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

MCU Serial Communications

Using the UART, LIN, CSI, and IIC Interfaces in NEC Electronics Microcontrollers

Document no. U18290EU1V0AN00

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 application note provides examples of the use of peripherals included in NEC Electronics microcontrollers. This application note includes:

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

The Applilet is a software tool that generates driver code for the peripherals, providing a convenient means of generating code for quick evaluation.

Please see the user manuals for your target microcontrollers for further details.

1.1 Overview of Serial Communication

Serial-communication protocols are built into most NEC Electronics microcontrollers. Many devices offer serial-communication interfaces using a universal asynchronous receiver transmitter (UART), Inter-Integrated Circuit Communication (IIC), Local Interconnect Network (LIN) or clocked serial I/O (CSI), also known as a 3-wire serial interface. All of these protocols move data quickly, in a simple manner, from one device to another. Each serial-communication protocol uses unique communication methods. This application note demonstrates each serial-communication method.

The UART is one of the most commonly used serial-communication protocols. Most NEC Electronics microcontrollers implement one or two UART channels. UARTs use separate transmit- and receive-data lines, allowing both transmission and reception to occur simultaneously—offering full-duplex operation. UARTs most commonly operates in asynchronous mode, communicating with a PC serial port using RS232 protocol.

IIC is a synchronous, master/slave protocol that allows a master device to initiate communication with one or more slave devices. Since IIC has separate clock and data lines, the clock can change without disrupting data. The data rate can increase or decrease according to the selected clock rate. The master device controls clock and data, and synchronizes data transfers with the clock. IIC is bidirectional, so data can flow in any direction on the IIC bus under the control of a master device. IIC enables low-cost implementations and is typically used for writing to EEPROM or peripheral-to-peripheral communications.

The Local Interconnect Network (LIN) is a low-speed, low-cost serial-bus protocol. LIN is primarily intended to connect sensors and actuators on a sub-bus to a main bus such as a Controller Area Network (CAN). A LIN network consists of one master and one or more slave nodes. The master, in single-master, multiple-slave configuration, controls the medium access. LIN uses single-wire communication, further reducing cost.

Clocked serial I/O uses three lines: Serial Clock (SCK), Data-Input (SI) and Data-Output (SO). In some cases, an additional line provides a handshake between the master and slave to provide simultaneous transmission and reception. Data transmission and reception is synchronized with the clock, making communication straightforward.

2. Serial Communication Using UART6

Almost all NEC Electronics microcontrollers incorporate one or more UART channels to provide standard asynchronous serial communication with personal computers, modems, and other devices. In Kx2 series MCUs, UART6 supports LIN functions.

When you do not require the UART pins for serial communications, they can serve as general-purpose I/O pins. Setting the appropriate control registers selects alternate functions.

The sample program demonstrates simple transmission and reception. It also shows how to initialize various registers for serial communication and set the appropriate baud rates. (For the interfaces described here, baud rate and data rate are the same.)

2.1 UART6 Features

The UART peripherals in NEC Electronics microcontrollers provide the following features:

- Full-duplex operation, with independent transmit and receive lines
- Dual transmit registers, for efficient continuous data transmission
- Selectable format for number of data bits (7 or 8)
- Selectable parity bit (none, mark (1), space (0), even, or odd)
- Selectable number of stop bits (1 or 2)
- Selectable LSB first (standard) or MSB first for data
- Available inversion of transmitted data
- Automatic generation of break state, with adjustable width
- Automatic recognition of break state (used in LIN operation)
- Error detection on received data (parity error, framing error, overrun error)
- On-board baud-rate generation by dividing the peripheral clock
- Support of standard and non-standard baud rates through flexible division of the peripheral clock
- Interrupt-based handling of transmit, receive, or reception error for program efficiency

The UART typically communicates with a personal computer or other devices through the RS232 interface. The RS232 interface uses voltage levels that differ from those of the microcontroller, so an RS232 transceiver converts the voltage levels. Many RS232 transceivers implement the handshake signals Request-To-Send (RTS) and Clear-To-Send (CTS). You can use the microcontroller’s port signals if the application requires these handshake signals

Figure 1. Simplified UART Block Diagram

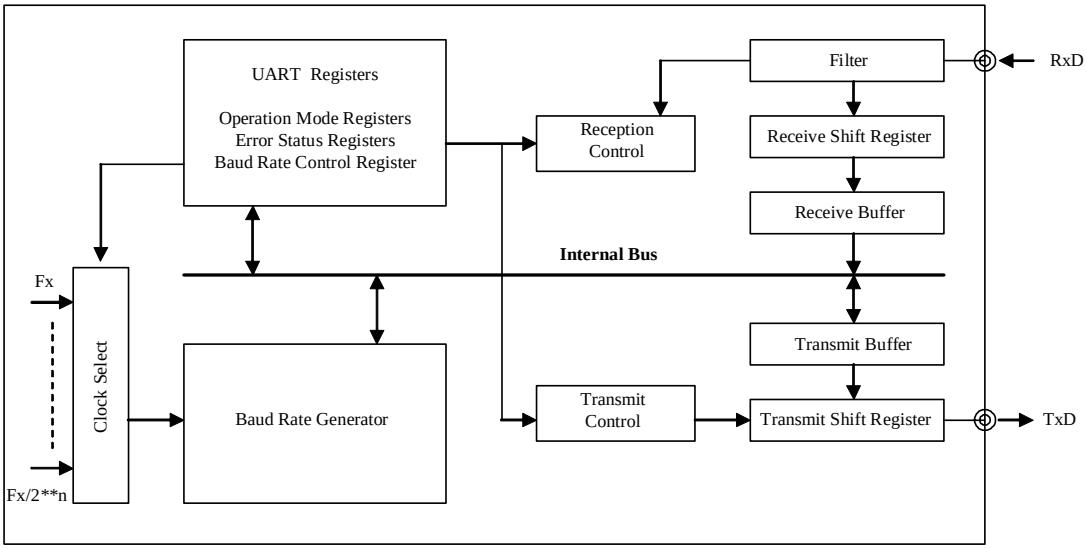


Table 1. UART Control Registers

Registers	Functions
Operation-Mode Register	Enable/Disable Clock, Reception, and Transmission Specify Parity, Receive/Transmit Data Length, Start/Stop Bits
Reception-Error Register	Status Flag indicating Parity, Framing, and Overrun errors for Reception
Transmit-Status Register	Status of Transmit Buffer Data
Clock-Selection Register	Selects Base Clock for UART
Baud-Rate Generator Control Register	Selects baud rate Generator Clock
Receive/Transmit Control Register	Synch Break Field (SBF) Reception Status (see note) SBF Transmit Width Control Sets MSB-First or LSB-First

Note: SBF is used for LIN communication protocol

To configure the UART6 peripheral:

- Set the port-mode register bits for input/output mode on the appropriate RXD6 and TXD6 pins, and set the port-output latch for the TXD6 pin.
- Set the number of data bits, parity, and number of stop bits using the ASIM6 register.
- Set LSB/MSB first and inverted/non-inverted TXD6 output using the ASICL6 register
- Define the baud rate by setting the clock source (CKSR6) and baud rate divider (BRGC6) registers.
- Set the priority for the receive error, receive, and transmit interrupts, clear the interrupt flags, and unmask the interrupts.

- Turn on the UART6 interface by setting the POWER6 bit in the ASIM6 register.
- Enable the transmitter and receiver by setting the TXE6 and RXE6 bits in the ASIM6 register.

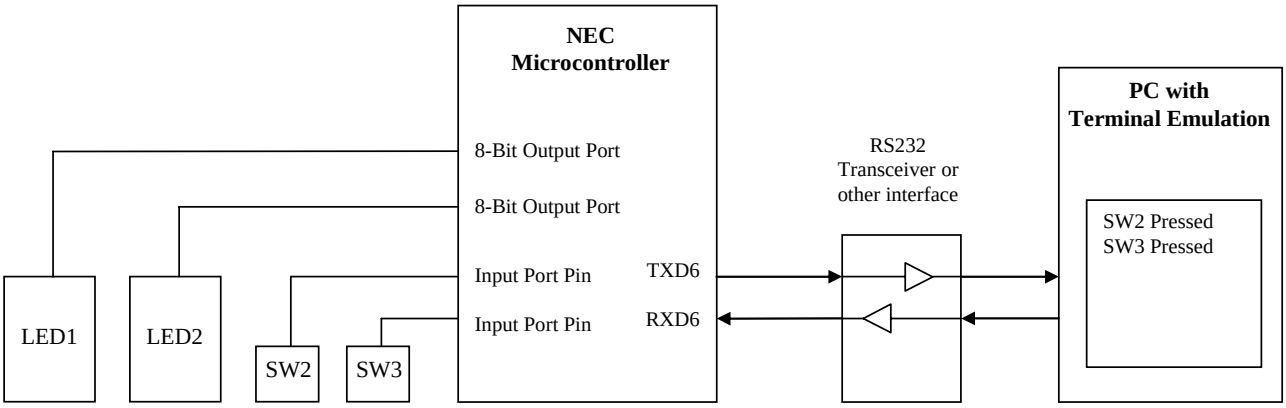
The UART6 peripheral is now ready. If it receives a character or sees a reception error, it generates an interrupt. The UART stores the error status in the ASIS6 register and received data in the RXB6 register.

To transmit a character, write data to the TXB6 register. When the transmission completes, an interrupt occurs. The UART can now write the next character. For greater speed, UART6 supports a dual-transmit-register structure. This means that once you write data to the TXB6 register, the data transfers to the TXS6 shift register for transmission. After the transfer, you can write a new character to the TXB6 register for immediate transmission.

2.2 Program Description and Specification

The demonstration program shows UART6 communication between an NEC Electronics microcontroller and a personal computer or other terminal. Two pushbutton switches and two-digit 7-segment LED displays connect to the port pins of the microcontroller. The UART6 Port of the microcontroller connects to an RS232 transceiver or other communication interface, which connects to a personal computer running a terminal emulation program.

Figure 2. UART Communication Block Diagram



When you press one of the switches, the microcontroller detects the switch input and transmits a message to the PC, which displays the message. The messages are:

- SW2 “SW2 Pressed”
- SW3 “ SW3 Pressed” (offset to make difference plainer)

To demonstrate the receive function, type any character on the PC keyboard. The microcontroller receives the character on the RXD6 line, reads the character data, and displays the hexadecimal code value of the character on the two LED displays.

If the microcontroller detects an error in a received character (framing error, parity error), or if it receives a second character before it reads the first character from the serial buffer (overrun error), the display shows "E" plus an error code indicating the error type.

Specifications:

- UART6 set for 9600 baud rate, generated from a 20-MHz external crystal
- Data LSB sent first, with one start bit, 8 data bits, no parity, and 1 stop bit
- Transmission-complete interrupts initiate transmission of the next character
- Handles receive-character and reception-error interrupts separately

2.3 Software Flow Charts

The demonstration program consists of the following major sections related to UART6 operation:

- Program-initialization code, including UART6 initialization, called before the main() program starts
- The main program loop, which checks switches and transmits data through the UART, checks for received data and displays values
- Subroutines for sending and receiving UART6 data, and interrupt-service routines related to UART6 operation

The program also includes sections not related directly to UART6 operation:

- Subroutines generated by the Applilet for port initialization for switches and LED display
- Subroutines generated by the Applilet to handle timer starting, stopping, and interval setting
- Switch-debounce subroutines with user code for handling timer interrupts
- Subroutines for switch input and LED display output

The flowcharts that follow describe initialization, the main program, and subroutines related to UART6 operation.

