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

Displaying Data With 78K0/Lx2 LCD Controllers

Document no. U18273EU1V0AN00

The information in this document is current as of July 2006. 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

Revision History

Date	Revision	Section	Description
July 2006	—	—	First release

Contents

1. Introduction

This document demonstrates use of the peripherals included in NEC Electronics' microcontroller devices, through simple examples.

The document includes:

- Descriptions of peripheral features
- Example program descriptions and specifications
- Software flow charts
- Applilet reference drivers
- Description of the demonstration platforms
- Hardware block diagram
- Lists and descriptions of software modules

The Applilet is a software tool that generates driver code for the peripherals, providing a means to generate code for the on-chip peripherals for quick evaluation.

For further details, please see the user manual for the specific device.

1.1 Overview of LCD Controllers

NEC Electronics' microcontrollers offer complete, ready-to-connect-and-use LCD controllers able to drive a variety of segment sizes. These controllers support, as standard features, reference-voltage generation, automatic display of data read from memory, and display modes ranging from static to multiple time-division display.

2. LCD Data Display with 78K0/Lx2 Microcontrollers

To display data on the LCD, simply write prepared data in the memory. The controller then drives the appropriate segments of an LCD panel. The integral controller handles all complex biasing tasks, reference-voltage generation and time-division duty cycle control. The example presented in this application note uses the 78K0/LG2 (µPD78F0397) – a member of the NEC Electronics 78K0/Lx2 microcontroller family.

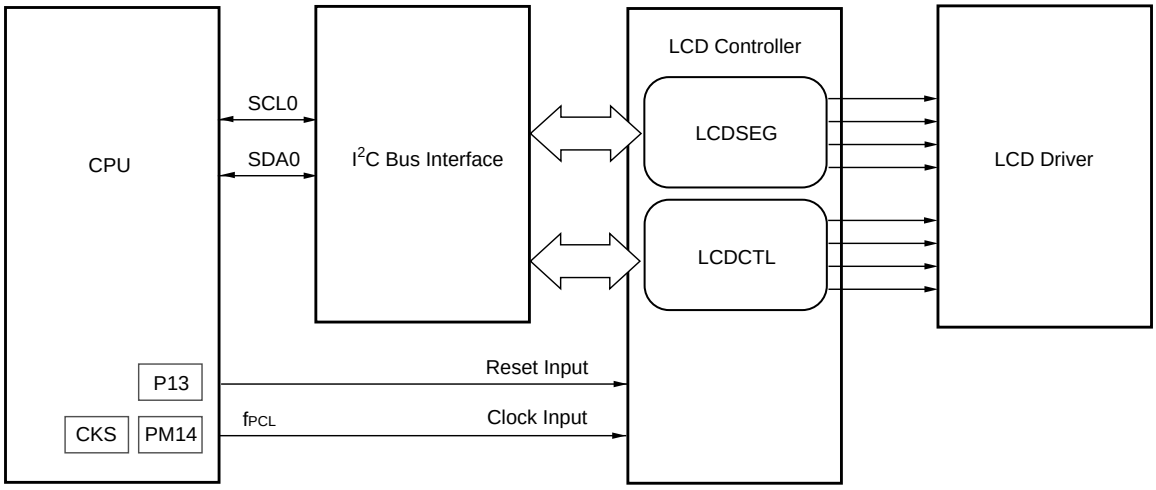
2.1 Features of LCD Controllers

The 78K0/LG2 microcontroller implements an on-chip LCD controller, which provides:

- Internal LCD voltage generation, with optional boost for higher-voltage LCD panels
- Optional external LCD voltage reference through a resistor divider
- Five display modes:
  - Static
  - 1/2 duty, 1/2 bias
  - 1/3 duty, 1/2 bias
  - 1/3 duty, 1/3 bias
  - 1/4 duty, 1/3 bias
- Internal LCD-frame clock, with selectable frame frequencies
- Four common and 40 segment lines, capable of driving up to 160 LCD segments

Figure 1 shows the block diagram of the 78K0/LG2 CPU and its LCD controller. The LCDSEG block generates and controls segment signals. The LCDCTL block provides control-register and display-mode settings.

Figure 1. CPU and LCD Controller Block Diagram



The CPU block provides the LCD controller with clocks and data via the IIC bus interface. These are all internal signals.

Table 1. Settings for System Configuration

Item		Configuration
LCD controller/driver	Display outputs (LCDSEG)	40 segment signals 4 common signals (COM0 to COM3)
	Control registers (LCDCTL)	LCD mode setting register (LCDMD) LCD display mode register (LCDM) LCD clock control register (LCDC) LCD voltage boost control register 0 (VLCG0)
CPU	Control registers	Clock output selection register (CKS) Port register 13 (P13) Port mode register 14 (PM14)

Table 2. Registers Controlling LCD Operation

Symbol	Register Name	Register Function Descriptions
LCDMD	Mode Setting Register	Sets number of segments and LCD reference voltage generator
LCDM	Display Mode Register	LCD Display Enable/Disable
		Segment-Pin and Common-Pin Output Level Select
		Voltage Booster Circuit Enable/Disable
		LCD Display Mode Select: Time-Division and Biasing Select
LCDC	Clock Control Register	LCD Clock Source and Frequency Selection
VLCG0	Voltage Boost Control Register-0	Controls voltage booster level during voltage booster operation
CKS	Clock Output Selection Register	Enable/Disable Clock Output (PCL) to the LCD Controller
		Select PLC Output Clock Frequencies
P13	Port Register 13	P13 sets or release Reset for the LCD Controller
PM14	Port Mode Register 14	Enable/Disable clock output or the LCD Controller

When setting the LCD voltage-booster control register (VLCG0), refer to the LCD panel-voltage specification.

The LCD controller in the 78K0/LG2 supports 40-segment (S0 - S39) outputs and 4-common (COM0 - COM3) operation in 1/2-, 1/3- and 1/4-duty time-division. Table 3 summarizes the specific features of the 78K0/LG2. For applications requiring other segment sizes and numbers of common signals you can use other members of the 78K0 family.

Table 3. Internal 78K0/LG2 LCD Controller Features

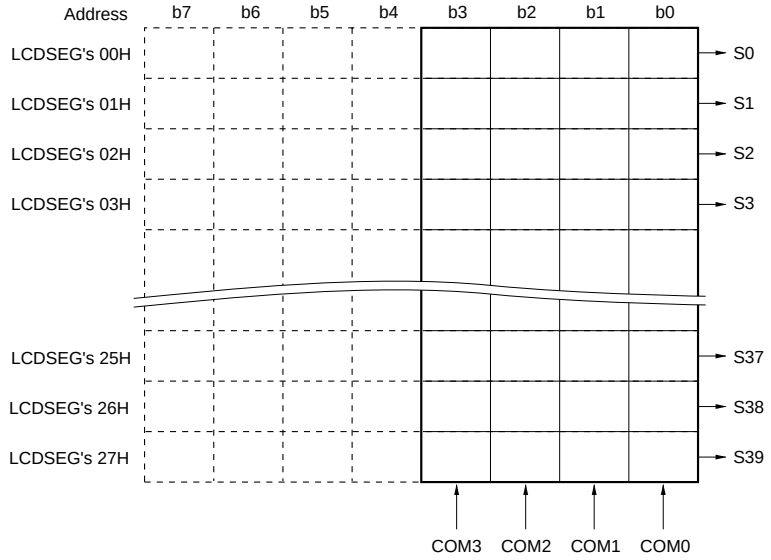
LCD Driver Reference Voltage Generator	Bias Mode	Number of Time Slices	Common Signals Used	Number of Segments	Maximum Number of Pixels
• External-resistance division • Internal-resistance division	–	Static	COM0 (COM1 to COM3)	40	40 (40 segment signals, 1 common signal)Note 1
	1/2	2	COM0, COM1		80 (40 segment signals, 2 common signals)Note 2
		3	COM0 to COM2		120 (40 segment signals, 3 common signals)Note 3
• Internal voltage boosting • External-resistance division • Internal-resistance division	1/3	3	COM0 to COM2		
		4	COM0 to COM3		160 (40 segment signals, 4 common signals)Note 4

Notes:

1. 5-digit LCD panel, each digit having an 8-segment configuration
2. 10-digit LCD panel, each digit having a 4-segment configuration
3. 15-digit LCD panel, each digit having a 3-segment configuration
4. 20-digit LCD panel, each digit having a 2-segment configuration

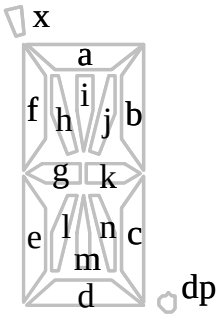
The controller maps LCD display memory at addresses 00h - 27h of LCDSEG. The controller displays the data stored in these memory locations. Figure 2 shows the relationship between data in the LCD display memory and the segment/common signal outputs.

Figure 2. Segment and Common Signal Output, Based on Display Data



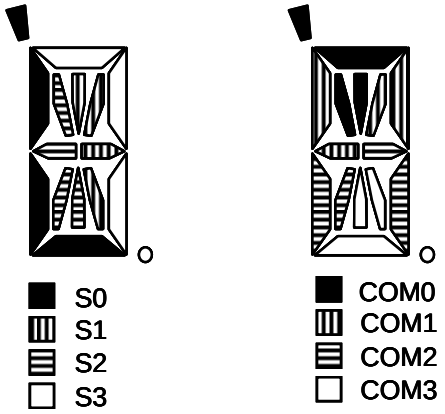
For example, to display data on a 14-segment alphanumeric LCD panel, arrange the segments, decimal point, and additional ‘x’ segment as shown in Figure 3.

Figure 3. Arrangement of 14-Segment Alphanumeric Display



This arrangement of segments provides high-quality representation of letters and numbers. To drive the LCD module, the 78K0/LG2 uses a combination of segment lines (S0–S39) and common lines (COM0–COM3). Figure 4 shows how the segment and common lines connect to display a single character.

Figure 4. Connections of Segment and Common Line



The 78K0/LG2 LCD controller automatically drives waveforms on the common and segment lines to turn desired segment in a digit on or off. Writing either a zero or a one into a bit of internal memory causes the appropriate segment to turn on (if the bit is 1) or off (if the bit is 0). Given this connection of segment and common lines, Table 4 shows the correspondence of internal LCD memory bits to segments.

Table 4. Correspondence of Display Memory to Segments

RAM Address	Bits							
	7	6	5	4	3	2	1	0
	Not Used by LCD Controller				COM3	COM2	COM1	COM0
SEG+0	0	0	0	0	d	e	f	x
SEG+1	0	0	0	0	n	k	j	i
SEG+2	0	0	0	0	m	l	g	h
SEG+3	0	0	0	0	dp	c	b	a

Each of the display's digits requires four RAM locations to specify whether the segments are on or off. The controller uses only the lower four bits of each location, the COM0–COM3 areas; the address of the RAM location specifies the segment(s) to be driven. For example, to display the integer "8" the controller turns on segments a, b, c, d, e, f, g, and k. All other segments remain off.

Table 5. Common Lines to Display the Integer "8"

RAM Address	Bits								
	7	6	5	4	3	2	1	0	
	Not Used by LCD Controller				COM3	COM2	COM1	COM0	
SEG+0	0	0	0	0	1	1	1	0	= 0x0E
SEG+1	0	0	0	0	0	1	0	0	= 0x04
SEG+2	0	0	0	0	0	0	1	0	= 0x02
SEG+3	0	0	0	0	0	1	1	1	= 0x07

To configure the LCD controller for operation:

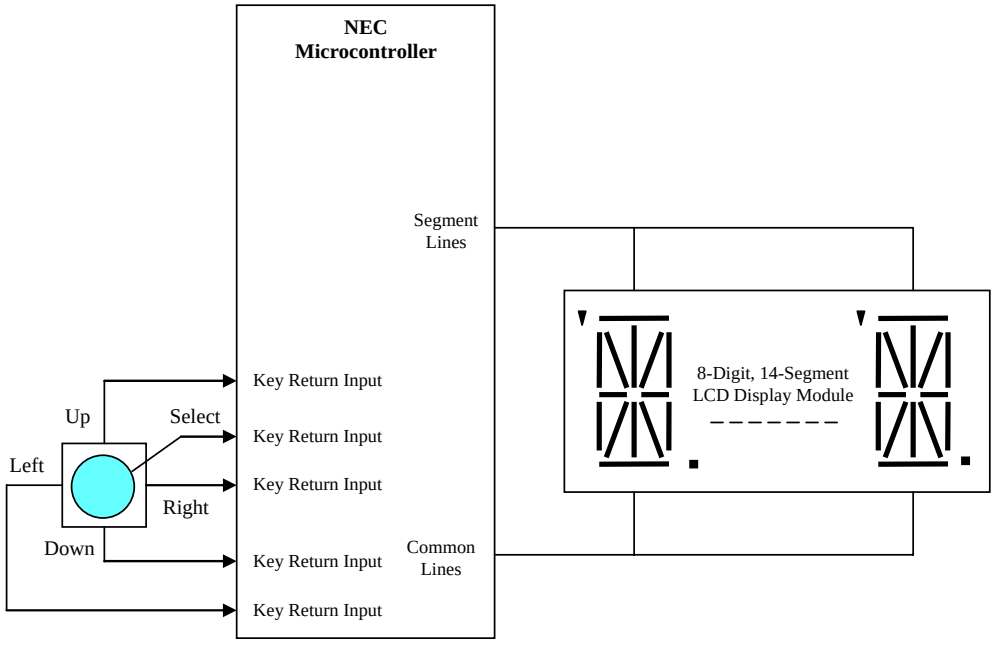
- Release the LCD controller reset, set the clock source and turn on the clock output, using the P13.0, PM14.0, and CKS registers.
- Set the internal-voltage boost, if used, with the LCDMD register.
- Set the initial data in LCDSEG memory.
- Set the LCD display mode with the LCDM register.
- Set the LCD voltage-boost and frame frequency with the LCDC register.
- Set the voltage gain using the VLCG0 register.
- Turn on the voltage booster using the VLCON bit in the LCDM register.
- Wait for a specified amount of time, depending on gain and device LVDD voltage.
- Disconnect LCD common and segment lines from GND by setting the SCOC bit of the LCDM register to 1.
- Set the LCDON bit of the LCDM register to 1, to enable the display.

At this point, the display is enabled and displaying data. Writing to the LCDSEG memory changes the display data.

2.2 Program Description and Specification

The demonstration program displays data on an 8-digit, 14-segment LCD panel, using a navigation switch for input. When you press the navigation switch, the display shows switch descriptions.

Figure 5. Demonstration Block Diagram



- Pressing LEFT Display "LEFT" on 14-Segment Alphanumeric Display
- Pressing RIGHT Display "RIGHT"
- Pressing UP Display "UP"
- Pressing DOWN Display "DOWN"
- Pressing SELECT Display "SELECT"

Specifications:

- The controller considers a switch debounced when the data is constant 5 milliseconds after input.
- The LCD uses internal voltage generation, with boost for 4.5V.
- The microcontroller expects to see LVDD as 5.0V.
- The controller configures the display for 1/3 bias, 1/4 duty cycle.
- The display clock runs at 32.768 kHz.
- The LCD voltage boost frequency is 32.768 kHz.
- The LCD frame frequency is 128 Hz (32-Hz full refresh at 4 time slices).
- IIC communications to the controller uses a 90,900-Hz bit clock.

2.3 Software Flow Charts

The demonstration program consists of:

- Program-initialization code, called before the main() program starts
- IIC0 controller and LCD controller initialization
- The main program loop, which checks the status of switches and displays strings on the LCD
- LCD subroutines to display data by writing into the LCD segment memory
- Subroutines generated by the Applilet for IIC0 communication
- Subroutines generated by the Applilet to handle timer starting, stopping, and interval setting
- Subroutines with user code for handling key return interrupts (Applilet-generated stub interrupt service routines, with additional user code)

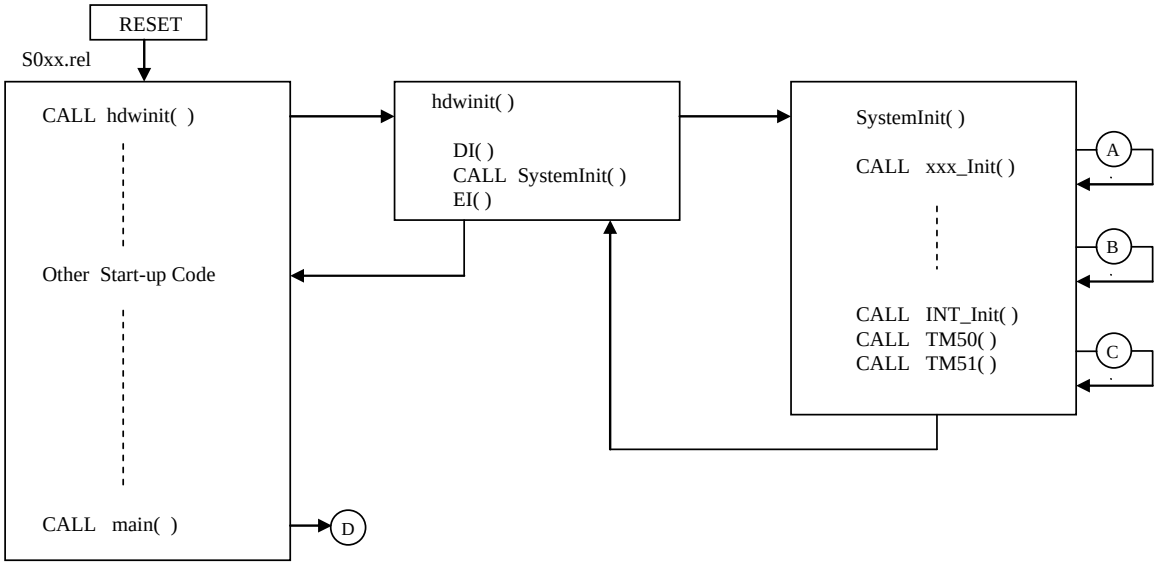
The flowcharts describe initialization, the main program, IIC0 and LCD initialization, IIC0 communication, LCD data transfer, and Timer 00-related and interrupt-service routines.

2.3.1 Program Startup and Initialization

For 78K0 programs written in the C language, an object code file such as s01.rel, linked to the user program, supplies the startup code. This startup code calls a function named hdwinit( ); you can place any hardware-initialization code here.

When generating a C program for the 78K0, the Applilet automatically adds the hdwinit( ) function to the user program, and calls the function SystemInit( ). The SystemInit( ) function, in turn, calls initialization routines for each peripheral.

Figure 6. Flowchart for System Initialization



When the hdwinit( ) function finishes, the startup code calls the main( ) function of the user program. Thus, when the main( ) function starts, peripheral initialization is already complete for key return interrupt and timers, and the main( ) function does not need to call these individual peripheral-initialization routines. The program calls initialization routines for the LCD controllers and the IIC communication channel later.

2.3.2 INT_Init( ) - Key Return Interrupt Initialization

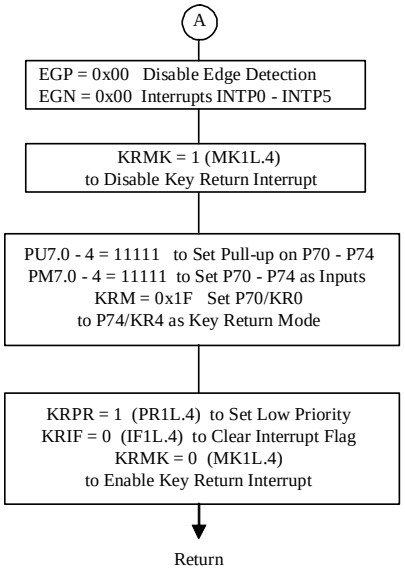


Figure 7. Initializing Key-Return Interrupt

SystemInit() calls the INT_Init() routine to set up the key-return function. Pins P70/KR0 through P74/KR4 serve as key-return inputs from the navigation switch. The initialization sets the EGP and EGN registers to disable other external interrupts (specifically, P75/KR5, P76/KR6 and P77/KR7).

The subroutine sets the key-return interrupt mask bit KRMK to 1, which masks the interrupt while the routine modifies other registers.

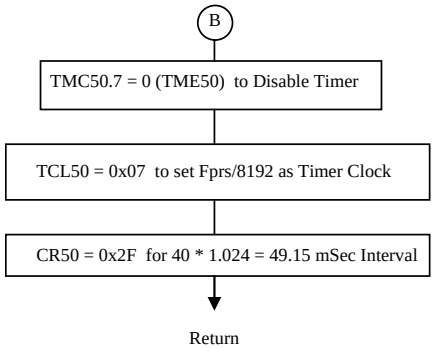
INT_Init() writes 0x1F to the PU7 and PM7 registers, configuring the P70-P74 port pins as inputs and providing pull-up resistors. The routine also writes 0x1F to the key-return mode register KRM to specify that the key-return interrupt (sources KR0 through KR4) trigger a key-return interrupt. With this setting, a negative edge on any of pins P70 through P74 causes the interrupt.

INT_Init() sets the interrupt priority to low, clears the interrupt flag (KRIF), and sets the mask bit back to zero, enabling the key-return interrupt INTKR.

The section of this application note on the MD_INTKR( ) key-return interrupt-service routine explains what the program does when the key-return interrupt occurs.

2.3.3 TM50_Init( ) - Timer-50 Initialization

Figure 8. Flowchart for Initializing Timer 50



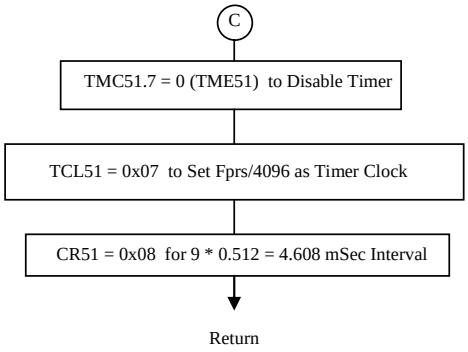
SystemInit() calls the TM50_Init() routine to initialize Timer50 as an interval timer of about 50 milliseconds duration. First the initialization routine sets the Timer50 enable bit TME50 to zero to disable the timer while the program changes the values of the control registers.

TM50_Init() sets the TCL50 register to 0x07, which establishes the Timer50 clock as $f_{PRS}/8192$. The demonstration program sets the 78K0/LG2 to run with the peripheral clock (f_{PRS}) coming from the 8-MHz internal high-speed oscillator. Thus, the timer clock is 8,000,000/8192, or 976.5 Hz, providing a clock count every 1.024 milliseconds.

TM50_Init() writes 0x2F (47) to the CR50 compare register. This value causes TM50 to compare with CR50 every 48 counts, for an interval duration of 48 * 1.024 = 49.15 milliseconds, or about every 50 milliseconds. The program uses this interval in the Wait(fiftymsec) routine, causing the system to wait in intervals of 50 milliseconds.

