control register */

#undef uint
```

G.10 Slave: lamp.h

```
// lamp.h
//
// This header contains definitions for the lamp module
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001215  Initial version
// R1A  010212  Released version

#ifndef LAMP
#define LAMP

// Initialization
extern void lamp_Init(void);

// Handles new data
extern void lamp_HandleData(void);

// Set the lamp to failure level
extern void lamp_FailureLevel(void);

// Interrupt handler for the timer interface module
extern @interrupt void lamp_Tick(void);

#endif
```

G.11 Slave: lamp.c

```
// lamp.c
//
// This module controls the lamp
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1C
//
// Rev  Date    Reason/description
// P1A  001215  Initial version
// P1B  000201  Improved current measurement
// R1A  010212  Released version
// R1B  011016  Adapted to new clock speed
// R1C  011210  Changed to buffered PWM
//
// The timer interface module (TIM) and the analog-to-digital converter (ADC)
   are used

#include "common.h"
#include "iokx8.h"
#include "command.h"

// Flash memory data definitions

#define START_OF_FLASH_BUFFER       (unsigned char *)0xFD80   // 0xFDE0
#define STOP_OF_FLASH_BUFFER        (unsigned char *)0xFD9F   // 0xFDFF

#define POWER_ON_LEVEL_ADDR         (unsigned char *)0xFD80   // Power-on level
                                                                 address
#define SYSTEM_FAILURE_LEVEL_ADDR   (unsigned char *)0xFD81   // System failure
                                                                 level address
#define MIN_LEVEL_ADDR              (unsigned char *)0xFD82   // Minimum level
                                                                 address
#define MAX_LEVEL_ADDR              (unsigned char *)0xFD83   // Maximum level
                                                                 address
#define FADE_RATE_ADDR              (unsigned char *)0xFD84   // Fade rate address
#define FADE_TIME_ADDR              (unsigned char *)0xFD85   // Fade time address
#define SHORT_ADDRESS_ADDR          (unsigned char *)0xFD86   // Short address
                                                                 address
#define RANDOM_ADDRESS_H_ADDR       (unsigned char *)0xFD87   // Random address
                                                                 high address
#define RANDOM_ADDRESS_M_ADDR       (unsigned char *)0xFD88   // Random address
                                                                 middle address
#define RANDOM_ADDRESS_L_ADDR       (unsigned char *)0xFD89   // Random address low
                                                                 address
#define GROUP_0_7_ADDR              (unsigned char *)0xFD8A   // Group 0-7 address
#define GROUP_8_15_ADDR             (unsigned char *)0xFD8B   // Group 8-15 address
#define SCENE_0_15_ADDR             (unsigned char *)0xFD8C   // Scene level array
                                                                 0-15 address
```

```
#define POWER_ON_LEVEL         *(POWER_ON_LEVEL_ADDR)         // Power-on level
#define SYSTEM_FAILURE_LEVEL   *(SYSTEM_FAILURE_LEVEL_ADDR)   // System failure
                                                                 level
#define MIN_LEVEL              *(MIN_LEVEL_ADDR)              // Minimum level
#define MAX_LEVEL              *(MAX_LEVEL_ADDR)              // Maximum level
#define FADE_RATE              *(FADE_RATE_ADDR)              // Fade rate
#define FADE_TIME              *(FADE_TIME_ADDR)              // Fade time
#define SHORT_ADDRESS          *(SHORT_ADDRESS_ADDR)          // Short address
#define RANDOM_ADDRESS_H       *(RANDOM_ADDRESS_H_ADDR)       // Random address high
#define RANDOM_ADDRESS_M       *(RANDOM_ADDRESS_M_ADDR)       // Random address
                                                                 middle
#define RANDOM_ADDRESS_L       *(RANDOM_ADDRESS_L_ADDR)       // Random address low
#define GROUP_0_7              *(GROUP_0_7_ADDR)              // Group 0-7
#define GROUP_8_15             *(GROUP_8_15_ADDR)             // Group 8-15

// Permanent data definitions

#define VERSION_NUMBER      0x00    // The current version number 0.0
#define PHYSICAL_MIN_LEVEL  0x01    // The physically lowest possible level

// Definition for the timer states

#define IDLE_STATE 0x00             // No change
#define INIT_STATE 0x01             // State during the first 600 ms or until the
first data arrives
#define FADE_STATE 0x02             // During this state the lamp is changing to a
                                       new level