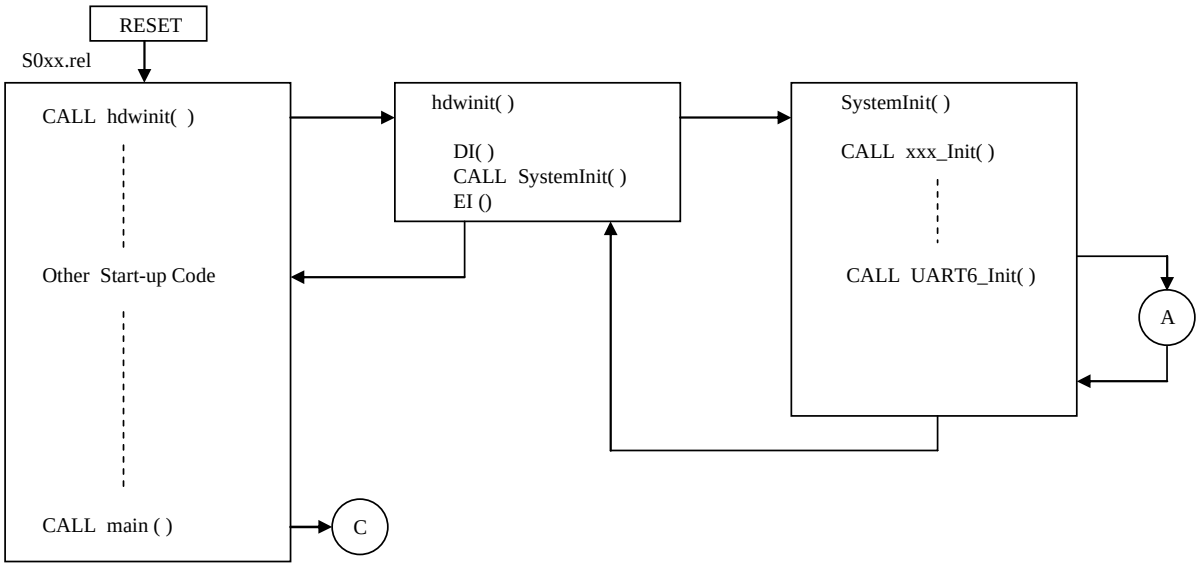
This document does not include flowcharts for timer and port initialization, switch input, LED display output, and timer-related subroutines and interrupt-service routines; however, the software listings include this code.

2.3.1 Program Startup and Initialization

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

When the Applilet generates a C program for the 78K0, the tool 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.

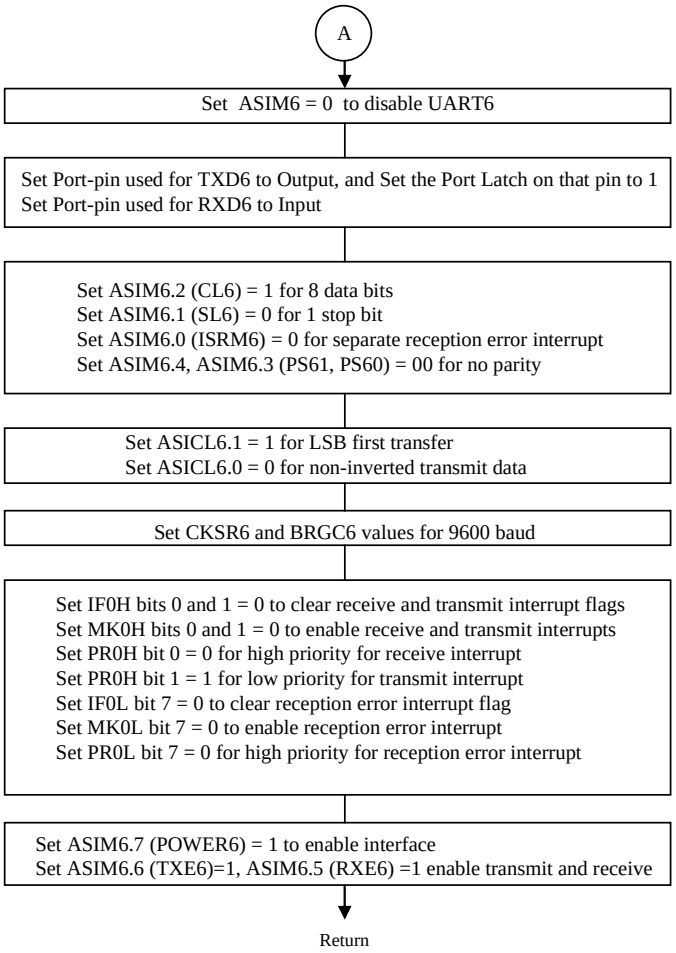
Figure 3. Flowchart for Program Initialization



After the hdwinit() function finishes, the startup code calls the main() function of the user program. So when the main() function starts, all initialization is complete, and the main() function does not need to call individual peripheral-initialization routines.

2.3.2 UART6_Init( ) – UART6 Initialization, Control Registers and Baud Rate Settings

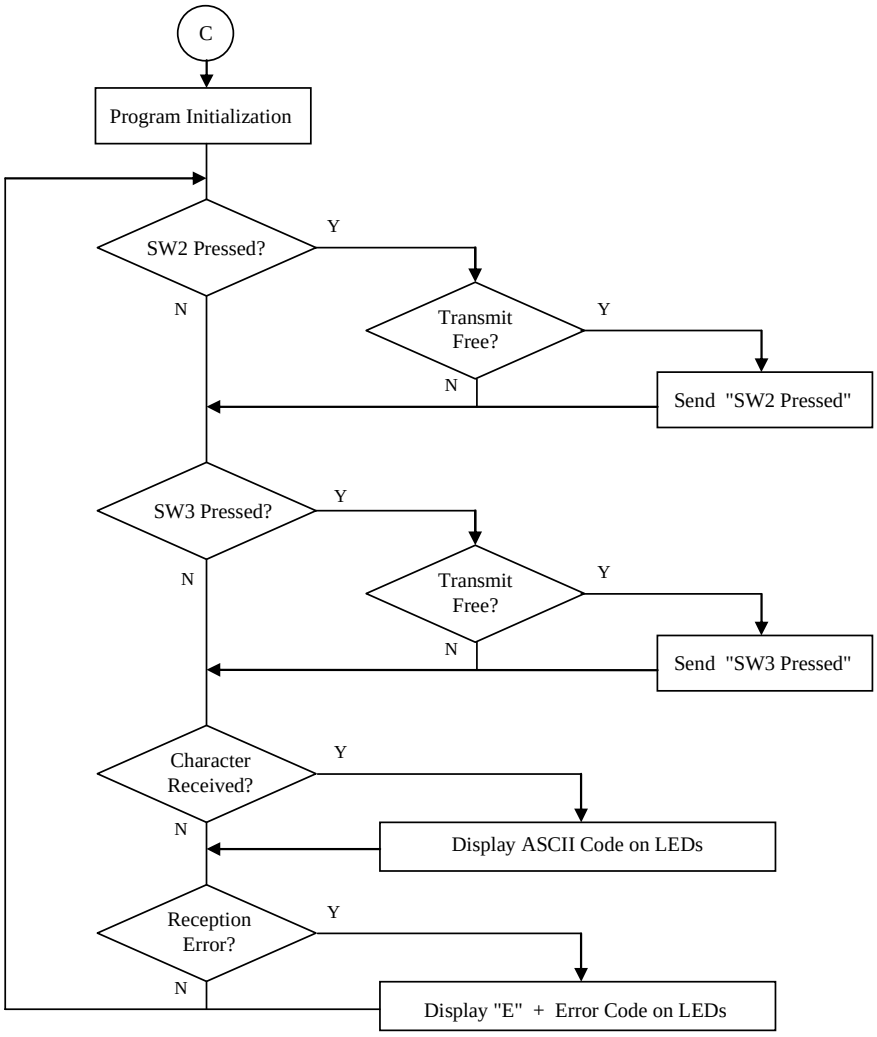
Figure 4. Flowchart for UART6 Initialization



Calling the UART6_Init() function initializes the UART6 peripheral. The demonstration program configures the UART6 for 8 data bits, one stop bit, no parity, LSB transmission first, non-inverted output, separate interrupt for receive errors, and a 9600 baud transmit and receive rate. The routine also initializes the port pins the UART uses, and sets the interrupt flag, mask, and priority registers to enable interrupts for transmit-complete, received-character, or reception errors.

2.3.3 Main( ) – Main Program – UART6 Operation

Figure 5. Flowchart for Main Program



The main() program initializes handling of the switches and LED display, sets up for UART6 data reception by calling UART6_ReceiveData(), and enters the main program loop.

In the main loop, Main() checks the state of the switches, taking an action when you press one of the two switches. If you press SW2, main() checks the transmit-free flag to see if a transmission is in process. If not, the routine calls UART6_SendData() to send the string “SW2 Pressed.” If you press SW3, the routine sends the message “SW3 Pressed.”

When a UART interrupt occurs (indicating character reception or a reception error), the interrupt-service routine sets the appropriate flag. Main() checks the status of these reception flags. If they indicate character

reception, the controller displays the character's ASCII value on the LED display. If they indicate a reception error, the LEDs display an "E" plus the error code.

After receiving and processing a character, main() sets up for the next data reception by calling UART_ReceiveData().

2.4 Applilet's Reference Drivers

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

The Applilet produces the basic initialization code and main function for the program, driver code for the UART, as well as code to manage the I/O ports used for switch input and display output. After the Applilet produces the basic code, adding additional code customizes the program.

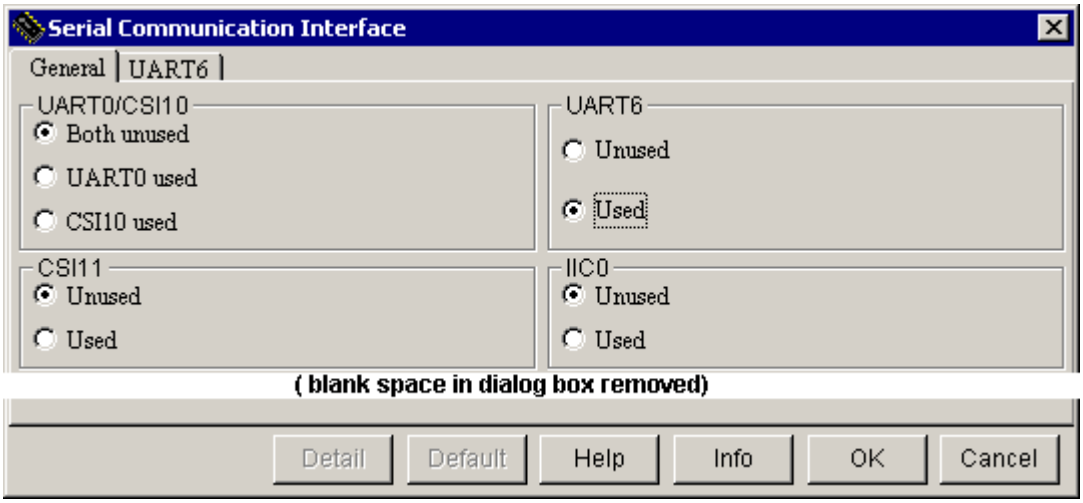
This section describes how to set up the Applilet to produce code for the UART and lists the routines produced.

