To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

RENESAS

Application Note

Digital Addressable Lighting Interface (DALI)

78K0 Series
8-Bit Single-Chip Microcontrollers

Document No. U17333EE1V0AN00
Date Published September 2004

Printed in Germany

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

- The information in this document is current as of 29.09, 2004. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.

- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such NEC Electronics products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.

- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC Electronics no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.

- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.

- NEC Electronics products are classified into the following three quality grades: “Standard”, “Special” and “Specific”.

The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated “quality assurance program” for a specific application. The recommended applications of NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

"Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.

"Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).

"Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is “Standard” unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact NEC Electronics sales representative in advance to determine NEC Electronics 's willingness to support a given application.

Notes: **1.** " NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.

2. " NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02.10

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics America Inc.
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 1101
Fax: 0211-65 03 1327

Sucursal en España
Madrid, Spain
Tel: 091- 504 27 87
Fax: 091- 504 28 60

Succursale Française
Vélizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

Filiale Italiana
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

Branch The Netherlands
Eindhoven, The Netherlands
Tel: 040-244 58 45
Fax: 040-244 45 80

Branch Sweden
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

United Kingdom Branch
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
Singapore
Tel: 65-6253-8311
Fax: 65-6250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

This document shows how to implement a DALI (DIGITAL ADRESSABLE LIGHTING INTERFACE) sub-unit on a general purpose NEC microcontroller. Furthermore, it illustrates the communication protocol, the hardware and the software which is used to control a lamp. In this case, the µPD78F0148H on the demo-board “Taste-it” is used with an additional PCB for the DALI-interface and the power-control of the lamp.
The main unit is done with the PC-interface DALI-SCI, the Power-supply DALI-PS1 and the Control-software WINDIM made by TRIDONIC™.

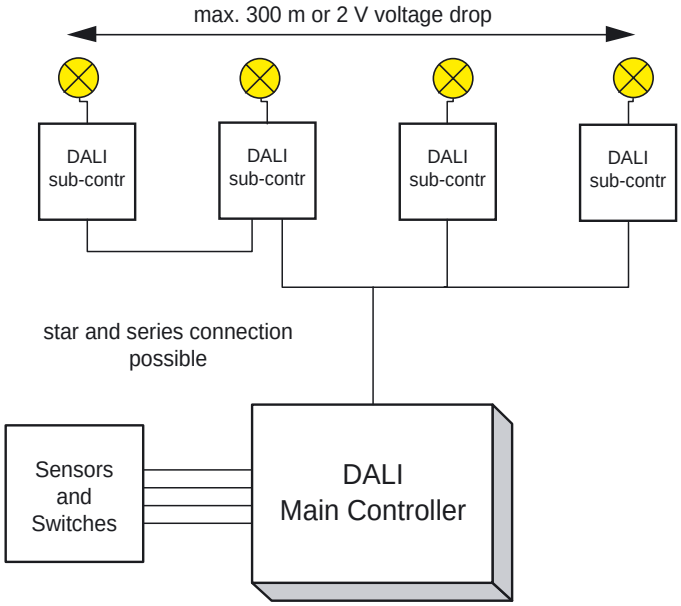
What is DALI?

DALI is an acronym and stands for “Digital Addressable Lighting Interface“. It is an international standard that guarantees the exchangeability of dimmable ballast from different manufacturers. The DALI-interface has been described in the fluorescent lamp ballast standard IEC 60929 under Annex E.

DALI is the ideal, simplified, digital way of communication tailored to the needs of present day lighting technology.

It closes the gap between the analog 1-10 V interface and the more complex building management systems like EIB or LON etc.

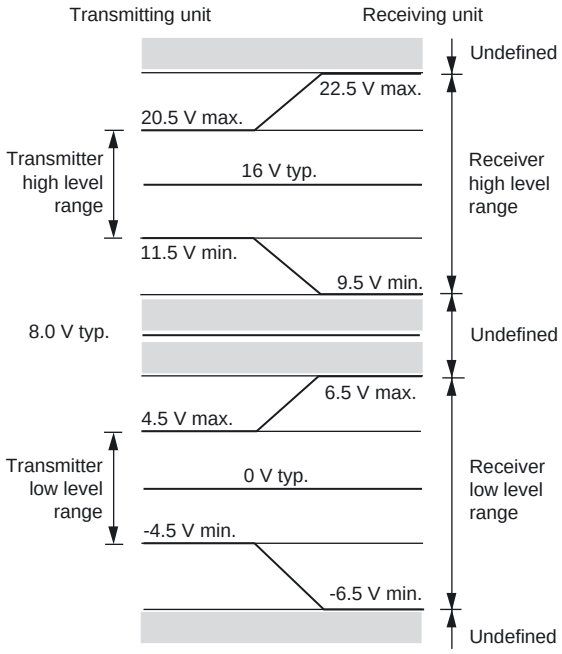
Figure 1-1: DALI Block Diagram



Chapter 2 Electrical specification

The DALI bus is a two-wire interface, the used voltage levels are shown in Figure 2-1. The bus is powered either by the main controller or by an external power-supply. The maximum supply current has to be limited to 250 mA and the current consumption of each device at the DALI-line is max. 2 mA. The communication is performed by sinking the current on the line. Due to the fact that the communication lines are galvanically separated from the mains voltage, all devices can operate on different phases.

Figure 2-1: DALI used voltage levels



2.1 Characteristics and Capability

- Standardization in IEC 60 929, so the combination of devices from different manufactures is possible.
- Slow communication speed (1200 bit/s) and wide disturbing voltage distance guarantees a secure operation.
- Manchester code used, to identify communication errors.
- Simple two-wire interface (non-polarity / potential isolated) with a max. length of 300 m between first and last unit.
  - Power-lines and control-lines can be in the same cable.
  - No terminating resistor required.
- Limited system size, 64 sub-unit can be controlled by one main-controller.
  Broadcast, group or single addressing possible.
- Programmable fade times, fade rates.
  Up to 16 scenes.
  Programmable levels for power-up, power-fail, system-fail etc.

2.2 The Protocol

DALI uses a Manchester encoded unidirectional serial protocol with a transmission rate of 1.2 kHz. So the bit time is 833 µs ±10%.

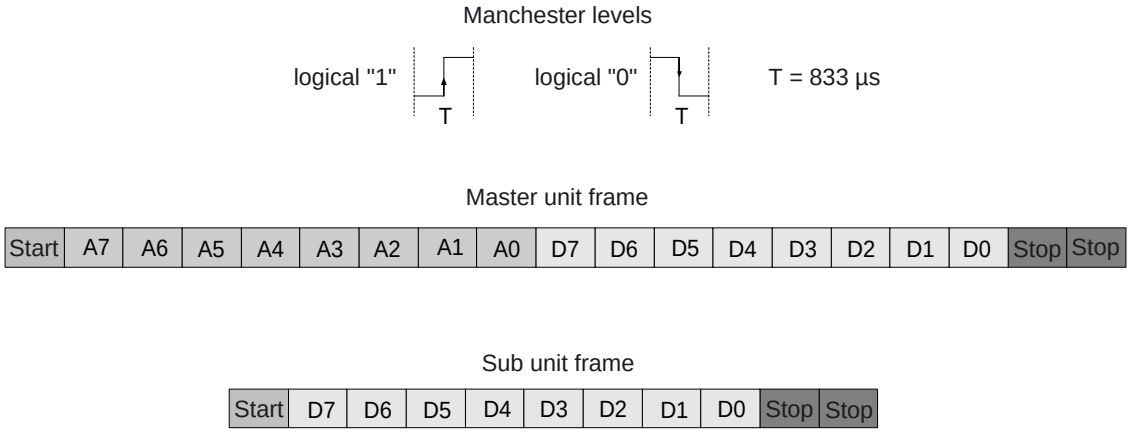
The frame of the main unit consists of 19 bits:

- 1 start-bit (logical 1)
- 8 address bits
- 8 data bits
- 2 stop bits (physical high level)

The answer frame of the sub unit consists of 11 bits:

- 1 start-bit (logical 1)
- 8 data bits
- 2 stop bits (physical high level)

Figure 2-2: DALI Protocol Configuration



To select the devices by the different addressing modes (broadcast, group, single), the following address types are used:

Broadcast:	1111111S	
Group:	100AAAAS	AAAA = 0 to 15
Single:	0AAAAAAS	AAAAAA = 0 to 63

Special command:		101CCCC1	
	or:	110CCCC1	CCCC = special command number

The S distinguishes between an direct arc power command and a DALI command. If S equals 0 the following data is interpreted as a power level with a value between 0x00 (off) and 0xFE (max. power level).

In the tables on the next pages the standard and special commands are listed.

Table 2-1: Standard commands (1/2)

Command Number	Address	Command / Data	Command Name
-	YAAAAAA0	XXXX XXXX	DIRECT ARC POWER CONTROL
0	YAAAAAA1	00000000	OFF
1	YAAAAAA1	00000001	UP
2	YAAAAAA1	00000010	DOWN
3	YAAAAAA1	00000011	STEP UP
4	YAAAAAA1	00000100	STEP DOWN
5	YAAAAAA1	00000101	RECALL MAX LEVEL
6	YAAAAAA1	00000110	RECALL MIN LEVEL
7	YAAAAAA1	00000111	STEP DOWN AND OFF
8	YAAAAAA1	00001000	ON AND STEP UP
9-15	YAAAAAA1	00001XXX	RESERVED
16-31	YAAAAAA1	0001XXXX	GO TO SCENE
32	YAAAAAA1	00100000	RESET
33	YAAAAAA1	00100001	STORE ACTUAL LEVEL IN THE DTR
34-41	YAAAAAA1	0010XXXX	RESERVED
42	YAAAAAA1	00101010	STORE THE DTR AS MAX LEVEL
43	YAAAAAA1	00101011	STORE THE DTR AS MIN LEVEL
44	YAAAAAA1	00101100	STORE THE DTR AS SYSTEM FAILURE LEVEL
45	YAAAAAA1	00101101	STORE THE DTR AS POWER ON LEVEL
46	YAAAAAA1	00101110	STORE THE DTR AS FADE TIME
47	YAAAAAA1	00101111	STORE THE DTR AS FADE RATE
48-63	YAAAAAA1	0011XXXX	RESERVED
64-79	YAAAAAA1	0100XXXX	STORE THE DTR AS SCENE
80-95	YAAAAAA1	0101XXXX	REMOVE FROM SCENE
96-111	YAAAAAA1	0110XXXX	ADD TO GROUP
112-127	YAAAAAA1	0111XXXX	REMOVE FROM GROUP
128	YAAAAAA1	10000000	STORE DTR AS SHORT ADDRESS
129-143	YAAAAAA1	1000XXXX	RESERVED
144	YAAAAAA1	10010000	QUERY STATUS
145	YAAAAAA1	10010001	QUERY BALLAST
146	YAAAAAA1	10010010	QUERY LAMP FAILURE
147	YAAAAAA1	10010011	QUERY LAMP POWER ON
148	YAAAAAA1	10010100	QUERY LIMIT ERROR
149	YAAAAAA1	10010101	QUERY RESET STATE
150	YAAAAAA1	10010110	QUERY MISSING SHORT ADDRESS _
151	YAAAAAA1	10010111	QUERY VERSION NUMBER
152	YAAAAAA1	10011000	QUERY CONTENT DTR
153	YAAAAAA1	10011001	QUERY DEVICE TYPE
154	YAAAAAA1	10011010	QUERY PHYSICAL MINIMUM LEVEL
155	YAAAAAA1	10011011	QUERY POWER FAILURE