// Table containing conversions between light level values and logarithmic dimming
   level
static const unsigned int log_level[253] = {
                  0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002,
                  0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002,
                  0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003,
                  0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004,
                  0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004,
                  0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006,
                  0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007,
                  0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009,
                  0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B,
                  0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D,
                  0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x0010, 0x0010, 0x0011,
                  0x0011, 0x0011, 0x0012, 0x0012, 0x0013, 0x0013, 0x0014, 0x0015,
                  0x0015, 0x0016, 0x0016, 0x0017, 0x0018, 0x0018, 0x0019, 0x001A,
                  0x001A, 0x001B, 0x001C, 0x001D, 0x001D, 0x001E, 0x001F, 0x0020,
                  0x0021, 0x0022, 0x0023, 0x0023, 0x0024, 0x0025, 0x0027, 0x0028,
                  0x0029, 0x002A, 0x002B, 0x002C, 0x002D, 0x002F, 0x0030, 0x0031,
                  0x0033, 0x0034, 0x0035, 0x0037, 0x0038, 0x003A, 0x003C, 0x003D,
                  0x003F, 0x0041, 0x0042, 0x0044, 0x0046, 0x0048, 0x004A, 0x004C,
                  0x004E, 0x0050, 0x0053, 0x0055, 0x0057, 0x005A, 0x005C, 0x005F,
                  0x0061, 0x0064, 0x0067, 0x006A, 0x006D, 0x0070, 0x0073, 0x0076,
```

```
                0x0079, 0x007D, 0x0080, 0x0084, 0x0087, 0x008B, 0x008F, 0x0093,
                0x0097, 0x009B, 0x009F, 0x00A4, 0x00A8, 0x00AD, 0x00B2, 0x00B7,
                0x00BC, 0x00C1, 0x00C6, 0x00CC, 0x00D1, 0x00D7, 0x00DD, 0x00E3,
                0x00E9, 0x00F0, 0x00F7, 0x00FD, 0x0104, 0x010C, 0x0113, 0x011B,
                0x0122, 0x012A, 0x0133, 0x013B, 0x0144, 0x014D, 0x0156, 0x0160,
                0x0169, 0x0173, 0x017E, 0x0188, 0x0193, 0x019E, 0x01AA, 0x01B5,
                0x01C2, 0x01CE, 0x01DB, 0x01E8, 0x01F5, 0x0203, 0x0212, 0x0220,
                0x022F, 0x023F, 0x024F, 0x025F, 0x0270, 0x0281, 0x0293, 0x02A5,
                0x02B8, 0x02CB, 0x02DF, 0x02F3, 0x0308, 0x031E, 0x0334, 0x034A,
                0x0362, 0x037A, 0x0392, 0x03AC, 0x03C6, 0x03E0, 0x03FC, 0x0418,
                0x0435, 0x0453, 0x0472, 0x0491, 0x04B1, 0x04D3, 0x04F5, 0x0518,
                0x053C, 0x0561, 0x0587, 0x05AE, 0x05D7 };

// Number of timer ticks for each fade time
static const unsigned int fade_tick[15] = { 35, 50, 71, 100, 141, 200, 283, 400,
                                            566, 800, 1131, 1600, 2263, 3200, 4526 };

// Number of level steps during 200 ms
static const unsigned char fade_step[15] = { 72, 51, 36, 25, 18, 13, 9, 6, 4, 3, 2,
                                             2, 1, 1, 1 };

static unsigned char level;             // The light level

static unsigned char search_address_h;  // Search address high
static unsigned char search_address_m;  // Search address middle
static unsigned char search_address_l;  // Search address low

static unsigned char dtr;               // The DTR register

static unsigned char last_channel;      // Last channel register used

static unsigned char state_no;          // Holds the current state for the timer
static unsigned int state_time;         // Time until state or level is changed

static unsigned char fade_start_level;  // Start level before fading
static unsigned char fade_stop_level;   // Stop level after fading
static unsigned int fade_total;         // Total time for fade

static unsigned char repeat_time;       // Time until repetation time expires
static unsigned char repeat_address;    // Address that should be repeated
static unsigned char repeat_command;    // Command that should be repeated

static unsigned int address_time;       // Time until addressing commands are
                                           illegal

static unsigned char power_failure;     // Flag is true until certain commands has
                                           arrived

static unsigned char limit_error;       // True if the last requested level was out
                                          of limit

static unsigned char measure_time;      // Time until lamp current can be measured
```

```
// Write data into flash at address
static void lamp_WriteFlash(unsigned char *flash_address, unsigned char new_data)
{
    unsigned char index;

    // Set all variables necessary
    *CTRLBYT = 0x00;                              // Page erase
    *CPUSPD = 0x4F;                               // 4 * 19.6608
    *LADDR = (unsigned int)STOP_OF_FLASH_BUFFER; // Last address in the flash memory

    // Copy data to buffer
    for (index = 0; index<32; index++)
    {
        *(DATA+index) = *(START_OF_FLASH_BUFFER+index);
    }

    // Merge the new data into the data buffer
    *(DATA+(unsigned char)(flash_address-START_OF_FLASH_BUFFER)) = new_data;

    di(); // Disable all interrupts during flash erase and flash write

    // Place erase start address in H:X and call the erase routine
    flash_erase();

    // Place write start address in H:X and call the write routine
    flash_write();

    ei(); // Enable all interrupts
}

// Reset flash content
static void lamp_ResetFlash(void)
{
    unsigned char index;

    // Set all variables necessary
    *CTRLBYT = 0x00;                              // Page erase
    *CPUSPD = 0x4F;                               // 4 * 19.6608
    *LADDR = (unsigned int)STOP_OF_FLASH_BUFFER; // Last address in the flash memory

    // Copy data to buffer
    for (index = 0; index<32; index++)
    {
        *(DATA+index) = *(START_OF_FLASH_BUFFER+index);
    }