When you start the Applilet and select the target device, save your settings to a new project (.prx) file. The Applilet displays a dialog box that lets you select the peripheral blocks you want to set up.

2.4.1 Configuring Applilet for UART

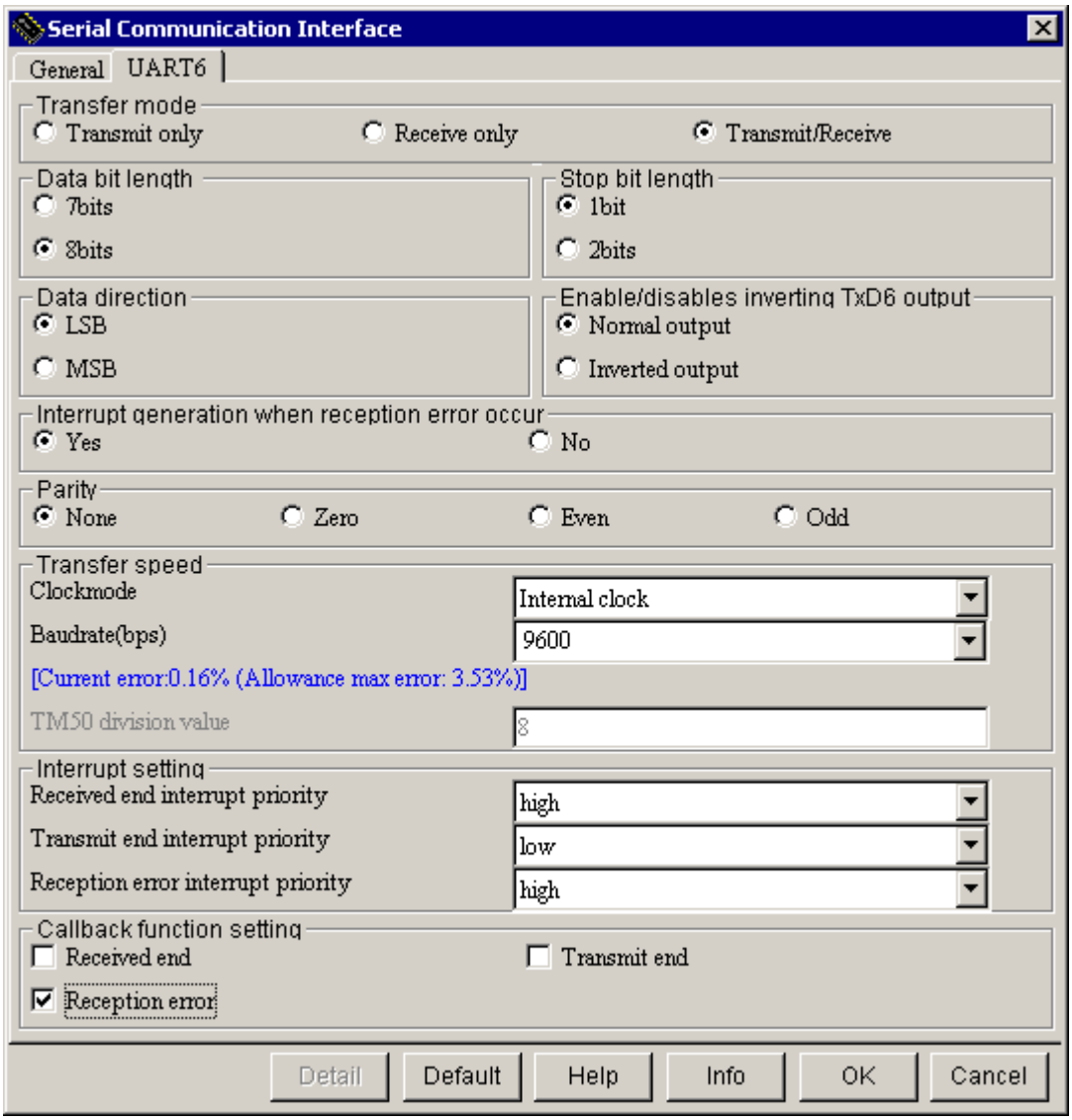
When running the Applilet, selecting **Serial Communications Interface** brings up a dialog box showing the various serial blocks. Select **UART6** as **Used**.

Figure 6. Serial Communication Dialog



Once you have selected **UART6**, click **Detail** to see the UART6 settings.

Figure 7. UART Dialog



Select **Transmit/Receive** for the **Transfer mode**, as the demonstration requires both transmit and receive functions. Set **Data Bit Length** to 8, **Stop bit** length to 1, **Data Direction** to LSB transmitted first, **Enable/disables inverting TxD6 output** to normal (non-inverted) output. Check **Yes** for **Interrupt generation,** and set **Parity** to **None**. Set **Clockmode** to **Internal clock** and **Baudrate** to **9600**.

UART6 generates interrupts on transmit-character end, received character, and receive error. The Applilet creates interrupt-service routines for these interrupts. You can choose to have receive errors generate a

separate interrupt or generate the same interrupt as received characters. Selecting separate interrupts causes the Applilet to create a separate interrupt-service routine.

In the dialog box, select high priority for receive and reception errors, and low priority for transmit. Normally you want to make sure that no received character is dropped, so reception takes higher priority than transmission.

Under **Callback function setting,** select **Reception error** to provide hooks for user code when a reception error occurs. The program does not require callback functions for transmit or receive.

2.4.2 Generating Code with Applilet

Selecting **Generate code** causes the Applilet to show a list of peripherals and functions, and lets you select a directory for storing the source code.

When you click **Generate,** 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 UART6 serial device, 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.

2.4.3 Applilet-Generated UART Files and Functions

The files serial.c, serial_user.c, and serial.h contain the code generated for UART6 support.

2.4.3.1 Serial.h

The header file serial.h contains definitions for the UART6 functions and some definitions of values for UART6 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.

2.4.3.2 Serial.c

The source file serial.c contains the following functions generated by the Applilet:

void UART6_Init(void)
This code initializes the UART6 peripheral as specified in the Applilet UART6 dialog. The routine calls UART6_User_Init() after the peripheral is initialized. Note void UART6_Init(void) requires

slight modification to correct an error in the Applilet-generated code for setting interrupt registers related to the INTSRE6 reception-error interrupt.

MD_STATUS UART6_ReceiveData(UCHAR* rxbuf, UCHAR rxnum)
This routine sets up a receive operation, requesting rxnum characters be received to the rxbuf buffer.

MD_STATUS UART6_SendData(UCHAR* txbuf, UCHAR txnum)
This routine sets up transmit of txnum characters from the txbuf buffer and starts the transmission operation by sending the first one or two characters.

__interrupt void MD_INTSR6(void)
This interrupt-service routine for the receive interrupt INTSR6 checks for reception errors (in case INTSRE6 is using the same vector). The routine puts the received character in the rxbuf buffer and decrements the receive count.

__interrupt void MD_INTST6(void)
This interrupt-service routine for the transmit interrupt INTST6 ignores the interrupt if transmission is complete. Otherwise, the UART sends the next character and decrements the transmit count.

__interrupt void MD_INTSRE6(void)

The Applilet creates this interrupt-service routine for the reception-error interrupt INTSRE6 if you select the option in the UART6 dialog. The routine reads the offending character for possible analysis and detects the error type, then calls the callback function CALL_UART6_Error() to report the error.

2.4.3.3 Serial_user.c

The source file Serial_user.c contains stub functions for user code. These functions are empty on code generation; you add application-specific code.

void CALL_UART6_Error(UCHAR err_type)
This routine is called with the error type when a reception error occurs. The user program needs to take appropriate action. The demonstration program has code to set a global error flag and save the error type for display.

2.4.4 Applilet-Generated Files Not Related to UART

For the demonstration program, the Applilet generates several other files.

Table 2. Applilet-Generated 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
Port.h	Definitions for default I/O port states
Port.c	PORT_Init() function
Timer.h	Definitions for timer settings and functions
Timer.c	Timer-management functions
Timer_user.c	Timer interrupt ISR (debounces pushbuttons)
Option.inc	Option-byte, POC, and security definitions
Option.asm	Option-byte, POC, and security data

2.4.5 Demonstration Program Files Not Generated by Applilet

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

Table 3. Files Not Generated by Applilet

File	Function
Sw_0537.h	Header file for pushbutton switch input
Sw_0537.c	Code to read and debounce pushbutton switches
Led_0537.h	Header file for seven-segment LED patterns and functions
Led_0537.c	Code to display data in seven-segment LED displays

2.5 Demonstration Platform

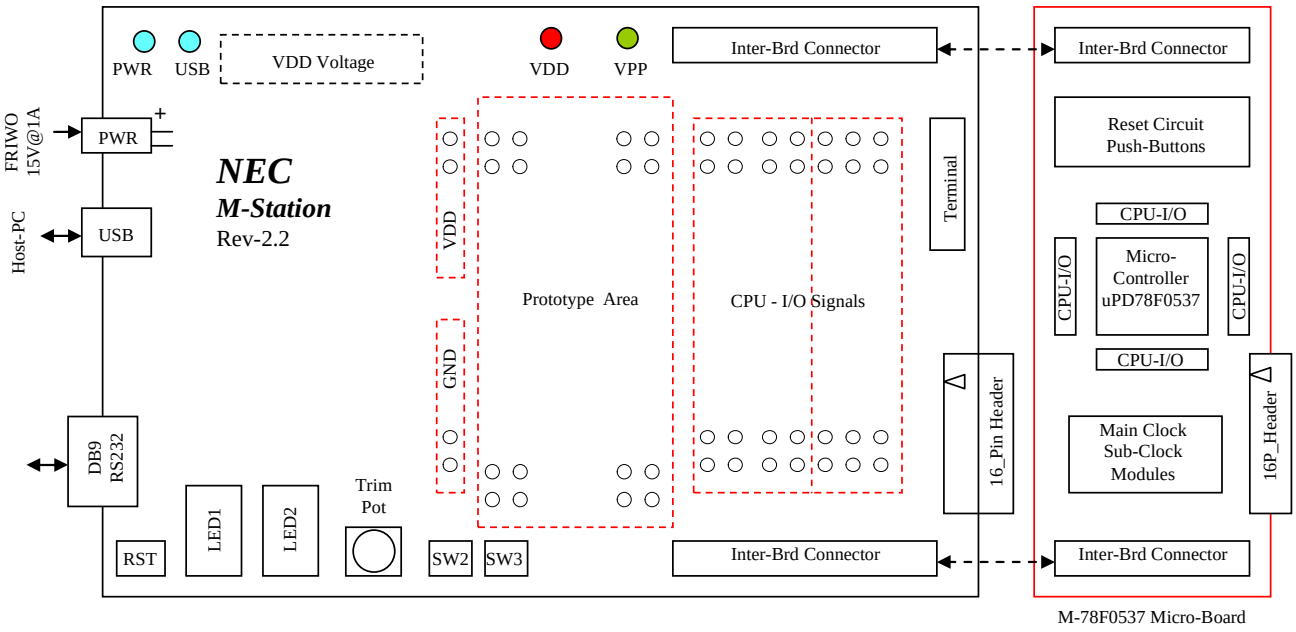
The demonstration uses a development board from NEC Electronics. You may be able to duplicate the hardware with off-the-shelf components along with the NEC Electronics microcontroller of interest.