Table 2-1: Standard commands (2/2)

Command Number	Address	Command / Data	Command Name
156-159	YAAAAAA1	100111XX	RESERVED
160	YAAAAAA1	10100000	QUERY ACTUAL LEVEL
161	YAAAAAA1	10100001	QUERY MAX LEVEL
162	YAAAAAA1	10100010	QUERY MIN LEVEL
163	YAAAAAA1	10100011	QUERY POWER ON LEVEL
164	YAAAAAA1	10100100	QUERY SYSTEM FAILURE LEVEL
165	YAAAAAA1	10100101	QUERY FADE TIME / FADE RATE
166-175	YAAAAAA1	1010XXXX	RESERVED
176-191	YAAAAAA1	1011XXXX	QUERY SCENE LEVEL (SCENES 0-15)
192	YAAAAAA1	11000000	QUERY GROUPS 0-7
193	YAAAAAA1	11000001	QUERY GROUPS 8-15
194	YAAAAAA1	11000010	QUERY RANDOM ADDRESS (H)
195	YAAAAAA1	11000011	QUERY RANDOM ADDRESS (M)
196	YAAAAAA1	11000100	QUERY RANDOM ADDRESS (L)
197-223	YAAAAAA1	10XXXXX	RESERVED

Table 2-2: Special Commands

Command Number	Address	Command / Data	Command Name
224-255	YAAAAAA1	11XXXXXX	QUERY APPLICATION EXTEND. COMMANDS
256	10100001	00000000	TERMINATE
257	10100011	XXXXXXXX	DATA TRANSFER REGISTER (DTR)
258	10100101	XXXXXXXX	INITIALISE
259	10100111	00000000	RANDOMISE
260	10101001	00000000	COMPARE
261	10101011	00000000	WITHDRAW
262	10101101	00000000	RESERVED
263	10101111	00000000	RESERVED
264	10110001	HHHHHHHH	SEARCHADDRH
265	10110011	MMMMMMMM	SEARCHADDRM
266	10110101	LLLLLLLL	SEARCHADDRL
267	10110111	0AAAAAA1	PROGRAM SHORT ADDRESS
268	10111001	0AAAAAA1	VERIFY SHORT ADDRESS
269	10111011	00000000	QUERY SHORT ADDRESS
270	10111101	00000000	PHYSICAL SELECTION
271	10111111	XXXXXXXX	RESERVED
272	11000001	XXXXXXXX	ENABLE DEVICE TYPE X
273-287	110XXXX1	XXXXXXXX	RESERVED

The DTR (Data transfer register) is a temporary memory in the sub-unit to store data which is shifted to different parameters.

If an answer from the sub-unit is required, a 0xFF will be recognized as a “YES”, no answer after 9.17 ms is a “NO”, otherwise if data should be transmitted every value is accepted.

Every sub-unit has to store several parameters, which are listed in the table below.
Most of these parameters are stored in a non-volatile memory (EEPROM or Flash) to guarantee a proper start-up after a power blackout. In this application a serial EEPROM (3-wire) is used.

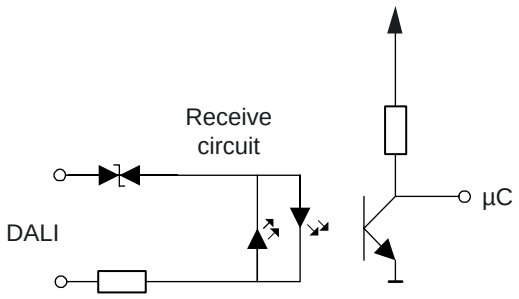
Table 2-3: DTR (Data Transfer Register) Parameters

VARIABLE	DEFAULT VALUE	RESET VALUE	RANGE OF VALIDITY	PERSISTENT MEMORY
ACTUAL DIM LEVEL	????????	254	0, min. - max	(1 byte RAM)
POWER ON LEVEL	254	254	1 -254,	1 byte
SYSTEM FAILURE LEVEL	254	254	0-255 ('MASK')	1 byte
MIN LEVEL	PHYSICAL MIN LEVEL	PHYSICAL MIN LEVEL	PHYS. MIN = MAX LEVEL	1 byte
MAX LEVEL	254	254	MIN LEVEL-254	1 byte
FADE RATE	7 (45 steps/sec)	7 (45 steps/sec)	1-15	1 byte
FADE TIME	0 no fade	0 no fade	0-15	1 byte
SHORT-ADDRESS	255 (MASK) no address	no change	0-63, 255('MASK')	1 byte
SEARCH-ADDRESS	FFFFFF	FFFFFF	000000-FFFFFF	(3 bytes RAM)
RANDOM-ADDRESS	FFFFFF	FFFFFF	000000-FFFFFF	3 bytes
GROUP 0-7	00000000 (no group)	00000000 (no group)	0-255	1 byte
GROUP 8-15	00000000 (no group)	00000000 (no group)	0-255	1 byte
SCENE 0-15	255 MASK	255 MASK	0 - 255 MASK	16 bytes
STATUS INFORMATION	????????	0?100???	0-255	(1 byte RAM)
VERSION NUMBER	factory burn-in	factory burn-in	0-255	(1 byte ROM)
PHYS.MIN. LEVEL	factory burn-in	factory burn-in	1 -254	(1 byte ROM)

Chapter 3 Examples for the receive and transmit circuit

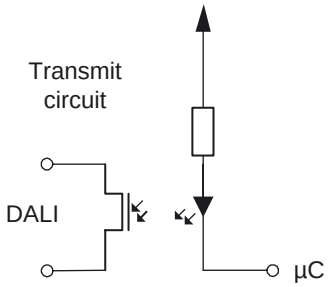
It is determined in the IEC standard, that the devices have to be protected against wrong polarity, so to make the receive circuit insensitive to the polarity an AC-opto-coupler is used. The current is limited by a resistor and a bi-directional Zener diode guarantees the voltage levels.

Figure 3-1: Receive Circuit Example



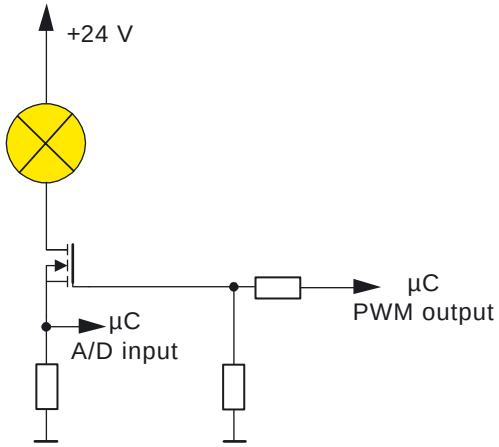
The transmit circuit consists of a high speed optical relay with low switching times, which can sink the current of 250 mA.

Figure 3-2: Transmit Circuit Example



The output circuit is a MOS-FET which is controlled by a PWM-output of the micro-controller. The current through the lamp is sensed by the voltage drop over a shunt-resistor. This voltage is measured with 1 channel of the built-in A/D-converter of the microcontroller.

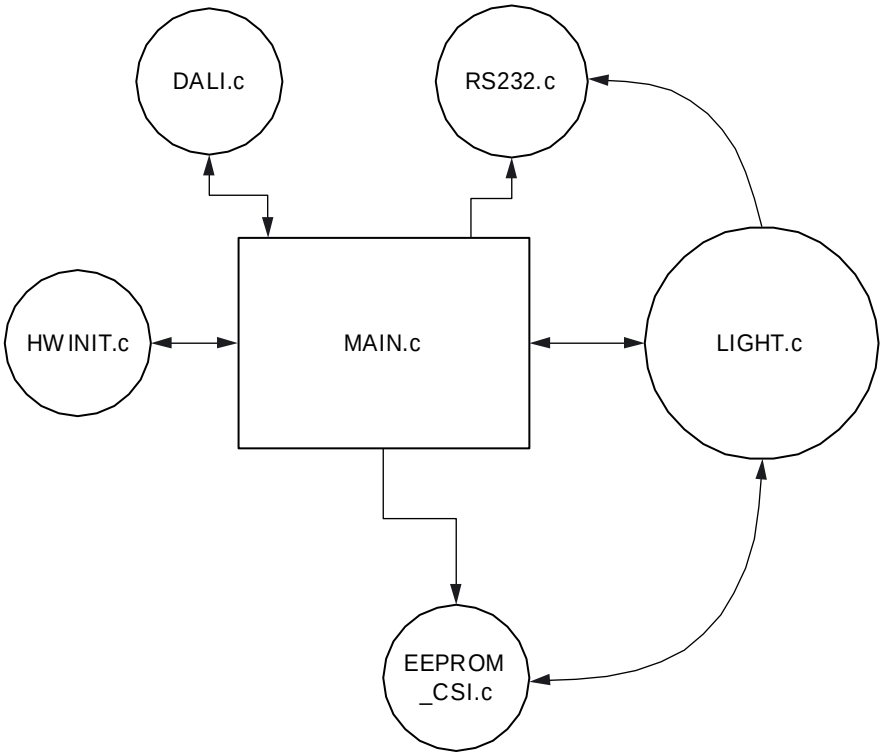
Figure 3-3: PWM Output Circuit Example



Chapter 4 Software

The software for the sub-unit is divided into functional modules.
The structure is shown in the picture below.
The source code is attached at the end of this document.