2.3.4 TM51_Init( ) - Timer-51 Initialization

Figure 9. Flowchart for Initializing Timer 51



SystemInit() calls the TM51_Init() routine to initialize Timer51 as an interval timer with a 5-millisecond duration. TM51_Init() sets the Timer51 enable bit TME51 to zero to disable the timer while changing control registers.

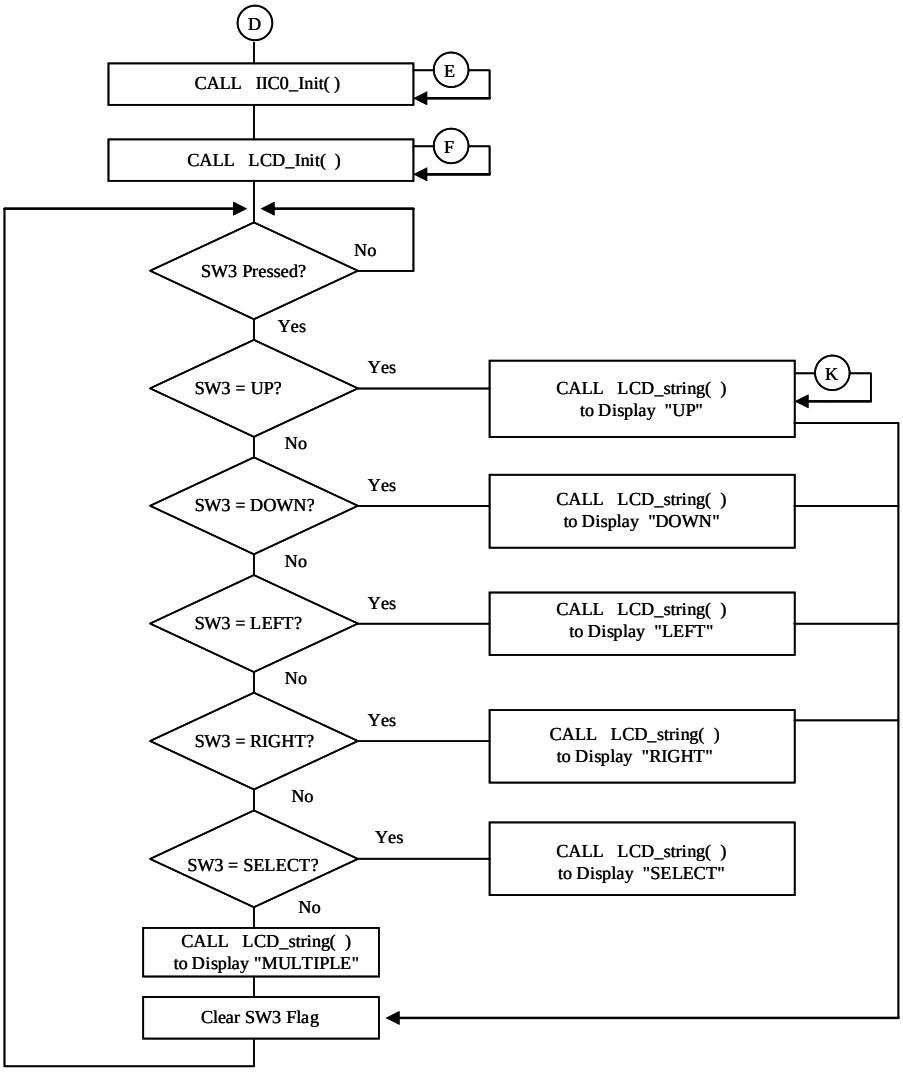
TM51_Init() sets the TCL51 register to 0x07, which establishes the clock to Timer51 as $f_{PRS}/4096$. Note that the same value in the TCL5x registers for Timer50 and Timer51 produces different clocks. For Timer 51, with the 8-MHz peripheral clock, the timer clock is 8,000,000/4096 or 1953.125 Hz; each clock count is 512 microseconds, or 0.512 milliseconds.

TM51_Init() sets the CR51 compare register to 8, which causes TM51 to compare with CR51 every 9 counts, for an interval duration of 9 * 0.512 = 4.608 milliseconds, or approximately every 5 milliseconds.

The demonstration program uses TM51 in the key-return interrupt-service routine to check the switch data 5 milliseconds after the initial interrupt.

2.3.5 Main( ) – Main Program – LCD Display

Figure 10. Flowchart for Main Program



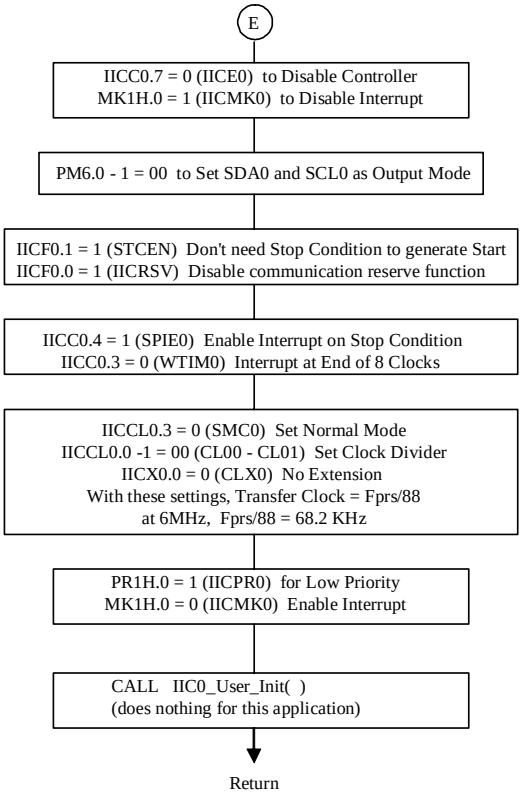
By executing startup code, the main() function starts the demonstration program. Main() calls IIC0_Init() to initialize the IIC controller, which communicates with the LCD controller. Then Main() calls LCD_Init() to initialize the LCD controller.

After this initialization, the program checks to see if you have pressed SW3 by checking the value in the sw3_in variable. The key-return interrupt-service routine sets this value, which will be zero if no switch is pressed. If you have pressed SW3, the key-return interrupt-service routine ensures the switch is debounced, then sets sw3_in to the value of the switch or switches that are closed. Main() then can determine which switch is closed and call the appropriate LCD_string() function to display on the LCD.

After acting on the switch closure, the program sets the sw3_in variable to zero again to respond to the next SW3 closure.

2.3.6 IIC0_Init( ) - IIC Controller Initialization

Figure 11. Flowchart for Initializing Bus Controller



The IIC0_Init() function initializes the IIC0 peripheral to prepare for IIC communication. The initialization routine first disables the controller and masks the IIC0 interrupt INTIIC0.

The IIC0 controller communicates with the LCD controller and external IIC devices via two pins, SCL0 (IIC serial clock) and SDA0 (IIC serial data). IIC0_Init() sets the port-mode register PM6 bits 0 and 1 to 0, which configures the pins as outputs.

IIC0_Init() sets the IICF0 (IIC0 Flag register) bits STCEN and IICRSV to 1. When STCEN=1 the controller generates a start condition without requiring a stop condition to be present. IICRSV=1 disables the communication-reserve function.

IIC0_Init() sets the IICC0 (IIC0 control register) bit SPIE0 to 1, to enable generation of an INTIIC0 interrupt on a stop condition. The initialization sets bit WTIM0 to 0, providing for an IIC0 interrupt on the eighth bit of data. Normally this setting supports operation in slave mode. After initialization, the IIC0

controller operates in slave mode until the demonstration program starts a master mode function by setting the WTIM0 bit to 1, to enable interrupts on the ninth bit—standard for master mode operation.

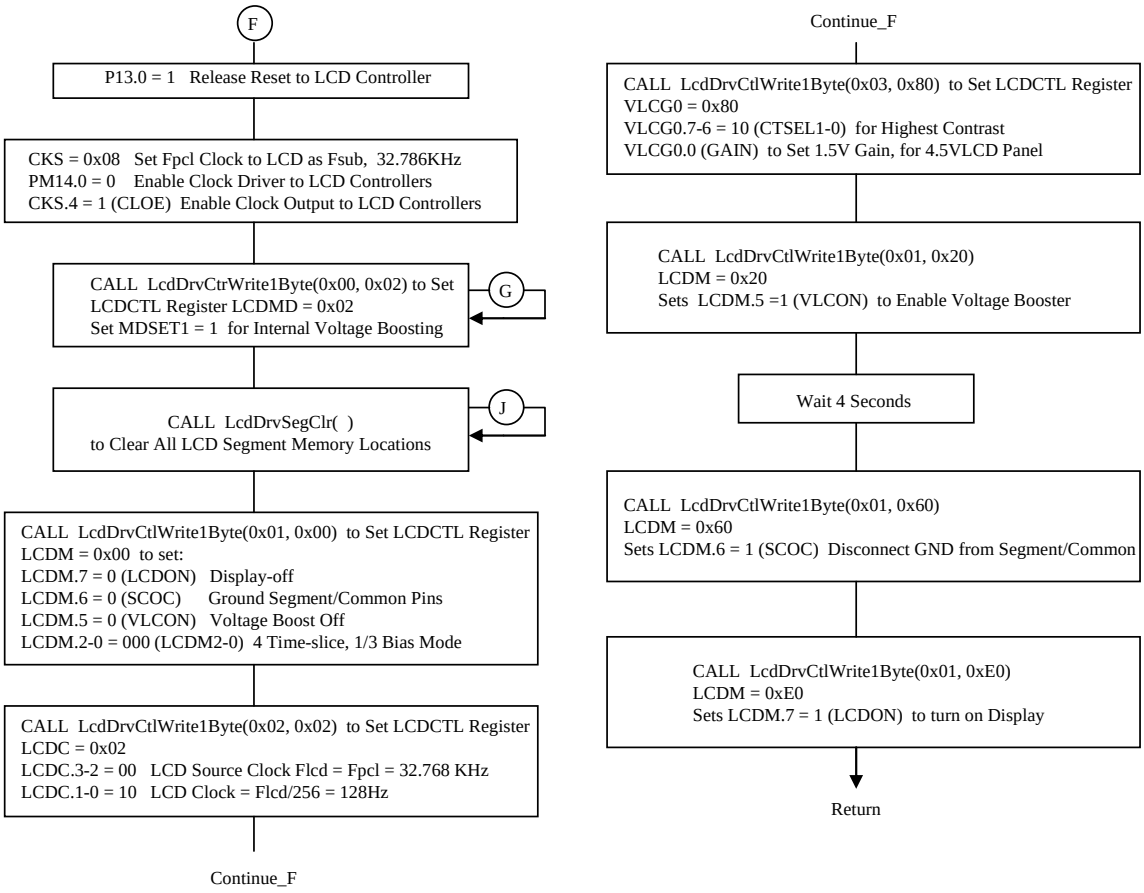
IIC0_Init() sets the IICCL0 (IIC0 clock register) and ICCX0 (IIC0 extension register) to specify IIC communication speeds. In this case, the setting is for normal mode with the clock set at f_{PRS}/88. The 8-MHz peripheral clock thus generates an IIC clock of 8,000,000/88 = 90.9 kHz—within the standard IIC maximum of 100 kHz for normal-speed devices.

IIC0_Init() sets the IIC0 interrupt INTIIC0 to low priority and sets the interrupt mask. Thus, the IIC0 peripheral is ready but not enabled. Enabling the peripheral requires setting IICC0 bit 7 (IICE0) to 1, which is handled by IIC0_MasterStart().

Finally, IIC0_Init() calls the IIC0_User_Init( ) function. This function accommodates hardware-specific initialization. For this demonstration the routine is empty and just returns.

2.3.7 LCD_Init( ) - LCD Controller Initialization

Figure 12. Flowchart for Controller Initialization



The LCD_Init() routine initializes the LCD controller, first by setting the µPD78F0397 registers to control the enable and clock to the LCD controller, and then by using the IIC0 peripheral to write to the LCD controller's control and segment registers.

P13.0 controls the reset signal to the LCD controller; LCD_Init() sets this signal to 1 to release reset. The initialization routine then sets the CKS register to specify the LCD clock as the 32.768-kHz subclock and sets PM14.0 to allow the LCD clock to be output to the LCD controller. LCD_Init() sets CKS register bit CLOE to enable clock output to the LCD controller.

At this point, the controller is out of reset and receiving a clock, but the display remains disabled because the LCD controller registers contain their default reset values.

LCD_Init() calls LcdDrvCtlWrite1Byte() to write values to the LCD controller LCDCTL registers. This routine sets the LCD controller display-mode register LCDMD to 0x02, which configures the controller for internal voltage boost. The LCD panel operates at 4.5V and the internal voltage boosting raises the LCD bias voltage to this level. This same value in LCDMD also sets LCDON to 0, turning the display off; sets SCOC to 0, connecting common and segment lines to GND (to discharge the display); and sets VLCON to 0, turning off the voltage booster.

Next, LCD_Init() calls LcdDrvSegClr() to write zeros to all segment bits in the LCD controller segment memory. These zeros leave the display blank after initialization.

LCD_Init() calls LcdDrvCtlWrite1Byte() to set the LCD controller display-mode register LCDM to 0x00, which disables the display and sets the mode as 4 time-slice, 1/3 bias.

The initialization routine sets the LCD controller clock register LCDC to 0x02, which establishes the LCD source clock (used for voltage boosting) at the same frequency as the 32.768 kHz clock to the LCD controller. (Note that this clock should be at least 32 kHz.) The lower two bits of LCDC set the LCD frame frequency as fLCD/256, or 128 Hz. (The frame frequency should be 128 Hz or lower.) With the 4 time-slice division, this frame frequency means the digit frequency is 128/4 = 32 Hz.

The routines sets the LCD voltage-boost control register VLCG0 to 0x80 to provide the highest contrast and set the voltage-boost gain to 1.5 (resulting in 4.5V output). The routine then sets LCDM to 0x20, turning the VLCON bit on and enabling the voltage booster.

The program then waits while the voltage booster generates the necessary LCD voltages. This waiting time should be 0.5 seconds if the LVDD pin to the 78F0397 is between 1.8 and 4.5V; if LVDD is between 4.5 and 5.5V, the wait time should be four seconds, as in this demonstration (with an LVDD of 5.0V).

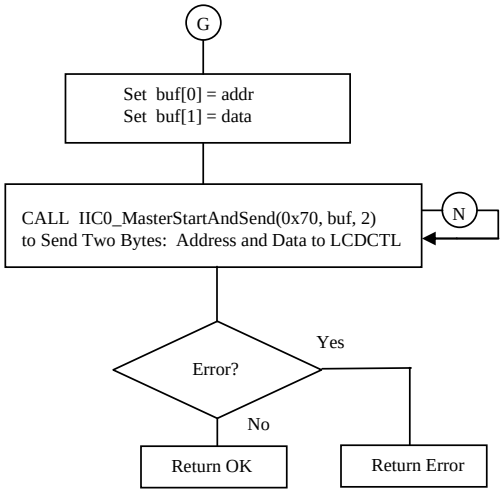
LCD_Init() waits by calling the Wait(number) subroutine, which waits for a specified number of 50-millisecond intervals. In this case, Wait(80) provides a wait of 80 * 50 = 4000 milliseconds, or 4 seconds.

LCD_Init() then sets LCDM to 0x60, which sets the SCOC bit to 1 to disconnect the LCD common and segment lines from GND. Finally, the initialization routine sets LCDM to 0xE0, which sets the LCDON bit to 1 and turns the display on.

At this point, the LCD controller is initialized and displaying all blank data. Writing to the LCDSEG registers changes the LCD segment display memory, and the controller displays those segments. There is no further need to write to the LCDCTL control registers.

2.3.8 LcdDrvCtlWrite1Byte(addr, data) – Write One Byte To LCDCTL Register

Figure 13. Flowchart for Writing to LCDCTL Register

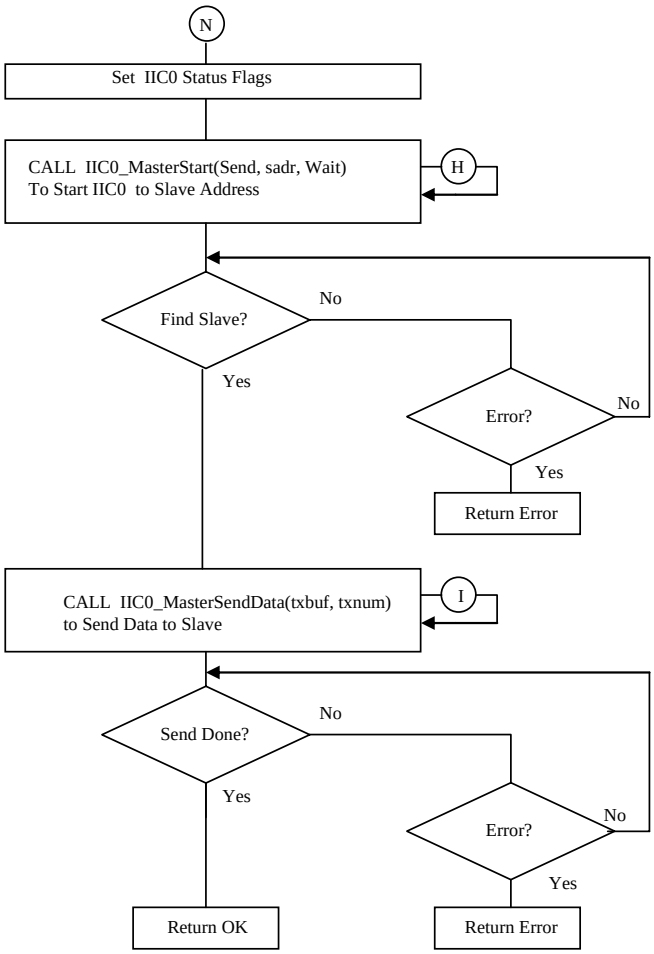


The LcdDrvCtrWrite1Byte(addr, data) function writes one byte of data to the LCDCTL register, whose address is specified by the addr parameter. Specify LCDMD with addr = 0, LCDM by addr = 1, LCDC by addr = 2, and VLCG0 by addr = 3.

The routine sets up a two-byte array buf[], storing the target-register address in buf[0] and the data in buf[1], and then calling the IIC0_MasterStartAndSend(CSLV_ID_LCDCTL, &buf[0], 2) function. This routine sends the two data bytes in the buffer to slave address 0x70—the address of the LCDCTL register. The first byte addresses the selected control register, and the second byte contains the data for that register.

2.3.9 IIC0_MasterStartAndSend(sadr, *txbuf, txnum) - Start IIC0, Send Data From Buffer

Figure 14. Flowchart for Sending Data From Buffer



The IIC0_MasterStartAndSend(sadr, *txbuf, txnum) routine is a combination of IIC0_MasterStart() and IIC0_MasterSendData() functions. The Applilet does not create this function, but the function is a simple combination of the two routines necessary for sending IIC data as a master.

The routine first sets IIC communication flags to their default settings; IIC0 callback functions during the IIC communication process use these flags. The routine sets UI_MasterError to MD_OK (to indicate no error), UI_MasterSendEnd to MD_ERROR to indicate that sending is not done, and UI_MasterFindSlave to MD_ERROR to indicate that a slave is not found at this point.

The routine then calls the IIC0_MasterStart(Send, sadr, 10) function, to start IIC as a master for sending data and specifying the slave address of “sadr”. The IIC0_MasterStart(Send, sadr, 10) returns, having either enabled the IIC0 peripheral and written the slave address to the IIC0 register for transmission, or reporting that the IIC communication channel is busy, which should not happen in this demonstration system.

The routine waits for either UI_MasterFindSlave to be OK or for an error condition in UI_MasterError.

On the ninth clock bit, the IIC0 peripheral gets an interrupt that invokes the MD_INTIIC0 interrupt-service routine. If the LCD controller has been properly addressed with one of the two slave addresses (0x70 for LCDCTL and 0x72 for LCDSEG), the controller responds with an ACK. The interrupt service routine sets the UI_MasterFindSlave flag. If there is no ACK, the interrupt service routine sets the UI_MasterError flag to MD_NACK.

If IIC0_MasterStartAndSend() does not find a slave, the routine simply returns an error code. Otherwise the routine calls IIC0_MasterSendData(txbuf, txnum) to queue up the data for sending.

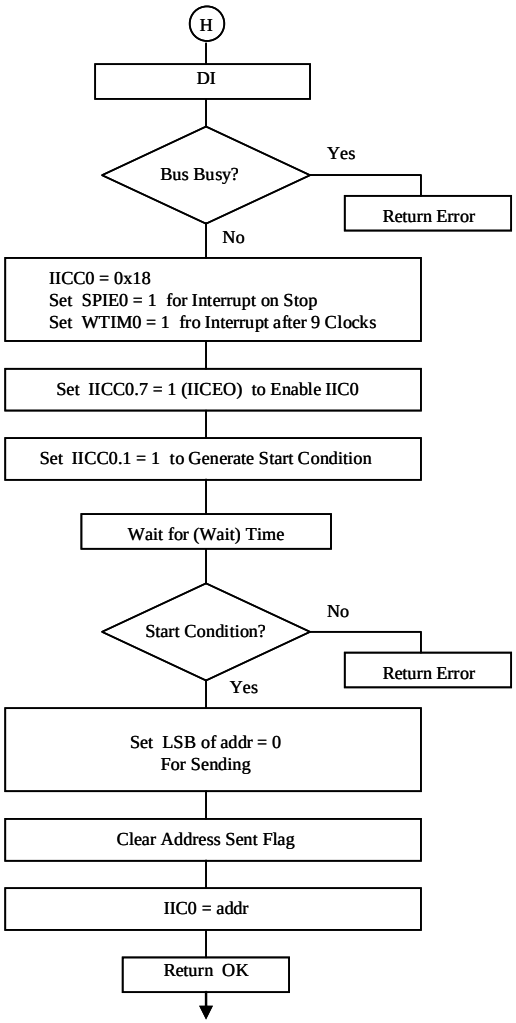
IIC0_MasterSendData() stores the buffer address and count, writes the first byte of data to the IIC0 register, and returns.

At this point, an INTIIC0 interrupt occurs after transmission of every data byte. The MD_INTIIC0 interrupt-service routine handles the interrupts in one of three ways: by sending the next data byte, by setting UI_MasterSendDone if all bytes have been sent and received, or by setting an error if there is no ACK from the slave.

The IIC0_MasterStartAndSend() routine waits for the UI_MasterSendDone condition to be true or UI_MasterError to be set to an error condition. The routine returns the operation status to the calling routine.

2.3.10 IIC0_MasterStart(Send, addr, wait) - Start IIC0 Sending To Slave Address

Figure 15. Flowchart to Start Sending to Slave Address



The IIC0_MasterStart(Send, addr, wait) function handles the starting of master-mode IIC communication.

The routine checks the status of the IIC bus. If the bus is busy, the routine returns an error. If the bus is available, the program sets the IICC0 register bits SPIE0 and WTIM0. These bits enable interrupts-on-stop conditions, and set the interrupt to occur after nine clocks (master mode). Then the routine sets IICC0 bit 7 (IICE0) to 1 to enable the IIC0 peripheral.