2.5.1 Resources For UART6 Demonstration

The UART6 demonstration program uses the following resources:

- M-78F0537 Micro-Board, with uPD78F0537 8-bit microcontroller mounted
- uPD78F0537 resources:
  - UART6 peripheral, using P13/TXD6 and P14/RXD6
  - External 20-MHz crystal connected to X1 and X2 pins
- M-Station II Evaluation System, using M-Station II resources:
  - 7-Segment LED displays LED1 and LED2
  - Pushbutton switches SW2 and SW3
  - Connection of TXD6 and RXD6 to PC through USB interface
- Personal computer, using PC resources:
  - USB port for connection to M-Station II
  - MSTTERM program for terminal emulation through M-Station II

Figure 8. Demonstration Platform



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

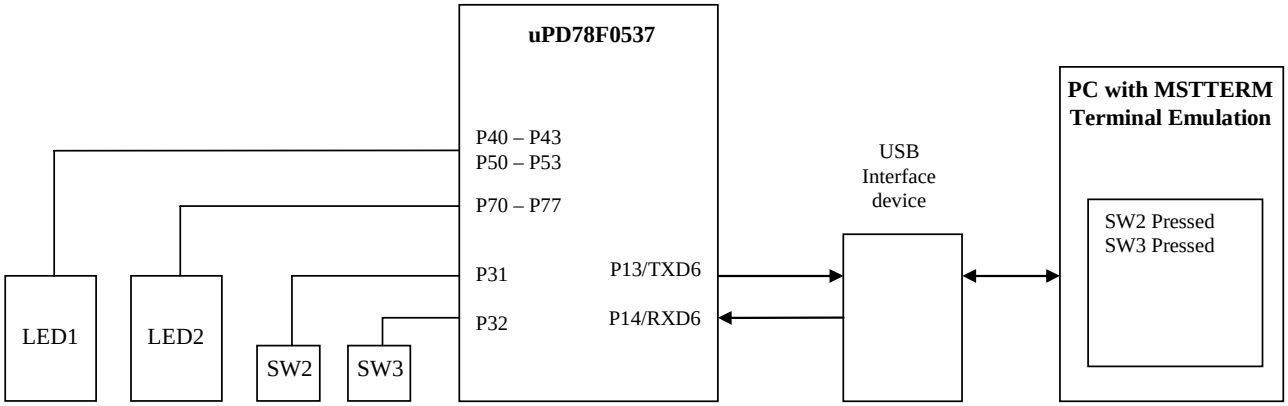
2.5.2 Demonstration of Program

With the hardware configured and the uPD78F0537 microcontroller programmed with the demonstration code, the demonstration is as follows:

- Press SW2 on M-Station II: PC displays “SW2 Pressed”
- Press SW3 on M-Station II: PC displays “ SW3 Pressed”
- Type “a” on PC keyboard M-Station II LED display shows “61” (ASCII code for lowercase a)
- Type “b” on PC keyboard M-Station II LED display shows “62” (ASCII code for b)

2.6 Hardware Block Diagram

Figure 9. Hardware Block Diagram



When you connect the M-78F0537 to the M-Station, pins of the uPD78F0537 connect to M-Station hardware, as shown below.

Table 4. M-Station Connections

M-Station-II Resource	uPD78F0537 Pins	Function
RXD_SI input	P13/TXD6	UART6 transmitted data output
TXD_SO output	P14/RXD6	UART6 received data
SW2 pushbutton	P31	Low when SW2 pressed
SW3 pushbutton	P32	Low when SW3 pressed
LED1 segments A through D	P40 – P43	On when P4 pin driven low
LED1 segments E through DP	P50 – P53	On when P5 pin driven low
LED2 segments A through DP	P70 – P77	On when P7 pin driven low

The P13/TXD6 and P14/RXD6 pins of the uPD78F0537 microcontroller connect to the RXD_SI and TXD_SO signals of the USB interface device on the M-Station II. You can access this UART communication channel through the MSTTERM program on the PC, which sends and receives UART

serial data through the USB interface. This is also the default channel that the M-Station II uses for flash programming of the uPD78F0537 microcontroller.

An alternative available on the M-Station II is to connect the P13/TXD6 and P14/RXD6 signals to an RS232 transceiver mounted on the M-Station II. Then you can use any terminal emulation program on the PC to communicate with UART6 on the uPD78F0537. Please consult the M-Station II user manual for further details.

2.7 Software Modules

The files shown in the table below make up the software modules for the demonstration programs. The table indicates which files the Applilet generated and which of those need modification to create the demonstration programs.

The listings for these files are in the Appendix.

Table 5. Software Modules in Demonstration Program

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
Serial.h	UART6-related definitions	Yes	Yes[Note 1]
Serial.c	UART6-related functions	Yes	Yes[Note 1]
Serial_user.c	User code for CSI callback routine	Yes	Yes[Note 1]
Port.h	Definitions for default I/O port states	Yes	No
Port.c	PORT_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
Timer_user.c	User code for timer-interrupt handling	Yes	Yes[Note 2]
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No
Led_0537.h	LED-display definitions	No	--
Led_0537.c	LED-display functions	No	--
Sw_0537.h	Switch-input definitions	No	--
Sw_0537.c	Switch-input functions	No	--

Note 1: Modify Serial.h to make some variables related to UART6 operation available to the main program. You need to modify Serial.c slightly to correct an error in the Applilet-generated code for initializing interrupt registers for the INTSRE6 reception-error interrupt. Serial_user.c requires modification to have the CALL_UART6_Error() callback routine set an error flag and save error data.

Note 2: Timer_user.c requires modification to have the MD_INTTM000() interrupt-service routine call the sw_isr() routine to debounce the switches.

3. Serial Communication Using LIN

The NEC Electronics UART6 peripheral supports Local Interconnect Network (LIN) data communication. The LIN interface is a low speed (1- to 20-Kbps) serial-communication protocol. This interface suits low-cost control-network applications and provides support for non-precise clock sources on slave nodes.

3.1 UART6 LIN Features

The UART6 peripheral supports LIN communication with the following features:

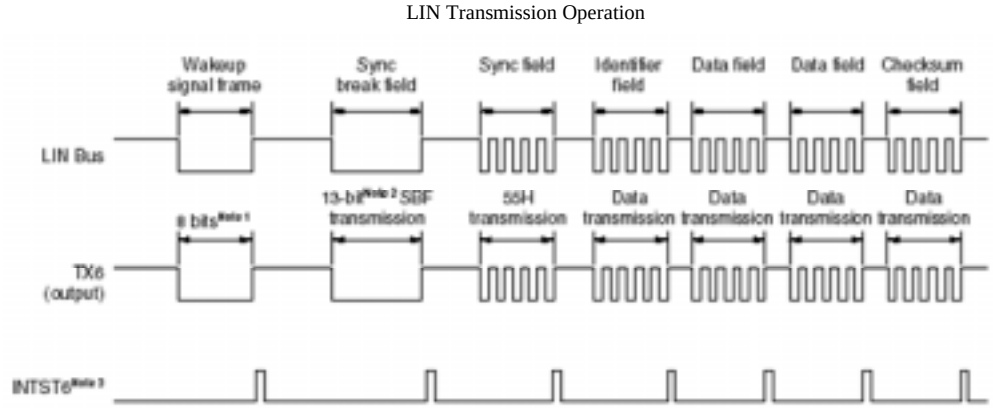
- Transmission and reception of data in LIN standard format (1 start bit, 8 data, no parity, 1 stop)
- On-board baud-rate generation by divisions of the main clock
- Baud-rate adjustments in fine increments to adapt slave node to a changing clock
- Interrupt-based handling of transmit end and receive end for program efficiency
- Automatic transmitter generation of Sync Break Frame (SBF)
- Ability of receiver to recognize SBF and reject other communications for synchronization
- Interrupt-on-end-of-transmission or reception of data byte
- Can use timer with RXD6 input for measurement or wakeup interrupt

The LIN bus uses a single-wire communication channel, connected to the nodes via LIN transceivers that comply with ISO9141. LIN has these characteristics:

- Low-speed (1- to 20-Kbps) serial-communication protocol
- Supports low-cost control-network applications
- Supports single-master, multi-slave configuration
- Connects up to 15 slaves to a master

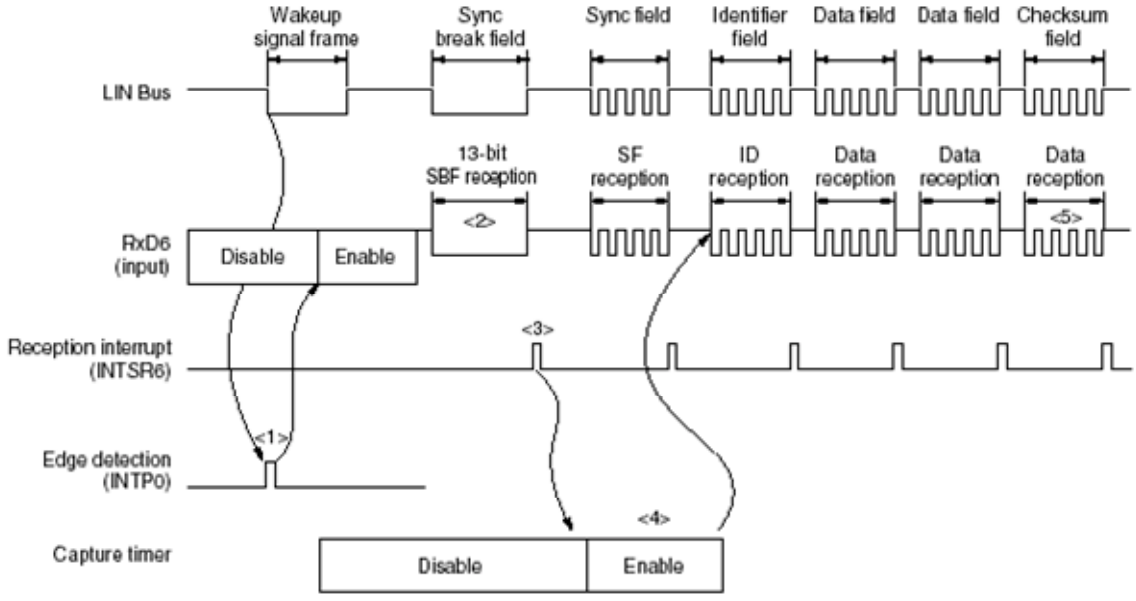
In the LIN protocol, the master transmits a message frame with a Sync Break Field (SBF) to start transmission and a **SYNC** byte (0x55) to provide baud-rate information. The slave recognizes the SBF and uses the **SYNC** byte to correct any baud-rate error. The system communicates even when the slave baud-rate error is ±15%, allowing the use of low-cost external RC oscillators or imprecise internal oscillators for slave nodes.