Figure 4-1: Software Structure



4.1 MAIN.c

This module initializes the microcontroller via the HWINIT.c module, handles the data received by the DALI-bus and will start the transmission of data if requested.
New values are copied from the shadow ram area in the EEPROM.
The complete DALI-communication is send via RS232-interface, so it can be monitored with a PC.

A terminal program will show the following messages:

Message from the main-unit with answer from another subunit:
Addr: 003
Data: 144
Answ: 004

Message from the main-unit with answer from this subunit:
Addr: 003
Data: 144
?Stat <- this shortcut shows the interpreted command, here: query status
MyAnsw: 004

4.2 RS232.c

This module contains the subroutines for the PC communication. The UART of the micro is initialized to 115200 Bd, 8 bit, 1 Stop bit and no parity. Also the decimal-conversion from byte to character is done.

4.3 DALI.c

Here all necessary routines for the DALI-communication are located. There are two interrupt service routines, one for an external interrupt to recognize the start-bit and another one from a timer to handle the bit-timing of transmission/reception and timeout of the communication. These routines can handle both ways of communication for the main unit (16bits) and for the sub-unit (8 bits).

4.4 EEPROM_CSI.c

Some small routines to initialize the CSI (3-wire serial interface) and to communicate with the external EEPROM to store the parameters.

4.5 LIGHT.c

This is the interpreter of the DALI-commands and the control of the lamp. Two timers are used one for the HW-PWM and one for controlling the communication timing and the fading of the lamp. The interpreter-module checks first if the received command is valid for this sub-unit (broadcast, group, or single address). Then it decides if this command is a direct arc-power command, a standard command or a special command and calls the required functions.

4.6 References

http://www.tridonic.at

http://www.dali-ag.org/b_m.htm

IEC 60 929 Appendix E (VDE0712 part 23)

Chapter 5 DALI Source Code Files

5.1 hwinit.c

```
/* ========================================================================
** PROJECT     = dalidemo
** MODULE      = hwinit
** VERSION     = 0.0
** DATE        = 27.01.2004
** LAST CHANGE =
** ========================================================================
** Description: Initialization of CPU-peripherals
**
** ========================================================================
** Environment: Device:         uPD78014x
**              Assembler:      A78000         Version
**              C-Compiler:     ICC78000       Version
**              Linker:         XLINK          Version
**
** ========================================================================
** By:               NEC Electronics (Europe) GmbH
**                   Arcadiastrasse 10
**                   D-40472 Duesseldorf
**
**                   Ingo Scalabrin, NEC-EE, CES
** ========================================================================
Changes:
** ========================================================================
*/



#include <in78000.h>
#include "DF0148.h"

void hwinit(void)
{
    // clock generator setting
    PCC  = 0x00;            // switch with speed
    OSTS = 0x05;
    MOC  = 0x00;            // start main osc.
    while (OSTC<= 0x18)
        {
            _NOP();
        }
    MCM0 = 1;               // set cpu-clock = main osc.

    // watchdog timer setting
    WDTM = 0x77;            // Watchdog off
```

```
    // port setting
    PM0  = 0xFF;        // port 0 = output
    PM1  = 0xFF;        // port 1 = output
    PM3  = 0xFF;        // port 3 = output
    PM4  = 0xFF;        // port 4 = output
    PM5  = 0xFF;        // port 5 = output
    PM6  = 0xFF;        // port 6 = output
    PM7  = 0xFF;        // port 7 = output
    PM12 = 0xFF;        // port 12 = output
    PM14 = 0xFF;        // port 14 = output
    PU0  = 0x00;        // no pull up-resistors
    PU1  = 0x00;        // no pull up-resistors
    PU3  = 0x00;        // no pull up-resistors
    PU4  = 0x00;        // no pull up-resistors
    PU5  = 0x00;        // no pull up-resistors
    PU6  = 0x00;        // no pull up-resistors
    PU7  = 0x00;        // no pull up-resistors
    PU12 = 0x00;        // no pull up-resistors
    PU14 = 0x00;        // no pull up-resistors

    ISC  = 0x00;        // Input control

    // interrupt setting
    IF0L = 0x00;
    IF0H = 0x00;
    IF1L = 0x00;
    MK0L = 0xFF;
    MK0H = 0xFF;
    MK1L = 0xFF;


    // internal memory settings
    IMS  = 0xCF;
    IXS  = 0x0A;


}
```

5.2 eeprom_csi.c

```
/* ========================================================================
** PROJECT     = dalidemo
** MODULE      = eeprom_csi.c
** VERSION     = 0.0
** DATE        = 29.01.2004
** LAST CHANGE =
** ========================================================================
** Description: EEprom_routines (CSI
**
** ========================================================================
** Environment: Device:         uPD78014x
**              Assembler:      A78000         Version
**              C-Compiler:     ICC78000       Version
**              Linker:         XLINK          Version
**
** ========================================================================
** By:               NEC Electronics (Europe) GmbH
**                   Arcadiastrasse 10
**                   D-40472 Duesseldorf
**
**                   Ingo Scalabrin, NEC-EE, EAD-TPS
** ========================================================================
Changes:
** ========================================================================
*/

/*
========================================================================
** pragma
** ========================================================================
*/
#pragma language = extended

/* ========================================================================
** include
** ========================================================================
*/
#include <in78000.h>
#include "DF0148.h"

#define CSOT10 CSIM10.0
#define CS_EEP_EN P0.2 = 0
#define CS_EEP_DIS P0.2 = 1
```

```
void init_csi(void)
{
    PU1.1  = 1;
    PM1.1  = 1;          // SI
    P1.2   = 0;
    PM1.2  = 0;          // SO
    P1.0   = 1;
    PM1.0  = 0;          // SCK
    P0.2   = 0;
    PM0.2  = 0;          // CS_EEprom
    P0.2   = 1;          // deselect EEprom
    CSIC10 = 0x00;       // Mode1 fx/2
    CSIM10 = 0x40;       // Receive/transmit / MSB first
    CSIM10 = 0x80;       // start CSI
}


unsigned char CHK_EEP(void)
{
    while (CSOT10);
    CS_EEP_EN;
    SOTB10 = 0x05;       // command read status
    while (CSOT10);
    SOTB10 = 0x00;       // send dummy
    while (CSOT10);
    CS_EEP_DIS;
    return SIO10;
}

void WR_EEP (unsigned int eep_addr, unsigned char eep_data)
{
    while (CSOT10);
    CS_EEP_EN;
    SOTB10 = 0x06;       // command write enable
    while (CSOT10);
    CS_EEP_DIS;
    CS_EEP_EN;
    SOTB10 = 0x02;       // command write
    while (CSOT10);
    SOTB10 = eep_addr;   // send address
    while (CSOT10);
    SOTB10 = eep_data;   // send data
    while (CSOT10);
    CS_EEP_DIS;
}
```

```
unsigned char RD_EEP (unsigned int eep_addr)
{
    while (CSOT10);
    CS_EEP_EN;
    SOTB10 = 0x03;      // command read
    while (CSOT10);
    SOTB10 = eep_addr;  // send address
    while (CSOT10);
    SOTB10 = 0x00;      //send dummy
    while (CSOT10);
    CS_EEP_DIS;
    return SIO10;
}
```

5.3 rs232.c

```
/* ========================================================================
** PROJECT     = dalidemo
** MODULE      = rs232.c
** VERSION     = 0.0
** DATE        = 29.01.2004
** LAST CHANGE =
** ========================================================================
** Description: UART0-Test
**
** ========================================================================
** Environment: Device:         uPD78014x
**              Assembler:      A78000         Version
**              C-Compiler:     ICC78000       Version
**              Linker:         XLINK          Version
**
** ========================================================================
** By:               NEC Electronics (Europe) GmbH
**                   Arcadiastrasse 10
**                   D-40472 Duesseldorf
**
**                   Ingo Scalabrin, NEC-EE, EAD-TPS
** ========================================================================
Changes:
** ========================================================================
*/

/*
========================================================================
** pragma
** ========================================================================
*/
#pragma language = extended

/* ========================================================================
** include
** ========================================================================
*/
#include <in78000.h>
#include "DF0148.h"


/* constant definitions */


// transmit variables
bit transmit_ready, data_received;
extern saddr unsigned char s[4];
```

```
void rs232_init(void)
{

    //UART6 setting

    //for play it

    PM14 &= 0xFC;
    P14.0 = 0;
    P14.1 = 1;

    P1.3  = 1;
    PM1.3 = 0;
    PM1.4 = 1;
    CKSR6 = 0x00;
    BRGC6 = 0x23;       // 115200 bd @8 MHz
    ASIM6 = 0x81;       // Power-up UART6,error with receive-interrupt
    ASIM6 = 0xE5;       // transmit/receive enable 8, n, 1
    STMK6 = 0;          // transmit interrupt enable
    SRMK6 = 1;          // receive interrupt disable
    transmit_ready=1;
    data_received=0;
}


interrupt [INTST6_vect] void transmit(void)
{
    transmit_ready=1;
}

interrupt [INTSR6_vect] void receive(void)
{

}

void put_one_char (char c)
{
    if (c == '\n')
        {
            while (!transmit_ready);
            transmit_ready=0;
            TXB6 = '\r';
        }
    while (!transmit_ready);
    transmit_ready=0;
    TXB6 = c;
}
```

```
void Print_String (char *s)
{
    while (*s!= '\0')
        put_one_char (*s++);
}

void bytetochar (unsigned char vari)
{
    s[0] =((vari/100)+'0');
    s[1] =(((vari/10)%10)+'0');
    s[2] =((vari%10)+'0');
    s[3] =0;
}
```