    // Merge the new data into the data buffer
    *(DATA+((unsigned char)(POWER_ON_LEVEL_ADDR-START_OF_FLASH_BUFFER))) = 254;
    *(DATA+((unsigned char)(SYSTEM_FAILURE_LEVEL_ADDR-START_OF_FLASH_BUFFER)))
      = 254;
```

```
    *(DATA+((unsigned char)(MIN_LEVEL_ADDR-START_OF_FLASH_BUFFER))) =
      PHYSICAL_MIN_LEVEL;
    *(DATA+((unsigned char)(MAX_LEVEL_ADDR-START_OF_FLASH_BUFFER))) = 254;
    *(DATA+((unsigned char)(FADE_RATE_ADDR-START_OF_FLASH_BUFFER))) = 7;
    *(DATA+((unsigned char)(FADE_TIME_ADDR-START_OF_FLASH_BUFFER))) = 0;
    *(DATA+((unsigned char)(RANDOM_ADDRESS_H_ADDR-START_OF_FLASH_BUFFER))) = 0xFF;
    *(DATA+((unsigned char)(RANDOM_ADDRESS_M_ADDR-START_OF_FLASH_BUFFER))) = 0xFF;
    *(DATA+((unsigned char)(RANDOM_ADDRESS_L_ADDR-START_OF_FLASH_BUFFER))) = 0xFF;
    *(DATA+((unsigned char)(GROUP_0_7_ADDR-START_OF_FLASH_BUFFER))) = 0x00;
    *(DATA+((unsigned char)(GROUP_8_15_ADDR-START_OF_FLASH_BUFFER))) = 0x00;
    for (index = 0; index<16; index++)
    {
       *(DATA+((unsigned char)(SCENE_0_15_ADDR-START_OF_FLASH_BUFFER+index))) = 255;
    }

    di(); // Disable all interrupts during flash erase and flash write

    // Place erase start address in H:X and call the erase routine
    flash_erase();

    // Place write start address in H:X and call the write routine
    flash_write();

    ei(); // Enable all interrupts
}

// Set the light level
static void lamp_SetLevel(unsigned char wanted_level)
{
    unsigned char new_level;

    limit_error = FALSE;
    new_level = wanted_level;
    if (new_level>=1 && new_level<=254)
    {
        if (new_level<MIN_LEVEL)
        {
            new_level = MIN_LEVEL;
            limit_error = TRUE;
        }
        if (new_level>MAX_LEVEL)
        {
            new_level = MAX_LEVEL;
            limit_error = TRUE;
        }
    }
```

```
    if (new_level==0)
    {
        if (last_channel==0)
        {
            TCH1 = 0x0000;                  // Lamp off
            last_channel = 1;
        }
        else
        {
            TCH0 = 0x0000;                  // Lamp off
            last_channel = 0;
        }
        TSC0 &= ~0x01;                      // Not 100% duty cycle
        level = new_level;
    }
    if (new_level>=1 && new_level<=253)
    {
        if (last_channel==0)
        {
            TCH1 = log_level[new_level-1];  // Set the PWM period
            last_channel = 1;
        }
        else
        {
            TCH0 = log_level[new_level-1];  // Set the PWM period
            last_channel = 0;
        }
        TSC0 &= ~0x01;                      // Not 100% duty cycle
        level = new_level;
    }
    if (new_level==254)
    {
        TSC0 |= 0x01;                       // 100% duty cycle
        level = new_level;
    }
}

// Test if lamp is functional, returns TRUE if bad lamp
static unsigned char lamp_TestFailure(void)
{
    unsigned char current_level;

    current_level = level;

    lamp_SetLevel(254);             // Turn on lamp for measurement

    measure_time = 5;               // Wait until level is stabilized (100 ms)
    while (measure_time!=0)
    {
        COPCTL = 0x00;              // Clear the COP counter
    }
```

```
    // Measure voltage over shunt resistor by using the built-in A/D-converter
    ADICLK = 0x80;                  // Set the ADC input clock to approximately 1 MHz
    ADSCR = 0x03;                   // Select channel 3 and start conversion
    while ((ADSCR & 0x80)==0)       // Wait until conversion is ready
        ;

    lamp_SetLevel(current_level);  // Restore the previous level

    if (ADR>0)
    {
        return (FALSE);             // Lamp is working
    }
    return (TRUE);
}

// Start dimming using the fade time
static void lamp_StartDimFadeTime(unsigned char new_level)
{
    if (new_level!=255)
    {
        // Not mask
        if (FADE_TIME==0)
        {
            // No fade
            lamp_SetLevel(new_level);
            state_no = IDLE_STATE;
        }
        else
        {
            // Fade with fade time
            fade_start_level = level;
            fade_stop_level = new_level;
            fade_total = fade_tick[FADE_TIME-1];
            state_time = fade_total;
            state_no = FADE_STATE;
        }
    }
}