Figure 10. LIN Transmission Operation



Preceding the message frame, an optional wakeup byte brings slave nodes out of sleep mode. Slaves may also transmit a wakeup to bring the master node out of sleep mode. The master transmits the SBF, followed by the SYNC byte and an Identifier (**ID**) field. Double parity bits protect the **ID** field from corruption. Depending on the meaning of the **ID** for the master and slave nodes, the master may transmit data or wait for a slave to transmit data. The data length is 2, 4, or 8 bytes, specified by bits in the **ID** code. A checksum byte following the data ensures data integrity.

Figure 11. LIN Reception



The LIN reception operation includes the following steps:

1. The node detects a wakeup signal and enables RXD for SBF reception.
2. SBF reception continues until the node detects a stop bit.
3. If SBF reception completes correctly, the operation generates an interrupt.
4. The node calculates the baud-rate error from the bit intervals of the sync field.
5. Software distinguishes the checksum field.

For slave-node reception, the wakeup frame can generate an interrupt to cancel power-down, stop, or halt modes. The UART6 receiver can then be set to recognize the Sync Break Field and reject normal data bytes. This capability allows the slave to wake up in the middle of a transmission and wait for the start of the next message frame to begin operation.

An internal timer connected to the RXD6 input can measure the **SYNC** byte's bit widths. After the **SYNC** byte is transmitted, the slave adjusts its baud rate, adapting to the most recent message frame for clock synchronization. After receiving the **ID** field, and depending on its content, the slave may ignore the following data and checksum, receive and use the data from the master, or transmit data to the master.

To configure the UART6 peripheral for use in LIN protocol communication, use the following steps:

- Set port-mode register bits to select input/output mode on appropriate RXD6 and TXD6 pins, and set the port-output latch for the TXD6 pin to the appropriate state.
- Set 8 data bits, no parity, and 1 stop bit using the ASIM6 register.
- Set LSB first and non-inverted TXD6 output using the ASICL6 register.
- Set the baud rate by setting the clock source (CKSR6) and baud rate divider (BRGC6) registers.
- Set the priority for the receive error, receive, and transmit interrupts, clear the interrupt flags, and unmask the interrupts.
- Turn on the UART6 interface by setting the POWER6 bit in the ASIM6 register.
- Enable the transmitter and receiver by setting the TXE6 and RXE6 bits in the ASIM6 register.

To start Sync Break Field transmission in the master, set the SBTT6 bit in the ASICL6 register. To transmit data, write data to the TXB6 register.

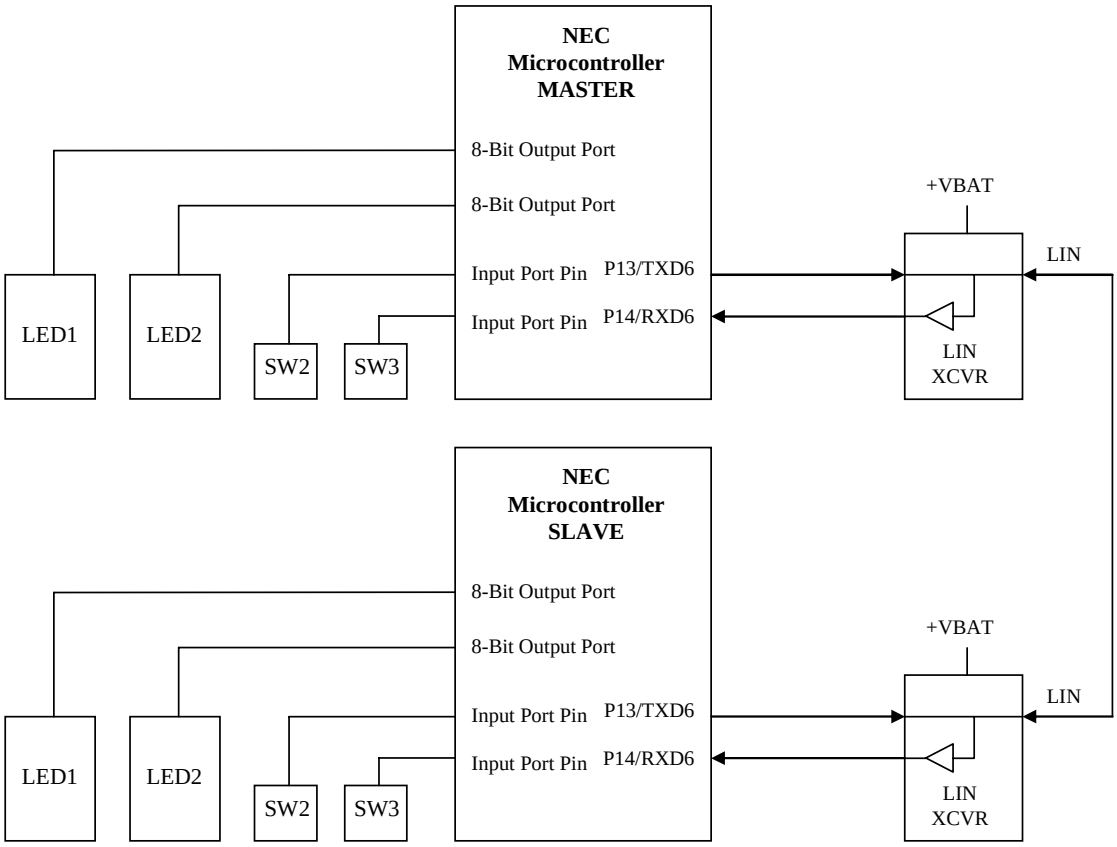
To prepare the slave to receive the Sync Break Field, set the SBRT6 bit in the ASICL6 register. When the SBF ends, the SBRF6 flag will be set in ASICL6. To read received data, read the RXB6 register.

3.2 Program Description and Specification

The LIN serial command and data transmission demonstration uses two NEC Electronics microcontrollers that interface to the LIN bus with the UART6 peripheral. One microcontroller acts as the master device and sends commands, while the other microcontroller acts as the slave device and receives commands. Both the master and slave can send or receive data. Both devices have local two-digit LEDs for showing data and two input switches to control operation.

The master and slave connect to the LIN bus via a LIN transceiver, which connects both the transmit and receive lines to the single-wire LIN bus. Each transceiver has its own power supply, +VBAT. The transceivers share a common ground connection to provide a voltage reference for the LIN bus.

Figure 12. Block Diagram for LIN Communication Scheme



The master device controls the test and initiates LIN message frame transfers. The master maintains a local count variable, which it can display, change, send to the slave, or set in response to data received from the slave. The slave displays its own count variable, which it can change locally or on command from the master.

Specifications:

- The master uses UART6 with a 19200 baud rate generated from a 20-MHz external crystal.
- The slave uses UART6 with an initial 19200 baud rate generated from an internal 8-MHz oscillator.
- Data transmits LSB first, with 1 start bit, 8 data bits, no parity, and 1 stop bit.
- The slave baud rate adapts to the master baud rate by measuring the **SYNC**-byte bit width.
- All transmission and reception is in LIN protocol V1.3 message frames.
- Master and slave recognize the following commands:
  - The master sends a two-byte count to the slave, and the slave sets the local count to the received value.
  - The master requests a two-byte count from the slave, and the master sets the count to the received value.
  - The master requests a four-byte identifier from the slave and displays the results.

3.3 Software Flowcharts

The LIN demonstration uses two programs, one for sending commands (master) and one for receiving commands (slave). Both the master and slave can send or receive data.

The demonstration programs consist of the following major sections related to LIN operation:

- Program-initialization code (including UART6 initialization), called before the main() program starts
- The main program loop, which checks switches, transmits data through the LIN bus, and checks for received data and displays values
- Subroutines for sending and receiving LIN data, and interrupt-service routines related to UART6 and TM00 operation
- A subroutine for setting TM00 for bit-width measurements and an interrupt-service routine for TM00 (slave only)

The programs also include sections not related directly to LIN operation:

- Subroutines generated by the Applilet to initialize ports for switches and LED display
- Subroutines generated by the Applilet to handle interval timer starting, stopping, and interval setting
- Subroutines with user code for handling timer interrupts used for debouncing switches
- Subroutines for switch input and LED display output

The flowcharts describe initialization, the main programs, subroutines and interrupt routines related to LIN operation. There are no flowcharts for other initialization, other peripheral-operation subroutines, and other peripheral interrupt-service routines. However, the software listings include this code.

3.3.1 LIN_XFER_STRUCT – Structure to Manage LIN Message Frame Transfer

Both the master and slave programs manage the sending and receiving of commands and data with a data structure of type LIN_XFER_STRUCT. The definition of this structure is:

```
typedef struct {
   UCHAR state;            /* one of LIN_STATE_xxx above */
   UCHAR id;          /* ID byte (will define dir and len */
   UCHAR dir;         /* direction, either LIN_ID_DIR_OUT or LIN_ID_DIR_IN */
   UCHAR len;         /* length of transfer: 2, 4, or 8 bytes */
   UCHAR count;            /* count of data transferred */
   UCHAR data[8];     /* buffer for data to send/received */
   UCHAR checksum;    /* checksum to send, or expected to receive */
   UCHAR rcvsum;           /* actual received checksum byte */
} LIN_XFER_STRUCT;
```

The **state** member of this structure defines where the master or slave is in the transfer process. The states are defined as:

```
#define LIN_STATE_IDLE     0x00  /* not in transmission/receive */
#define LIN_STATE_SBF      0x01  /* send/receive Synch Break Field */
#define LIN_STATE_SYNC     0x02  /* send/receive SYNC byte */
#define LIN_STATE_ID       0x03  /* send/receive ID field */
#define LIN_STATE_DATAW    0x04  /* writing data */
#define LIN_STATE_SUMW     0x05  /* writing checksum */
#define LIN_STATE_DATAR    0x06  /* reading data */
#define LIN_STATE_SUMR     0x07  /* reading checksum */
```

For example, when the master is between transfers, it is IDLE. When it sends the Synch Break Field and is waiting for that field to be sent, the master is in the SBF state. When it finishes sending the SBF field, the master sends the Sync byte and waits in the SYNCH state for that byte to be sent.

When the slave is between commands, it is waiting to receive the Synch Break Field and is therefore not in the IDLE state, but in the SBF state. When the slave receives the Synch Break Field and is waiting for the Sync byte, then the slave is in the SYNC state.

Once the **SYNC** field transfers, the master transmits the LIN **ID** byte and is in the ID state. The slave is also in the ID state, waiting to receive the **ID**. Bits 6 and 7 of the **ID** byte serve as double-parity bits to protect against corrupted commands.

The ID command determines the direction of data and checksum transfers, and the number of data bytes. In this application, **ID** bits 4 and 5 determine the length (2, 4 or 8 bytes), and bit 3 determines the direction.

The states DATAW, SUMW, DATAR, and SUMR accommodate data and checksum transfers, which can be from master to slave or vice versa. The suffix “W” denotes writing from master to slave; the “R” suffix denotes master reading from slave.

The state DATAW indicates that the master sends data to the slave. When the slave is receiving data, it is in the DATAW state. DATAR indicates that the master reads data from the slave to the master. When the slave is sending data, it is in the DATAR state. The same is true for the checksum states.