5.4 dali.c

```
/* ========================================================================
** PROJECT     = dalidemo
** MODULE      = dali.c
** VERSION     = 0.0
** DATE        = 29.01.2004
** LAST CHANGE =
** ========================================================================
** Description: Dali-interface routines
**
** ========================================================================
** Environment: Device:         uPD78010x
**              Assembler:      A78000         Version
**              C-Compiler:     ICC78000       Version
**              Linker:         XLINK          Version
**
** ========================================================================
** By:               NEC Electronics (Europe) GmbH
**                   Arcadiastrasse 10
**                   D-40472 Duesseldorf
**
**                   Ingo Scalabrin, NEC-EE, EAD-TPS
** ========================================================================
Changes:
** ========================================================================
*/

/* ========================================================================
** pragma
** ========================================================================
*/
#pragma language = extended

/* ========================================================================
** include
** ========================================================================
*/
#include <in78000.h>
#include "DF0148.h"


/* constant definitions */

#define dali_rxdir PM3.0
#define dali_rx P3.0                  // must be interrupt
#define dali_int INTP1_vect
#define dali_int_mk PMK1
#define dali_int_if PIF1
#define dali_int_dirlow EGN.1
#define dali_int_dirhigh EGP.1
#define dali_int_pullup PU3.0


#define dali_txdir PM0.0
#define dali_tx P0.0
```

```
#define dali_timer_control TMC50
#define dali_timer_compare CR50
#define dali_timer_clocksel TCL50
#define dali_timer_if TMIF50
#define dali_timer_mk TMMK50
#define dali_timer_int INTTM50_vect
#define dali_timer_start TCE50
#define dali_timer_prescaler 0x05 // fx/64 =  125 kHz @ 8 MHz
#define dali_timer_initvalue 0x00
#define dali_timer_start_tick0x19 // 26    => 208 us
#define dali_timer_tick      0x33 // 52    => 416 us


bit dali_rx_status, dali_rx_bit, dali_rx_bit_previous, dali_tx_status,
dali_data_received, dali_8_bit;
saddr unsigned char dali_bit_cnt, dali_send_cnt_max, dali_byte1, dali_byte2,
dali_status, dali_err;
saddr unsigned int dali_signal_loss_cnt;



interrupt [dali_int] void dali_rx_interrupt(void)
{
    dali_int_mk = 1;
    dali_timer_start = 0;
    dali_timer_compare = dali_timer_start_tick;
    dali_timer_start = 1;
    dali_timer_mk = 0;
    dali_rx_status = 1;
    dali_status = 0;
    dali_byte1=dali_byte2=0;
    dali_bit_cnt = 0;
    dali_err = 0;
    dali_rx_bit = 0;
    dali_rx_bit_previous = 1;
    dali_data_received=0;
}
```

```
interrupt [dali_timer_int] void dali_timer(void)
{
    if (dali_rx_status)
        {
           dali_rx_bit =!dali_rx;      // signal inverted by optocoupler

            switch (dali_status)
                {
                case 0:                // start bit low level
                    if (!dali_rx_bit)
                        {
                            dali_status++;
                            dali_timer_compare = dali_timer_tick;
                        }
                    else
                        {
                          dali_err = 1;// was spike, dali error = 1
                        }
                    break;

                case 1:                // start bit high level
                    if (dali_rx_bit)
                        {
                            dali_status++;
                        }
                    else
                        {
                            dali_rx_status = 0;
                            dali_int_if = 0;
                            dali_int_mk = 0;
                                       // dali_err = 2;
                                       // low signal was too long, dali
                                       // error=2
                        }
                    break;

                case 2:                // databit & 1.stopbit first half
                    dali_rx_bit_previous = dali_rx_bit;
                    dali_status++;
                    break;
```

```
                case 3:                // databit & 1.stopbit second half
                    if ((dali_rx_bit!= dali_rx_bit_previous)&&
(dali_bit_cnt<16))
                        {
                            if (dali_bit_cnt<8)
                                {
                                    dali_8_bit=1;
                                    dali_byte1=dali_byte1<<1;
                                    if (dali_rx_bit)
                                        dali_byte1|=0x01;
                                }
                            else
                                {
                                    dali_8_bit=0;
                                    dali_byte2=dali_byte2<<1;
                                    if (dali_rx_bit)
                                        dali_byte2|=0x01;
                                 }
                            dali_bit_cnt++;
                            dali_status--;
                        }
                    else
                        {
                        if ((dali_rx_bit)&(dali_rx_bit_previous))
                                    // possible stop bit
                                {
                                 if ((dali_bit_cnt ==8)||(dali_bit_cnt==16))
                                        {
                                          dali_status++;
                                        }
                                    else
                                      dali_err = 3;// stop bit too early
                                }
                            else
                                {
                                  dali_err = 4;    // stop bit not detected
                                }
                        }
                    break;
```

```
            case 4:
                if (dali_rx_bit)
                    {
                      dali_bit_cnt++;
                      if (dali_bit_cnt>1)
                            {
                                //receive completed
                                dali_rx_status=0;
                                dali_timer_start=0;
                                dali_int_if = 0;
                                dali_int_mk = 0;
                                dali_data_received=1;

                            }
                    }
                else
                    {
                        dali_err = 4;// stop bit not detected
                    }
                break;

            default:
                dali_rx_status=0;
                dali_timer_start=0;
                dali_int_if = 0;
                dali_int_mk = 0;
                dali_err = 5;        // not possible
                break;

    }
        if (dali_err!=0)
            {
                //stop receive
                dali_rx_status=0;
                dali_timer_start=0;
                dali_int_if = 0;
                dali_int_mk = 0;
                dali_data_received=1;
            }
    }
```

```
if (dali_tx_status)
    {
        switch (dali_status)
            {
            case 0:
                dali_tx=1;          // start-bit high level
                dali_status++;
                break;

            case 1:                 // first half of bit
                if (dali_bit_cnt<dali_send_cnt_max)
                    {
                        if(dali_bit_cnt<8)
                            {
                                dali_tx=!((dali_byte1&0x80)>0);
                                dali_byte1=dali_byte1<<1;
                            }
                        else
                            {
                                dali_tx=!((dali_byte2&0x80)>0);
                                dali_byte2=dali_byte2<<1;
                            }
                        dali_status++;
                    }
                else
                    {
                        dali_tx=1;   // 1. half of first stop bit
                        dali_status=3;
                        dali_bit_cnt = 0;
                    }

                break;

            case 2:                 // second half of bit
                dali_tx =!dali_tx;
                dali_bit_cnt++;
                dali_status--;
                break;

            case 3:
                if (dali_bit_cnt++>3)// send 3 more high levels for stop bit
                    {
                        dali_tx_status=0;
                        dali_timer_start=0;
                        dali_int_if = 0;
                        dali_int_mk = 0;

                    }
                break;

            default:
                dali_tx_status=0;
                dali_timer_start=0;
                dali_int_if = 0;
                dali_int_mk = 0;
                break;              // normally not possible

            }
```

```
        }

    if (!dali_rx)               // signal inverted by optocoupler
        {
            dali_signal_loss_cnt=0;
        }
    else
        if (dali_signal_loss_cnt++>1201)//1201 *416 us = 500 ms
            {
                dali_signal_loss_cnt=0;
                dali_err = 2;    // low signal was too long, dali error = 2
            }

        }


void dali_init(void)
{
    dali_int_pullup = 1;
    dali_int_dirhigh = 1;
    dali_int_dirlow = 0;
    dali_tx = 1;
    dali_txdir = 0;
    dali_rx = 1;
    dali_rxdir = 1;
    dali_timer_compare = dali_timer_start_tick;
    dali_timer_clocksel = dali_timer_prescaler;
    dali_timer_control = dali_timer_initvalue;
    dali_tx_status = 0;
    dali_rx_status = 0;
    dali_data_received=0;
    dali_timer_if = 0;
    dali_timer_mk = 1;
    dali_int_if = 0;
    dali_int_mk = 0;
    dali_signal_loss_cnt = 0;
    if (dali_rx)                // signal inverted by opto coupler
        {
            dali_timer_compare = dali_timer_tick;
            dali_timer_start = 1;
            dali_timer_mk = 0;
        }

}
```

```
void dali_send (unsigned char addr, unsigned char data, unsigned char twobyte)
{
    if (twobyte==1)
        {
            dali_byte1=addr;
            dali_byte2=data;
            dali_send_cnt_max = 16;
        }
    else
        {
            dali_byte1=data;
            dali_send_cnt_max = 8;
        }
    dali_status = 0;
    dali_bit_cnt = 0;
    dali_tx_status = 1;
    dali_rx_status = 0;
    dali_timer_compare=dali_timer_tick;
    dali_tx=0;
    dali_timer_if = 0;
    dali_timer_mk = 0;
    dali_int_if = 0;
    dali_int_mk = 1;
    dali_timer_start=1;
}
```

5.5 dali_cmd.h

```
/* ========================================================================
** PROJECT     = dalidemo
** MODULE      = dali_cmd.h
** VERSION     = 0.0
** DATE        = 27.01.2004
** LAST CHANGE =
** ========================================================================
** Description: header-file for dali commands
**
** ========================================================================
** Environment: Device:         uPD78014x
**              Assembler:      A78000         Version
**              C-Compiler:     ICC78000       Version
**              Linker:         XLINK          Version
**
** ========================================================================
** By:               NEC Electronics (Europe) GmbH
**                   Arcadiastrasse 10
**                   D-40472 Duesseldorf
**
**                   Ingo Scalabrin, NEC-EE, CES
** ========================================================================
Changes:
** ========================================================================
*/


// direct control commands
#define OFF 0x00
#define UP 0x01
#define DOWN 0x02
#define STEPUP 0x03
#define STEPDOWN 0x04
#define SET_MAX_LEVEL 0x05
#define SET_MIN_LEVEL 0x06
#define DOWN_AND_OFF 0x07
#define ON_AND_UP 0x08
#define SELECT_SCENE 0x10             // -0x1F
#define RESET 0x20

// parameters settings
#define STO_VALUE_IN_DTR 0x21         // 0x22-0x29 reserved
#define STO_DTR_MAX_LEVEL 0x2A
#define STO_DTR_MIN_LEVEL 0x2B
#define STO_DTR_SYS_FAIL_LEVEL 0x2C
#define STO_DTR_PWR_ON_LEVEL 0x2D
#define STO_DTR_FADE_TIME 0x2E
#define STO_DTR_FADE_RATE 0x2F        // 0x30 - 0x3F reserved
#define STO_DTR_SCENE 0x40            // - 0x4F
#define REM_FROM_SCENE 0x50           // -0x5F
#define ADD_TO_GROUP 0x60             // -0x6F
#define REM_FROM_GROUP 0x70           // -0x7F
#define STO_DTR_SHRT_ADDR 0x80        // 0x81 - 0x8F reserved
```

```
// status requests
#define QUY_STATUS 0x90
#define QUY_COMMUNICATE 0x91
#define QUY_LAMP_FAIL 0x92
#define QUY_LAMP_PWR_ON 0x93
#define QUY_LIMIT_ERROR 0x94
#define QUY_RESET_STATE 0x95
#define QUY_MISS_SHRT_ADDR 0x96
#define QUY_VERS_NR 0x97
#define QUY_CONTENT_DTR 0x98
#define QUY_DEVICE_TYPE 0x99
#define QUY_PHYS_MIN_LEVEL 0x9A
#define QUY_PWR_FAIL 0x9B             // 0x9C - 0x9F reserved

// Queries related to arc power parameters settings
#define QUY_ACT_LEVEL 0xA0
#define QUY_MAX_LEVEL 0xA1
#define QUY_MIN_LEVEL 0xA2
#define QUY_PWR_ON_LEVEL 0xA3
#define QUY_SYS_FAIL_LEVEL 0xA4
#define QUY_FADE 0xA5                 // 0xA6 - 0xAF

// Queries related to system parameters settings
#define QUY_SCENE_LEVEL 0xB0          // -0xBF
#define QUY_GROUP_0_7 0xC0
#define QUY_GROUP_8_15 0xC1
#define QUY_rnd_ADDR_H 0xC2
#define QUY_rnd_ADDR_M 0xC3
#define QUY_rnd_ADDR_L 0xC4           // OxC5 - 0xDF reserved

// 0xE0 - 0xFF system specific extended commands


// Terminate special processes
#define TERMINATE 0xA1
// Download information to the dtr
#define DATA_TRANSFER_REGISTER 0xA3

// Addressing commands
#define INIT 0xA5
#define GEN_RND 0xA7
#define COMPARE 0xA9
#define WITHDRAW 0xAB
#define SEARCHADDRH 0xB1
#define SEARCHADDRM 0xB3
#define SEARCHADDRL 0xB5
#define PRG_SHRT_ADDR 0xB7
#define VRF_SHRT_ADDR 0xB9
#define QUY_SHRT_ADDR 0xBB
#define USE_PHYS_SELECT 0xBD
```