// Handle special commands
static void lamp_HandleSpecialCommands(void)
{
    static unsigned withdraw;
    static unsigned selection;

    unsigned long random_address;
    unsigned long search_address;
```

```
// Special command
if (address>=0xA7 && address<=0xBD  && address_time==0)
{
    // Not a valid command until INITIALISE has been received
    return;
}
switch (address)
{
case TERMINATE:
    address_time = 0;
    break;
case DATA_TRANSFER_REGISTER:
    dtr = command;
    break;
case INITIALISE:
    if (command==0x00 || ((command & 0x7E)>>1)==SHORT_ADDRESS)
    {
        address_time = 45000;  // Accept addressing commands for the next
                                  15 minutes
        withdraw = FALSE;
        selection = FALSE;
    }
    break;
case RANDOMISE:
    // The algorithm used to get a random number can be improved
    lamp_WriteFlash(RANDOM_ADDRESS_H_ADDR, TCNTL);  // Read timer counter to get
                                                       a random number
    lamp_WriteFlash(RANDOM_ADDRESS_M_ADDR, TCNTL);  // Read timer counter to get
                                                       a random number
    lamp_WriteFlash(RANDOM_ADDRESS_L_ADDR, TCNTL);  // Read timer counter to get
                                                       a random number
    break;
case COMPARE:
    random_address = RANDOM_ADDRESS_H<<16 | RANDOM_ADDRESS_M<<8 |
        RANDOM_ADDRESS_L;
    search_address = search_address_h<<16 | search_address_m<<8 |
        search_address_l;
    if (random_address!=search_address || withdraw==FALSE)
    {
        if (random_address<=search_address)
        {
            answer = 0xFF; // YES
            flag |= SEND_ANSWER;
        }
    }
    break;
case WITHDRAW:
    withdraw = TRUE;
    break;
```

```
case SEARCHADDRH:
    search_address_h = command;
    break;
case SEARCHADDRM:
    search_address_m = command;
    break;
case SEARCHADDRL:
    search_address_l = command;
    break;
case PROGRAM_SHORT_ADDRESS:
    if (selection==TRUE)
    {
        if (lamp_TestFailure()==TRUE)
        {
            if (command==255)
            {
                lamp_WriteFlash(SHORT_ADDRESS_ADDR, 255);
            }
            else
            {
                if (((dtr & 0x7E)>>1)<=63)
                {
                    lamp_WriteFlash(SHORT_ADDRESS_ADDR, (command & 0x7E)>>1);
                }
            }
        }
    }
    else
    {
        random_address = RANDOM_ADDRESS_H<<16 | RANDOM_ADDRESS_M<<8 |
            RANDOM_ADDRESS_L;
        search_address = search_address_h<<16 | search_address_m<<8 |
            search_address_l;
        if (random_address==search_address)
        {
            if (command==255)
            {
                lamp_WriteFlash(SHORT_ADDRESS_ADDR, 255);
            }
            else
            {
                if (((dtr & 0x7E)>>1)<=63)
                {
                    lamp_WriteFlash(SHORT_ADDRESS_ADDR, (command & 0x7E)>>1);
                }
            }
        }
    }
    break;
```

```
    case VERIFY_SHORT_ADDRESS:
        if (SHORT_ADDRESS==((command & 0x7E)>>1))
        {
            answer = 0xFF; // YES
            flag |= SEND_ANSWER;
        }
        break;
    case QUERY_SHORT_ADDRESS:
        if (selection==TRUE)
        {
            if (lamp_TestFailure()==TRUE)
            {
                answer = SHORT_ADDRESS;
                flag |= SEND_ANSWER;
            }
        }
        else
        {
            random_address = RANDOM_ADDRESS_H<<16 | RANDOM_ADDRESS_M<<8 |
                RANDOM_ADDRESS_L;
            search_address = search_address_h<<16 | search_address_m<<8 |
                search_address_l;
            if (random_address==search_address)
            {
                answer = SHORT_ADDRESS;
                flag |= SEND_ANSWER;
            }
        }
        break;
    case PHYSICAL_SELECTION:
        selection = TRUE;
        break;
    default:
        break;
    }
}

// Handle normal commands
static void lamp_HandleNormalCommands(void)
{
    // Normal commands
    switch (command)
    {
    case OFF:
        power_failure = FALSE;
        lamp_SetLevel(0);
        state_no = IDLE_STATE;
        break;
```

```
    case UP:
        if (level!=0 && level!=MAX_LEVEL)
        {
            fade_start_level = level;
            if (fade_step[FADE_RATE-1]>(MAX_LEVEL-level))
            {
                fade_stop_level = MAX_LEVEL;
                fade_total = (unsigned
int)(MAX_LEVEL-level)*10/fade_step[FADE_RATE-1];   // No of ms until MAX_LEVEL
                                                       is reached
            }
            else
            {
                fade_stop_level = level+fade_step[FADE_RATE-1];
                fade_total = 10; // 200 ms
            }
            state_time = fade_total;
            state_no = FADE_STATE;
        }
        break;
    case DOWN:
        if (level!=0 && level!=MIN_LEVEL)
        {
            fade_start_level = level;
            if (fade_step[FADE_RATE-1]>(level-MIN_LEVEL))
            {
                fade_stop_level = MIN_LEVEL;
                fade_total = (unsigned
int)(level-MIN_LEVEL)*10/fade_step[FADE_RATE-1];   // No of ms until MIN_LEVEL
                                                       is reached
            }
            else
            {
                fade_stop_level = level-fade_step[FADE_RATE-1];
                fade_total = 10; // 200 ms
            }
            state_time = fade_total;
            state_no = FADE_STATE;
        }
        break;
    case STEP_UP:
        if (level!=0 && level!=MAX_LEVEL)
        {
            lamp_SetLevel(level+1);
            state_no = IDLE_STATE;
        }
        break;
    case STEP_DOWN:
        if (level!=0 && level!=MIN_LEVEL)
        {
            lamp_SetLevel(level-1);
            state_no = IDLE_STATE;
        }
        break;
```