If the **ID** direction bit is OUT (master to slave), both the master and slave shift into the DATAW state. Once the data bytes have been transferred, both master and slave change to the SUMW state, sending and receiving the checksum. When the checksum has been sent, the master returns to the IDLE state. The slave acts on the data sent and then enters the SBF state to receive the next command.

If the **ID** direction bit is IN (slave to master), both master and slave shift into the DATAR state. The slave prepares data (as determined by the command sent), and the master waits to receive the data. After receiving the data, the master acts on the data received and returns to the IDLE state. The slave enters the SBF state to be ready for the next command.

The UART6 transmit-done and received-character interrupt routines shift the master and slave states. This arrangement frees the main programs for other actions during data transfers.

The slave uses the sync byte (0x55) to measure the master's baud rate. The slave can then adapt its baud rate accordingly. Timer 00 (TM00) measures the pulse width, and the timer's interrupt-service routine handles the slave's transition from SYNC to ID.

The other members of the LIN_XFER_STRUCT structure are fairly self-explanatory. The id member holds the **ID** byte sent or received. The dir and len fields are set according to the appropriate bits in the **ID** byte, with len set to 2, 4 or 8 depending on the number of bytes to be sent. The count member holds the actual number of bytes sent or received so far. The data[8] member is an array of up to eight bytes (the maximum sent in standard LIN protocol), holding the bytes to send or storing received bytes.

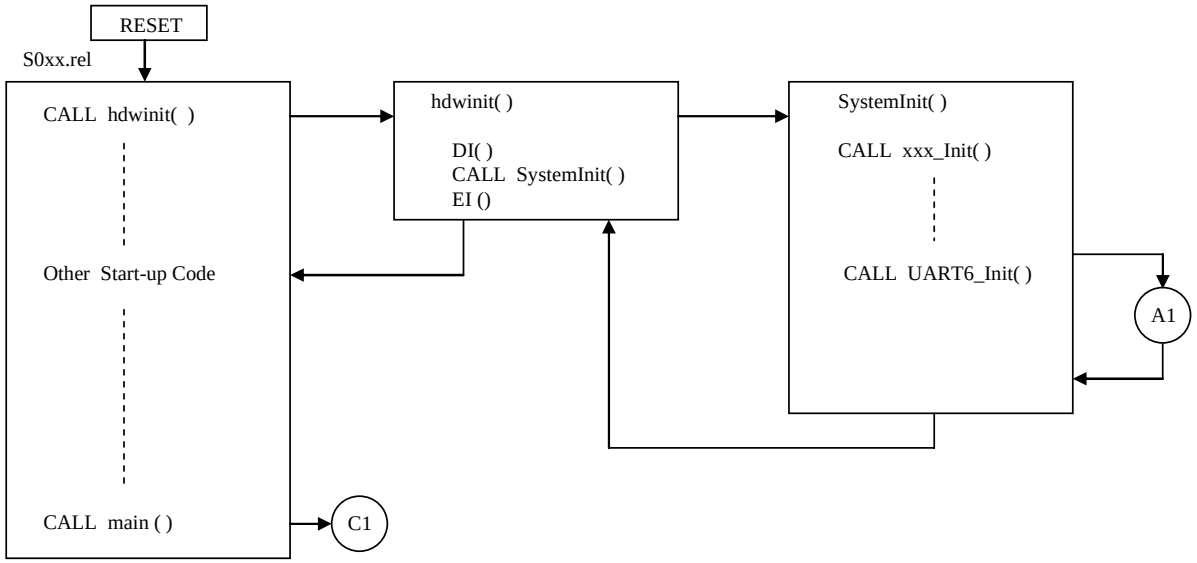
The checksum member holds either the calculated checksum of the data bytes to be sent or the checksum expected on the data bytes received. The rcvsum member holds the actual checksum byte received by the slave to use in error handling if the checksum is not correct.

3.3.2 Master: Program Startup and Initialization

For 78K0 programs written in the C language, an object code file such as s0l.rel is linked into the user's program and provides the program startup code. This startup code calls a function named hdwinit(); you can place any hardware initialization code here.

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

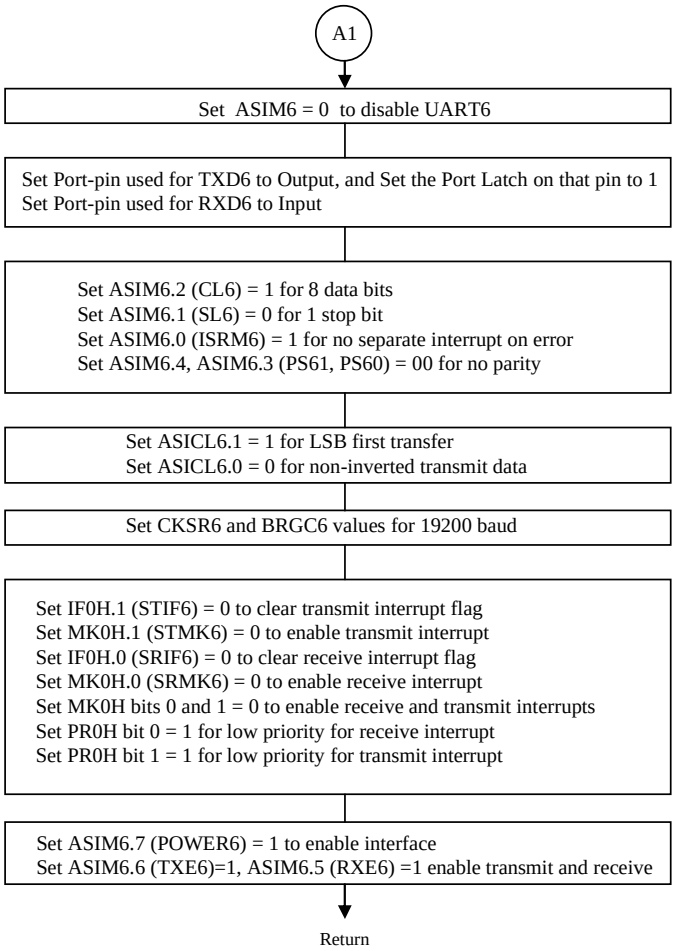
Figure 13. Initializing the System at Startup



After the hdwinit() function finishes, the startup code calls the program's main() function. When the main() function starts, all peripheral initialization is complete, and the main() function does not need to call peripheral-initialization routines.

3.3.3 Master: UART6_Init() – Initialization of UART6 for LIN

Figure 14. Flowchart for Initializing UART6



SystemInit() calls the UART6_Init() routine to initialize the UART6 peripheral. The flowchart above applies to both the master and slave programs; only the values used to set baud rate differ.

First, the routine clears the ASIM6 register (which controls UART6) to zero to disable the UART. The routine sets the port-mode register (PM1) to configure the P13/TXD6 pin as an output and sets the P13 output latch high to allow the P13/TXD6 pin to transmit data. The routine configures P14/RXD6 as an input.

The routine sets the ASIM6 register bits controlling UART data transfers for 8-bit data, 1 stop bit, and no parity. This is the standard setting used for LIN data. Setting ASIM6.0 to 1 causes the INTSR6 interrupt to occur (rather than the INTSRE6 interrupt) if there is a reception error. Thus, the INTSR6 interrupt serves for both normal and error reception.

The initialization routine sets the CKSR6 and BRCG6 registers for 19200 baud. To run the master program with a 20-MHz peripheral clock (f_{PRS}), UART6_Init() sets CKSR6 to 0x02, which sets the clock to the UART6 baud rate generator to $f_{PRS}/4$ or 5 MHz. The routine sets BRCG6 to 0x82 (130 decimal). The bit rate for UART6 transfers is calculated as:

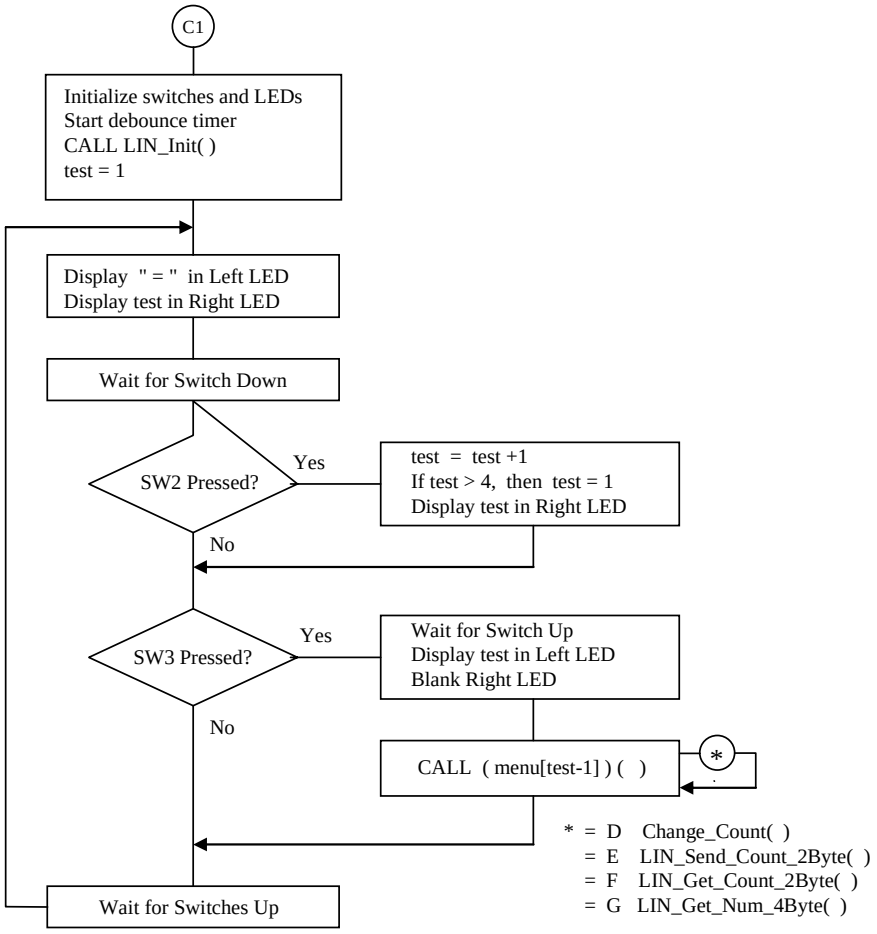
CKSR6 clock / BRGC6 * 2 = 5,000,000 / 130 * 2

= 19231 bits per second

The program enables interrupts INTSR6 and INTST6 (the interrupts controlling UART6) by setting the appropriate registers and sets ASIM6 to enable data transfers. Then the routine sets the POWER6 bit to enable the UART6 interface, and the TXE6 and RXE6 bits to enable transmit and receive at 19200 baud, with an error of 0.16%. After initialization, UART6 is ready to send by writing to TXB6 and to receive data in RXB6.

3.3.4 Master: Main() – Main Program for LIN Master

Figure 15. Flowchart for LIN Master Main Program



When the startup code completes, it calls the main() routine. Main() initializes the switch and LED display routines, and calls LIN_Init() for any additional LIN hardware or software initialization. In the demonstration program, LIN_Init() simply sets the global variable LIN_Active to true. Main sets the **test** variable to 1, and enters a test-selection loop.