5.6 light.c

```
/* ========================================================================
** PROJECT     = dalidemo
** MODULE      = light.c
** VERSION     = 0.0
** DATE        = 7.06.2004
** LAST CHANGE =
** ========================================================================
** Description: Dali-Demo
**
** ========================================================================
** Environment: Device:         uPD78014x
**              Assembler:      A78000         Version
**              C-Compiler:     ICC78000       Version
**              Linker:         XLINK          Version
**
** ========================================================================
** By:               NEC Electronics (Europe) GmbH
**                   Arcadiastrasse 10
**                   D-40472 Duesseldorf
**
**                   Ingo Scalabrin, NEC-EE, EAD-TPS
** ========================================================================
Changes:
** ========================================================================
*/

/* ========================================================================
** pragma
** ========================================================================
*/
#pragma language = extended

/* ========================================================================
** include
** ========================================================================
*/
#include <in78000.h>
#include <stdlib.h>

#include "DF0148.h"
#include "dali_cmd.h"

#define eeprom 0xF400   // ram address for eeprom mirror

#define true 1
#define false 0

#define ntd 0           // nothing to do
#define init 1          // init state
#define fade 2          // lamp in fade-mode
```

```
// dali parameters

#define act_dim_level 0  // ram offset address for actual dim value (254)
#define pwr_on_level 1   // eeprom offset address for power-on value (254)
#define sys_fail_level 2 // eeprom offset address for system failure
                         // value (254)
#define min_level 3      // eeprom offset address for minimum level value
                         // (physical min. value)
#define max_level 4      // eeprom offset address for maximum level value
                         // (physical max. value)
#define fade_rate 5      // eeprom offset address for fade rate value (7)
#define fade_time 6      // eeprom offset address for fade time value
                         // (0\no change)
#define shrt_addr 7      // eeprom offset address for short address
                         // (no change)
#define rnd_addr_h 8     // eeprom offset address for random address
                         // high(255)
#define rnd_addr_m 9     // eeprom offset address for random address
                         // mid (255)
#define rnd_addr_l 10    // eeprom offset address for random address
                         // low (255)
#define group_0_7 11     // eeprom offset address for group 0-7 (0)
#define group_8_15 12    // eeprom offset address for group 8-15 (0)
#define scene_0_15 13    // -28 eeprom offset address + array offset for
                         // 16 values scene 0-15 (255)
#define version 0x00     // 0x version number / subversion
#define phys_min_level 1 // min. phys. level


// Table for dimming light level values
static const unsigned int dt_light_value[255] =
    {0, 0, 2, 4, 6, 9, 14, 18, 36, 41, 46, 47, 47, 48, 48, 50, 53, 56, 58,
68, 69, 69, 70, 73, 77, 80, 81, 81, 82, 84, 86, 88, 91, 94, 98, 102, 106,
111, 116, 120, 124, 128, 133, 138, 143, 148, 152, 156, 160, 164, 169, 174,
178, 182, 186, 190, 194, 197, 200, 202, 209, 216, 222, 228,238, 248, 250,
253, 271, 279, 285, 301, 309, 317, 323, 333, 349, 363, 371, 378, 387, 399,
411, 419, 427, 449, 453, 463, 470, 478, 495, 508, 528, 543, 556, 677, 702,
730, 752, 766, 801, 823, 855, 889, 923, 957, 975, 1011, 1030, 1068, 1110,
1146, 1184, 1228, 1266, 1305, 1361, 1397, 1443, 1491, 1550, 1604, 1654,
1714, 1766, 1817, 1883, 1949, 2011, 2041, 2106, 2182, 2261, 2321, 2413,
2485, 2555, 2658, 2740, 2817, 2911, 3001, 3090, 3190, 3290, 3383, 3489,
3606, 3718, 3822, 3945, 4056, 4180, 4318, 4440, 4578, 4714, 4853, 4991,
5138, 5301, 5457, 5611, 5763, 5912, 6087, 6270, 6442, 6624, 6789, 6998,
7200, 7414, 7633, 7852, 8035, 8297, 8547, 8768, 8998, 9227, 9537, 9793,
10075, 10370, 10677, 10988, 11294, 11629, 11943, 12275, 12621, 12984, 13369,
13749, 14126, 14536, 14931, 15363, 5773, 16213, 16664, 17129, 17592, 18074,
18586, 19120, 19608, 20144, 20725, 21305, 21891, 22497, 23146, 23706, 24391,
25048, 25743, 26467, 27124, 27875, 28657, 29430, 30234, 31053, 32087, 32760,
33676, 34482, 35465, 36421, 36954, 37480, 38960, 39452, 40555, 41167, 42451,
43543,45091, 47300, 48518, 49978, 51373, 52730, 54058, 55501, 57295, 59218,
59510, 61415, 62695, 63945, 64725};
```

```
// Number of timer ticks for each fade time
static const unsigned int fade_tick[16] =
     {0, 43, 61, 86, 122, 172, 244, 345, 488, 690, 976, 1376, 1953, 2762,
3906, 5524};

// Number of level steps during 200 ms
static const unsigned char fade_step[16] =
    {0, 72, 51, 36, 25, 18, 13, 9, 6, 4, 3, 2, 2, 1, 1, 1};


typedef struct
        {
    unsigned char l;
    unsigned char m;
    unsigned char h;
        } characters;

typedef union
        {
    unsigned long l;
    characters b;
        } bit 24;



extern saddr unsigned char addr, data;

saddr unsigned int tm_meas, tm_rpt, tm_adr, tm_state;
saddr unsigned char l_state, dtr, level, search_addr_h, search_addr_m,
search_addr_l;
saddr unsigned char l_fadelevel_start,l_fadelevel_end,fade_diff;
saddr unsigned int l_fade_total;
saddr unsigned char rpt_addr,rpt_data;
saddr unsigned int rnd_cnt;
saddr unsigned withdraw;
saddr unsigned selection;

saddr unsigned char pwmcnt = 0,pwmcnt_value=0;

saddr unsigned long eep_upd_flag;

bit power_failure,limit_error;

bit data_to_transmit;
bit new_fade_data;

saddr unsigned char test,test1;


extern void WR_EEP (unsigned int eep_addr, unsigned char eep_data);
extern unsigned char RD_EEP (unsigned int eep_addr);
extern unsigned char CHK_EEP (void);
extern void Print_String (char *s);
```

```
void init_timer(void);
unsigned char reepr (unsigned char offset);
void wrepr (unsigned char offset, unsigned char data);
void light_init(void);
void l_set level (unsigned char value);
void light_init_eepr(void);
void init_PWM_module(void);
void exec_norm void);
void exec_spec(void);



interrupt [INTTMH0_vect] void timerH0(void)
{
    if (pwmcnt==255)
        pwmcnt=0;
    if (pwmcnt>=pwmcnt_value)
        CMP10 = test;
    else
        CMP10 = test1;
    pwmcnt++;

}


interrupt [INTWTI_vect] void timer(void)
{
    unsigned char diff, nf_level;
    static unsigned int tm_l_fadelevel_start, tm_l_fadelevel_end,
    tm_l_fade_total, tm_fade_diff;

    WTIMK = 1;                  // disable WTI interrupt
    _EI();

    rnd_cnt++;                  // random counter for address
    if (tm_meas!=0) tm_meas--;
    if (tm_rpt!=0) tm_rpt--;
    if (tm_adr!=0) tm_adr--;
    switch (l_state)
        {
        case ntd:               // nothing to do
            break;

        case init:              // init the light
            tm_state--;
            if (tm_state==0)
                {
                    l_state = ntd;
                    l_setlevel(reepr(pwr_on_level));
                }
            break;
```

```
        case fade:                // light is fading
            if (new_fade_data)
                {
                    new_fade_data = 0;
                    tm_l_fadelevel_start = l_fadelevel_start;
                    tm_l_fadelevel_end = l_fadelevel_end;
                    tm_l_fade_total = l_fade_total;
                    tm_fade_diff = fade_diff;

                }

            tm_state--;
            if (tm_state==0)
                {
                    l_state = ntd;
                    if (tm_l_fadelevel_end==0)
                        nf_level = 0;
                }
           diff = (unsigned long)(tm_fade_diff*(tm_l_fade_total-tm_state))/
            tm_l_fade_total;
           nf_level = (tm_l_fadelevel_start>tm_l_fadelevel_end)?
            tm_l_fadelevel_start-diff: tm_l_fadelevel_start+diff;
            if (nf_level!=reepr(act_dim_level))
                l_setlevel(nf_level);
            break;
}
    _DI();
    WTIMK = 0;                    // enable WTI interrupt

}




unsigned char reepr (unsigned char offset)
{
    return (*((unsigned char *)(eeprom+offset)));
}

void wrepr (unsigned char offset, unsigned char data)
{
                                  //WR_EEP (offset, data);

    *((unsigned char *)(eeprom+offset))=data;
    if (offset >0)      // act_dim_level ram-only
        {
            eep_upd_flag |= 1<<offset;
        }

}
```

```
void l_setlevel(unsigned char value)
{
    limit_error = false;
    if (value>=1 && value<=254)  // check limits
        {
            if (value<reepr(min_level))
                {
                    limit_error = true;
                    value = reepr(min_level);
                }
            if (reepr(max_level)<value)
                {
                    limit_error = true;
                    value = reepr(max_level);
                }
        }

    switch (value)
        {
        case 0:                  // lamp off
            P1.5 = 0;
            TMHMD0.0 = 0;        // PWM-timer off output latch = 0
            wrepr(act_dim_level,0);
            break;

        case 254:
            P1.5 = 1;            // lamp on overrule PWM-output
            wrepr(act_dim_level,254);
            break;

        case 255:                // no action
            break;

        default:                 // values between
            pwmcnt_value=dt_light_value[value]%255;
            test = ((unsigned int)(dt_light_value[value]/255));
            test1=test+1;
            CMP10 = test;
            TMHMD0.0 = 1;
            P1.5 = 0;
            wrepr(act_dim_level,value);
            break;

        }

}
```

```
// Initialization
void light_init(void)
{
    unsigned char i;
    // Reset values that are not stored in flash
    power_failure = true;        // This is true until certain data has arrived
    limit_error = false;         // No error
    dtr = 0;
    level = 0;
    search_addr_h = 0xFF;
    search_addr_m = 0xFF;
    search_addr_l = 0xFF;
    // Initialize flash values if necessary
    if (reepr(pwr_on_level)==255)// Check if eeprom has been written
        {
            // Write factory information into eeprom
            wrepr(shrt_addr,255);// No short address
            light_init_eepr();
        }
    for (i=1;i<=28;i++)
        {
          *((unsigned char *)(eeprom+i))= RD_EEP(i);
                                 // get values from eeprom
        }
    tm_rpt  = 0;                 // Initialize when starting
    tm_adr  = 0;                 // Initialize when starting
    // Start timer for initial lamp values
    tm_state= 36;                // ~600 ms
    l_state = init;
    // Initialize the timer interface module
    init_timer();
    init_PWM_module();
}

void init_timer(void)
{
  WTM   = 0x61;                  // enable watch interval timer with
                                 // 16.384ms @ 8MHz
  WTIIF = 0;
  WTIMK = 0;                     // enable WTI interrupt
}
```

```
void light_init_eepr(void)
{
    unsigned char i;

    while ((CHK_EEP()& 01)==01);
    WR_EEP(pwr_on_level,254);
    *((unsigned char *)(eeprom+pwr_on_level))=254;
    while ((CHK_EEP()& 01)==01);
    WR_EEP(sys_fail_level,254);
    *((unsigned char *)(eeprom+sys_fail_level))=254;
    while ((CHK_EEP()& 01)==01);
    WR_EEP(min_level,phys_min_level);