IIC0_MasterStart(Send, addr, wait) sets IICC0.1 (STT0) to 1 to generate a start condition, then the program waits to allow the peripheral to register the start and set IICS0.1 (STD0) to indicate that a start condition exists. IF IICS0.1 does not indicate a start condition, the routine returns an error.

The start routine sets the LSB of the slave address to 0 (indicating a master-to-slave data transmission) and writes the slave address to the IIC0 register to start the IIC transfer. The program then clears the address-

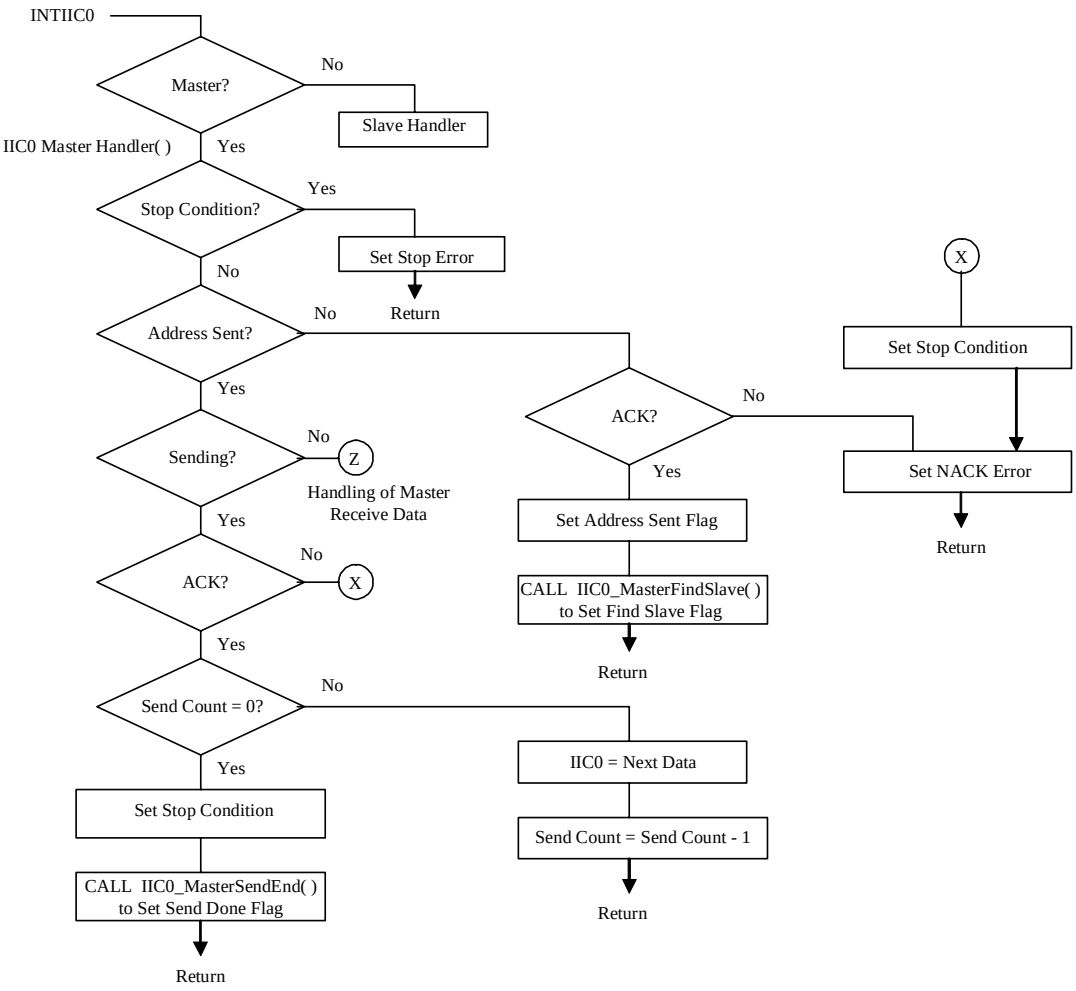
sent flag, which the MD_INTIIC0 interrupt-service routine uses to determine what action to take on interrupt.

The routine then returns, having started the communication to the slave.

Note that the IIC0_MasterStart() function can also start a receive operation, transferring data slave-to-master, if the first parameter is “Receive” instead of “Send”.

2.3.11 MD_INTIIC0( ) and IIC0_MasterHandler( ) - IIC0 Interrupt-Service Routines

Figure 16. Flowchart for IIC0 Interrupt-Service Routines



The MD_INTIIC0() interrupt-service routine, and the functions it calls, do most of the work involved in IIC transfers. The interrupt-service routine determines whether a master or slave operation is in progress and calls the appropriate handler routine, either IIC0_MasterHandler() or IIC0_SlaveHandler().

Communication with the LCD controller requires master communication, so the flowchart in Figure 16 shows the IIC0_MasterHandler() flow.

The first interrupt of a master transmission occurs after the master has sent the slave address and the ninth clock has been set.

The handler first checks for a stop condition, which should not occur in normal master-mode communication. If a stop condition does occur, the handler calls UI_MasterError() to indicate a stop error.

The handler then checks the slave address-sent flag. If the flag has not been set, the routine checks for an ACK from the slave. If this signal has not been received, the routine sets the NACK error condition. If the ACK has been received, then the slave responded to the address properly. The interrupt-service routine sets the UI_MasterFindSlave flag, sets the “address sent” flag true, and then returns.

After the first interrupt, further interrupts occur after additional data transmissions or when a receive operation starts. The handler checks to see if the transfer is sending (master-to-slave) or receiving (slave-to-master). This application note deals with sending data from the master to the LCD controller slave, so the receive data flow is not shown.

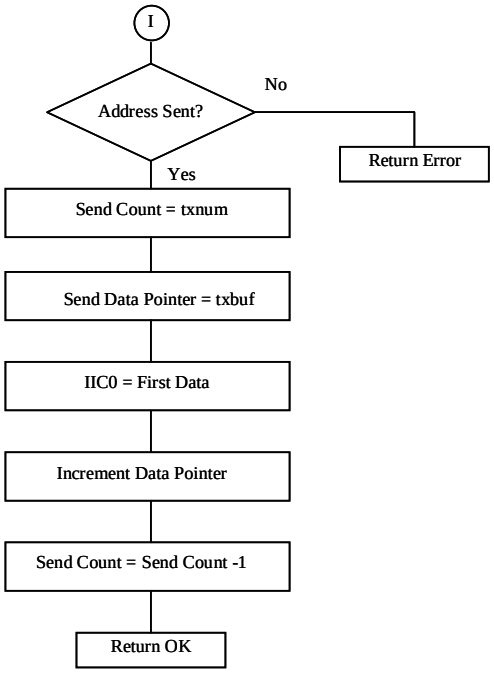
If sending, the handler expects to receive an ACK from the slave after transmission of each data byte. If the handler does not detect the ACK , the handler sets the NACK error condition and returns.

After receiving the ACK, the handler checks to see if there are any bytes left to send. If so, the handler writes the next byte to IIC0, updates the count and buffer pointers, and returns.

When there are no more bytes to send, the handler sets a stop condition to inform the slave that the transfer is done, sets the UI_MasterSendDone flag, and returns.

2.3.12 IIC0_MasterSendData(*txbuf, txnum) – Queue Data For IIC0 Transmission

Figure 17. Flowchart for Queuing Data



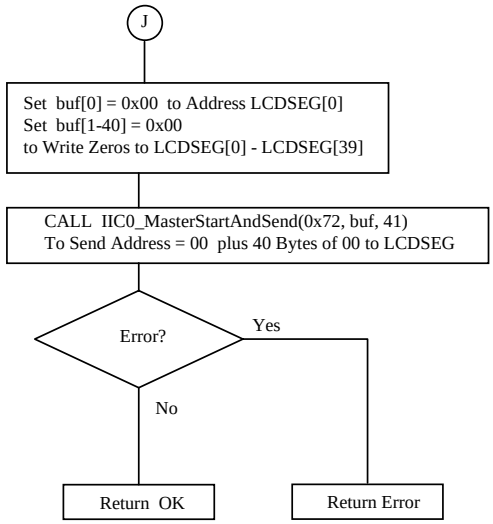
Once a program calls IIC0_MasterStart(), and after the UI_MasterFindSlave flag has been set to indicate that the slave is there and ready for data, the main program calls the IIC0_MasterSendData(*txbuf, txnum).

The routine checks the status of the address-sent flag. If the flag is not set, the routine returns an error. Calling this routine before the INTIIC0 interrupt occurs at the end of slave-address transmission results in this error.

The routine then sets the buffer address for data to send to the location pointed to by txbuf, sets the count of bytes to send to txnum, and writes the first byte to the IIC0 register. The routine also increments the pointer and decrements the count, so the second and following bytes are sent by the MD_INTIIC0 interrupt service routine.

2.3.13 LcdDrvSegClr( ) - Clear All LCD Segment Data

Figure 18. Flowchart to Clear LCD Segment Data



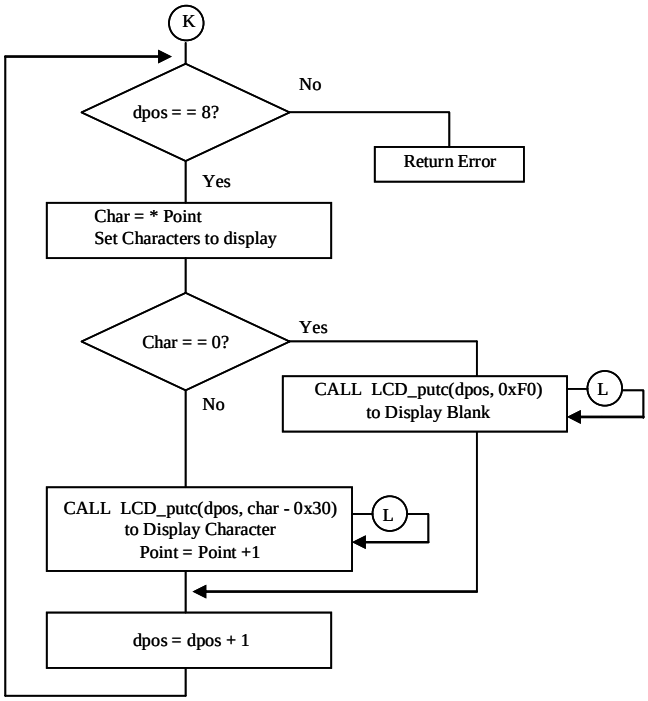
The LcdDrvSegClr() routine clears all LCDSEG memory locations in the LCD controller to zero, blanking the LCD.

The function sets up a 41-byte buffer. Buf[0] contains 0, as the address of the first LCDSEG location. The function also sets data buf[1] through buf[40] to zero, to specify the data (all zeros) to be written to LCDSEG locations 0 through 39.

The routine then calls IIC0_MasterStartAndSend(CSLV_ID_LCDSEG, &buf[0], 41) to send the 41 bytes of data to the slave address CSLV_ID_LCDSEG, which is 0x72. The LCD controller takes the first byte sent (buf[0] == 0x00) as the LCDSEG address to begin writing data to. The following 40 bytes of 0x00 write zeros to the LCDSEG addresses 0 through 39, turning off the segment bits at these addresses.

2.3.14 LCD_string(*point, dpos) - Write String To LCD At Specified Digit Position

Figure 19. Flowchart for Writing String to LCD



The LCD_string() function writes a string (an array of character bytes, terminated by zero) to the LCD display at a specified display position. Dpos = 0 means that the string should be written starting in the left-most display character. Dpos = 1 would leave the left-most character as is and start writing the string in the next character location.

The routine loops, incrementing the display position, until dpos = 8, thus supporting an LCD panel of eight characters, with the leftmost position defined as dpos=0, and the rightmost position as dpos=7.

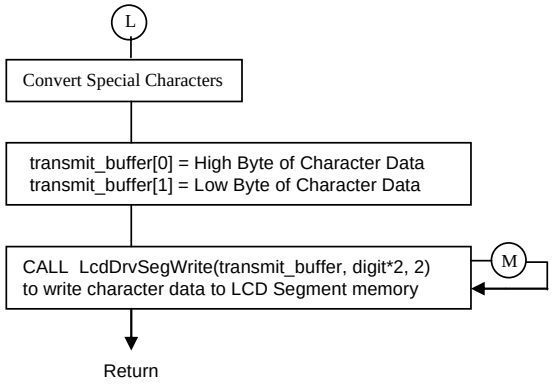
For each dpos position, the routine reads the character to display from the *point array. If the character is zero, indicating the end of the string, the routine sets a blank for that position and all further rightward positions.

If the character is not zero, the routine calls function LCD_putc(dpos, char – 0x30) to write the character's segment data to the display. This function subtracts 0x30 from the ASCII character value to create an index into the character-segment data array. For example, the character values are changed from 0x30 ('0') to 0x00, or 0x41 ('A') to 0x11, so that the converted values point to the respective locations in the character-segment data array.

The routine displays the current character, then updates dpos and the character pointer (unless a zero was found). The routine continues until dpos is greater than or equal to eight.

2.3.15 LCD_putc(digit, data) - Write One Character to LCD At Specified Digit Position

Figure 20. Flowchart for Writing One Character to LCD



LCD_putc() displays one data character (with a bias of – 0x30 applied) at a selected display position.

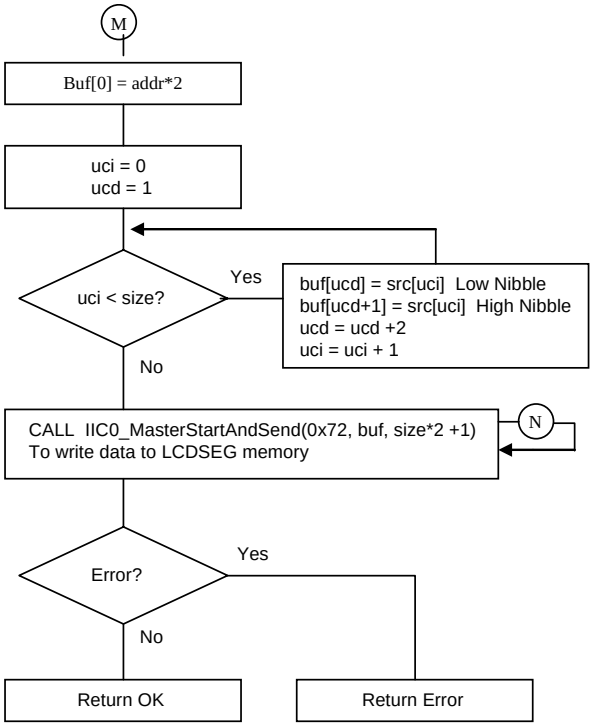
The routine checks the character data value and corrects the index into the character-segment data array. Each character requires 16 bits of data, written to four successive 4-bit LCDSEG locations. The character-segment data array consists of an array of 16-bit locations, with the data for each character contained in the 16 bits.

The data index addresses the character-segment data array, and the LCD_putc() writes the 16-bit value into the 2-byte array transmit_buffer[]. The routine then calls LcdDrvSegWrite(&transmit_buffer[0], digit * 2, 2), to write the two bytes into four successive locations addressed by digit * 2.

The input to the LCD_putc(digit, data) routine ranges from 0 to 7, specifying the character positions. LcdDrvSegWrite() specifies a byte address of 0 to 14 as the starting location in segment memory. LcdDrvSegWrite() multiplies this value by two, to convert to LCDSEG locations.

2.3.16 LcdDrvSegWrite(*src, addr, size) - Write Bytes of Character Data to LCD Segment Memory

Figure 21. Flowchart for Writing Character Data to LCD



The LcdDrvSegWrite() routine writes data bytes to the LCDSEG memory. The *src parameter specifies a pointer to an array of bytes. The addr parameter specifies the byte address in LCDSEG memory for the data, and this parameter's size indicates the number of bytes to write.

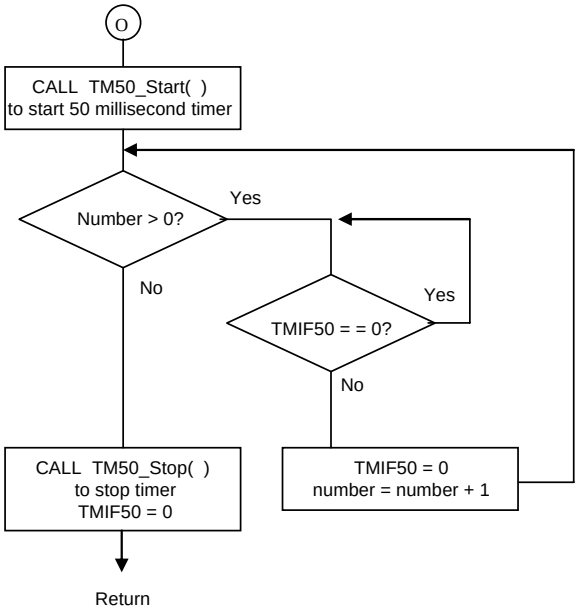
The routine sets up a data buffer to write using the IIC0_MasterStartAndSend() routine. The first byte in the array, buf[0], contains the address in LCDSEG memory for the addressed bytes. Since each byte address corresponds to two LCDSEG four-bit locations, the byte address is multiplied by two to get the LCDSEG location.

The routine then converts the byte data in the *src array into four-bit values in the buf[] array for writing to each LCDSEG location. The low four bits of each byte go into the lower buf[] location, and the high four bits of each byte go into the next location.

After this translation, the buf[] array contains the address and data for LCDSEG memory. IIC0_MasterStartAndSend(CSLV_ID_LCDSEG, &buf[0], size * 2 + 1) is called to send this data to the IIC slave at address CLSV_ID_LCDSEG, which is 0x72. The first byte (addr * 2) provides the LCDSEG location. The routine writes the rest of the segment data in the array to the addressed location and those following.

2.3.17 Wait(number) – Wait For Specified Number Of 50-Millisecond Intervals

Figure 22. Flowchart for 50-msec Wait



LCD_Init(), calls the Wait(number) function to pause for the specified number of 50-millisecond intervals.

Wait( ) calls TM50_Start( ), to start TM50 counting. The initialization routines have already programmed TM50_Init() so that the timer has CR50 match TM50 after approximately 50 milliseconds. When TM50 matches CR50, the TM50 peripheral sets the TMIF50 interrupt flag. The interrupt-mask flag (TMMK50) for this interrupt remains set, however, so no interrupt occurs.

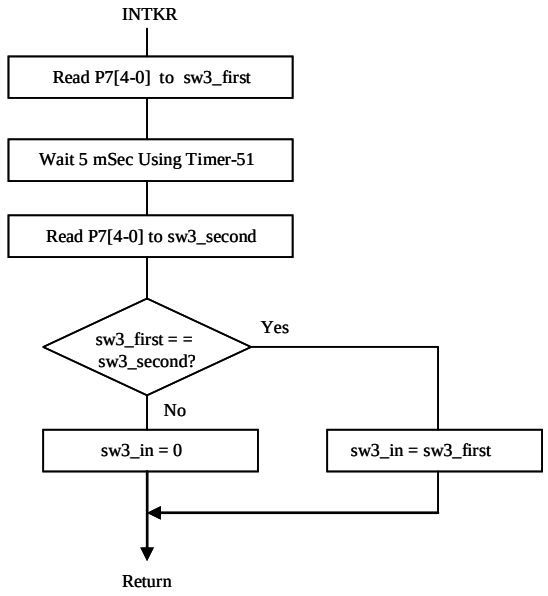
The Wait( ) function checks the state of the TMIF50 flag, looping until the flag is set. At this point, 50 milliseconds have elapsed. Wait( ) then resets the TMIF50 interrupt flag.

The Wait( ) function loops for the number of intervals specified in its input parameter, counting down the variable to zero. At the end of the looping, the specified number of 50-milliseconds intervals have elapsed.

Wait( ) stops the timer and ensures that the interrupt flag is reset.

2.3.18 MD_INTKR( ) – Key-Return Interrupt Service Routine

Figure 23. Flowchart for Key-Interrupt Service Routine



The MD_INTKR() routine services the INTKR interrupt. When any of the SW3 navigation switches closes, the key-return input pin is connected to GND. The resulting negative edge on the key-return pin causes the INTKR interrupt.

The MD_INTKR() routine reads the input port attached to the switches to get the current state of the pins. Timer TM51 measures 5-millisecond intervals. When the timer completes a 5-millisecond interval, the routine reads the input ports again.

If the first and second readings agree, then the switches have been seen in the same state over a 5 millisecond interval and are considered stable, or debounced. In this case, the routine sets the global variable sw3_in to the value read, and the routine returns.

If the first and second readings do not agree, the routines sets sw3_in to zero, to indicate that no stable switch setting has been found, and then returns.

If the switches are still bouncing, the next negative edge on the key return inputs cause another INTKR interrupt, and the routine checks the switch state again.

2.4 Applilet's Reference Driver

NEC Electronics' Applilet program generator automatically generates C or assembly language source code to manage peripherals for NEC Electronics microcontrollers. Please see the Appendix for the version of the Applilet used.

The Applilet produces the basic initialization code and main functions for the program, as well as driver code for the key-return interrupt, for the 8-bit timers, and for the IIC0 peripheral used to communicate with the LCD controller. After the Applilet produces the basic code, you can add additional code to customize the program.

NEC Electronics provides sample program code for LCD controller support, which is adapted for this demonstration. This code calls Applilet-generated IIC0 routines to communicate with the LCD controller.