The LED's left digit displays " = ", and the right digit shows the current value of **test**. At startup, the display shows " =1 ". Main() waits for a switch closure, looping until this happens. Pressing SW2 increments the test variable, which wraps back to 1 if it exceeds 4.

If you press SW3, main() waits for the switches to open again, puts the value of **test** on the left LED, blanks the right LED, and then calls a subroutine. Main() gets the subroutine's address from the menu[ ] array, using **test** – 1 as an index into the array. So if **test** = 1, main() calls the subroutine whose address is in menu[0]—the Change_Count() routine. This arrangement lets you select and run four different tests. The table below shows the values of **test**, what is displayed on the LEDs when you press SW3, the menu-array function accessed, and a description of the test.

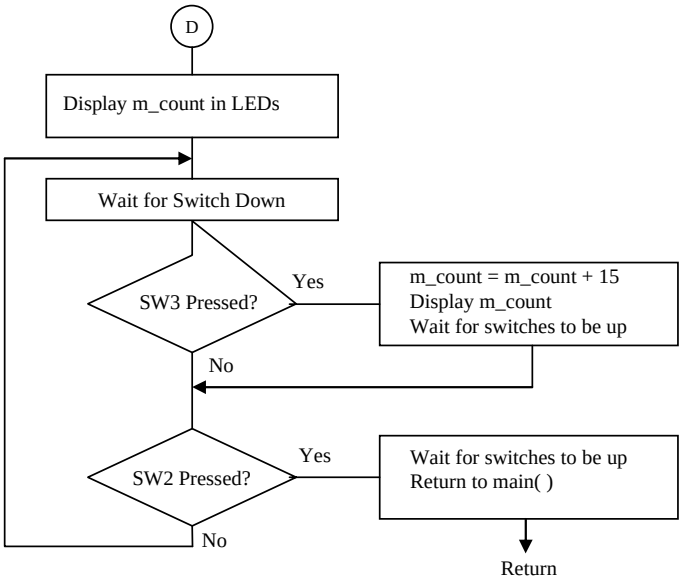
Table 6. Test Functions

test	LED	menu[test - 1]	Function Operation
1	"1 "	Change_Count()	Adds to global variable **m_count**
2	"2 "	LIN_Send_Count_2Byte()	Sends **m_count** to slave
3	"3 "	LIN_Get_Count_2Byte()	Gets value from slave, sets **m_count**
4	"4 "	LIN_Get_Num_4Byte()	Gets and displays four bytes from slave

When the subroutine returns, main() returns to the top of the loop, displays **test**, and resumes waiting for a switch closure.

3.3.5 Master: Change_Count() – Change Master Local Count

Figure 16. Flowchart for Changing Master Local Count



When **test** is 1, main() calls the Change_Count() routine, which changes the global variable **m_count**.

The master count variable **m_count** shows the transfer of data to and from the slave. This variable can be sent to the slave (**test** = 2), or set by a value received from the slave (**test** = 3).

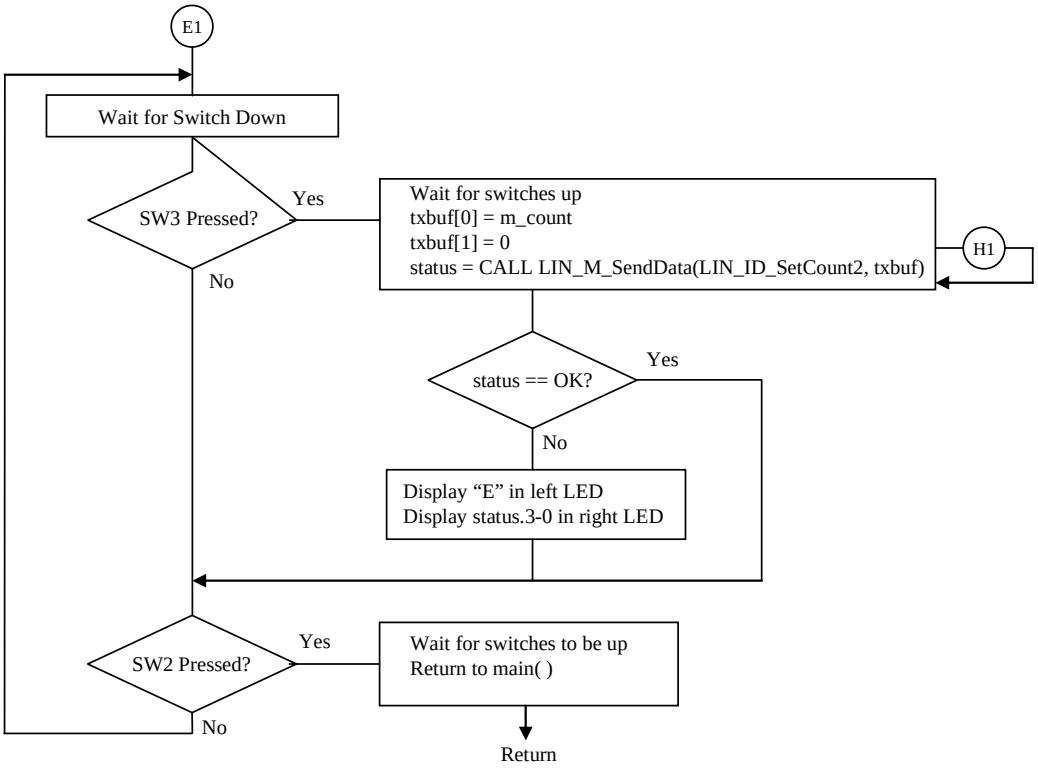
The Change_Count() routine displays the current value of **m_count**, and then waits for you to press a switch.

If you press SW3, the routine increases the value of **m_count** by adding 15 to it, then displays the new value. When the master program starts, **m_count** is zero. In this case, repeated presses of SW2 show 00, 0F, 1E, 2D, etc. If **m_count** held a different value that it received from the slave, that value would be modified in the same way.

If you press SW2, Change_Count() waits for the switch open again, then returns to main().

3.3.6 Master: LIN_Send_Count_2Byte() – Transfer Count Data to Slave

Figure 17. Flowchart to Transfer Count Data to Slave



When **test** =2, main() calls the LIN_Send_Count_2Byte() routine. This routine sends the current value of the master count global variable **m_count** to the slave.

The routine waits for you to press a switch. If you press SW3, LIN_Send_Count_2Byte() waits for the switch to open and then sets up a two-byte array (txbuf) by setting txbuf[0] to the value of **m_count**, and setting txbuf[1] to zero.

LIN_Send_Count_2Byte() calls the LIN master function LIN_M_SendData(LIN_ID_SetCount2, txbuf). The first parameter is the **ID** byte in the LIN transfer. In both master and slave application programs, this command specifies sending a two-byte count from the master to the slave. The second parameter is the address of the buffer, txbuf, containing the two-byte data to send.

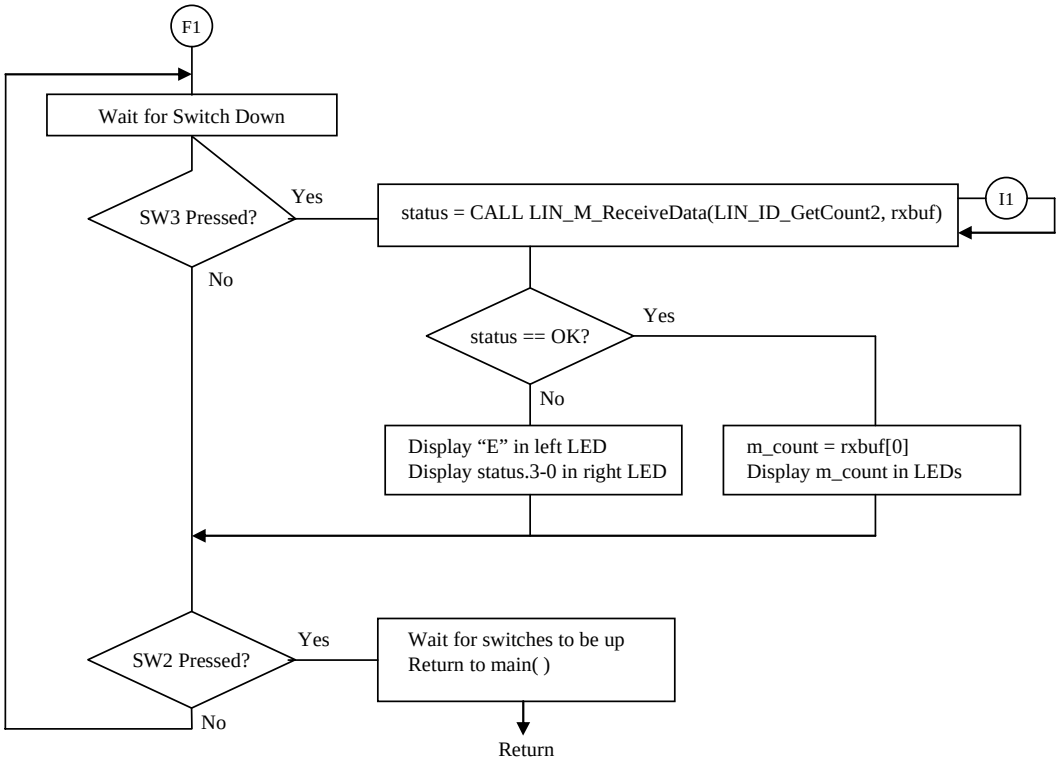
The LIN_M_SendData() routine transfers the data and returns a status code indicating whether the transfer was successful or whether there was an error. If the transfer is successful, the display does not change.

If an error occurs while sending the command and data, the LED displays an “E” in the left LED and a hex digit in the right LED corresponding to the low four bits of the status variable set by the LIN_M_SendData() routine.

If you press SW2, the LIN_Send_Count_2Byte() routine waits for the switch to open, then returns to main().

3.3.7 Master: LIN_Get_Count_2Byte() – Request Count Data from Slave

Figure 18. Flowchart for Request Count Data from Slave



When **test** is 3, main() calls the LIN_Get_Count_2Byte() routine. This routine gets a value from the slave and sets the master count global variable **m_count** to that value.

LIN_Get_Count_2Byte() waits for you to press a switch. If you press SW3, the routine waits for the switch to open, then calls the LIN master function LIN_M_ReceiveData(LIN_ID_GetCount2, rxbuf). The first parameter is the **ID** byte. In both master and slave application programs, this command specifies sending a two-byte count from the slave to the master. The second parameter gives the address of a two-byte buffer (rxbuf) for storing received data.

The LIN_M_ReceiveData() routine transfers the data and returns a status code indicating whether the command was sent successfully and data was received without error, or whether there was an error in the transfer.