    *((unsigned char *)(eeprom+min_level))=phys_min_level;
    while ((CHK_EEP()& 01)==01);
    WR_EEP(max_level,254);
    *((unsigned char *)(eeprom+max_level))=254;
    while ((CHK_EEP()& 01)==01);
    WR_EEP(fade_rate,7);
    *((unsigned char *)(eeprom+fade_rate))=7;
    while ((CHK_EEP()& 01)==01);
    if(reepr(shrt_addr)==255)
        {
            WR_EEP(fade_time,0);
            *((unsigned char *)(eeprom+fade_time))=0;
        }
    while ((CHK_EEP()& 01)==01);
    WR_EEP(rnd_addr_h,255);
    *((unsigned char *)(eeprom+rnd_addr_h))=255;
    while ((CHK_EEP()& 01)==01);
    WR_EEP(rnd_addr_m,255);
    *((unsigned char *)(eeprom+rnd_addr_m))=255;
    while ((CHK_EEP()& 01)==01);
    WR_EEP(rnd_addr_l,255);
    *((unsigned char *)(eeprom+rnd_addr_l))=255;
    while ((CHK_EEP()& 01)==01);
    WR_EEP(group_0_7,0);
    *((unsigned char *)(eeprom+group_0_7))=0;
    while ((CHK_EEP()& 01)==01);
    WR_EEP(group_8_15,0);
    *((unsigned char *)(eeprom+group_8_15))=0;

    for (i=0;i<16;i++)
        {
            while ((CHK_EEP()& 01)==01);
            WR_EEP(scene_0_15+i,255);
            *((unsigned char *)(eeprom+scene_0_15+i))=255;
        }
}
```

```
void init_PWM_module(void)
{
    PM5.0  = 1;
    P1.5   = 0;
    PM1.5  = 0;
    CMP00  = 0xFF;
    CMP10  = reepr(pwr_on_level);
    wrepr(act_dim_level, reepr(pwr_on_level));
    TMHMD0 = 0x19;                // PWM H0 with fx/16 low-level output enable
    TMHE0  = 1;                   // Start PWM H0
    TMIFH0 = 0;
    TMMKH0 = 0;
}


// Start dimming using the fade time
static void l_StartDimFadeTime (unsigned char value)
{
    if (value!=255)               // Not mask
        {
         if (reepr(fade_time)==0)// No fade
                {
                    l_set level (value);
                    l_state = ntd;
                }
            else                  // Fade with fade time
                {
                    l_fadelevel_start = reepr (act_dim_level);
                    l_fadelevel_end = value;
                   fade_diff = (l_fadelevel_start>l_fadelevel_end)?
l_fadelevel_start-l_fadelevel_end: l_fadelevel_end-l_fadelevel_start;

                    l_fade_total = fade_tick[reepr(fade_time)];
                    tm_state = l_fade_total;
                    l_state = fade;
                    new_fade_data = 1;
                }
        }
}


// Set the light to failure level
void set_l_f_level(void)
{
    l_state = ntd;
    l_setlevel(reepr(sys_fail_level));
}
```

```
unsigned char l_fail_test (void)
{
    unsigned int temp_AD;
    unsigned char temp_act_dim_level;
    ADM = 0x29;                         // conversion time 120/fx, A/D reference on
    ADS = 0x00;                         // select channel 0
    temp_act_dim_level = reepr(act_dim_level);
    l_setlevel(reepr(max_level));     //
    // tm_meas = 3;
    // while (tm_meas!=0) _NOP();
    ADIF = 0;
    ADCS = 1;
    while (!ADIF);
    temp_AD = ADCR>>6;
    ADM = 0x28;                         // stop AD-conversion
    l_setlevel(temp_act_dim_level);
    return (temp_AD>5)? false:true;
}

// Handles new data
void Process_data(void)
{
    unsigned int group;
    if ((addr & 0xE1)!=0xA1 && (addr & 0xE1)!=0xC1 && (addr & 0xFE)!=0xFE) /
/ no special data or broadcast message
        {
            if ((addr & 0xE0)==0x80)  // check Group address
                {
                    group = reepr(group_8_15)<<8 | reepr(group_0_7);
                    if ((group>>((addr & 0x1E)>>1) & 0x01)==0)
                        return;       // Not in this group
                }
            if ((addr & 0x80)==0x00)  // check Short address
                {
                    if (((addr & 0x7E)>>1)!=reepr(shrt_addr))
                        return;       // no short address match
                }
        }

    // Message execution
    // Check for special data which must be repeated in 100 ms
    if (tm_rpt!=0)
        {
            if (addr!=rpt_addr || data!=rpt_data)
                {
                 tm_rpt = 0;          // Ignore message and clear repeat timer
                    return;
                }
        }


    if ((addr & 0x01)==0)             // check direct arc power control command
        {
            power_failure = false;
            l_StartDimFadeTime(data);
            return;
        }
```

```
    if ((addr & 0xE0)==0xA0 || (addr & 0xE0)==0xC0)
                                        // Check for special commands
        {
            if (addr==INIT || addr==GEN_RND)
                                        // These commands shall be repeated
                {
                    if (tm_rpt==0)
                        {
                            rpt_addr = addr;
                            rpt_data = data;
                            tm_rpt = 6;// 100 ms
                            return;
                        }
                }
        }
    else
        {
            if (data>=0x20 && data<=0x80)
                                        // These commands has to be repeated
                {
                    if (tm_rpt==0)
                        {
                            rpt_addr = addr;
                            rpt_data = data;
                            tm_rpt = 6;// 100 ms
                            return;
                        }
                }
        }
    tm_rpt = 0;

    if ((addr & 0xE0)==0xA0 || (addr & 0xE0)==0xC0)
                                        // Check for special commands
        exec_spec();
    else
        exec_norm();
}
```

```
void exec_norm(void)                        // Normal commands

{
    switch (data)
        {
        case OFF:
            Print_String ("OFF");
            Print_String ("\n");
            power_failure = false;
            l_setlevel(0);
            l_state = ntd;
            break;

        case UP:
            Print_String ("UP");
            Print_String ("\n");
           if (reepr(act_dim_level)!=0 && reepr(act_dim_level)!=reepr(max_level))
                {
                    l_fadelevel_start = reepr(act_dim_level);
                    if (fade_step[reepr(fade_rate)]>(reepr(max_level)-
reepr(act_dim_level)))
                        {
                         l_fadelevel_end = reepr(max_level);
                         l_fade_total = (unsigned int)(reepr(max_level)-
reepr(act_dim_level))*12/fade_step[reepr(fade_rate)];
                                                // No of ms until MAX_LEVEL is
                                                // reached
                        }
                    else
                        {
                         l_fadelevel_end = reepr(act_dim_level) +
fade_step[reepr(fade_rate)];
                         l_fade_total = 12;// 200 ms
                }
                    fade_diff = (l_fadelevel_start>l_fadelevel_end)?
l_fadelevel_start-l_fadelevel_end: l_fadelevel_end-l_fadelevel_start;
                    tm_state = l_fade_total;
                    l_state = fade;
                    new_fade_data = 1;
        }
            break;
```

```
        case DOWN:
            Print_String ("Down");
            Print_String ("\n");
            if (reepr(act_dim_level)!=0 && reepr(act_dim_level)!=
reepr(min_level))
                {
                    l_fadelevel_start = reepr(act_dim_level);
                    if (fade_step[reepr(fade_rate)]>(reepr(act_dim_level)-
reepr(min_level)))
                        {
                         l_fadelevel_end = reepr(min_level);
                         l_fade_total = (unsigned int)(reepr(act_dim_level)-
reepr(min_level))*12/fade_step[reepr(fade_rate)];
                                                // No of ms until MAX_LEVEL is
                                                // reached
                        }
                    else
                        {
                         l_fadelevel_end = reepr(act_dim_level)-
fade_step[reepr(fade_rate)];
                         l_fade_total = 12;// 200 ms
                        }
                    fade_diff = (l_fadelevel_start>l_fadelevel_end)?
l_fadelevel_start-l_fadelevel_end: l_fadelevel_end-l_fadelevel_start;
                    tm_state = l_fade_total;
                    l_state = fade;
                    new_fade_data = 1;
                }
            break;

        case STEPUP:
            Print_String ("Step-Up");
            Print_String ("\n");
            if (reepr(act_dim_level)!=0 && reepr(act_dim_level)!=
reepr(max_level))
                {
                    l_setlevel(reepr(act_dim_level)+1);
                    l_state = ntd;
                }
            break;

        case STEPDOWN:
            Print_String ("StepDown");
            Print_String ("\n");
            if (reepr(act_dim_level)!=0 && reepr(act_dim_level)!=
reepr(min_level))
                {
                    l_setlevel(reepr(act_dim_level)-1);
                    l_state = ntd;
                }
            break;

        case SET_MAX_LEVEL:
            Print_String ("SetMax");
            Print_String ("\n");
            power_failure = false;
            l_setlevel(reepr(max_level));
            l_state = ntd;
```

```
        break;

    case SET_MIN_LEVEL:
        Print_String ("SetMin");
        Print_String ("\n");
        power_failure = false;
        l_set level (reepr(min_level));
        l_state = ntd;
        break;

    case DOWN_AND_OFF:
        Print_String ("Down_OFF");
        Print_String ("\n");
        power_failure = false;
        if (reepr(act_dim_level)!=0)
            {
                if (reepr(act_dim_level)==reepr(min_level))
                    l_setlevel(0);
                else
                    l_setlevel(reepr(act_dim_level)-1);
                l_state = ntd;
            }
        break;

    case ON_AND_UP:
        Print_String ("ON_UP");
        Print_String ("\n");
        power_failure = false;
        if (reepr(act_dim_level)!=reepr(max_level))
            {
                if (reepr(act_dim_level)==0)
                    l_set level (reepr(min_level));
                else
                    l_setlevel(reepr(act_dim_level)+1);
                l_state = ntd;
            }
        break;

    case RESET:
        Print_String ("Reset");
        Print_String ("\n");
        search_addr_h = 0xFF;
        search_addr_m = 0xFF;
        search_addr_l = 0xFF;
        light_init_eepr();
        l_setlevel(254);
        l_state = ntd;
        break;

    case STO_VALUE_IN_DTR:
        Print_String ("Sto_DTR");
        Print_String ("\n");
        dtr = reepr(act_dim_level);
        break;
```

```
case STO_DTR_MAX_LEVEL:
    Print_String ("Sto_MAX");
    Print_String ("\n");
    if (dtr>=reepr(min_level) && dtr<=254)
        wrepr(max_level, dtr);
    break;

case STO_DTR_MIN_LEVEL:
    Print_String ("Sto_MIN");
    Print_String ("\n");
    if (dtr>=phys_min_level && dtr<=reepr(max_level))
        wrepr(min_level, dtr);
    break;

case STO_DTR_SYS_FAIL_LEVEL:
    Print_String ("Sto_SysF");
    Print_String ("\n");
    wrepr (sys_fail_level, dtr);
    break;

case STO_DTR_PWR_ON_LEVEL:
    Print_String ("Pwr_ON");
    Print_String ("\n");
    if (dtr>=1 && dtr<=254)
        wrepr(pwr_on_level, dtr);
    break;

case STO_DTR_FADE_TIME:
    Print_String ("Sto_Fadetime");
    Print_String ("\n");
    wrepr (fade_time, dtr & 0x0F);
    break;

case STO_DTR_FADE_RATE:
    Print_String ("Sto_Faderate");
    Print_String ("\n");
    if (dtr & 0x0F)
        wrepr (fade_rate, dtr & 0x0F);
    break;

case STO_DTR_SHRT_ADDR:
    Print_String ("Sto_SADDR");