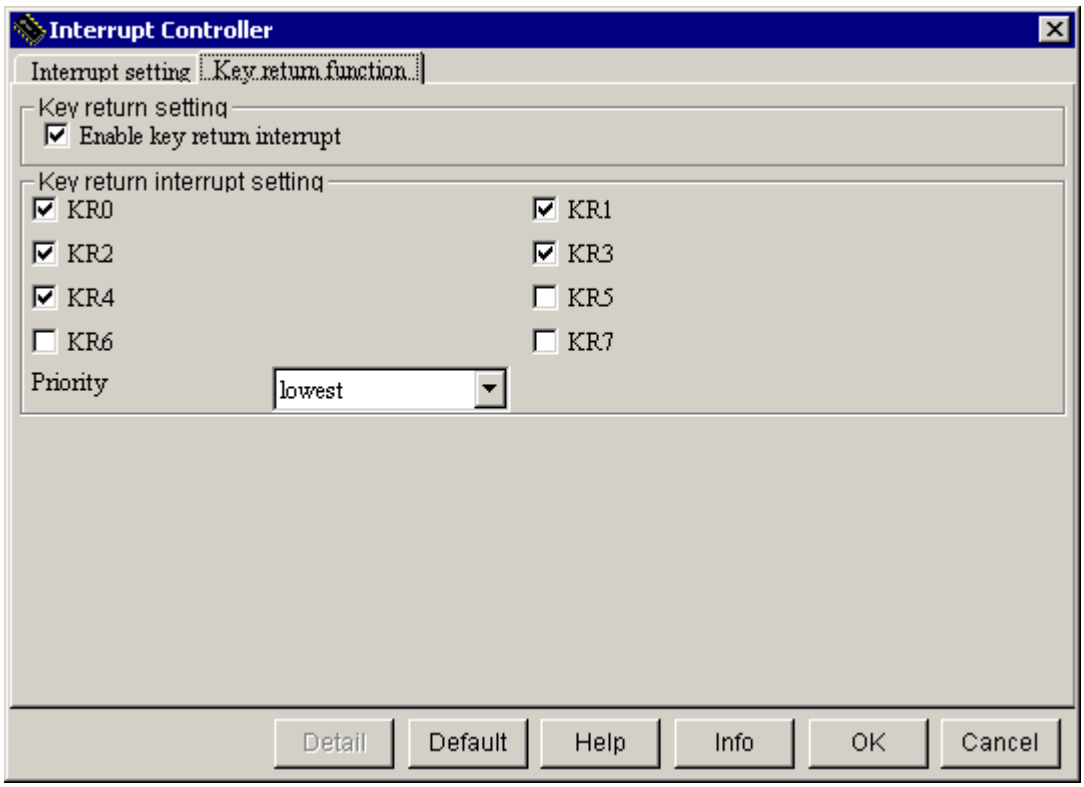
This section describes how to set up the Applilet to produce code for the key-return interrupt, timers TM50 and TM51, and the IIC0 peripheral. This section also lists the routines produced.

On startup the Applilet creates a new project (.prx) file. The Applilet then displays a screen that lets you select among different peripheral blocks for setup.

2.4.1 Configuring Applilet for Key-Return Interrupt

In the first Applilet dialog, select **Interrupt controller**. Then select the **Key return function** tab to set details for the key-return interrupt.

Figure 24. Applilet Dialog For Selecting Interrupt-Controller Operation



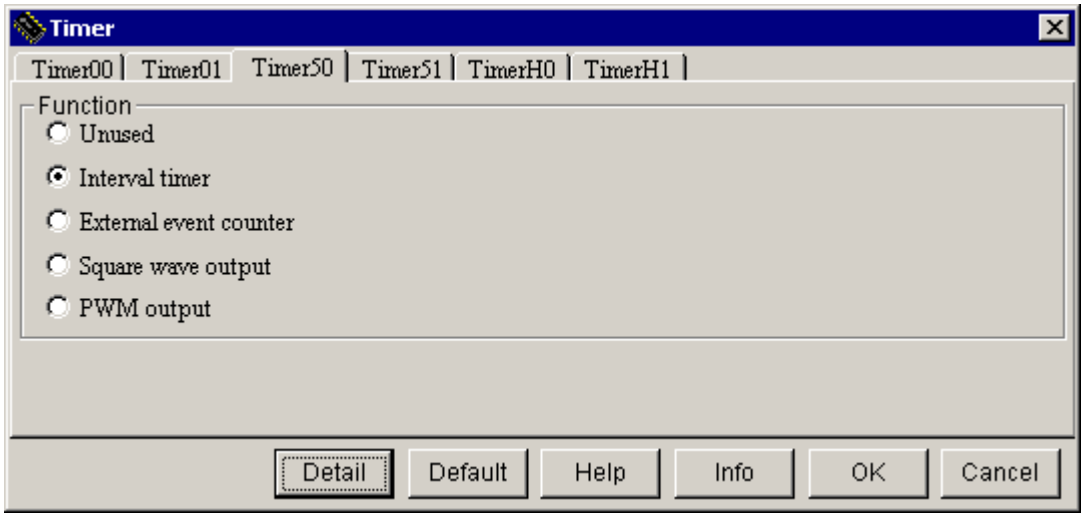
Check **Enable key return interrupt** to generate an interrupt when the key-return pins go low. Set the interrupt for **lowest** priority.

The navigation switch connects to the microcontroller via pins P70/KR0 through P74/KR4, so select the checkboxes for **KR0** through **KR4**. Because this demonstration does not use KR5, KR6 and KR7, the associated pins P75/KR5, P76/KR6 and P77/KR7 are available as general-purpose I/O pins without affecting the key-return interrupt function.

2.4.2 Configuring Applilet for Timer TM50

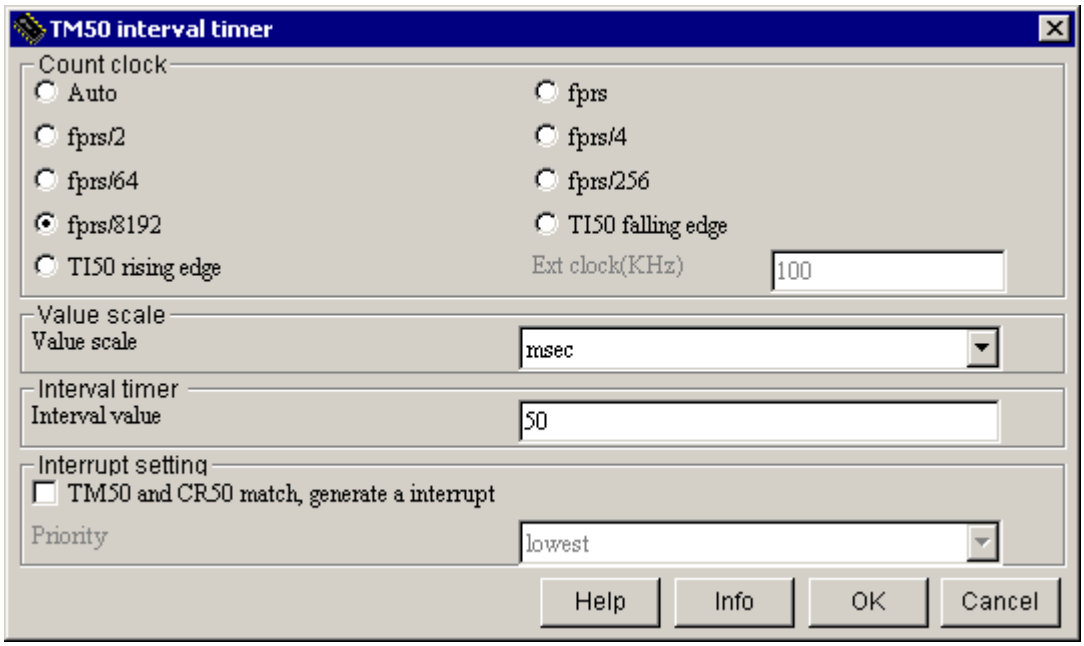
Selecting **Timer** brings up the selection screen for the various timer blocks. Select **Timer50** then click **Interval timer**.

Figure 25. Applilet Timer-Selection Dialog



Once you have selected TM50, click **Detail** to bring up the TM50 detail dialog box, where you can define interval timer operation.

Figure 26. Timer Detail Dialog



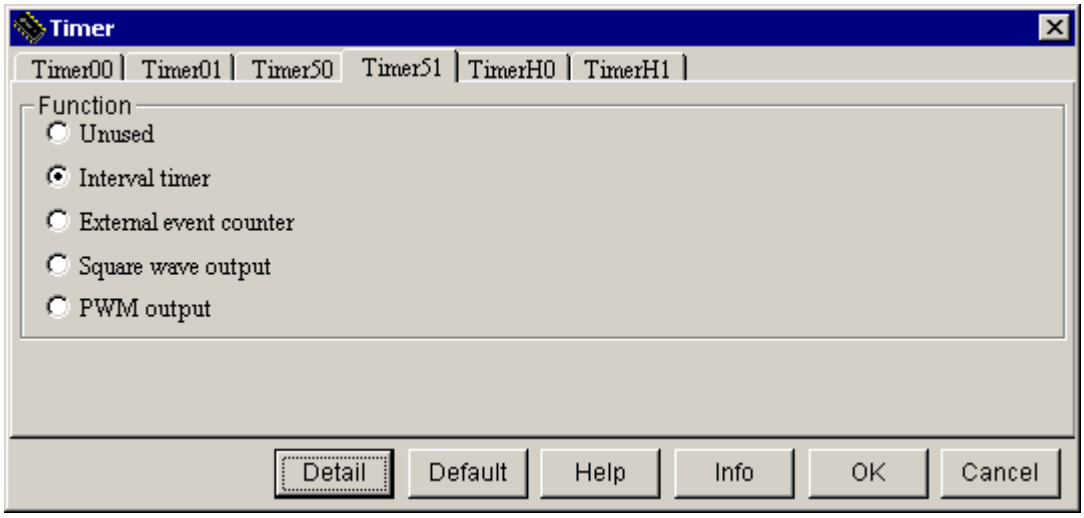
You want the timer to generate an interrupt every 50 milliseconds, so set the value scale to **msec** (milliseconds), and the interval value to **50**. Set **Count clock** to **fprs/8192**. The Applilet calculates an appropriate setting for the timer-comparison register to achieve a 50-millisecond interval.

Do not check **Interrupt setting** so that the timer will not generate an interrupt when the timer interval completes.

2.4.3 Configuring Applilet for Timer TM51

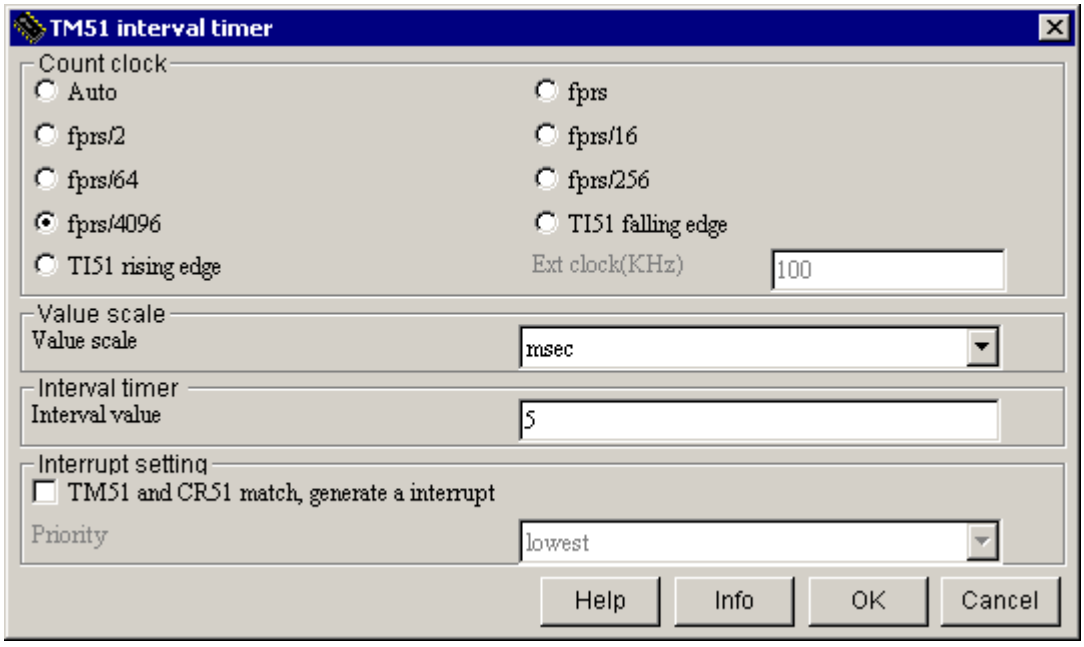
In the **Timer** block, select **Timer51** and click **Interval timer.**

Figure 27. Setting Up Timer51 as Interval Timer



Once you have selected TM51 as an interval timer, clicking **Detail** brings up the detail dialog box for interval-timer settings.

Figure 28. Timer51 Detail Dialog



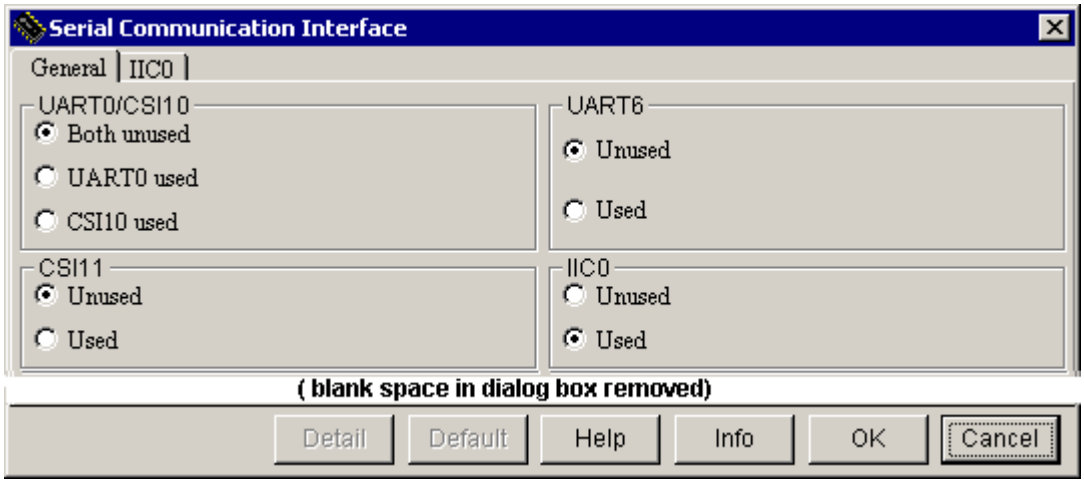
To cause the timer to generate an interrupt every 5 milliseconds, set **Value scale** to **msec** (milliseconds), and set **Interval value** to **5**. Set **Count clock** to **fprs/4096**, to select the basic clock to TM51. The Applilet then calculates an appropriate setting for the timer-comparison register to provide a 5-millisecond interval.

To prevent interrupt generation when the timer interval completes, leave **Interrupt setting** unchecked.

2.4.4 Configuring Applilet for IIC0 Communication

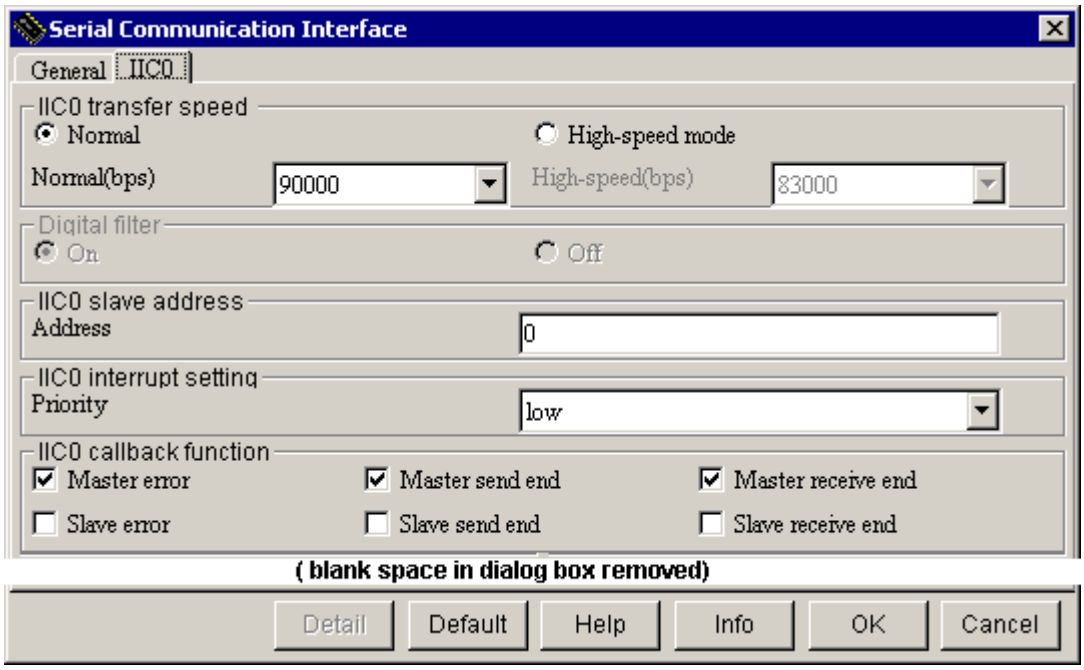
Click **Serial Communication Interface** and select **IIC0** as **Used**.

Figure 29. Configuring Serial Communications



Once you have selected the IIC0 as **Used,** select the tab labeled **IIC0** to see the detail dialog box.

Figure 30. Detail Dialog Box for IICO



Set the IIC0 peripheral interface to operate in **Normal** mode. With this mode selected, a drop-down list lets you choose among a range of communication speeds. These speeds are based on divisions of the peripheral clock. Select the highest available speed—in this case, 90000 bps.

The IIC0 peripheral operates in both slave and master modes, and can switch between the two modes. Unless in the middle of a transfer initiated as a master, the device is in slave mode and responds to IIC transmissions from an external master that matches the appropriate slave address. Because this demonstration does not use the device in slave mode, leave the slave address at the default of 0.

Set **IIC0 interrupt setting** priority for **low**.

The Applilet generates several standard routines for operation and has the option of generating blank callback routines, where you can insert code to deal with particular events. Some of these callback routines support master communication, so check **Master error**, **Master send end** and **Master receive end**. Leave the checkboxes for slave callbacks unchecked.

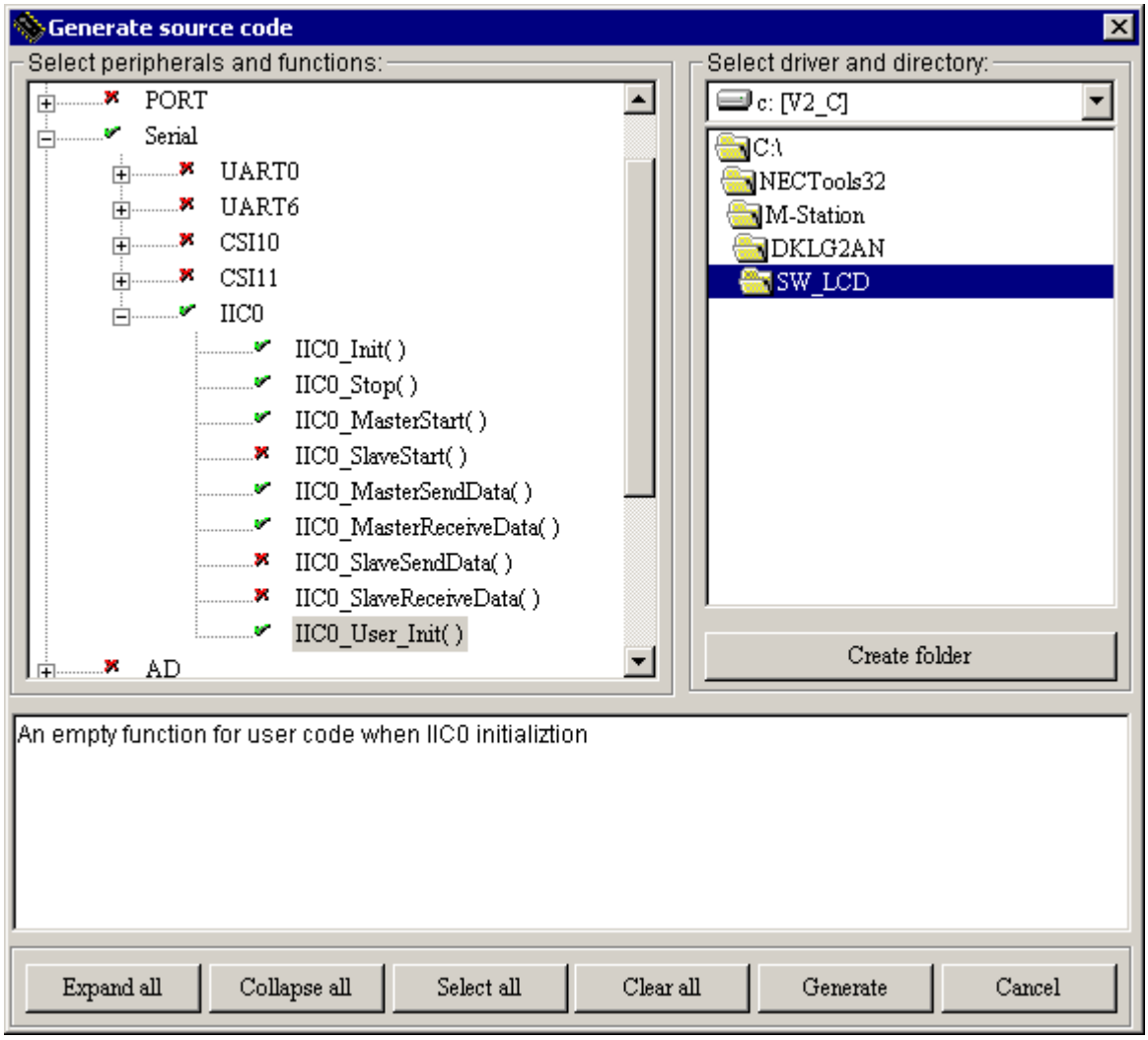
For the IIC0 peripheral, the Applilet can also generate optional IIC0 peripheral functions that are not selected in the IIC0 tab. You select these additional functions either in the Applilet function view (described in the next section) or during code generation.

2.4.5 Generating Code with Applilet, Selecting Optional IIC0 Routines

Once you have set up the IIC0 dialog box, select **Generate code**. The Applilet displays a list of peripherals and functions and allows you to select a directory for storing the source code.

To select additional optional IIC0 routines, the peripheral/function tree in the left pane expands to show the currently selected functions for IIC0 support.

Figure 31. Applilet's Generate-Code Dialog



For a new project, only IIC0_Init() may be shown. For this demonstration, select the additional functions IIC0_Stop(), IIC0_MasterStart(), IIC0_MasterSendData(), IIC0_MasterReceiveData() and IIC0_User_Init(). Click **Generate** to have the Applilet create the selected routines.

The Applilet creates the code in several C-language source files (extension .c) and header files (extension .h), and shows the list of files created in a dialog box.

To support the key-return interrupt, the Applilet generates int.h, int.c, and int_user.c.

To support TM50 and TM51, the Applilet generates timer.h, timer.c, and timer_user.c.

To support IIC0, the Applilet generates serial.h, serial.c, and serial_user.c.

The Applilet generates several other files, including a main.c file with a blank main function.

2.4.6 Applilet-Generated Files and Functions for Key-Return Interrupt

The code generated for key-return interrupt support is in the files int.h, int.c, and int_user.c.

2.4.6.1 Int.h

The header file int.h contains declarations for the functions controlling the key-return interrupt and definitions of values for key-return initialization. The header file macrodriver.h, used for all of the Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

You must add to the Applilet-generated code here to include the external declaration of the sw3_in variable. This variable reports the state of the SW3 navigation switch. The external declaration of sw3_in allows other portions of the program to examine this variable by including int.h.

2.4.6.2 Int.c

The source file Int.c contains the following functions generated by the Applilet for the key-return interrupt:

void INT_Init(void)
The INT_Init() routine initializes the key-return register and interrupt.