```
case RECALL_MAX_LEVEL:
    power_failure = FALSE;
    lamp_SetLevel(MAX_LEVEL);
    state_no = IDLE_STATE;
    break;
case RECALL_MIN_LEVEL:
    power_failure = FALSE;
    lamp_SetLevel(MIN_LEVEL);
    state_no = IDLE_STATE;
    break;
case STEP_DOWN_AND_OFF:
    power_failure = FALSE;
    if (level!=0)
    {
        if (level==MIN_LEVEL)
        {
            lamp_SetLevel(0);
        }
        else
        {
            lamp_SetLevel(level-1);
        }
        state_no = IDLE_STATE;
    }
    break;
case ON_AND_STEP_UP:
    power_failure = FALSE;
    if (level!=MAX_LEVEL)
    {
        if (level==0)
        {
            lamp_SetLevel(MIN_LEVEL);
        }
        else
        {
            lamp_SetLevel(level+1);
        }
        state_no = IDLE_STATE;
    }
    break;
case RESET:
    search_address_h = 0xFF;
    search_address_m = 0xFF;
    search_address_l = 0xFF;
    lamp_ResetFlash();
    lamp_SetLevel(254);
    state_no = IDLE_STATE;
    break;
case STORE_ACTUAL_LEVEL_IN_THE_DTR:
    dtr = level;
    break;
```

```
    case STORE_THE_DTR_AS_MAX_LEVEL:
        if (dtr>=MIN_LEVEL && dtr<=254)
        {
            lamp_WriteFlash(MAX_LEVEL_ADDR, dtr);
        }
        break;
    case STORE_THE_DTR_AS_MIN_LEVEL:
        if (dtr>=PHYSICAL_MIN_LEVEL && dtr<=MAX_LEVEL)
        {
            lamp_WriteFlash(MIN_LEVEL_ADDR, dtr);
        }
        break;
    case STORE_THE_DTR_AS_SYSTEM_FAILURE_LEVEL:
        lamp_WriteFlash(MIN_LEVEL_ADDR, dtr);
        break;
    case STORE_THE_DTR_AS_POWER_ON_LEVEL:
        if (dtr>=1 && dtr<=254)
        {
            lamp_WriteFlash(MIN_LEVEL_ADDR, dtr);
        }
        break;
    case STORE_THE_DTR_AS_FADE_TIME:
        lamp_WriteFlash(FADE_TIME_ADDR, dtr & 0x0F);
        break;
    case STORE_THE_DTR_AS_FADE_RATE:
        if (dtr & 0x0F)
        {
            lamp_WriteFlash(FADE_RATE_ADDR, dtr & 0x0F);
        }
        break;
    case STORE_DTR_AS_SHORT_ADDRESS:
        if (dtr==255)
        {
            lamp_WriteFlash(SHORT_ADDRESS_ADDR, 255);
        }
        else
        {
            if (((dtr & 0x7E)>>1)<=63)
            {
                lamp_WriteFlash(SHORT_ADDRESS_ADDR, (dtr & 0x7E)>>1);
            }
        }
        break;
    case QUERY_STATUS:
        answer = 0x00;
        if (lamp_TestFailure()==TRUE)
        {
            answer |= 0x02;
        }
        if (level)
        {
            answer |= 0x04;
        }
```

```
        if (limit_error==TRUE)
        {
            answer |= 0x08;
        }
        if (state_no==FADE_STATE)
        {
            answer |= 0x10;
        }
        if (state_no==INIT_STATE)
        {
            answer |= 0x20;
        }
        if (SHORT_ADDRESS==255)
        {
            answer |= 0x40;
        }
        if (power_failure==TRUE)
        {
            answer |= 0x80;
        }
        flag |= SEND_ANSWER;
        break;
    case QUERY_BALLAST:
        answer = 0xFF; // YES
        flag |= SEND_ANSWER;
        break;
    case QUERY_LAMP_FAILURE:
        if (lamp_TestFailure()==TRUE)
        {
            answer = 0xFF; // YES
            flag |= SEND_ANSWER;
        }
        break;
    case QUERY_LAMP_POWER_ON:
        if (level)
        {
            answer = 0xFF; // YES
            flag |= SEND_ANSWER;
        }
        break;
    case QUERY_LIMIT_ERROR:
        if (limit_error==TRUE)
        {
            answer = 0xFF; // YES
            flag |= SEND_ANSWER;
        }
        break;
    case QUERY_RESET_STATE:
        if (state_no==INIT_STATE)
        {
            answer = 0xFF; // YES
```