    Print_String ("\n");
    if (dtr==255)
        wrepr (shrt_addr, 255);
    else
        if (((dtr & 0x7E)>>1)<=63)
            wrepr (shrt_addr, (dtr & 0x7E)>>1);
    break;
```

```
case QUY_STATUS:
    Print_String ("?Stat");
    Print_String ("\n");
    data = 0x00;
    if (l_fail_test()==true)
        data |= 0x02;
    if ((reepr(act_dim_level))!=0)
        data |= 0x04;
    if (limit_error==true)
        data |= 0x08;
    if (l_state==fade)
        data |= 0x10;
    if (l_state==init)
        data |= 0x20;
    if (reepr(shrt_addr)==255)
        data |= 0x40;
    if (power_failure==true)
        data |= 0x80;
    data_to_transmit = true;
    break;

case QUY_COMMUNICATE:
    Print_String ("?Comm");
    Print_String ("\n");
    data = 0xFF;         // YES
    data_to_transmit = true;
    break;

case QUY_LAMP_FAIL:
    Print_String ("?lfail");
    Print_String ("\n");
    if (l_fail_test()==true)
        {
            data = 0xFF; // YES
            data_to_transmit = true;
        }
    break;

case QUY_LAMP_PWR_ON:
    Print_String ("?lpwr");
    Print_String ("\n");
    if (reepr(act_dim_level)!=0)
        {
            data = 0xFF; // YES
            data_to_transmit = true;
        }
    break;

case QUY_LIMIT_ERROR:
    Print_String ("?limit");
    Print_String ("\n");
    if (limit_error==true)
        {
            data = 0xFF; // YES
            data_to_transmit = true;
        }
    break;
```

```
case QUY_RESET_STATE:
    Print_String ("?Reset");
    Print_String ("\n");
    if (l_state==init)
        {
            data = 0xFF; // YES
            data_to_transmit = true;
        }
    break;

case QUY_MISS_SHRT_ADDR:
    Print_String ("?miss_saddr");
    Print_String ("\n");
    if (reepr(shrt_addr)==255)
        {
            data = 0xFF; // YES
            data_to_transmit = true;
        }
    break;

case QUY_VERS_NR:
    Print_String ("?Ver");
    Print_String ("\n");
    data = version;
    data_to_transmit = true;
    break;

case QUY_CONTENT_DTR:
    Print_String ("?DTR");
    Print_String ("\n");
    data = dtr;
    data_to_transmit = true;
    break;

case QUY_DEVICE_TYPE:
    Print_String ("?DEV");
    Print_String ("\n");
    data = 0;
    data_to_transmit = true;
    break;

case QUY_PHYS_MIN_LEVEL:
    Print_String ("?Minlevel");
    Print_String ("\n");
    data = phys_min_level;
    data_to_transmit = true;
    break;

case QUY_PWR_FAIL:
    Print_String ("?PWR_fail");
    Print_String ("\n");
    if (power_failure==true)
        {
            data = 0xFF; // YES
            data_to_transmit = true;
        }
    break;
```

```
case QUY_ACT_LEVEL:
    Print_String ("?Act_lev");
    Print_String ("\n");
    data = reepr(act_dim_level);
    data_to_transmit = true;
    break;

case QUY_MAX_LEVEL:
    Print_String ("?Max_lev");
    Print_String ("\n");
    data = reepr(max_level);
    data_to_transmit = true;
    break;

case QUY_MIN_LEVEL:
    Print_String ("?Min_lev");
    Print_String ("\n");
    data = reepr(min_level);
    data_to_transmit = true;
    break;

case QUY_PWR_ON_LEVEL:
    Print_String ("?PWRON_lev");
    Print_String ("\n");
    data = reepr(pwr_on_level);
    data_to_transmit = true;
    break;

case QUY_SYS_FAIL_LEVEL:
    Print_String ("?Sysfail_lev");
    Print_String ("\n");
    data = reepr (sys_fail_level);
    data_to_transmit = true;
    break;

case QUY_FADE:
    Print_String ("?Fade");
    Print_String ("\n");
    data = reepr(fade_time)<<4 | reepr (fade_rate);
    data_to_transmit = true;
    break;

case QUY_GROUP_0_7:
    Print_String ("?GR 0-7");
    Print_String ("\n");
    data = reepr(group_0_7);
    data_to_transmit = true;
    break;

case QUY_GROUP_8_15:
    Print_String ("?GR 8-15");
    Print_String ("\n");
    data = reepr(group_8_15);
    data_to_transmit = true;
    break;
```

```
        case QUY_rnd_ADDR_H:
            Print_String ("?rnad_h");
            Print_String ("\n");
            data = reepr (rnd_addr_h);
            data_to_transmit = true;
            break;

        case QUY_rnd_ADDR_M:
            Print_String ("?rnad_m");
            Print_String ("\n");
            data = reepr (rnd_addr_m);
            data_to_transmit = true;
            break;

        case QUY_rnd_ADDR_L:
            Print_String ("?rnad_l");
            Print_String ("\n");
            data = reepr (rnd_addr_l);
            data_to_transmit = true;
            break;

        default:
            break;
                }
    switch (data & 0xF0)
        {
        case SELECT_SCENE:
            Print_String ("Sel_Scene");
            Print_String ("\n");
            power_failure = false;
            l_StartDimFadeTime(reepr(scene_0_15+(data & 0x0F)));
            break;

        case STO_DTR_SCENE:
            Print_String ("Sto_Scene");
            Print_String ("\n");
            wrepr(scene_0_15+(data & 0x0F), dtr);
            break;

        case REM_FROM_SCENE:
            Print_String ("Rem_Scene");
            Print_String ("\n");
            wrepr(scene_0_15+(data & 0x0F), 255);
            break;

        case ADD_TO_GROUP:
            Print_String ("Add_Gr.");
            Print_String ("\n");
            if ((data & 0x0F)<=7)
              wrepr(group_0_7,(reepr(group_0_7)|(1<<(data & 0x0F))));
            else
              wrepr(group_8_15,(reepr(group_8_15) | (1<<((data & 0x0F)-8))));
            break;
```

```
        case REM_FROM_GROUP:
            Print_String ("Rem_Gr.");
            Print_String ("\n");
            if ((data & 0x0F)<=7)
                wrepr(group_0_7, (reepr(group_0_7) & ~(1<<(data & 0x0F))));
            else
               wrepr(group_8_15, (reepr(group_8_15) & ~(1<<((data & 0x0F)-8))));
            break;

        case QUY_SCENE_LEVEL:
            Print_String ("?Scene_lev");
            Print_String ("\n");
            data = reepr(scene_0_15+(data & 0x0F));
            data_to_transmit = true;
            break;

        default:
            break;
        }
}

void exec_spec(void)
{
    unsigned char uc_help;
    bit24 rnd_addr;
    bit24 srch_addr;
    // Special data
    if (addr>=0xA7 && addr<=0xBD && tm_adr==0)
                        // Not a valid data until INIT has been executed
        return;

    switch (addr)
        {

        case TERMINATE:
            Print_String ("Terminate");
            Print_String ("\n");
            tm_adr = 0;
            withdraw = false;
            selection = false;
            break;

        case DATA_TRANSFER_REGISTER:
            Print_String ("Lo_data");
            Print_String ("\n");
            dtr = data;
            break;
```

```
        case INIT:
            Print_String ("Init");
            Print_String ("\n");
            if (data==0x00 || ((data & 0x7E)>>1)==reepr(shrt_addr) ||
(data==0xFF && (reepr(shrt_addr)==0xFF)))
                {
                 tm_adr    = 54931;     // Accept addressing data for the
                                        // next 15 minutes
                 withdraw  = false;
                 selection = false;
                }
                 break;

        case GEN_RND:
            Print_String ("Gen_Rnd");
            Print_String ("\n");
            srand (rnd_cnt);
            uc_help=rand();
            wrepr(rnd_addr_h, uc_help);// store high address
            srand (rnd_cnt+uc_help);
            uc_help=rand();
            wrepr(rnd_addr_m, uc_help);// store mid address
            srand (rnd_cnt+uc_help);
            uc_help=rand();
            wrepr(rnd_addr_l, uc_help);// store low address
            break;

        case COMPARE:
            Print_String ("Comp");
            Print_String ("\n");
            rnd_addr.b.h = reepr (rnd_addr_h);
            rnd_addr.b.m = reepr (rnd_addr_m);
            rnd_addr.b.l = reepr (rnd_addr_l);

            srch_addr.b.h = search_addr_h;
            srch_addr.b.m = search_addr_m;
            srch_addr.b.l = search_addr_l;
            // if (rnd_addr.l!=srch_addr.l || withdraw==false)
            // {
            if ((rnd_addr.l<=srch_addr.l)&&(withdraw==false))
                {
                    data = 0xFF;       // YES
                    data_to_transmit = true;
                }
            // }
            break;
```

```
case WITHDRAW:
    Print_String ("Withdr.");
    Print_String ("\n");
    rnd_addr.b.h = reepr (rnd_addr_h);
    rnd_addr.b.m = reepr (rnd_addr_m);
    rnd_addr.b.l = reepr (rnd_addr_l);

    srch_addr.b.h = search_addr_h;
    srch_addr.b.m = search_addr_m;
    srch_addr.b.l = search_addr_l;
    if (rnd_addr.l==srch_addr.l)
        withdraw = true;
    break;

case SEARCHADDRH:
    Print_String ("Search_H");
    Print_String ("\n");
    search_addr_h = data;
    break;

case SEARCHADDRM:
    Print_String ("Search_M");
    Print_String ("\n");
    search_addr_m = data;
    break;

case SEARCHADDRL:
    Print_String ("Search_L");
    Print_String ("\n");
    search_addr_l = data;
    break;
```

```
case PRG_SHRT_ADDR:
    Print_String ("Prg_Saddr");
    Print_String ("\n");
    if (selection==true)
        {
            if (l_fail_test()==true)
                {
                    if (data==255)
                        wrepr (shrt_addr, 255);
                    else
                        {
                            if (((data & 0x7E)>>1)<=63)
                                wrepr (shrt_addr, (data & 0x7E)>>1);
                        }
                }
        }
    else
        {
            rnd_addr.b.h = reepr (rnd_addr_h);
            rnd_addr.b.m = reepr (rnd_addr_m);
            rnd_addr.b.l = reepr (rnd_addr_l);
            srch_addr.b.h= search_addr_h;
            srch_addr.b.m= search_addr_m;
            srch_addr.b.l= search_addr_l;
            if ((rnd_addr.l==srch_addr.l)&&(withdraw==false))
                {
                    if (data==255)
                        wrepr (shrt_addr, 255);
                    else
                        {
                            if (((data & 0x7E)>>1)<=63)
                                wrepr (shrt_addr, (data & 0x7E)>>1);
                        }
                }
        }
    break;

case VRF_SHRT_ADDR:
    Print_String ("Vrf_Saddr");
    Print_String ("\n");
    if (reepr (shrt_addr)==((data & 0x7E)>>1))
        {
            data = 0xFF;      // YES
            data_to_transmit = true;
        }
    break;
```

```
        case QUY_SHRT_ADDR:
            Print_String ("?Saddr");
            Print_String ("\n");
            if (selection==true)
                {
                    if (l_fail_test()==true)
                        {
                            data = (reepr(shrt_addr)<<1)|0x01;
                            data_to_transmit = true;
                        }
                }
            else
                {
                    rnd_addr.b.h = reepr (rnd_addr_h);
                    rnd_addr.b.m = reepr (rnd_addr_m);
                    rnd_addr.b.l = reepr (rnd_addr_l);
                    srch_addr.b.h= search_addr_h;
                    srch_addr.b.m= search_addr_m;
                    srch_addr.b.l= search_addr_l;
                    if (rnd_addr.l== srch_addr.l)
                        {
                            data = (reepr(shrt_addr)<<1)|0x01;
                            data_to_transmit = true;
                        }
                }
            break;

        case USE_PHYS_SELECT:
            Print_String ("Phy_Sel");
            Print_String ("\n");
            selection = true;
            break;


        default:
            break;
    }
}
```