2.4.6.3 Int_user.c

The source file int_user.c contains stub functions for user code. These functions are empty on code generation. You can add application-specific code.

__interrupt void MD_INTKR(void)
This is the interrupt-service routine for the key-return interrupt INTKR, generated when a negative-going edge is seen on one of the key-return pins.

On generation by the Applilet, this interrupt service routine is blank. You need to add code to use the key-return interrupt to debounce and read the SW3 navigation switch.

The file also contains an addition of the declaration of the sw3_in variable.

2.4.7 Applilet-Generated Files and Functions for TM50 and TM51

The code generated for TM50 and TM51 support is in the files timer.h, timer.c, and timer_user.c.

2.4.7.1 Timer.h

The header file timer.h contains declarations for the functions controlling TM50 and TM51, and definitions of values for timer initialization. The header file macrodriver.h, used for all Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

2.4.7.2 Timer.c

The source file Timer.c contains the following functions generated by the Applilet for TM50 and TM51:

void TM50_Init(void)
The TM50_Init() routine initializes the TM50 peripheral.

void TM50_Start(void)
The TM50_Start() routine starts TM50 operation by enabling the timer..

void TM50_Stop(void)
The TM51_Stop() routine stops TM50 operation by disabling the timer.

MD_STATUS TM50_ChangeTimerCondition(USHORT value)
The TM50_ChangeTimerCondition() function changes the value in the CR50 compare registers and therefore changes the interval period of TM50.

void TM51_Init(void)
void TM51_Start(void)
void TM51_Stop(void)
MD_STATUS TM51_ChangeTimerCondition(USHORT value)
These routines perform the same functions for TM51 as the similar routines described above for TM50.

2.4.7.3 Timer_user.c

The source file timer_user.c contains stub functions for user code. These functions are empty on code generation. You can add application-specific code.

2.4.8 Applilet-Generated Files and Functions for IIC0 Communication

The code generated for IIC0 support is in the files serial.h, serial.c, and serial_user.c.

2.4.8.1 Serial.h

The header file serial.h contains declarations for the functions controlling IIC0 and definitions of values for IIC0 initialization. The header file macrodriver.h, used for all of the Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

The code requires the addition of external declarations for the variables used to signal IIC0 states:

- extern MD_STATUS UI_MasterError;
- extern MD_STATUS UI_MasterSendEnd;
- extern MD_STATUS UI_MasterReceiveEnd;
- extern MD_STATUS UI_MasterFindSlave;

The code also requires the addition of the functions IIC0_Init( ) and IIC0_MasterStartAndSend( ).

2.4.8.2 Serial.c

The source file Serial.c contains the following Applilet-generated functions:

void IIC0_Init(void)
This routine initializes the IIC0 peripheral.

MD_STATUS IIC0_MasterStart(TransferMode mode, UCHAR adr, UCHAR wait)
This routine starts a master-mode communication operation. The **mode** parameter is either “Send” or “Receive” to specify the direction; the **adr** parameter specifies the IIC slave address; **wait** specifies a number of loop times to wait for a start condition to occur. This routine creates a start condition and sends the slave address.

void IIC0_Stop(void)
This routine stops the IIC0 peripheral. The demonstration program does not use this routine.

MD_STATUS IIC0_MasterSendData(UCHAR* txbuf , UINT txnum)
The program calls this routine after IIC0_MasterStart( ) has started a “Send” transfer. The **txbuf** parameter points to an array of bytes to send, and the **txnum** parameter specifies the number of bytes to send. This routine sets the buffer pointer and count, and sends the first byte in the buffer. Further bytes are sent in the MD_INTIIC0( ) interrupt-service routine.

MD_STATUS IIC0_MasterReceiveData(UCHAR* rxbuf, UINT rxnum)
This routine is called after IIC0_MasterStart( ) has started a “Receive” transfer. The **rxbuf** parameter specifies the address of a buffer to store received characters, and **rxnum** specifies the number of characters to receive. The demonstration program does not use this routine.

__interrupt void MD_INTIIC0(void) ;

This is the IIC0 interrupt-service routine, invoked by the INTIIC0 interrupt. This routine calls either IIC0_MasterHandler( ) or IIC0_SlaveHandler( ), depending on the IIC0 master or slave state.

MD_STATUS IIC0_SlaveHandler(void)
This routine, called by MD_INTIIC0( ), handles IIC0 interrupts in master mode.

MD_STATUS IIC0_MasterHandler(void)
This routine, called by MD_INTIIC0( ), handles IIC0 interrupts in slave mode.

2.4.8.3 Serial_user.c

The source file serial_user.c contains stub functions for user code. These functions are empty on code generation. You can add application-specific code.

You must add the following variables to enable subroutines to check the state of IIC0 communication:

- MD_STATUS UI_MasterError; stores master errors
- MD_STATUS UI_MasterSendEnd; reports that sending is done
- MD_STATUS UI_MasterReceiveEnd; reports that receiving is done
- MD_STATUS UI_MasterFindSlave; reports that slave has responded, send/receive can start

void IIC0_User_Init(void)
This routine is called by IIC0_Init( ) and is unused.

void CALL_IIC0_MasterError(MD_STATUS flag)
This routine is called by the IIC0_MasterHandler( ) routine when an error is detected. You must add code to store the **flag** parameter in UI_MasterError.

void CALL_IIC0_MasterReceiveEnd(void)
This routine is called by IIC0_MasterHandler( ) when the receive count reaches zero. You must add code to set the UI_MasterReceiveEnd flag to MD_OK. The demonstration program does not use this routine.

void CALL_IIC0_MasterSendEnd(void)
This routine is called by IIC0_MasterHandler( ) when the last transmit byte has been sent. You must add code to set the UI_MasterSendEnd flag to MD_OK.

void CALL_IIC0_SlaveAddressMatch(void)
This routine is called by IIC0_SlaveHandler( ) when the slave address received in slave mode matches the set slave address for the IIC0 peripheral. The demonstration program does not use this routine.

void CALL_IIC0_MasterFindSlave(void)
This routine is called by IIC0_MasterHandler( ) when an ACK has been received after the slave address has been sent, indicating that a slave has responded to the slave address. This routine sets the UI_MasterFindSlave flag to MD_OK.

The following routine is not generated by the Applilet, but necessary for this demonstration program:

MD_STATUS IIC0_MasterStartAndSend(UCHAR sadr, UCHAR* txbuf, UINT txnum)
This routine combines the IIC0_MasterStart() and IIC0_MasterSendData() functions, since these are used together to send data to a particular slave. First IIC0_MasterStart(Send, sadr, 10) sets up a sending transfer to the specified slave address; then the routine waits for UI_MasterFindSlave to be set, indicating that the slave has responded.

Then IIC0_MasterSendData(txbuf, txnum) sends the specified data. This routine then waits for the UI_MasterSendDone flag to be set, indicating the transfer is complete.

The routine monitors the UI_MasterError flag while waiting, to check for error conditions in the sending of the slave address or transfer of data.

2.4.9 Other Applilet-Generated Files

For the demonstration program, the Applilet generates several other source files. The files and their functions are listed in Table 6.

Table 6. Additional Applilet-Generated Source Files

File	Function
Macrodriver.h	General header file for Applilet-generated programs
Systeminit.c	SystemInit() and hdwinit() functions for initialization
Main.c	The main program function
System.h	Clock-related definitions
System.c	Clock_Init() function
Option.asm	Defines the option byte and security bytes
Option.inc	Defines settings for the option byte and security settings

2.4.10 Demonstration Program Files Not Generated by Applilet

The demonstration program also includes the following files, not generated by the Applilet.

Table 7. Program Files Not Generated By Applilet

File	Function
define.h	Definitions of navigation switch values
lcd.h	Header file for high-level LCD functions
lcd.c	Code to write characters and strings to the LCD display
LcdDrvApp.h	Header file for LCD driver functions
LcdDrvApp.c	Code to initialize, write and read the LCD display controller; These functions call Applilet-generated IIC0 routines

2.5 Demonstration Platform

The demonstration platform for LCD display is a development board from NEC Electronics. You may be able to duplicate the same hardware using off-the-shelf components along with the NEC Electronics microcontroller of interest.

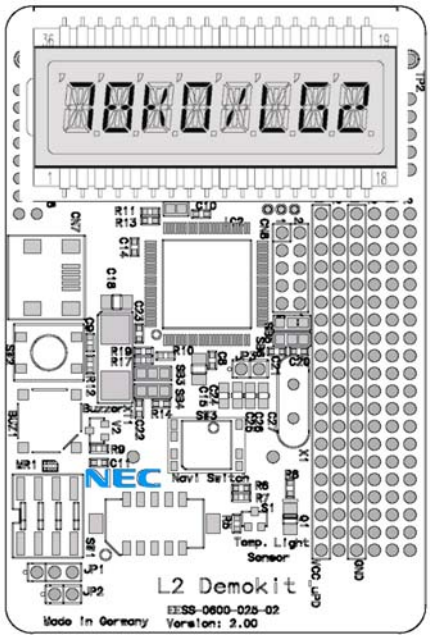
2.5.1 Resources

The program demonstration uses:

- DemoKit-LG2 demonstration board, with µPD78F0397 8-bit microcontroller mounted
- DemoKit-LG2 resources:
  - 5-position navigation switch SW3
  - 8-character LCD

For details on the hardware listed, please refer to the appropriate user manual, available from NEC Electronics upon request.

Figure 32. Demonstration Hardware



2.5.2 Demonstration of Program

With the hardware configured and the µPD78F0397 microcontroller programmed with the demonstration code, the demonstration includes the following steps:

- Press RIGHT switch. Observe "RIGHT" on display.
- Press LEFT switch. Observe "LEFT" on display.
- Press UP switch. Observe "UP" on display.
- Press DOWN switch. Observe "DOWN" on display.
- Press SELECT switch. Observe "SELECT" on display.

2.6 Hardware Block Diagram

Figure 33. Demonstration Hardware Block Diagram

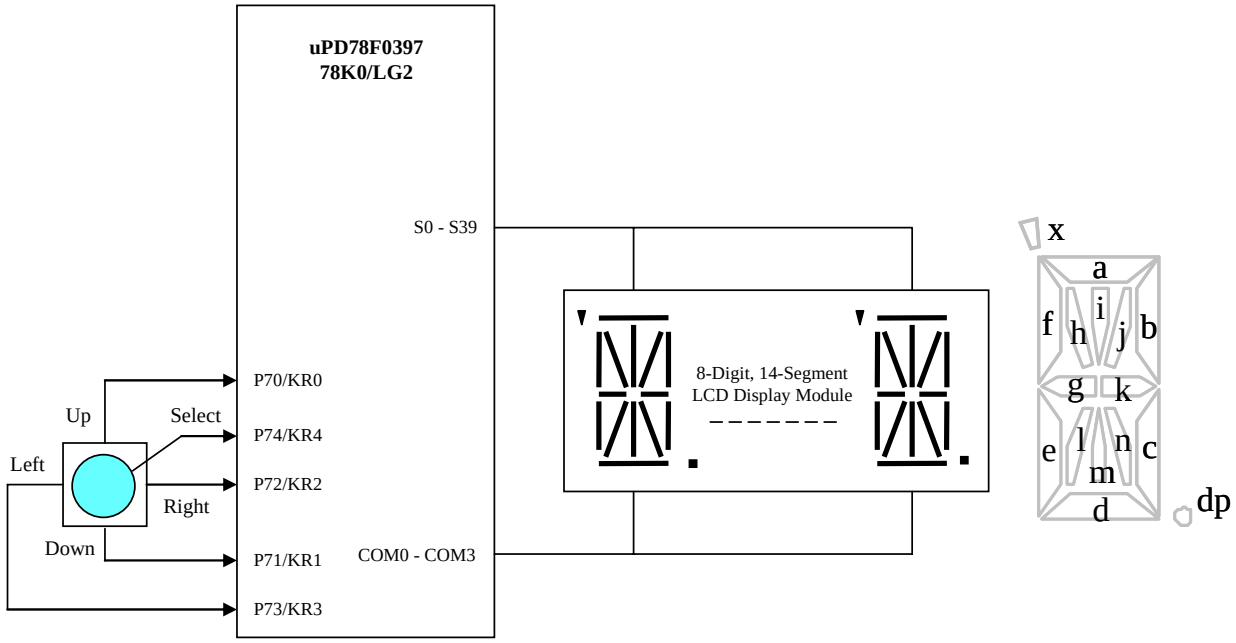


Table 8. LCD Pin Assignments and Connections

LCD					78K0/LG2 MCU	LCD					78K0/LG2 MCU
Pin	COM1	COM2	COM3	COM4		Pin	COM1	COM2	COM3	COM4	
1	X1	1F	1E	1D	S0	36	1H	1G	1L	1M	S2
2	1I	1J	1K	1N	S1	35	1A	1B	1C	DP1	S3
3	X2	2F	2E	2D	S4	34	2H	2G	2L	2M	S6
4	2I	2J	2K	2N	S5	33	2A	2B	2C	DP2	S7
5	X3	3F	3E	3D	S8	32	3H	3G	3L	3M	S10
6	3I	3J	3K	3N	S9	31	3A	3B	3C	DP3	S11
7	X4	4F	4E	4D	S12	30	4H	4G	4L	4M	S14
8	4I	4J	4K	4N	S13	29	4A	4B	4C	DP4	S15
9	X5	5F	5E	5D	S16	28	5H	5G	5L	5M	S18
10	5I	5J	5K	5N	S17	27	5A	5B	5C	DP5	S19
11	X6	6F	6E	6D	S20	26	6H	6G	6L	6M	S22
12	6I	6J	6K	6N	S21	25	6A	6B	6C	DP6	S23
13	X7	7F	7E	7D	S24	24	7H	7G	7L	7M	S26
14	7I	7J	7K	7N	S25	23	7A	7B	7C	DP7	S27
15	X8	8F	8E	8D	S28	22	8H	8G	8L	8M	S30
16	8I	8J	8K	8N	S29	21	8A	8B	8C	DP8	S31
17	NC	NC	NC	COM4	COM4	20	COM0	NC	NC	NC	COM0
18	NC	NC	COM3	NC	COM3	19	NC	COM1	NC	NC	COM1

2.7 Software Modules

Table 9. Demonstration Program Software Modules

File	Purpose	Generated By Applilet	Modified By User
Main.c	Main program	Yes	Yes
Macrodriver.h	General definitions used by the Applilet	Yes	No
System.h	Clock-related definitions	Yes	No
Systeminit.c	SystemInit() and hdwinit() functions	Yes	No
System.c	Clock_Init() function	Yes	No
Int.h	Interrupt-related definitions	Yes	Yes[Note 1]
Int.c	Key-return interrupt-related functions	Yes	No
Int_user.h	User code for key-return interrupt	Yes	Yes[Note 1]
Serial.h	IIC0-related definitions	Yes	Yes[Note 2]
Serial.c	IIC0-related functions	Yes	Yes[Note 2]
Serial_user.c	User code for IIC0 callback routines	Yes	Yes[Note 2]
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
TIMER_user.c	User code for timer interrupt handling	Yes	Yes
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No
define.h	Definitions of navigation switch values	--	Yes
Lcd.h	LCD high-level function definition	--	Yes
Lcd.c	LCD high-level functions	--	Yes
LcdDrvApp.h	LCD-controller driver definitions	--	Yes
LcdDrvApp.c	LCD-controller driver functions	--	Yes

Note 1: Int.h requires modification to add the declaration of sw3_in, the variable used to store the navigation switch debounced input. Int_user.c requires modification to add the variable sw3_in and to add the code for handling the key-return interrupt.

Note 2: Serial.h requires modification to add the declarations of IIC0-related global variables, defined in Serial_user.c, and also to declare the functions IIC0_Init( ) (not declared by default) and IIC0_MasterStartAndSend( ). Serial.c requires slight modification to remove settings for P6 not supported for 78K0/LG2 devices, and to replace asm versions of DI and EI instructions with NEC Electronics' C Compiler equivalents. Serial_user.c requires modification to have IIC0 callback functions set the IIC0-related global variables and to add the IIC0_MasterStartAndSend( ) function.

3. Appendix A - Development Tools

This application note makes use of the following software and hardware tools:

3.1 Software Tools

Table 10. Software Tools Used in Application Note

Tool	Version	Comments
Applilet for 78K0KX2	V1.51	Source-code generation tool for 78K0/KE2 devices
PM Plus	V5.20	Project manager for program compilation and linking
CC78K0	V3.60	C Compiler for NEC Electronics' 78K0 devices
RA78K0	V3.70	Assembler for NEC Electronics' 78K0 devices
DF0397.78K	V1.01	Device file for µPD78F0397 device

Note: The version of the Applilet used produces code for the 78K0/Kx2 family of microcontrollers—in this case, specifically the 78K0/KE2 (uPD78F0537_64). This device is very similar to the 78K0/LG2 device (µPD78F0397) used in the DemoKit-LG2.

3.2 Hardware Tools

Table 11. Hardware Tools used in Application Note

Tool	Version	Comments
DemoKit-LG2	V2.00	Demonstration kit for 78K0397 (78K0/LG2)

4. Appendix B – Software Listings

4.1 Main.c

```
/*  main.c for NEC LCD Application Note */
/*
************************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename : main.c
**  Abstract : This file implements main function
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :          uPD78F0537
**
**  Compiler:  NEC/CC78K0
**
************************************************************************************
*/
/*
************************************************************************************
**  Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "int.h"
#include "timer.h"
#include "serial.h"
/*
************************************************************************************
**  MacroDefine
************************************************************************************
*/

/* include files for DemoKit-LG2 LCD */
#include "defines.h"
#include "lcd.h"
#include "LcdDrvApp.h"

//-------------------------------------------------------------------------
// Global string constants
//-------------------------------------------------------------------------
const unsigned char *s_UP               = "   UP   ";
const unsigned char *s_DOWN                     = "  DOWN  ";
const unsigned char *s_LEFT             = "LEFT    ";
const unsigned char *s_RIGHT            = "   RIGHT";
const unsigned char *s_SELECT           = " SELECT ";
const unsigned char *s_MULTIPLE     = "MULTIPLE";

/*
**-------------------------------------------------------------------------
```

```
**
**  Abstract:
**          main function
**
**  Parameters:
**          None
**
**  Returns:
**          None
**
**---------------------------------------------------------------------------
*/
void main( void )
{
        IMS = MEMORY_IMS_SET;
        IXS = MEMORY_IXS_SET;
        /* TODO. add user code */

        /* initialize IIC */
        IIC0_Init();

        /* initialize LCD */
        LCD_Init();

        while(!sw3_in) {
                LCD_string_shift((unsigned char *)"NEC DEMOKIT-LG2");
                LCD_string_shift((unsigned char *)"LCD APP NOTE");
                LCD_string_shift((unsigned char *)"PRESS SW3 TO START");
        }
        sw3_in = 0;

        LCD_string((unsigned char *)" -SW3-  ", 0);

        while(1){
                if (sw3_in != 0) {
                        switch(sw3_in) {
                        case UP:
                                LCD_string(&s_UP[0],0);
                                break;
                        case DOWN:
                                LCD_string(&s_DOWN[0],0);
                                break;
                        case LEFT:
                                LCD_string(&s_LEFT[0],0);
                                break;
                        case RIGHT:
                                LCD_string(&s_RIGHT[0],0);
                                break;
                        case SELECT:
                                LCD_string(&s_SELECT[0],0);
                                break;
                        default:
                                LCD_string(&s_MULTIPLE[0],0);
                                break;
                        }
                        sw3_in=0;
                }
        }
}
```

4.2 Macrodriver.h

```
/*
************************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename : macrodriver.h
**  Abstract : This is the general header file
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :        uPD78F0537
**
**  Compiler:  NEC/CC78K0
**
************************************************************************************
*/
#ifndef    _MDSTATUS_
#define _MDSTATUS_

#pragma sfr
#pragma di
#pragma ei
#pragma NOP
#pragma HALT
#pragma STOP

/* data type defintion */
typedef    unsigned long       ULONG;
typedef    unsigned int        UINT;
typedef    unsigned short      USHORT;
typedef    unsigned char       UCHAR;
typedef    unsigned char       BOOL;

#define    ON      1
#define    OFF     0

#define    TRUE    1
#define    FALSE   0

#define IDLE       0                       /* idle status */
#define READ       1                       /* read mode */
#define WRITE      2                       /* write mode */

#define SET        1
#define CLEAR      0

#define MD_STATUS                  unsigned short
#define MD_STATUSBASE              0x0

/* status list definition */
#define MD_OK                              MD_STATUSBASE+0x0     /* register setting OK */
#define MD_RESET                           MD_STATUSBASE+0x1     /* reset input */
#define MD_SENDCOMPLETE                            MD_STATUSBASE+0x2     /* send data complete */
```

```
#define MD_OVF                          MD_STATUSBASE+0x3     /* timer count overflow */

/* error list definition */
#define MD_ERRORBASE            0x80
#define MD_ERROR                MD_ERRORBASE+0x0      /* error */
#define MD_RESOURCEERROR        MD_ERRORBASE+0x1      /* no resource available */
#define MD_PARITYERROR                  MD_ERRORBASE+0x2      /* UARTn parity error */
#define MD_OVERRUNERROR                 MD_ERRORBASE+0x3      /* UARTn overrun error */
#define MD_FRAMEERROR                   MD_ERRORBASE+0x4      /* UARTn frame error */
#define MD_ARGERROR             MD_ERRORBASE+0x5      /* Error agrument input error */
#define MD_TIMINGERROR                  MD_ERRORBASE+0x6      /* Error timing operation error */
#define MD_SETPROHIBITED        MD_ERRORBASE+0x7      /* setting prohibited */
#define MD_DATAEXISTS           MD_ERRORBASE+0x8      /* Data to be transferred next exists in TXBn register */
#define MD_SPT                  MD_STATUSBASE+0x8     /*IIC stop*/
#define MD_NACK                         MD_STATUSBASE+0x9     /*IIC no ACK*/
#define MD_SLAVE_SEND_END       MD_STATUSBASE+0x10    /*IIC slave send end*/
#define MD_SLAVE_RCV_END        MD_STATUSBASE+0x11     /*IIC slave receive end*/
#define MD_MASTER_SEND_END      MD_STATUSBASE+0x12     /*IIC master send end*/
#define MD_MASTER_RCV_END       MD_STATUSBASE+0x13    /*IIC master receive end*/

/* main clock and subclock as clock source */
enum ClockMode { HiRingClock, SysClock };

/* the value for IMS and IXS */
#define  MEMORY_IMS_SET                 0xCC
#define  MEMORY_IXS_SET                 0x00
/* clear IO register bit and set IO register bit */
#define ClrIORBit(Reg, ClrBitMap)     Reg &= ~ClrBitMap
#define SetIORBit(Reg, SetBitMap)     Reg |= SetBitMap

enum INTLevel { Highest, Lowest };


#define    SYSTEMCLOCK  8000000
#define    SUBCLOCK     32768
#define    MAINCLOCK    8000000
#define    FRCLOCK      8000000
#define    FRCLOCKLOW   240000


#endif
```