```
        flag |= SEND_ANSWER;
    }
    break;
case QUERY_MISSING_SHORT_ADDRESS:
    if (SHORT_ADDRESS==255)
    {
        answer = 0xFF; // YES
        flag |= SEND_ANSWER;
    }
    break;
case QUERY_VERSION_NUMBER:
    answer = VERSION_NUMBER;
    flag |= SEND_ANSWER;
    break;
case QUERY_CONTENT_DTR:
    answer = dtr;
    flag |= SEND_ANSWER;
    break;
case QUERY_DEVICE_TYPE:
    answer = 0; // For low voltage halogen lamps this is 3 but no extended
                  commands are available
    flag |= SEND_ANSWER;
    break;
case QUERY_PHYSICAL_MINIMUM_LEVEL:
    answer = PHYSICAL_MIN_LEVEL;
    flag |= SEND_ANSWER;
    break;
case QUERY_POWER_FAILURE:
    if (power_failure==TRUE)
    {
        answer = 0xFF; // YES
        flag |= SEND_ANSWER;
    }
    break;
case QUERY_ACTUAL_LEVEL:
    answer = level;
    flag |= SEND_ANSWER;
    break;
case QUERY_MAX_LEVEL:
    answer = MAX_LEVEL;
    flag |= SEND_ANSWER;
    break;
case QUERY_MIN_LEVEL:
    answer = MIN_LEVEL;
    flag |= SEND_ANSWER;
    break;
case QUERY_POWER_ON_LEVEL:
    answer = POWER_ON_LEVEL;
    flag |= SEND_ANSWER;
    break;
```

```
case QUERY_SYSTEM_FAILURE_LEVEL:
    answer = SYSTEM_FAILURE_LEVEL;
    flag |= SEND_ANSWER;
    break;
case QUERY_FADE:
    answer = FADE_TIME<<4 | FADE_RATE;
    flag |= SEND_ANSWER;
    break;
case QUERY_GROUPS_0_7:
    answer = GROUP_0_7;
    flag |= SEND_ANSWER;
    break;
case QUERY_GROUPS_8_15:
    answer = GROUP_8_15;
    flag |= SEND_ANSWER;
    break;
case QUERY_RANDOM_ADDRESS_H:
    answer = RANDOM_ADDRESS_H;
    flag |= SEND_ANSWER;
    break;
case QUERY_RANDOM_ADDRESS_M:
    answer = RANDOM_ADDRESS_M;
    flag |= SEND_ANSWER;
    break;
case QUERY_RANDOM_ADDRESS_L:
    answer = RANDOM_ADDRESS_L;
    flag |= SEND_ANSWER;
    break;
default:
    break;
}
switch (command & 0xF0)
{
case GO_TO_SCENE:
    power_failure = FALSE;
    lamp_StartDimFadeTime(*(SCENE_0_15_ADDR+(command & 0x0F)));
    break;
case STORE_THE_DTR_AS_SCENE:
    lamp_WriteFlash(SCENE_0_15_ADDR+(command & 0x0F), dtr);
    break;
case REMOVE_FROM_SCENE:
    lamp_WriteFlash(SCENE_0_15_ADDR+(command & 0x0F), 255);
    break;
case ADD_TO_GROUP:
    if ((command & 0x0F)<=7)
    {
        lamp_WriteFlash(GROUP_0_7_ADDR, GROUP_0_7 | (1<<(command & 0x0F)));
    }
    else
    {
        lamp_WriteFlash(GROUP_8_15_ADDR, GROUP_8_15 | (1<<((command &
        0x0F)-8)));
```

```
        }
        break;
    case REMOVE_FROM_GROUP:
        if ((command & 0x0F)<=7)
        {
            lamp_WriteFlash(GROUP_0_7_ADDR, GROUP_0_7 & ~(1<<(command & 0x0F)));
        }
        else
        {
            lamp_WriteFlash(GROUP_8_15_ADDR, GROUP_8_15 & ~(1<<((command &
            0x0F)-8)));
        }
        break;
    case QUERY_SCENE_LEVEL:
        answer = *(SCENE_0_15_ADDR+(command & 0x0F));
        flag |= SEND_ANSWER;
        break;
    default:
        break;
    }
}

// Initialization
void lamp_Init(void)
{
    // Reset values that are not stored in flash
    power_failure = TRUE;   // This is true until certain commands has arrived

    limit_error = FALSE;    // No error

    dtr = 0;

    level = 0;

    search_address_h = 0xFF;
    search_address_m = 0xFF;
    search_address_l = 0xFF;

    // Initialize flash values if necessary
    if (POWER_ON_LEVEL==255) // Check if flash has been written
    {
        // Write factory information into flash
        lamp_WriteFlash(SHORT_ADDRESS_ADDR, 255);   // No short address
        lamp_ResetFlash();
    }

    repeat_time = 0;    // Initialize when starting
    address_time = 0;   // Initialize when starting