5.7 main.c

```
/* ========================================================================
** PROJECT     = dalidemo
** MODULE      = main.c
** VERSION     = 0.0
** DATE        = 29.01.2002
** LAST CHANGE =
** ========================================================================
** Description: dali-demo
**
** ========================================================================
** Environment: Device:         uPD78010x
**              Assembler:      A78000         Version
**              C-Compiler:     ICC78000       Version
**              Linker:         XLINK          Version
**
** ========================================================================
** By:               NEC Electronics (Europe) GmbH
**                   Arcadiastr. 10
**                   D-40472 Duesseldorf
**
**                   Ingo Scalabrin, NEC-EE, CES-TPS
** ========================================================================
Changes:
** ========================================================================
*/

/* ========================================================================
** pragma
** ========================================================================
*/
#pragma language = extended

/* ========================================================================
** include
** ========================================================================
*/
#include <in78000.h>
#include "DF0148.h"


extern void hwinit(void);
extern void rs232_init(void);
extern void Print_String (char *s);
extern void dali_init(void);
extern void light_init(void);
extern void Process_data(void);
extern void dali_send (unsigned char addr, unsigned char data, unsigned char
twobyte);
extern void init_csi(void);

extern saddr unsigned char dali_byte1,dali_byte2;// receive buffer
extern saddr unsigned char dali_err;             // error message
extern bit dali_data_received,dali_8_bit;        // dali status flags

extern void WR_EEP (unsigned int eep_addr, unsigned char eep_data);
extern unsigned char RD_EEP (unsigned int eep_addr);
```

```
extern unsigned char CHK_EEP (void);
extern unsigned char reepr (unsigned char offset);
extern saddr unsigned long eep_upd_flag;

extern void bytetochar (unsigned char vari);
extern bit data_to_transmit;     // flag, if answer is required


saddr unsigned char s[4],addr,data;
saddr unsigned char dali_channel;



void main(void)
{
    unsigned char i;

    _DI();                       // interrupt disable
    hwinit ();                   // peripheral settings
    _EI();                       // interrupt enable
    rs232_init();
    init_csi();
    dali_init();
    light_init();

    Print_String ("Dali_Test V1.0 \n");

    while(1)                     // endless loop - main loop
        {
            if (dali_data_received)
                {
                    if (!dali_8_bit)
                        {
                            addr = dali_byte1;
                            data = dali_byte2;
                            Print_String ("Addr: ");
                            bytetochar (addr);
                            Print_String (s);
                            Print_String ("\n");
                            Print_String ("Data: ");
                            bytetochar (data);
                            Print_String (s);
                            Print_String ("\n");
                            Process_data();
                        }
                    else
                        {
                            // Answer from other slave
                            data = dali_byte1;
                            dali_data_received = 0;
                            Print_String ("Answ: ");
                            bytetochar (data);
                            Print_String (s);
                            Print_String ("\n");
                        }
                    dali_data_received = 0;
                }
```

```
            if (data_to_transmit)
                {
                    Print_String ("MyAnsw: ");
                    bytetochar (data);
                    Print_String (s);
                    Print_String ("\n");
                    dali_send(0,data,0);
                    data_to_transmit = 0;
                }

            if (dali_err!=0)
                {
                    Print_String ("Error: ");
                    bytetochar (dali_err);
                    Print_String (s);
                    Print_String ("\n");
                    dali_err=0;
                }

            if (eep_upd_flag>0)
                if ((CHK_EEP()& 01)!=01)
                    {
                        for(i=1;i<=28;i++)
                            {
                                if ((eep_upd_flag & (1<<i))>0)
                                    {
                                        eep_upd_flag &= ~(1<<i);
                                        WR_EEP (i,reepr(i));
                                        break;
                                    }
                            }

                    }


        }
}
```

NEC

Facsimile Message

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

From:

Name

Company

Tel. FAX

Address

Thank you for your kind support.

North America
NEC Electronics America Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Market Communication Dept.
Fax: +49(0)-211-6503-1344

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-6250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: +81- 44-435-9608

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	☐	☐	☐	☐
Technical Accuracy	☐	☐	☐	☐
Organization	☐	☐	☐	☐

[MEMO]