4.3 System.h

```
/*
************************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      system.h
**  Abstract :      This file implements device driver for SYSTEM module.
```

```
**  APIlib :        NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :        uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
************************************************************************************
*/
#ifndef   _MDSYSTEM_
#define   _MDSYSTEM_
/*
************************************************************************************
**  MacroDefine
************************************************************************************
*/
#define   CG_X1STAB_SEL0x5
#define   CG_X1STAB_STA         0x1f
#define   CG_CPU_CLOCKSEL       0x0

enum CPUClock { SystemClock, Sys_Half, Sys_Quarter, Sys_OneEighth, Sys_OneSixteen, Sys_SubClock };
enum PSLevel { PS_STOP, PS_HALT };
enum StabTime { ST_Level0, ST_Level1, ST_Level2, ST_Level3, ST_Level4 };

void  Clock_Init( void );

#endif
```

4.4 Systeminit.c

```
/*
************************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      systeminit.c
**  Abstract :      This file implements macro initialization.
**  APIlib :        NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :        uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
************************************************************************************
*/
/*
************************************************************************************
**  Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
```

```
#include "int.h"
#include "timer.h"
#include "serial.h"
/*
************************************************************************************
**  MacroDefine
************************************************************************************
*/

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**          Init every Macro
**
**  Parameters:
**          None
**
**  Returns:
**          None
**
**-------------------------------------------------------------------------
*/
void SystemInit( void )
{
            /* Clock generator initiate */
            Clock_Init();
            /* INT initiate */
            INT_Init();
            /* TM50 initiate */
            TM50_Init();
            /* TM51 initiate */
            TM51_Init();
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**          Init hardware setting
**
**  Parameters:
**          None
**
**  Returns:
**          None
**
**-------------------------------------------------------------------------
*/
void hdwinit( void )
{
            DI( );
            SystemInit( );
            EI( );
}
```

4.5 System.c

```
/*
```

```
************************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      system.c
**  Abstract :      This file implements device driver for System module.
**  APIlib :        NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :        uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
************************************************************************************
*/
/*
************************************************************************************
**  Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
/*
************************************************************************************
**  MacroDefine
************************************************************************************
*/

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**      Init the Clock Generator and Oscillation stabilization time.
**
**  Parameters:
**      None
**
**  Returns:
**      None
**
**-------------------------------------------------------------------------
*/
void Clock_Init( void )
{
    ClrIORBit(MCM, 0x05);                   /* High-Internal-OSC operate for CPU */

    SetIORBit(MCM, 0x01);                   /* peripheral hardware clock:frh */
    SetIORBit(PM12, 0x18);                  /* P123/124 input mode */
    ClrIORBit(OSCCTL, 0x20);            /* XT1 input mode */
    SetIORBit(OSCCTL, 0x10);
    SetIORBit(MOC, 0x80);                   /* stop X1 clock */
    PCC = CG_CPU_CLOCKSEL;
}
```

4.6 Int.h

```
/*
************************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      int.h
**  Abstract :      This file implements device driver for INT module.
**  APIlib :        NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :        uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
************************************************************************************
*/

#ifndef   _MDINT_
#define   _MDINT_
/*
************************************************************************************
**  MacroDefine
************************************************************************************
*/
#define   EGP_INT           0x0
#define   EGN_INT           0x0
#define   PU7_KR 0x1f
#define   PM7_KR 0x1f
#define   KRM_KR            0x1f

enum ExternalINT {
          EX_INTP0, EX_INTP1, EX_INTP2, EX_INTP3,
          EX_INTP4, EX_INTP5, EX_INTP6, EX_INTP7
          };

enum INTInputEdge {
          None, RisingEdge, FallingEdge, BothEdge
          };

enum MaskableSource {
          INT_LVI, INT_INTP0, INT_INTP1, INT_INTP2,
          INT_INTP3, INT_INTP4, INT_INTP5, INT_SRE6,
          INT_SR6, INT_ST6, INT_CSI10_ST0, INT_TMH1,
          INT_TMH0, INT_TM50, INT_TM000, INT_TM010,
          INT_AD, INT_SR0, INT_WTI, INT_TM51,
          INT_KR, INT_WT, INT_INTP6, INT_INTP7,
          INT_IIC0_DMU, INT_CSI11, INT_TM001, INT_TM011
          };
void INT_Init( void );
__interrupt void MD_INTKR( void );

/* global value used for debounced switch input */
extern __sreg volatile unsigned char sw3_in;
```

```
#endif
```

4.7 Int.c

```
/*
************************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      int.c
**  Abstract :      This file implements device driver for INT module.
**  APIlib :        NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :        uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
************************************************************************************
*/

/*
************************************************************************************
**  Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "int.h"
/*
************************************************************************************
**  MacroDefine
************************************************************************************
*/

/*
**-----------------------------------------------------------------------
**
**  Abstract:
**          This function initializes the external interrupt, key return function.
**
**  Parameters:
**          None
**
**  Returns:
**          None
**
**-----------------------------------------------------------------------
*/
void INT_Init( void )
{
            EGP = EGP_INT;
            EGN = EGN_INT;
```

```
    KRMK = 1;                           /* disable INTKR */
    PU7 |= PU7_KR;
    PM7 |= PM7_KR;
    KRM = KRM_KR;                       /* set KR input mode */
    KRPR = 1;
    KRIF = 0;
    KRMK = 0;                           /* enable INTKR */
}
```

4.8 Int_user.c

```
/*
************************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      int_user.c
**  Abstract :      This file implements device driver for INT module.
**  APIlib :        NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :        uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
************************************************************************************
*/

#pragma  interrupt  INTKR   MD_INTKR

/*
************************************************************************************
**  Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "int.h"
/*
************************************************************************************
**  MacroDefine
************************************************************************************
*/

#include "timer.h"   // for TM51 routines and values
/* global value used for debounced switch input */
__sreg volatile unsigned char sw3_in;

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
```

```
**          INTKR Interrupt service routine.
**
**  Parameters:
**          None
**
**  Returns:
**          None
**
**----------------------------------------------------------------------------
*/
__interrupt void MD_INTKR( void )
{
unsigned char sw3_first,sw3_second;
   sw3_first= (~P7) & 0x1f;     // read SW3 first time
   TM51_ChangeTimerCondition(TM_TM51_INTERVALVALUE);
   TM51_Start();
   while(!TMIF51);              // wait for Timer51 Interrupt
   TM51_Stop();
   TMIF51=0;
   sw3_second= (~P7) & 0x1f;     // read SW3 second time
   if(sw3_first==sw3_second)
      sw3_in=sw3_first; // debounce SW3
   else
      sw3_in=0;
}
```

4.9 Serial.h

```
/*
*******************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      serial.h
**  Abstract :      This file implements device driver for SERIAL module.
**  APIlib :        NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :        uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
*******************************************************************************
*/
#ifndef _MDSERIAL_
#define _MDSERIAL_
/*
*******************************************************************************
**  Global variables
*******************************************************************************
*/
#define   IIC0_SLAVEADDRESS      0x0
```

```
enum TransferMode { Send, Receive };
MD_STATUS IIC0_MasterStart( enum TransferMode , UCHAR , UCHAR );
MD_STATUS IIC0_MasterSendData( UCHAR* , UINT );
MD_STATUS IIC0_MasterReceiveData( UCHAR* , UINT );
void IIC0_Stop( void );
__interrupt void MD_INTIIC0( void ) ;
void IIC0_User_Init( void );
MD_STATUS IIC0_SlaveHandler( void );
MD_STATUS IIC0_MasterHandler( void );
void CALL_IIC0_SlaveAddressMatch( void );
void CALL_IIC0_MasterFindSlave( void );
void CALL_IIC0_MasterSendEnd( void  );
void CALL_IIC0_MasterReceiveEnd( void  );
void CALL_IIC0_MasterError( MD_STATUS flag );

/* added flags set by callback routines for use by upper level routine */
extern MD_STATUS UI_MasterError;
extern MD_STATUS UI_MasterSendEnd;
extern MD_STATUS UI_MasterReceiveEnd;
extern MD_STATUS UI_MasterFindSlave;

/* added definition for initialize routine */
void IIC0_Init( void );

/* added combined function */
MD_STATUS IIC0_MasterStartAndSend(UCHAR sadr, UCHAR* txbuf, UINT txnum);

#endif
```

4.10 Serial.c

```
/*
************************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      serial.c
**  Abstract :      This file implements device driver for SERIAL module.
**  APIlib :        NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :        uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
************************************************************************************
*/


#pragma interrupt   INTIIC0             MD_INTIIC0
```

```
/*
************************************************************************************
**  Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"
UCHAR  iic0_m_sta_flag;             /* start flag for send address check by master mode */
UCHAR  *iic0_m_send_pbuf;                   /* send data pointer by master mode */
UINT   iic0_m_send_size;            /* send data size by master mode */
UCHAR  *iic0_m_rev_pbuf;            /* receive data pointer by master mode */
UINT   iic0_m_rev_size;             /* receive data size by master mode */
UCHAR  iic0_s_sta_flag;             /* start flag for send address check by slave mode */
UCHAR  *iic0_s_send_pbuf;           /* send data pointer by slave mode */
UINT   iic0_s_send_size;            /* send data size by slave mode */
UCHAR  *iic0_s_rev_pbuf;            /* receive data pointer by slave mode */
UINT   iic0_s_rev_size;             /* receive data size by slave mode */

/*
**-------------------------------------------------------------------------
**
** Abstract:
**      This function initializes IIC0 module.
**
** Parameters:
**      None
**
** Returns:
**      None
**
**-------------------------------------------------------------------------
*/
void IIC0_Init( void )
{
        ClrIORBit(IICC0, 0x80);                 /* stop IIC0 */

        SetIORBit(MK1H, 0x1);                   /* disable interrupt */

        ClrIORBit(PM6, 0x03);                   /* port setting */
// remove setting of P6 for 78F0397 (78K0/LG2)
//      ClrIORBit(P6, 0x03);

        SetIORBit(IICF0, 0x02);                 /* start-condition doesn't need stop-condition */
        SetIORBit(IICF0, 0x01);                 /* comunication reserve - disable */
        SetIORBit(IICC0, 0x10);                 /* stop-condition interrupt - enable */
        ClrIORBit(IICC0, 0x08);                 /* interrupt control - 8 clock falling edge */

        /* transfer clock */
        /* fprs/88 */
        ClrIORBit(IICCL0, 0x08);        /* normal mode */
        ClrIORBit(IICCL0, 0x3);                 /* CL00 = 0 CL01 = 0 */
        ClrIORBit(IICX0, 0x1);                  /* disable extension */

        /* selection interrupt priority */
        SetIORBit(PR1H, 0x1);                   /* interrupt priority low */

        ClrIORBit(MK1H, 0x1);                   /* enable interrupt */

        IIC0_User_Init( );

        return;
}
```

```
/*
**-------------------------------------------------------------------------
**
** Abstract:
**      This function is responsible for start IIC0 by master mode.
**
** Parameters:
**      enum TransferMode mode :   select transfer mode
**              Send :              send data
**              Receive : receive data
**      UCHAR adr :     set address for select slave
**      UCHAR wait :    set wait for need waiting when get start condition
**
** Returns:
**      MD_OK
**      MD_ERROR
**      MD_ARGERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS IIC0_MasterStart( enum TransferMode mode, UCHAR adr, UCHAR wait )
{
        /* bus check */
//      __asm("di");        // replace in-line assembly with psuedo-function
        DI();
        if( IICF0 & 0x40 ){
//              __asm("ei");                    /* bus busy */
                EI();       // replace in-line assembly with psuedo-function
                return MD_ERROR;
        }

        /* start  IIC0 */
        SetIORBit(IICC0, 0x18);                 /* SPIE0 = WTIM0 = 1 */
        SetIORBit(IICC0, 0x80);                 /* IICE0 = 1 */

        SetIORBit(IICC0, 0x02);                 /* generate start condition */
//      __asm("ei");
        EI();       // replace in-line assembly with psuedo-function

        /* wait */
        while( wait-- );

        if( !(IICS0 & 0x2) ){           /* check start condition */
                return MD_ERROR;
        }

        /* set transfer mode to address */
        /* slave would be selected trans or receive from bit0 at address */
        if( mode == Send ){
                ClrIORBit(adr, 0x01);       /* if master is send mode, clear bit0 */
        }
        else if( mode == Receive ){
                SetIORBit(adr, 0x01);       /* if master is receive mode, set bit0 */
        }
        else{
                return MD_ARGERROR;
        }

        iic0_m_sta_flag = 0;
        IIC0 = adr;                             /* send address */

        return MD_OK;
```

```
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**          This function is responsible for stop IIC0.
**
** Parameters:
**          None
**
** Returns:
**          None
**
**-------------------------------------------------------------------------
*/
void IIC0_Stop( void )
{
          ClrIORBit(IICC0, 0x80);                   /* stop transfer */
          return;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**          This function is responsible for IIC0 data transfering by master mode.
**
** Parameters:
**          UCHAR* txbuf :    transfer buffer pointer
**          UINT txnum :      buffer size
**
** Returns:
**          MD_OK
**          MD_ERROR :        cannot send address
**
**-------------------------------------------------------------------------
*/
MD_STATUS IIC0_MasterSendData(UCHAR* txbuf, UINT txnum)
{
          if( iic0_m_sta_flag == 0 ){
                    return MD_ERROR;                  /* cannnot send address */
          }

          /* set parameter */
          iic0_m_send_size = txnum;
          iic0_m_send_pbuf = txbuf;

          IIC0 = *iic0_m_send_pbuf ++ ;               /* start transfer */
          iic0_m_send_size--;

          return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**          This function is responsible for IIC0 data receiving by master mode.
**
** Parameters:
**          UCHAR* rxbuf :    receive buffer pointer
**          USHORT rxnum :    buffer size
```

```
**
** Returns:
**          MD_OK
**          MD_ERROR :      cannot send address
**
**------------------------------------------------------------------------------
*/
MD_STATUS IIC0_MasterReceiveData(UCHAR* rxbuf, UINT rxnum)
{
        if( iic0_m_sta_flag == 0 ){
                return MD_ERROR;                /* cannnot send address */
        }

        /* set parameter */
        iic0_m_rev_size = rxnum;
        iic0_m_rev_pbuf = rxbuf;

        ClrIORBit(IICC0, 0x08);                 /* clear WTIM0 */
        SetIORBit(IICC0, 0x04);                 /* set ACKE0 */

        SetIORBit(IICC0, 0x20);                 /* start receive */

        return MD_OK;
}

/*
**------------------------------------------------------------------------------
**
** Abstract:
**          IIC0 interrupt service routine
**
** Parameters:
**          None
**
** Returns:
**          None
**
**------------------------------------------------------------------------------
*/
__interrupt void MD_INTIIC0( void )
{
        MD_STATUS sta;
        if( IICS0 & 0x80 ) {
                sta = IIC0_MasterHandler();
        }
        else {
                sta = IIC0_SlaveHandler();
        }
}

/*
**------------------------------------------------------------------------------
**
** Abstract:
**          The function call at IIC0 interrupt request
**
** Parameters:
**          None.
**
** Returns:
**          MD_OK
**          MD_ERROR :      cannot get address
**                          not slave mode
```

```
**          MD_SLAVE_RCV_END : all data received
**          MD_SLAVE_SEND_END : all data sended
**          MD_SPT : get stop condition
**
**----------------------------------------------------------------------------
*/
MD_STATUS IIC0_SlaveHandler( void )
{
        /* control for stop condition */
        if( IICS0 & 0x01 ){                                     /* get stop condition */
                /* slave send end and get stop condition */
                if( iic0_s_sta_flag &&( iic0_s_send_size == 0 ) ){
                        return MD_SLAVE_SEND_END;
                } else {
                        return MD_SPT;
                }
        }

        /* control for get address */
        if( iic0_s_sta_flag == 0 ){
                if( !(IICS0 & 0x20) ){                          /* check EXC0 -> external code */
                        if( IICS0 & 0x10 ){                     /* check COI0 -> address */
                                iic0_s_sta_flag = 1;
                                CALL_IIC0_SlaveAddressMatch();  /* slave address match */
                        }
                        else{
                                return MD_ERROR;
                        }
                }
                else{
                        return MD_ERROR;
                }
        }

        /* slave send control */
        else if( IICS0 & 0x08 ){
                if(!( IICS0 & 0x04 )){                          /* check ACKD0 -> acknowledge */
                        return MD_NACK;
                }
                IIC0 = *iic0_s_send_pbuf ++ ;
                iic0_s_send_size--;
        }
        /* slave receive control */
        else{
                *iic0_s_rev_pbuf ++ = IIC0;
                iic0_s_rev_size--;
                SetIORBit(IICC0, 0x20);                                 /* WREL1 = 1 start receive */
                if( iic0_s_rev_size == 0 ){                     /* check all data received */
                        ClrIORBit(IICC0, 0x04);                         /* clear ACKE0 */
                        return MD_SLAVE_RCV_END;
                }
        }
        return MD_OK;
}

/*
**----------------------------------------------------------------------------
**
** Abstract:
**          The function call at IIC0 interrupt request.
**
** Parameters:
**          None.
```

```
**
** Returns:
**          MD_OK
**          MD_ERROR :      cannot get ack after sended address
**                                  not master mode
**                                  slave did not send ack
**          MD_MASTER_RCV_END : all data received
**          MD_MASTER_SEND_END : all data sended
**
**--------------------------------------------------------------------------
*/
MD_STATUS IIC0_MasterHandler( void )
{
        /* control for stop condition */
        if( !( IICF0 & 0x40 ) ){                        /* get stop condition */
                CALL_IIC0_MasterError(MD_SPT);
                return MD_SPT;
        }

        /* control for sended condition */
        if( !iic0_m_sta_flag ){
                if( IICS0 & 0x4 ){                      /* check ACK */
                        iic0_m_sta_flag = 1;                    /* address complete */
                        CALL_IIC0_MasterFindSlave();
                }
                else{
                        CALL_IIC0_MasterError(MD_NACK);
                        return MD_NACK;
                }
        }
        /* master send control */
        else if( IICS0 & 0x8 ){
                if( !(IICS0 & 0x4) ){                   /* check ACK */
                        SetIORBit(IICC0, 0x01);                 /* generate stop condition */
                        CALL_IIC0_MasterError(MD_NACK);
                        return MD_NACK;
                }

                if( !iic0_m_send_size ){                /* sended finish */
                        SetIORBit(IICC0, 0x01);                 /* generate stop condition */
                        CALL_IIC0_MasterSendEnd( );
                        return MD_MASTER_SEND_END;
                }
                IIC0 = *iic0_m_send_pbuf ++ ;                   /* send data */
                iic0_m_send_size--;
        }
        /* master receive control */
        else {
                *iic0_m_rev_pbuf ++ = IIC0;             /* receive data */
                iic0_m_rev_size--;
                if( iic0_m_rev_size == 0 ){             /* receive finish */
                        ClrIORBit(IICC0, 0x04);                 /* ACK STOP */
                        SetIORBit(IICC0, 0x01);                 /* generate stop condition */
                        CALL_IIC0_MasterReceiveEnd( );
                        return MD_MASTER_RCV_END;
                }
                SetIORBit(IICC0, 0x20);                         /* start receive */
        }
        return MD_OK;
}
```

4.11 Serial_user.c

```
/*
************************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      serial_user.c
**  Abstract :      This file implements device driver for SERIAL module.
**  APIlib :        NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :        uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
************************************************************************************
*/
/*
************************************************************************************
**  Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"

/* added flags set by callback routines for use by upper level routine */
MD_STATUS UI_MasterError;
MD_STATUS UI_MasterSendEnd;
MD_STATUS UI_MasterReceiveEnd;
MD_STATUS UI_MasterFindSlave;

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**      This function is an empty function for user code when IIC0 initializing
**
**  Parameters:
**      None
**
**  Returns:
**      None
**
**-----------------------------------------------------------------------------
*/

void IIC0_User_Init( void )
{

}
```

```
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**          Master Error,
**          callback function open for users operation
**
**  Parameters:
**          MD_STATUS flag
**
**  Returns:
**          None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_MasterError( MD_STATUS flag )
{
        /* user operation */
        UI_MasterError = flag;
        return;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**          Master recevice finish,
**          callback function open for users operation
**
**  Parameters:
**          None
**
**  Returns:
**          None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_MasterReceiveEnd( void )
{
        /* user operation */
        UI_MasterReceiveEnd = MD_OK;
        return;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**          Master send finish,
**          callback function open for users operation
**
**  Parameters:
**          None
**
**  Returns:
**          None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_MasterSendEnd( void )
{
        /* user operation */
        UI_MasterSendEnd = MD_OK;
```

```
        return;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**      IIC0 slave address match
**      callback function for users operation
**
**  Parameters:
**      None
**
**  Returns:
**      None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_SlaveAddressMatch( void )
{
        /* user operation */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**      Master find the slave address
**      callback function open for users operation
**
**  Parameters:
**      None
**
**  Returns:
**      None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_MasterFindSlave( void )
{
        /* user operation */
        UI_MasterFindSlave = MD_OK;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**      Combines IIC0_MasterStart(Send, (sadr), 10)
**    and IIC0_MasterSendData(UCHAR* txbuf, UINT txnum)
**
**  Parameters:
**      UCHAR sadr :      set address for select slave
**      UCHAR* txbuf :    transfer buffer pointer
**      UINT txnum :      buffer size
**
**  Returns:
**      MD_OK
**      MD_ERROR
**      MD_ARGERROR
**  MD_NACK - timeout on slave address
**
**-------------------------------------------------------------------------
```

```
*/

MD_STATUS IIC0_MasterStartAndSend(UCHAR sadr, UCHAR* txbuf, UINT txnum)
{
MD_STATUS status;
int i,j;

        // Init IIC in case it needs it
//      IIC0_Init();

        // set up for first operation
        UI_MasterError = MD_OK;
        UI_MasterSendEnd = MD_ERROR;
        UI_MasterFindSlave = MD_ERROR;

        status = IIC0_MasterStart(Send, sadr, 10);
        if (status != MD_OK) {
                return status;
        }

        i = 10000;
        do {
                i--;
        } while ( (UI_MasterFindSlave == MD_ERROR) && (UI_MasterError == MD_OK) && ( i > 0 ) );

        if (i == 0) {
                return MD_NACK;
        }

        if (UI_MasterError != MD_OK) {
                return UI_MasterError;
        }

        // got slave address ok here
        status = IIC0_MasterSendData(txbuf, txnum);     // send data bytes
        if (status != MD_OK) {
                return status;
        }
        i = 10000;
        do {
                i--;
        } while ( (UI_MasterError == MD_OK) && (UI_MasterSendEnd == MD_ERROR) && (i > 0) );

        if (i == 0) {
                return MD_NACK;
        }
        if (UI_MasterError != MD_OK) {
                return UI_MasterError;
        }
        if (UI_MasterSendEnd != MD_OK) {
                return UI_MasterSendEnd;
        }

        // fix to interact with other LCD code for now
//      SetIORBit(MK1H, 0x1);                   /* disable interrupt */
//      ClrIORBit(IICC0, 0x10);                 /* clear SPIE0 bit */
//      ClrIORBit(IF1H, 0x01);                  /* clear IICIF0 bit */
        return MD_OK;     /* no error */
}
```