    // Start timer for initial lamp values
    state_time = 30;    // 600 ms
```

```
    state_no = INIT_STATE;

    // Initialize the timer interface module
    TSC = 0x76;         // Stop and reset counter, Set prescaler to internal bus/64,
                           Enable timer overflow interrupt
    TMOD = 0x0600;      // Period 50 Hz

    // Initialize channel 0 for PWM
    TCH0 = 0x0000;      // Clear channel 0 compare register
    last_channel = 0;
    TSC0 = 0x3A;        // Set channel 0 to buffered PWM mode

    TSC &= ~0x20;       // Start counter
}

// Handles new data
void lamp_HandleData(void)
{
    unsigned int group;

    // Check if this message is for this unit
    if ((address & 0xE1)!=0xA1 && (address & 0xE1)!=0xC1 && (address & 0xFE)!=0xFE)
// Special command or broadcast message
    {
        if ((address & 0xE0)==0x80)
        {
            // Group address
            group = GROUP_8_15<<8 | GROUP_0_7;
            if ((group>>((address & 0x1E)>>1) & 0x01)==0)
            {
                // Not in this group
                return;
            }
        }
        if ((address & 0x80)==0x00)
        {
            // Short address
            if (((address & 0x7E)>>1)!=SHORT_ADDRESS)
            {
                // Wrong short address
                return;
            }
        }
    }

    // Message is intended for his unit

    // Check if repetetion shall be interrupted
    if (repeat_time!=0)
    {
        // Waiting for repeated message
```

```
    if (address!=repeat_address || command!=repeat_command)
    {
        // Ignore message and clear repeat timer
        repeat_time = 0;
        return;
    }
}

if ((address & 0x01)==0)
{
    // Direct arc power control command
    power_failure = FALSE;
    lamp_StartDimFadeTime(command);
    return;
}

// Check if commands shall be repeated
if ((address & 0xE0)==0xA0 || (address & 0xE0)==0xC0)
{
    // Special command
    if (address==INITIALISE || address==RANDOMISE)
    {
        // These commands shall be repeated
        if (repeat_time==0)
        {
            repeat_address = address;
            repeat_command = command;
            repeat_time = 5;            // 100 ms
            return;
        }
    }
}
else
{
    // Normal command
    if (command>=0x20 && command<=0x80)
    {
        // These commands shall be repeated
        if (repeat_time==0)
        {
            repeat_address = address;
            repeat_command = command;
            repeat_time = 5;            // 100 ms
            return;
        }
    }
}
repeat_time = 0;

if ((address & 0xE0)==0xA0 || (address & 0xE0)==0xC0)
{
```

```
        // Handle special commands
        lamp_HandleSpecialCommands();
    }
    else
    {
        // Handle normal commands
        lamp_HandleNormalCommands();
    }
}

// Set the lamp to failure level
void lamp_FailureLevel(void)
{
    state_no = IDLE_STATE;
    lamp_SetLevel(SYSTEM_FAILURE_LEVEL);
}

// Interrupt handler for timer interface module
@interrupt void lamp_Tick(void)
{
    unsigned char new_level;
    unsigned char diff_level;

    TSC &= ~0x80;   // Clear the TIM Overflow Flag Bit
    if (measure_time!=0)
    {
        measure_time--;
    }
    if (repeat_time!=0)
    {
        repeat_time--;
    }
    if (address_time!=0)
    {
        address_time--;
    }
    switch (state_no)
    {
    case IDLE_STATE:
        break;
    case INIT_STATE:
        state_time--;
        if (state_time==0)
        {
            state_no = IDLE_STATE;
            lamp_SetLevel(POWER_ON_LEVEL);
        }
        break;
```

```
    case FADE_STATE:
        state_time--;
        if (state_time==0)
        {
            state_no = IDLE_STATE;
        }
        // Calculate fade
        if (fade_start_level>fade_stop_level)
        {
            // Dim down
            if (fade_stop_level==0)
            {
                if (state_time==0)
                {
                    new_level = 0;
                }
                else
                {
                    // Use MIN_LEVEL instead of 0
                    diff_level = (unsigned char)((((unsigned
long)fade_start_level-(unsigned long)MIN_LEVEL)*((unsigned long)fade_total-(unsigned
long)state_time))/(unsigned long)fade_total);
                    new_level = fade_start_level-diff_level;
                }
            }
            else
            {
                diff_level = (unsigned char)((((unsigned
long)fade_start_level-(unsigned long)fade_stop_level)*((unsigned
long)fade_total-(unsigned long)state_time))/(unsigned long)fade_total);
                new_level = fade_start_level-diff_level;
            }
        }
        else
        {
            // Dim up
            diff_level = (unsigned char)((((unsigned long)fade_stop_level-(unsigned
long)fade_start_level)*((unsigned long)fade_total-(unsigned
long)state_time))/(unsigned long)fade_total);
            new_level = fade_start_level+diff_level;
        }
        if (new_level!=level)
        {
            lamp_SetLevel(new_level);
        }
        break;
    default:
        break;
    }
}
```

G.12 Slave: main.c