4.12 Timer.h

```
/*
********************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename : timer.h
**  Abstract : This file implements a device driver for the timer module
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :          uPD78F0537
**
**  Compiler:  NEC/CC78K0
**
********************************************************************************
*/

#ifndef _MDTIMER_
#define _MDTIMER_
/*
********************************************************************************
**  MacroDefine
********************************************************************************
*/
#define   REGVALUE_MAX            0xff
#define   TM_TM00_CLOCK           0x0
#define   TM_TM00_INTERVALVALUE       0x00
#define   TM_TM00_SQUAREWIDTH         0x00
#define   TM_TM00_PPGCYCLE    0x00
#define   TM_TM00_PPGWIDTH    0x00
#define   TM_TM00_ONESHOTCYCLE        0x00
#define   TM_TM00_ONEPULSEDELAY       0x00
#define   TM_TM01_CLOCK           0x0
#define   TM_TM01_INTERVALVALUE       0x00
#define   TM_TM01_SQUAREWIDTH         0x00
#define   TM_TM01_PPGCYCLE    0x00
#define   TM_TM01_PPGWIDTH    0x00
#define   TM_TM01_ONESHOTCYCLE        0x00
#define   TM_TM01_ONEPULSEDELAY       0x00
#define   TM_TM50_CLOCK           0x7
#define   TM_TM50_INTERVALVALUE       0x2f
#define   TM_TM50_SQUAREWIDTH         0x2f
#define   TM_TM50_PWMACTIVEVALUE    0x2f
#define   TM_TM51_CLOCK           0x7
#define   TM_TM51_INTERVALVALUE       0x8
#define   TM_TM51_SQUAREWIDTH         0x8
#define   TM_TM51_PWMACTIVEVALUE    0x8
#define   TM_TMH0_CLOCK           0x0
#define   TM_TMH0_INTERVALVALUE       0x00
#define   TM_TMH0_SQUAREWIDTH         0x00
#define   TM_TMH0_PWMCYCLE    0x00
#define   TM_TMH0_PWMDELAY    0x00
#define   TM_TMH1_CLOCK           0x0
```

```
#define   TM_TMH1_INTERVALVALUE       0x00
#define   TM_TMH1_SQUAREWIDTH         0x00
#define   TM_TMH1_PWMCYCLE    0x00
#define   TM_TMH1_PWMDELAY    0x00
#define   TM_TMH1_CARRIERDELAY        0x00
#define   TM_TMH1_CARRIERWIDTH        0x00


/* timer00 to 01,50,51,H0,H1 configurator initiation */
void TM50_Init(void);
void TM51_Init(void);

/*timer start*/
void TM50_Start(void);
void TM51_Start(void);

/*timer stop*/
void TM50_Stop(void);
void TM51_Stop(void);
MD_STATUS TM50_ChangeTimerCondition(UCHAR value);
MD_STATUS TM51_ChangeTimerCondition(UCHAR value);

#endif                /* _MDTIMER_*/
```

4.13 Timer.c

```
/*
************************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename : timer.c
**  Abstract : This file implements a device driver for the timer module
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :          uPD78F0537
**
**  Compiler:  NEC/CC78K0
**
************************************************************************************
*/

/*
************************************************************************************
**  Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"
```

```
/*
************************************************************************************
**  MacroDefine
************************************************************************************
*/
/*TM00 pulse width measure*/

/*TM01 pulse width measure*/

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**          This function can initialize TM50_module.
**
**  Parameters:
**          None
**
**  Returns:
**          None
**
**-------------------------------------------------------------------------
*/
void TM50_Init( void )
{
            ClrIORBit(TMC50, 0x80);
            TCL50 = TM_TM50_CLOCK;                          /* countclock=fx/8192 */
            /* TM50 interval */
            CR50 = TM_TM50_INTERVALVALUE;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**          This function can start the TM50 counter.
**
**  Parameters:
**          None
**
**  Returns:
**          None
**
**-------------------------------------------------------------------------
*/
void TM50_Start( void )
{
            /* TM50 interval */
            SetIORBit(TMC50, 0x80);
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**          This function can stop the TM50 counter and clear the count register.
**
**  Parameters:
**          None
**
**  Returns:
**          None
**
```

```
**------------------------------------------------------------------------
*/
void TM50_Stop( void )
{
        ClrIORBit(TMC50, 0x80);
}

/*
**------------------------------------------------------------------------
**
**  Abstract:
**      This function can change TM50 condition.
**
**  Parameters:
**      UCHAR :         value
**  Returns:
**      MD_OK
**      MD_ERROR
**
**------------------------------------------------------------------------
*/
MD_STATUS TM50_ChangeTimerCondition(UCHAR value)
{
        CR50 =value;
        return MD_OK;
}

/*
**------------------------------------------------------------------------
**
**  Abstract:
**      This function Initializes TM51_module.
**
**  Parameters:
**      None
**
**  Returns:
**      None
**
**------------------------------------------------------------------------
*/
void TM51_Init( void )
{
        ClrIORBit(TMC51, 0x80);
        TCL51 = TM_TM51_CLOCK;                  /* countclock=fx/4096 */
        /* TM51 interval */
        CR51 = TM_TM51_INTERVALVALUE;
}

/*
**------------------------------------------------------------------------
**
**  Abstract:
**      This function starts the TM51 counter.
**
**  Parameters:
**      None
**
**  Returns:
**      None
**
**------------------------------------------------------------------------
*/
```

```
void TM51_Start( void )
{
        /* TM51 interval */
        SetIORBit(TMC51, 0x80);
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**      This function stops the TM51 counter and clear the count register.
**
**  Parameters:
**      None
**
**  Returns:
**      None
**
**-------------------------------------------------------------------------
*/
void TM51_Stop( void )
{
        ClrIORBit(TMC51, 0x80);
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**      This function can change TM51 condition.
**
**  Parameters:
**      UCHAR :         value
**  Returns:
**      MD_OK
**      MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS TM51_ChangeTimerCondition(UCHAR value)
{
        CR51 =value;
        return MD_OK;
}
```

4.14 TIMER_user.c

```
/*
*******************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
```

```
**
**  Filename : timer_user.c
**  Abstract : This file implements a device driver for the timer module
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :        uPD78F0537
**
**  Compiler:  NEC/CC78K0
**
*****************************************************************************
*/

#pragma sfr
/*
*****************************************************************************
**  Include files
*****************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"


/*
*****************************************************************************
**  MacroDefine
*****************************************************************************
*/

/* Timer00, Timer01 pulse width measure */
```

4.15 Option.inc

```
;*****************************************************************************
;**
;**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
;**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
;**
;**  Copyright(C) NEC Electronics Corporation 2002 - 2005
;**  All rights reserved by NEC Electronics Corporation.
;**
;**  This program should be used on your own responsibility.
;**  NEC Electronics Corporation assumes no responsibility for any losses
;**  incurred by customers or third parties arising from the use of this file.
;**
;**  Filename : option.asm
;**  Abstract : This file implements OPTION-BYTES/SECURITY-ID setting.
;**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;**  Device :        uPD78F0537
;**
;**  Compiler :      NEC/CC78K0
;**
;*****************************************************************************

;
;*****************************************************************************
;**  MacroDefine
;*****************************************************************************
;
```

```
OPTION_BYTE     EQU     00H
POC81   EQU     00H
POC82   EQU     00H
POC83   EQU     00H
CG_ONCHIP       EQU     02H
CG_SECURITY0    EQU     0ffH
CG_SECURITY1    EQU     0ffH
CG_SECURITY2    EQU     0ffH
CG_SECURITY3    EQU     0ffH
CG_SECURITY4    EQU     0ffH
CG_SECURITY5    EQU     0ffH
CG_SECURITY6    EQU     0ffH
CG_SECURITY7    EQU     0ffH
CG_SECURITY8    EQU     0ffH
CG_SECURITY9    EQU     0ffH
```

4.16 Option.asm

```
;*******************************************************************************
;**
;**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
;**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
;**
;**  Copyright(C) NEC Electronics Corporation 2002 - 2005
;**  All rights reserved by NEC Electronics Corporation.
;**
;**  This program should be used on your own responsibility.
;**  NEC Electronics Corporation assumes no responsibility for any losses
;**  incurred by customers or third parties arising from the use of this file.
;**
;**  Filename : option.asm
;**  Abstract : This file implements OPTION-BYTES/SECURITY-ID setting.
;**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;**  Device :        uPD78F0537
;**
;**  Compiler :      NEC/CC78K0
;**
;*******************************************************************************

;
;*******************************************************************************
;**  Include files
;*******************************************************************************
$ INCLUDE (option.inc)
        OPT_SET CSEG AT 80H
OPTION:DB       OPTION_BYTE
        DB      POC81
        DB      POC82
        DB      POC83
        ONC_SET CSEG AT 84H
ONCHIP:DB       CG_ONCHIP

        CSEG SECUR_ID
SECURITY0:      DB      CG_SECURITY0
SECURITY1:      DB      CG_SECURITY1
SECURITY2:      DB      CG_SECURITY2
SECURITY3:      DB      CG_SECURITY3
```

```
SECURITY4:      DB      CG_SECURITY4
SECURITY5:      DB      CG_SECURITY5
SECURITY6:      DB      CG_SECURITY6
SECURITY7:      DB      CG_SECURITY7
SECURITY8:      DB      CG_SECURITY8
SECURITY9:      DB      CG_SECURITY9
END
```

4.17 define.h

```
#define UP     0x01
#define DOWN   0x02
#define RIGHT  0x04
#define LEFT   0x08
#define SELECT 0x10

#define BAUDRATE 115200
```

4.18 Lcd.h

```
/*
************************************************************************************
**
**  This file was created for the NEC Application Notes
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2006
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      lcd.h
**  Abstract :      This file implements header for LCD functions on DemoKit-LG2
**
**  Device :        uPD78F0397
**
**  Compiler :      NEC/CC78K0
**
************************************************************************************
*/
#ifndef    _LCD_H_
#define    _LCD_H_

void Wait(unsigned char Number);
void LCD_Init(void);

__callt extern void LCD_putc(unsigned  char digit, unsigned  char data);
__callt extern void LCD_string(unsigned char const *point, unsigned char dpos);
__callt extern void LCD_string_shift(unsigned char const *point);

#endif /* _LCD_H_ */
```

4.19 Lcd.c

```
/*
********************************************************************************
**
**  This file was created for the NEC Application Notes
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2006
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      lcd.c
**  Abstract :      This file implements LCD functions on DemoKit-LG2
**          Modified from DemoKit-LG2 example code to use LcdDrvApp.h
**
**  Device :        uPD78F0397
**
**  Compiler :      NEC/CC78K0
**
********************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"   /* for Timer50 routines */
#include "int.h"     /* for sw3_in definition */
#include "defines.h"
#include "lcd.h"
#include "LcdDrvApp.h"

//-----------------------------------------------------------------------------
// Global constants: minimum ASCII table for 14 segment LCD panel
//-----------------------------------------------------------------------------
unsigned const short characters[43]   = {

                                                                    0x0e70, // '0'
                    0x2060, // '1'
                    0x4c32, // '2'
                    0x4872, // '3'
                    0x4262, // '4'
                    0x4a52, // '5'
                    0x4e52, // '6'
                    0x0070, // '7'
                    0x4e72, // '8'
                    0x4a72, // '9'
                    0x500a, // '+'
                    0x4002, // '-'
                    0x4232, // '°'
                    0x0800, // '_'
                    0x2004, // '/'
                    0x4c42, // 'o'
                    0x0000, // ' '  => space
                    0x4672, // 'A'
                    0x4e42, // 'B'
                    0x0e10, // 'C'
                    0x4c62, // 'D'
                    0x0e12, // 'E'
                    0x0612, // 'F'
```

```
                        0x4e50, // 'G'
                        0x4662, // 'H'
                        0x1008, // 'I'
                        0x0c70, // 'J'
                        0xa602, // 'K'
                        0x0e00, // 'L'
                        0x2661, // 'M'
                        0x8661, // 'N'
                        0x0e70, // 'O'
                        0x4632, // 'P'
                        0x8e70, // 'Q'
                        0xc632, // 'R'
                        0x4a52, // 'S'
                        0x1018, // 'T'
                        0x0e60, // 'U'
                        0x8061, // 'V'
                        0x9664, // 'W'
                        0xa005, // 'X'
                        0x2009, // 'Y'
                        0x2814  // 'Z'
                       };

// string of spaces for clearing display
const unsigned char *s_clear   ="       ";

//---------------------------------------------------------------------------
// Global variables
//---------------------------------------------------------------------------
__sreg unsigned char transmit_buffer[2];
__sreg char message_byte_count;

//---------------------------------------------------------------------------
// Module name: Wait
// Description: This module delays the program for (number * 50ms).
//---------------------------------------------------------------------------
void Wait(unsigned char number)
{
        TM50_Start();       // start 50 ms timer
        while (number > 0) {
                while (TMIF50 == 0)
                        ;                       // wait for flag at end of time
                TMIF50 = 0;         // clear flag
                number--;// count down
        }
        TM50_Stop();        // stop timer
        TMIF50 = 0;                     // insure flag is zero
}

//---------------------------------------------------------------------------
// Module name: LCD_Init
// Description: Calls LcdDrv functions to initialize LCD
//---------------------------------------------------------------------------
void LCD_Init(void)
{
        LcdDrvInit();
        LcdDrvOnWait();

        Wait(80);               // voltage boost wait time = 4s
        LcdDrvOn();
}

//---------------------------------------------------------------------------
// Module:   LCD_putc
```

```
// Description: Sent character to LCD controller
//-----------------------------------------------------------------------------
__callt void LCD_putc (unsigned char digit, unsigned char data)
{
    // convert special characters
    switch(data)
    {
      case 0x2f:  data = 0x0d;  // '_'
                  break;
      case 0xf0:  data = 0x10;  // ' ' => space
                  break;
      case 0x80:  data = 0x0c;  // '°'
                  break;
      case 0xfb:  data = 0x0a;  // '+'
                  break;
      case 0xfd:  data = 0x0b;  // '-'
                  break;
      case 0xff:  data = 0x0e;  // '/'
                  break;
      case 0x3f:  data = 0x0f;  // 'o'
                  break;
      default:    break;
    }

    // load transmit buffer
    transmit_buffer [0] = ((characters[data])& 0xff00)>>8;
    transmit_buffer [1] = ((characters[data])& 0x00ff);
    message_byte_count = 2;

    digit<<=1;

    LcdDrvSegWrite(&transmit_buffer[0],digit,message_byte_count);
}

//-----------------------------------------------------------------------------
// Module:   LCD_string
// Description: Send character string to LCD module
//-----------------------------------------------------------------------------
__callt void LCD_string(unsigned char const *point, unsigned char dpos)
{
  while(dpos<=7)
  {
      if(point[0])
      {
         LCD_putc(dpos,*point-0x30);
         *point++;
      }
      else
      {
         LCD_putc(dpos,0xf0);
      }
      dpos++;
  }
}

//-----------------------------------------------------------------------------
// Module:   LCD_string_shift
// Description: Send character string to LCD module
//-----------------------------------------------------------------------------
__callt void LCD_string_shift(unsigned char const *point)
{
  unsigned char dpos=7;
  while(dpos!=0xff)
```

```
  {
    if(sw3_in)
    {
     LCD_string(&s_clear[0],0);
     return;
    }
    LCD_string(point,dpos);
    dpos--;
    Wait(4);
  }
  *point++;
  dpos=0;
  while(point[0])
  {
    if(sw3_in)
    {
      LCD_string(&s_clear[0],0);
      return;
    }
    *point++;
    LCD_string(point,dpos);
    Wait(4);
  }
}
```

4.20 LcdDrvApp.h

```
/*****************************************************************************
;
;   NNNNNN        NN  EEEEEEEEEEEEEEEEE    CCCCCCCCCCCCCCC
;   NNNNNNNN      NN  EEEEEE           CCCCCC
;   NNNNNNNNNN    NN  EEEEEE           CCCCCC
;   NN  NNNNNNNN  NN  EEEEEEEEEEEEEEEEE  CCCCCC
;   NN   NNNNNNNN NN  EEEEEE           CCCCCC
;   NN    NNNNNNNNNN  EEEEEE           CCCCCC
;   NN       NNNNNN   EEEEEEEEEEEEEEEE    CCCCCCCCCCCCCCC
;
;   NEC Electronics   78K0/Lx2
;
;*****************************************************************************
;   78K0/Lx2   LCD Control Sample Program
;*****************************************************************************
;   LCD Controller/Driver Control   -- Function and constant definition file
;*****************************************************************************
;[History]
;   2005.06.--  Newly created
;   2006.01.18 - mod to use Applilet-generated IIC routines instead of iic.c
;              - mod for DemoKit-LG2, LCDC = 0x02 instead of 0x0C
;*****************************************************************************/
/*===========================================================================
;   External reference
;===========================================================================*/
#ifndef _LCDDRVAPP_H_
#define _LCDDRVAPP_H_

/*== Various interface functions ==*/
/* LCD driver initialization processing  */
extern  unsigned char LcdDrvInit( void );
/* LCD driver display ON wait start pre-processing
```

```
   (used only when internal step-up mode is selected) */
extern  unsigned char LcdDrvOnWait( void );
/* LCD driver display ON processing */
extern  unsigned char LcdDrvOn( void );
/* LCD driver display OFF processing */
extern  unsigned char LcdDrvOff( void );
/* LCD driver control register write processing */
extern  unsigned char LcdDrvCtrWrite( unsigned char *src,
                              unsigned char addr,
                              unsigned char size );
/* LCD driver segment data write processing */
extern  unsigned char LcdDrvSegWrite( unsigned char *src,
                              unsigned char addr,
                              unsigned char size );
/* LCD driver segment data clear processing */
extern  unsigned char LcdDrvSegClr( void );
/* Single byte write processing in LCD driver control register */
extern  unsigned char LcdDrvCtrWrite1Byte( unsigned char addr,
                                  unsigned char data );
/* Single byte write processing of LCD driver segment data */
extern  unsigned char LcdDrvSegWrite1Byte( unsigned char addr,
                                  unsigned char data );
/*== Control register address value ==*/
#define CLDR_ADDR_LCDMD 0x00    /* 0x00: LCD mode select register     */
#define CLDR_ADDR_LCDM  0x01    /* 0x01: LCD display mode register    */
#define CLDR_ADDR_LCDC  0x02    /* 0x02: LCD clock control register   */
#define CLDR_ADDR_VLCG0 0x03    /* 0x03: LCD step-up control register */

/*== Error type value ==*/
enum
{
   CLDR_ERR_NONE       /* 0: No errors                       */
,  CLDR_ERR_NACK       /* 1: NACK received                   */
,  CLDR_ERR_BUSY       /* 2: Busy (communication disabled)   */
,  CLDR_ERR_PARA       /* 3: Parameter error                 */
};

/*============================================================================
;   Definition of constants
;
;   Settings in registers that control the LCD controller/driver
;   and main unit's control register
;============================================================================*/
/*====================================================================
;   Settings in control register for LCD chip's LCD control unit
;====================================================================*/
/*
;[Communication format]
;
;   Address                 Bit
;           7      6      5      4      3      2      1      0
;           +-------+-------+-------+-------+-------+-------+-------+-------+
; 00H LCDMD   |SEGSET2|SEGSET1|SEGSET0|   0   |   0   |   0   |MDSET1 |MDSET0 |
;           +-------+-------+-------+-------+-------+-------+-------+-------+
;         MDSET1/0   : Selects LCD reference voltage generator
;         SEGSET2-0  : Sets number of segments (fixed as 40)
;
;           +-------+-------+-------+-------+-------+-------+-------+-------+
; 01H LCDM    | LCDON | SCOC  | VLCON |   0   |   0   | LCDM2 | LCDM1 | LCDM0 |
;           +-------+-------+-------+-------+-------+-------+-------+-------+
;          LCDM2-0   : Selects LCD controller/driver display mode
;          VLCON     : Enables/disables operation of step-up circuit
;          SCOC      : Controls segment pins/common pins output
```

```
;          LCDON     : Enables/disables LCD display
;
;           +-------+-------+-------+-------+-------+-------+-------+-------+
; 02H LCDC   |  0  |  0  |  0  |  0  | LCDC3 | LCDC2 | LCDC1 | LCDC0 |
;           +-------+-------+-------+-------+-------+-------+-------+-------+
;         LCDC1/0    : Sets LCD clock
;         LCDC3/2    : Sets LCD source clock
;
;           +-------+-------+-------+-------+-------+-------+-------+-------+
; 03H VLCG0   |CTSEL1 |CTSEL0 |  0  |  0  |  0  |  0  |  0  | GAIN  |
;           +-------+-------+-------+-------+-------+-------+-------+-------+
;         GAIN       : Sets  reference voltage level
;         CTSEL1/0   : Selects contrast adjustment
;
;
; **Selects each register's settings from the following patterns.
;
;=====================================================================*/
/*---------------------------------------------------------------
;   LCD mode select register (register address: 00H)
;---------------------------------------------------------------*/
/*--- Selects LCD reference voltage generator ---*/

//  #define CLDR_LCDMD  0b00000000  /* <1> External resistor division mode */
//  #define CLDR_LCDMD  0b00000001  /* <2> Internal resistor division mode */
    #define CLDR_LCDMD  0b00000010  /* <3> Internal step-up mode          */
/*          76543210                          */
/*          -----000 ..... 0: Bit must be set      */
/*          |||||||                           */
/*          |||||||  **Settings are from patterns <1> to <3> above       */
/*          |||||||                                    */
/*          |||||++--    MDSET1/0    Sets LCD reference voltage generator */
/*          |||+++----    <000 fixed>                          */
/*          +++-------    <000 fixed> Selects number of segments (SEGSET2/1/0) */

/*---------------------------------------------------------------
;   LCD display mode register (register address:  01H)
;---------------------------------------------------------------*/
/*-- Selects LCD controller/driver display mode --*/
/*                                    */
/*                                |   Resistor    |    Step-up    | */
/*                                | division mode |      mode     | */
/*                                |  Time  | Bias |  Time  | Bias | */
/*                                |division|method|division|method| */
    #define CLDR_LCDM   0b11100000  /* <1> |   4    | 1/3  |   4    | 1/3  | */
//  #define CLDR_LCDM   0b11100001  /* <2> |   3    | 1/3  |   3    | 1/3  | */
//  #define CLDR_LCDM   0b11100010  /* <3> |   2    | 1/2  |   4    | 1/3  | */
//  #define CLDR_LCDM   0b11100011  /* <4> |   3    | 1/2  |   3    | 1/3  | */
//  #define CLDR_LCDM   0b11100100  /* <5> |     Static    |Set prohibited | */
/*          76543210                          */
/*          XXX--000..... 0: Bit must be set, */
/*          |||||||       X: Bit setting not required, control by software  */
/*          |||||||                                    */
/*          |||||||  **Settings are from patterns <1> to <5> above         */
/*          |||||||  **Bits 7 to 5 are controlled by software              */
/*          |||||||                                    */
/*          ||||+++--    LCDM2/1/0   Selects LCD controller/driver disp mode */
/*          |||++-----    <00 fixed>                           */
/*          ||+-------    VLCON (1 fixed) Enables/disables step-up circuit    */
/*          |+--------    SCOC  (1 fixed) Controls segment/common pins output */
/*          +---------    LCDON (1 fixed) Enables/disables LCD display        */

/*---------------------------------------------------------------
```

```
;   LCD clock  control register (register address:  02H)
;-------------------------------------------------------------------*/
/*-- Selects LCD source clock (fLCD) & LCD clock  --*/
/*                                        */
/*                                | LCD source | LCD clock  | */
/*                                |clock (fLCD)| LCD clock  | */
//  #define CLDR_LCDC   0b00000000  /* <1> | fPCL       | fLCD/2^6   | */
//  #define CLDR_LCDC   0b00000001  /* <2> | fPCL       | fLCD/2^7   | */
    #define CLDR_LCDC   0b00000010  /* <3> | fPCL       | fLCD/2^8   | */
//  #define CLDR_LCDC   0b00000011  /* <4> | fPCL       | fLCD/2^9   | */
//  #define CLDR_LCDC   0b00001000  /* <5> | fPCL/2     | fLCD/2^6   | */
//  #define CLDR_LCDC   0b00001001  /* <6> | fPCL/2     | fLCD/2^7   | */
//  #define CLDR_LCDC   0b00001010  /* <7> | fPCL/2     | fLCD/2^8   | */
//  #define CLDR_LCDC   0b00001011  /* <8> | fPCL/2     | fLCD/2^9   | */
//  #define CLDR_LCDC   0b00001100  /* <9> | fPCL/2^2   | fLCD/2^6   | */
//  #define CLDR_LCDC   0b00001101  /* <10>| fPCL/2^2   | fLCD/2^7   | */
//  #define CLDR_LCDC   0b00001110  /* <11>| fPCL/2^2   | fLCD/2^8   | */
//  #define CLDR_LCDC   0b00001111  /* <12>| fPCL/2^2   | fLCD/2^9   | */
/*          76543210                                    */
/*          ----0000 ..... 0: Bit must be set           */
/*          |||||||                                     */
/*          |||||||  **Settings are from patterns <1> to <12> above */
/*          |||||||                                     */
/*          |||||++--    LCDC1/0     Sets LCD clock     */
/*          ||||++----    LCDC3/2     Sets LCD source clock  */
/*          ++++------    <0000 fixed>                   */
/*-------------------------------------------------------------------
;   LCD step-up control register (register address:03H)
;-------------------------------------------------------------------*/
/*-- Selects reference voltage (VLC2) level & contrast adjustment (TYP. value) -*/
/*                                        */
/*                                |Reference|       Contrast       | */
/*                                | voltage |      adjustment      | */
/*                                | (VLC2)  |     (TYP. value)     | */
/*                                |level *1 |  VLC0 |  VLC1 |  VLC2  | */
    #define CLDR_VLCG0  0b10000000  /* <1> |1.5V     | 4.89V | 3.27V | 1.633V | */
//  #define CLDR_VLCG0  0b11000000  /* <2> |1.5V     | 4.71V | 3.13V | 1.567V | */
//  #define CLDR_VLCG0  0b00000000  /* <3> |1.5V     | 4.50V | 3.00V | 1.500V | */
//  #define CLDR_VLCG0  0b01000000  /* <4> |1.5V     | 4.29V | 2.87V | 1.433V | */
//  #define CLDR_VLCG0  0b10000001  /* <5> |1.0V     | 3.29V | 2.27V | 1.133V | */
//  #define CLDR_VLCG0  0b11000001  /* <6> |1.0V     | 3.21V | 2.13V | 1.067V | */
//  #define CLDR_VLCG0  0b00000001  /* <7> |1.0V     | 3.00V | 2.00V | 1.000V | */
//  #define CLDR_VLCG0  0b01000001  /* <8> |1.0V     | 2.79V | 1.87V | 0.933V | */
/*          76543210                                    */
/*          00-----0 ..... 0: Bit must be set           */
/*          |||||||                                     */
/*          |||||||  **Settings are from patterns <1> to <8> above  */
/*          |||||||                                     */
/*          ||||||+--    GAIN        Sets reference voltage level  */
/*          ||+++++---    <00000 fixed>                  */
/*          ++--------    CTSEL1/0    Selects contrast adjustment  */
/*                                                      */
/*  *1: The reference voltage level is selected 1.5 V when the target LCD panel */
/*      is rated at 4.5V, and is selected as 1.0 V when the target LCD panel is */
/*      rated at 3.0 V.                                 */
/*=================================================================
;   Definition of main control register settings
;=================================================================*/
/*-----------------------------------------------------------------
;    Selects clock output
;-------------------------------------------------------------------*/
/*-- Selects PCL's output clock --*/
/*   fSUB=32.768kHz             */
```

```
/*   fPRS=peripheral clock       */
/*                                    | fPRS=10MHz  | fPRS=20MHz | */
//  #define CLDR_CKS    0b00000110  /* <1> | fPRS/2^6 | 156.25kHz   | 312.5 kHz  | */
//  #define CLDR_CKS    0b00000111  /* <2> | fPRS/2^7 |  78.125kHz  | 156.25kHz  | */
    #define CLDR_CKS    0b00001000  /* <3> | fSUB     |  32.768kHz  |  32.768KHz | */
/*          76543210                                                  */
/*          ---X0000 ..... 0: Bit must be set                         */
/*          |||||||        X: Bit setting not required, control by software     */
/*          |||||||                                                   */
/*          |||||||  **Settings are from patterns <1> to <3> above             */
/*          |||||||  **Bit 4 is controlled by software                  */
/*          |||||||                                                   */
/*          ||||++++--    CCS3/2/1/0  Selects PCL's output clock               */
/*          |||+------    CLOE (0 fixed)  Enables/disables clock output to LCD   */
/*          +++-------    <000 fixed>                                 */
/*----------------------------------------< END OF FILE >----------*/
#endif /* _LCDDRVAPP_H */
```

4.21 LcdDrvApp.c

```
/*****************************************************************************
;
;   NNNNNN        NN  EEEEEEEEEEEEEEEEE    CCCCCCCCCCCCCCC
;   NNNNNNNN      NN  EEEEEE           CCCCCC
;   NNNNNNNNNN    NN  EEEEEE           CCCCCC
;   NN  NNNNNNNN  NN  EEEEEEEEEEEEEEEEE  CCCCCC
;   NN   NNNNNNNN NN  EEEEEE           CCCCCC
;   NN    NNNNNNNNNN  EEEEEE           CCCCCC
;   NN       NNNNNN   EEEEEEEEEEEEEEEE    CCCCCCCCCCCCCCC
;
;   NEC Electronics   78K0/Lx2
;
;*****************************************************************************
;   78K0/Lx2   LCD control sample program
;*****************************************************************************
;   LCD controller/driver control   -- Processing file--
;*****************************************************************************
;[History]
;   2005.06.--  Newly created
;   2005.07.01  [050701] Provisionally added voltage supply to VLC0
;   2006.01.11  Modified to use Applilet-generated IIC routines - rdh
;   2006.01.27  Correction in LcdDrvOff in turning off VLCON - rdh
;   2006.03.30  Use routines for writing only for Application Notes
*****************************************************************************/
#pragma sfr

/*===========================================================================
;   INCLUDE
;===========================================================================*/
#include    "macrodriver.h"
#include    "serial.h"
#include    "LcdDrvApp.h"

static void LcdDrvClkOut( void );
static void LcdDrvClkStop( void );

/*===========================================================================
;   Definition of control area for LCD driver and various settings
;===========================================================================*/
```

```
/*===========
;   Slave ID
;==========*/
#define     CSLV_ID_LCDCTL  0b01110000      /* Control register (LCDCTL) */
#define     CSLV_ID_LCDSEG  0b01110010      /* Segment data (LCDSEG)  */

/*===========
;   Definition of control register settings on main side
;==========*/
/*-- Clock output select register --*/
#define     LDR_CKS     CKS
#define    LDR_CKS_CLOE    LDR_CKS.4       /* Clock output enable/disable */

/*-- Port/port mode register (directly connected in microcontroller) --*/
#define     PO_LDR_RST  P13.0           /* Reset to LCD chip            */
#define     PM_LDR_OUT  PM14.0          /* Clock output to LCD chip     */

/*-- Port/port mode register --*/                        /*[050701]>>*/
#define     PO_VLC0_HL  P7.7            /* Voltage supply to VLC0 */
#define     PM_VLC0_HL  PM7.7           /* Voltage supply to LLC0 [050701] */

/*===========
  Control register setting
==========*/
/* Selects LCD reference voltage generator: internal step-up mode */
#define     CLDR_LCDMD_VOL  0b00000010

/*****************************************************************************
;   LCD driver initialization
;-----------------------------------------------------------------
;   [I N]   -
;   [OUT]   0= Setting OK, 1 = NACK received, 2 = Busy
;*****************************************************************************/
unsigned char LcdDrvInit( void )
{
    register unsigned char result = CLDR_ERR_NONE;

    PO_LDR_RST = 1;                 /* Cancels LCD chip's reset status  */
    LDR_CKS = ( CLDR_CKS & 0b00001111);  /* Output clock setting (CCS3-0) */

    /*-- Enables clock output to LCD chip  --*/
    LcdDrvClkOut();

    /*--  Selects reference voltage generator --*/
    result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDMD, CLDR_LCDMD );

    /*-- Clears segment data  --*/
    if( result == CLDR_ERR_NONE ){
        result = LcdDrvSegClr();
    }

    /*-- Selects display mode --*/
    if( result == CLDR_ERR_NONE ){
        result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b00000111 ));
    }

    /*-- LCD clock setting --*/
    if( result == CLDR_ERR_NONE ){
        result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDC, CLDR_LCDC );
    }
    return  ( result );
}
```

```
/*****************************************************************************
; LCD driver display ON wait start pre-processing (when step-up mode is selected)
;-----------------------------------------------------------------
;   << Note >>
;
; Only N is called when internal step-up mode is selected for the reference
; voltage generator.  After this processing is called, a wait period
; of at least 500 ms should occur, then the "LcdDrvOn" should be called.
;
;-----------------------------------------------------------------
;   [I N]   -
;   [OUT]   0= Setting OK, 1 = NACK received, 2 = Busy
;*****************************************************************************/
unsigned char LcdDrvOnWait( void )
{
#if (CLDR_LCDMD==CLDR_LCDMD_VOL)
    /*  Reference voltage generator:  internal step-up mode */
    register unsigned char result = CLDR_ERR_NONE;

    /*-- Enables clock output to LCD chip  --*/
    LcdDrvClkOut();

    /*-- Sets LCD step-up level and contrast--*/
    result = LcdDrvCtrWrite1Byte( CLDR_ADDR_VLCG0, CLDR_VLCG0 );

    /*-- Enables LCD step-up  --*/
    if( result == CLDR_ERR_NONE ){
        result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b00100111 ));
    }
    /* -- After 500 ms, the "LcdDrvOnWait" function must be called -- */
    return  ( result );
#else/*!(CLDR_LCDMD==CLDR_LCDMD_VOL)*/
    /*  Reference voltage generator:  resistor division mode */
    return  ( 0 );
#endif/*(CLDR_LCDMD)*/
}

/*****************************************************************************
;   LCD driver ON processing
;-----------------------------------------------------------------
;   << Note >>
;
; When internal step-up mode has been selected for the reference voltage
; generator, after the "LcdDrvOnWait" has been called, a wait period of
; at least 500 ms must occur before calling the next function.
;
;-----------------------------------------------------------------
;   [I N]   -
;   [OUT]   0= Setting OK, 1 = NACK received, 2 = Busy
;*****************************************************************************/
unsigned char LcdDrvOn( void )
{
    register unsigned char result = CLDR_ERR_NONE;

#if (CLDR_LCDMD==CLDR_LCDMD_VOL)
    /*  Reference voltage generator: internal step-up mode */
    /*-- Setting of deselect potential output --*/
    result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b01100111 ));

    /*-- Display ON setting --*/
    if( result == CLDR_ERR_NONE ){
        result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b11100111 ));
    }
```

```
#else/*!(CLDR_LCDMD==CLDR_LCDMD_VOL)*/
   /*  Reference voltage generator: resistor division mode */
   /*-- Enables clock output to LCD chip  --*/
   LcdDrvClkOut();

   /*-- Setting of deselect potential output --*/
   result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b01000111 ));

   /*-- Display ON setting --*/
   if( result == CLDR_ERR_NONE ){
      result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b11000111 ));
   }

#endif/*(CLDR_LCDMD)*/
   return  ( result );
}

/*********************************************************************
;   LCD driver display OFF processing
;---------------------------------------------------------------------
;   [I N]   -
;   [OUT]   0= Setting OK, 1 = NACK received, 2 = Busy
;
;   *Clears AX register
;*********************************************************************/
unsigned char LcdDrvOff( void )
{
   register unsigned char result = CLDR_ERR_NONE;

   /*-- Clears segment data  --*/
   result = LcdDrvSegClr();

   /*-- Display OFF setting --*/
   if( result == CLDR_ERR_NONE ){
      result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b01100111 ));
   }

   /*-- Segment/common buffer output disable setting --*/
   if( result == CLDR_ERR_NONE ){
      result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b00100111 ));
   }

#if (CLDR_LCDMD==CLDR_LCDMD_VOL)

   /*-- LCD step-up disable setting --*/
   if( result == CLDR_ERR_NONE ){
#if 0   /* correction 060127 - bit 5 is 1, does not turn off VLCON */
      result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b00100111 ));
#else   /* correction turns off VLCON by having bit 5 as zero */
      result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b00000111 ));
#endif
   }

#endif/*(CLDR_LCDMD)*/

   if( result == CLDR_ERR_NONE ){
      /*-- Disables clock output to LCD chip --*/
      LcdDrvClkStop();
   }
   return  ( result );
}
```

```
/*********************************************************************
;   Writes LCD driver control data
;---------------------------------------------------------------------
; [I N]   src : Control register data's storage address
;     addr    : Control register address value
;     size    : Number of bytes to be transmitted (4 bytes maximum)
; [OUT]   0= Setting OK, 1 = NACK received, 2 = Busy, 3 = Parameter error
;*********************************************************************/
unsigned char SLDRCTLW( unsigned char *src,
                    unsigned char addr, unsigned char size )
{
   register unsigned char result = CLDR_ERR_NONE;
   register unsigned char cnt;
   MD_STATUS status;
   unsigned char buf[5];
   unsigned char uc;

   /*-- Enables clock output to LCD chip --*/
   LcdDrvClkOut();

   /*-- Checks parameters   --*/
   if(( size == 0 )||( addr > 0x03 )||(( 0x03+1 - addr ) < size )){
      result = CLDR_ERR_PARA;
   }
   buf[0] = addr;
   for (uc = 0; uc < size; uc++) {
      buf[uc+1] = src[uc];
   }

   status = IIC0_MasterStartAndSend( CSLV_ID_LCDCTL, buf, size + 1 );
   if (status != MD_OK)
      result = CLDR_ERR_NACK;
   return  ( result );
}

/*********************************************************************
;   Writes LCD driver segment data
;---------------------------------------------------------------------
; [I N]   src : Address of segment data to be written
;     addr    : Data address for start of write operation
;     size    : Number of bytes to be transmitted (maximum:  20 bytes)
; [OUT]   0= Setting OK, 1 = NACK received, 2 = Busy, 3 = Parameter error
;---------------------------------------------------------------------
;   ** Smooths data before transmitting segment data.
;
;   <Received data>
;                bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
;               +----+
;    1st byte (00H)|COM3|COM2|COM1|COM0|COM3|COM2|COM1|COM0|
;               +----+----+----+----+----+----+----+----+
;               |<-  segment:S1   ->|<-  segment:S0   ->|
;
;               +----+----+----+----+----+----+----+----+
;    2nd byte (01H)|COM3|COM2|COM1|COM0|COM3|COM2|COM1|COM0|
;               +----+----+----+----+----+----+----+----+
;                 :
;      :          :
;               +----+----+----+----+----+----+----+----+
;   19th byte (12H)|COM3|COM2|COM1|COM0|COM3|COM2|COM1|COM0|
;               +----+----+----+----+----+----+----+----+
;               +----+----+----+----+----+----+----+----+
;   20th byte (13H)|COM3|COM2|COM1|COM0|COM3|COM2|COM1|COM0|
;               +----+----+----+----+----+----+----+----+
```

```
;               |<-  segment:S39  ->|<-  segment:S38  ->|
;
;   <Sent data>
;                bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
;               +----+----+----+----+----+----+----+----+
;     1st byte(00H)|  0 |  0 |  0 |  0 |COM3|COM2|COM1|COM0|
;               +----+----+----+----+----+----+----+----+
;                                |<-  segment:S0   ->|
;                                   Address:00H    |
;
;               +----+----+----+----+----+----+----+----+
;     2nd byte (01H)|  0 |  0 |  0 |  0 |COM3|COM2|COM1|COM0|
;               +----+----+----+----+----+----+----+----+
;                :
;                :
;               +----+----+----+----+----+----+----+----+
;    39th byte (26H)|  0 |  0 |  0 |  0 |COM3|COM2|COM1|COM0|
;               +----+----+----+----+----+----+----+----+
;               +----+----+----+----+----+----+----+----+
;    40th byte (27H)|  0 |  0 |  0 |  0 |COM3|COM2|COM1|COM0|
;               +----+----+----+----+----+----+----+----+
;                                |<-  segment:S39  ->|
;                                |    Address:27H    |
;
;*********************************************************************/
unsigned char LcdDrvSegWrite( unsigned char *src,
                        unsigned char addr, unsigned char size )
{
   register unsigned char result = CLDR_ERR_NONE;
   register unsigned char cnt;
   MD_STATUS status;
   unsigned char buf[41];
   unsigned char uci,ucd;

   /*-- Enables clock output to LCD chip --*/
   LcdDrvClkOut();

   /*-- Checks parameters   --*/
   if(( size == 0 )||( addr > 0x13 )||(( 0x13+1 - addr ) < size )){
      result = CLDR_ERR_PARA;
   }

   buf[0] = addr*2;
   for (uci = 0, ucd = 1; uci < size; uci++, ucd = ucd + 2) {
      buf[ucd]   = src[uci] & 0x0f ;
      buf[ucd+1] = (src[uci] >> 4) & 0x0f;
   }

   status = IIC0_MasterStartAndSend( CSLV_ID_LCDSEG, buf, (size * 2) + 1 );
   if (status != MD_OK)
      result = CLDR_ERR_NACK;
   return  ( result );
}

/*********************************************************************
;   Clears LCD driver segment data
;---------------------------------------------------------------------
; [I N]   -
; [OUT]   0= Setting OK, 1 = NACK received, 2 = Busy
;*********************************************************************/
unsigned char LcdDrvSegClr( void )
{
   register unsigned char result = CLDR_ERR_NONE;
```

```
    register unsigned char cnt;
    MD_STATUS status;
    unsigned char buf[41];
    unsigned char uc;

    /*-- Enables clock output to LCD chip --*/
    LcdDrvClkOut();

    buf[0] = 0; // start at location zero
    for (uc = 1; uc < 41; uc++) {
        buf[uc] = 0;
    }

    status = IIC0_MasterStartAndSend( CSLV_ID_LCDSEG, buf, 41 );
    if (status != MD_OK)
        result = CLDR_ERR_NACK;
    return  ( result );
}

/*****************************************************************************
;   Writes to LCD driver control register (1 byte setting)
;----------------------------------------------------------------
; [I N]   addr    : control register  address value
;         data    : control register  data
; [OUT]   0= Setting OK, 1 = NACK received, 2 = Busy, 3 = Parameter error
;*****************************************************************************/
unsigned char LcdDrvCtrWrite1Byte( unsigned char addr, unsigned char data )
{
    register unsigned char result = CLDR_ERR_NONE;
    MD_STATUS status;
    unsigned char buf[2];
    unsigned char uc;

    /*-- Enables clock output to LCD chip --*/
    LcdDrvClkOut();

    buf[0] = addr;
    buf[1] = data;

    status = IIC0_MasterStartAndSend( CSLV_ID_LCDCTL, buf, 2 );
    if (status != MD_OK)
        result = CLDR_ERR_NACK;
    return  ( result );
}

/*****************************************************************************
;   Writes LCD driver segment data (1 byte setting)
;----------------------------------------------------------------
; [I N]   addr    : Control register address value
;         data    : control register  data
; [OUT]   0= Setting OK, 1 = NACK received, 2 = Busy, 3 = Parameter error
;*****************************************************************************/
unsigned char LcdDrvSegWrite1Byte( unsigned char addr, unsigned char data )
{
    register unsigned char result = CLDR_ERR_NONE;
    register unsigned char work;
    MD_STATUS status;
    unsigned char buf[5];
    unsigned char uc;

    /*-- Enables clock output to LCD chip --*/
    LcdDrvClkOut();
```

```
   buf[0] = addr;
   buf[1] = data & 0x0f;
   buf[2] = ( data >>4 ) & 0x0f;

   status = IIC0_MasterStartAndSend( CSLV_ID_LCDSEG, buf, 3 );
   if (status != MD_OK)
      result = CLDR_ERR_NACK;
   return  ( result );
}

/*****************************************************************************
;   Enables clock and power output to LCD chip
;-----------------------------------------------------------------
; [I N]   -
; [OUT]   -
;*****************************************************************************/
static void LcdDrvClkOut( void )
{
   PM_LDR_OUT   = 0;
   LDR_CKS_CLOE = 1;
#if (CLDR_LCDMD==CLDR_LCDMD_VOL)                    /*[050701]>>*/
   PO_VLC0_HL   = 0;   /* Low output (power is supplied to VLC0)*/
#else /*!(CLDR_LCDMD==CLDR_LCDMD_VOL)*/
   PO_VLC0_HL   = 1;   /* High output (power is not supplied to VLC0)*/
#endif/*(CLDR_LCDMD)*/                              /*[050701]<<*/
}

/*****************************************************************************
;   Disables clock and power output to LCD chip
;-----------------------------------------------------------------
; [I N]   -
; [OUT]   -
;*****************************************************************************/
static void LcdDrvClkStop( void )
{
   LDR_CKS_CLOE = 0;
   PM_LDR_OUT   = 1;
   PO_VLC0_HL   = 0;                        /*[050701]*/
}

/*-----------------------------------------------< END OF FILE >---*/
```