```
// main.c
//
// This module contains the main function
//
// Prepared: Motorola AB
//
// Functional level: Application
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// R1A  010212  Released version

#include "common.h"
#include "iokx8.h"
#include "cpu.h"
#include "rs232.h"
#include "dali.h"
#include "lamp.h"

// Global variables declaration

unsigned char address;              // The DALI address
unsigned char command;              // The DALI command

unsigned char answer;               // The DALI answer

unsigned int flag;                  // Keep track of all events

// Main function
void main(void)
{
    address = 0x00;                 // Initialize when starting
    command = 0x00;                 // Initialize when starting

    answer = 0x00;                  // Initialize when starting

    flag = 0x00;                    // Initialize when starting

    cpu_Init();                     // Initialize the cpu module
    rs232_Init();                   // Initialize the rs232 module
    dali_Init();                    // Initialize the dali module
    lamp_Init();                    // Initialize the lamp module

    ei();                           // Enable interrupt
```

```
    PTA &= ~0x08;                   // Switch on LED

    while (1==1)                    // Loop forever
    {
        COPCTL = 0x00;              // Clear the COP counter

        // NEW_DATA flag can be set from the dali module or from the rs232
           module
        if (flag & NEW_DATA)
        {
            flag &= ~NEW_DATA;      // Clear the NEW_DATA flag
            lamp_HandleData();
        }

        // SEND_ANSWER flag can be set from the lamp module
        if (flag & SEND_ANSWER)
        {
            flag &= ~SEND_ANSWER;   // Clear the SEND_ANSWER flag
            dali_SendAnswer();      // Send answer on the dali port
            rs232_SendAnswer();     // Send answer on the rs232 port
        }

        // SIGNAL_ERROR flag can be set from the dali module
        if (flag & SIGNAL_ERROR)
        {
            flag &= ~SIGNAL_ERROR;  // Clear the SIGNAL_ERROR flag
            lamp_FailureLevel();
        }
    }
}
```

G.13 Slave: rs232.h

```
// rs232.h
//
// This header contains definitions for the rs232 module
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001027  Initial version
// R1A  010212  Released version

#ifndef RS232
#define RS232

// Initialize
extern void rs232_Init(void);

// Sends the DALI answer on the rs232 port
extern void rs232_SendAnswer(void);

// Interrupt handler for received data on the rs232 port
extern @interrupt void rs232_ReceiveData(void);

#endif
```

G.14 Slave: rs232.c

```
// rs232.c
//
// This module handles all tasks related to the rs232 port
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1B
//
// Rev  Date    Reason/description
// P1A  001027  Initial version
// R1A  010212  Released version
// R1B  011016  Adapted to new clock speed
//
// The serial communication interface (SCI) module is used

#include "common.h"
#include "iokx8.h"

static unsigned char first; // Keeps track if incoming data is address or command

// Initialize
void rs232_Init(void)
{
    SCBR = 0x05;    // Baud rate 9600
    SCC1 = 0x40;    // Enable SCI
    SCC2 = 0x2C;    // Enable receive interrupt, transmitter enabled, receiver
enabled
    first = TRUE;   // Next incoming byte is an address
}

// Sends the answer on the rs232 port
void rs232_SendAnswer(void)
{
    while ((SCS1 & 0x80)==0x00) // Wait until transmit buffer is empty
        ;
    SCDR = answer;
}
```

```
// Interrupt handler for received data on the rs232 port
@interrupt void rs232_ReceiveData(void)
{
    if (SCS1 & 0x20)    // Received data available in SCDR
    {
        if (first==TRUE)
        {
            address = SCDR;
            first = FALSE;  // Next data is a command
        }
        else
        {
            command = SCDR;
            first = TRUE;   // Next data is an address
            flag |= NEW_DATA;
        }
    }
}
```

G.15 Slave: vector.c

```
// vector.c
//
// This module contains the interrupt vector
//
// Prepared: Motorola AB
//
// Functional level: Hardware
//
// Revision: R1A
//
// Rev  Date    Reason/description
// P1A  001023  Initial version
// R1A  010212  Released version

#include "cpu.h"
#include "rs232.h"
#include "dali.h"
#include "lamp.h"

void (* const _vectab[])() =
{
    dali_Tick,          // Timebase module
    cpu_Trap,           // ADC conversion complete
    cpu_Trap,           // Keyboard
    cpu_Trap,           // SCI transmit
    rs232_ReceiveData,  // SCI receive
    cpu_Trap,           // SCI receive error
    cpu_Trap,           // Reserved
    cpu_Trap,           // Reserved
    cpu_Trap,           // Reserved
    cpu_Trap,           // Reserved
    cpu_Trap,           // Reserved
    lamp_Tick,          // TIM overflow
    cpu_Trap,           // TIM channel 1
    cpu_Trap,           // TIM channel 0
    cpu_Trap,           // CMIREQ
    dali_Start,         // IRQ1
    cpu_Trap,           // SWI
    _stext              // Reset
};
```

DALI Slave Source Code Files

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

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

JAPAN:

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

ASIA/PACIFIC:

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

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://www.motorola.com/semiconductors

DRM004/D