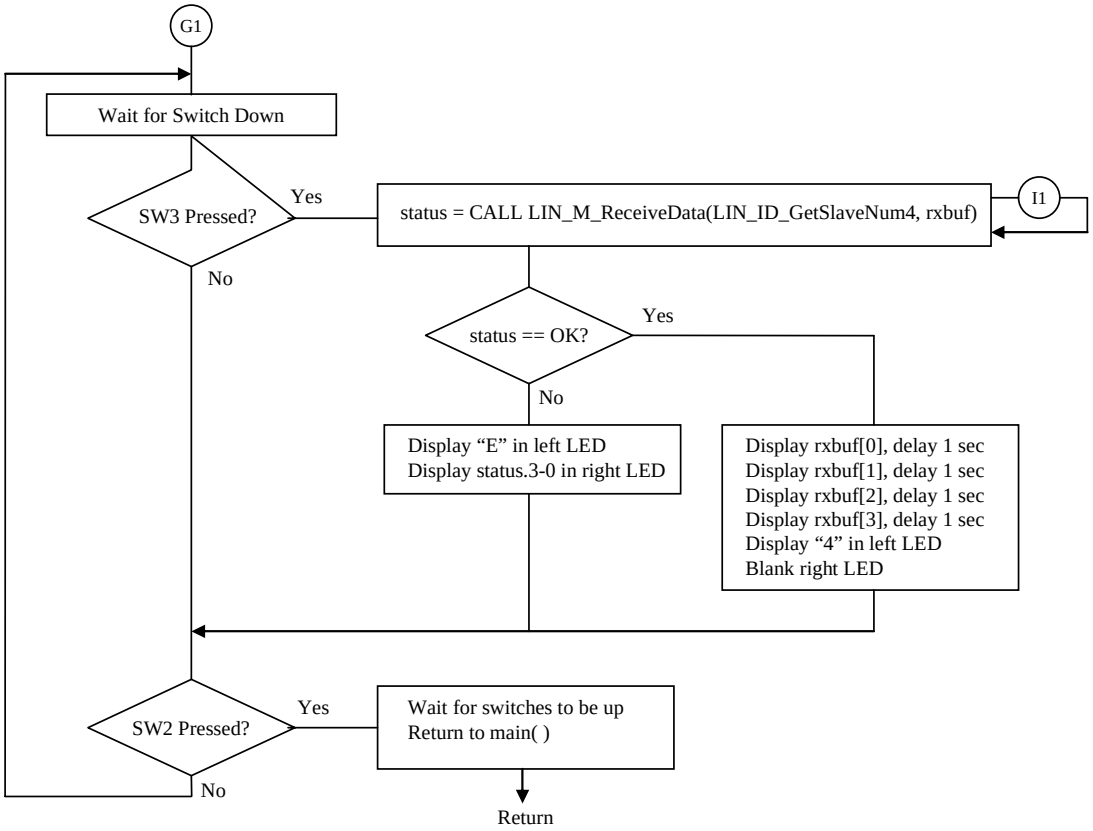
If there is no error, the routine sets **m_count** now stored in rxbuf[0] and shows the value in the LED display as a hexadecimal number. For instance, if the count received from the slave is 125, the display shows “7D”.

If an error occurs in sending the command or receiving data, the LED display shows an “E” in the left LED and a hex digit in the right LED corresponding to the low four bits of the **status** variable set by the LIN_M_ReceiveData() routine.

If you press SW2, the LIN_Get_Count_2Byte() routine waits for the switch to open and returns to the main() routine.

3.3.8 Master: LIN_Get_Num_4Byte() – Receive Four Data Bytes from Slave

Figure 19. Flowchart for Receiving Four Data Bytes from Slave



When **test** is 4, main() calls the LIN_Get_Num_4Byte() routine. This routine gets four bytes of data from the slave and displays them serially in the LED display.

The routine waits for a switch closure. If you press SW3, LIN_Get_Num_4Byte() waits for the switch to open, then calls the LIN master function LIN_M_ReceiveData(LIN_ID_GetSlaveNum4, rxbuf). The first parameter is the **ID** byte. In both master and slave application programs, this command specifies sending a four-byte slave identifier from the slave to the master. The second parameter is the address of a four-byte buffer (rxbuf**)** for the received data.

The LIN_M_ReceiveData() routine transfers the data, and returns a status code indicating whether the command was sent successfully and data was received without error, or whether there was an error in the transfer.

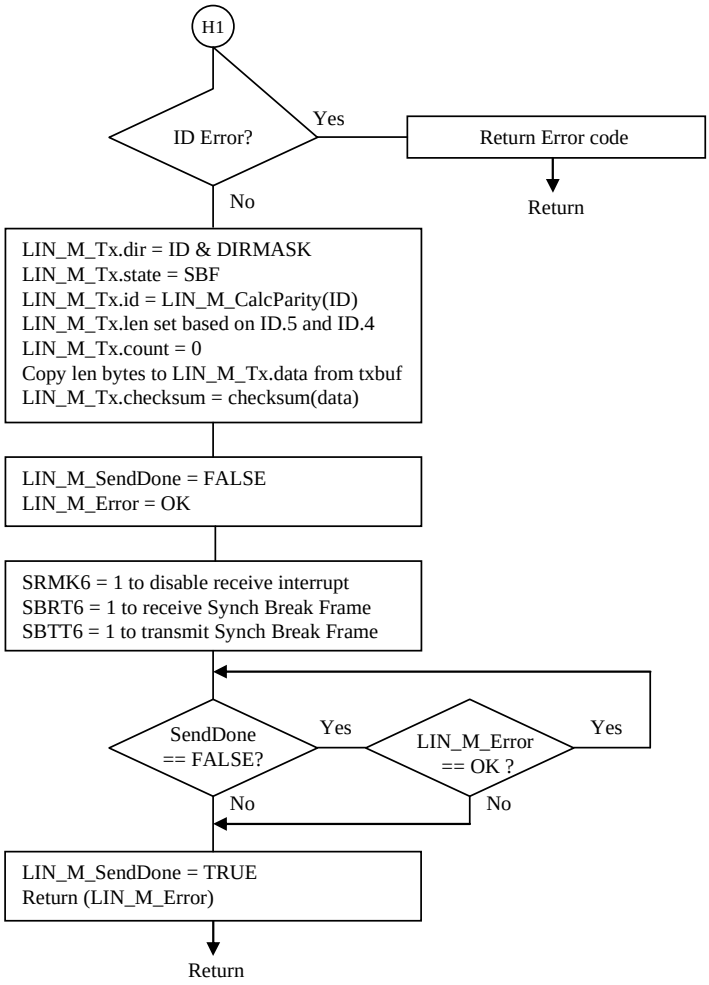
If there is no error, the routine displays the received data on the LED display one byte at a time, with a wait of about 1 second between values. The routine then displays “4” on the left LED and blanks the right one, showing the state at the start of the routine.

If an error occurs while sending the command or receiving data, the LED display shows an “E” in the left LED and a hex digit in the right that corresponds to the low four bits of the status variable set by the LIN_M_ReceiveData() routine.

If you press SW2, the LIN_Get_Num_4Byte() routine waits for the switch to open, and returns to main().

3.3.9 Master: LIN_M_SendData(ID, *txbuf) – Send Command and Data to Slave

Figure 20. Flowchart for Sending Command and Data to Slave



The LIN_M_SendData(ID, *txbuf) routine sends data from the master to the slave. First the routine checks the **ID** parameter, which specifies the command, direction, and length, and returns an error if **ID** is incorrect.

The LIN_M_Tx structure controls the transfer. This structure is of the type LIN_XFER_STRUCT and takes the **dir**, **id**, and **len** members from the **ID** parameter. The routine LIN_M_CalcParity(ID) calculates the double parity bits **id.7** and **id.6**. The routine copies the data into the data[ ] array from the buffer pointed to by the txbuf parameter.

The routine calculates the checksum member from the data bytes it will send. For the LIN protocol, the routine adds the data bytes together, incrementing the sum if a carry is generated for an addition and inverting the resulting sum for sending. By using the same algorithm for received data and adding together the received and calculated checksums, the result should be 0xFF.

The routine sets the global flag LIN_M_SendDone to false, and the global variable **LIN_M_Error** to OK.

The routine masks the interrupt for UART6 received data or errors. The LIN transceiver echoes transmitted characters, which UART6 receives. The transmit-done interrupt-service routine handles the receive-character interrupt flag. The routine sets the SBRT6 bit in the ASICL6 register to configure the UART6 receiver to look for a Sync Break Field. The routine then sets the SBTT6 bit in the ASICL6 register, causing UART6 to transmit a 13-bit Sync Break Field.

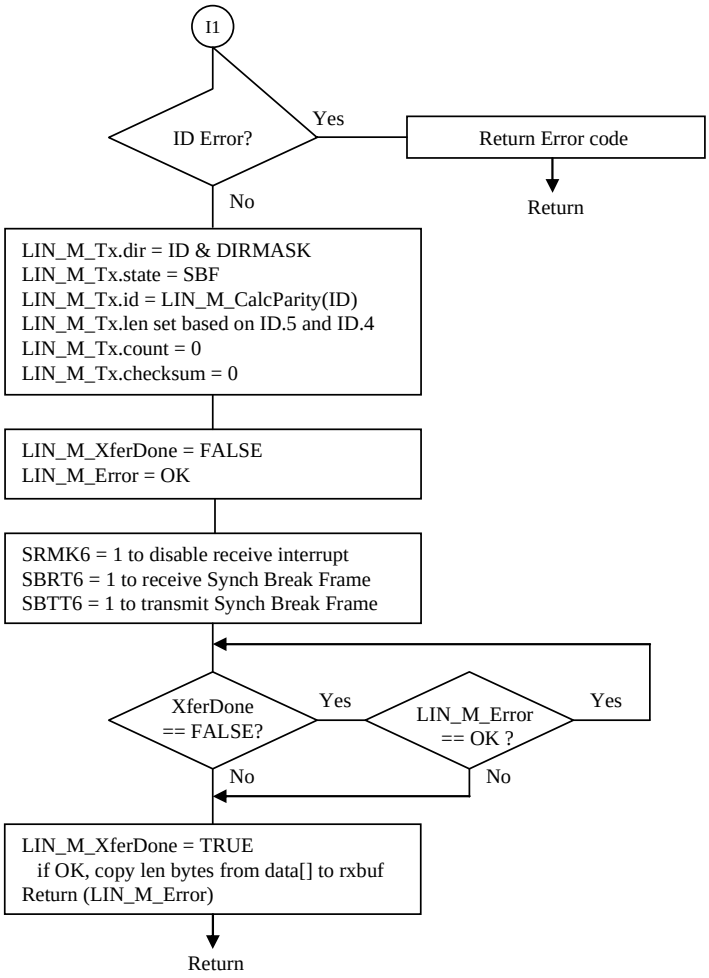
LIN_M_SendData() then loops, waiting for either LIN_M_SendDone to be true or LIN_M_Error to be set to something other than OK. The LIN_M_Tx_Handler() interrupt-service routine sets these flags. When one of these flags is set, LIN_M_SendDone is forced to true, and the value in the LIN_M_Error global variable is returned.

LIN_M_SendData() returns OK if LIN_M_Error is still set as OK, the transfer was completed successfully, and the routine was ended by LIN_M_SendDone being set to true.

If LIN_M_Error is not OK, an error occurred somewhere in the transfer, and LIN_M_SendData() returns the error code.

3.3.10 Master: LIN_M_ReceiveData(ID, *rxbuf) – Send Command, Receive Data from Slave

Figure 21. Flowchart for Sending Command, Receive Data from Slave

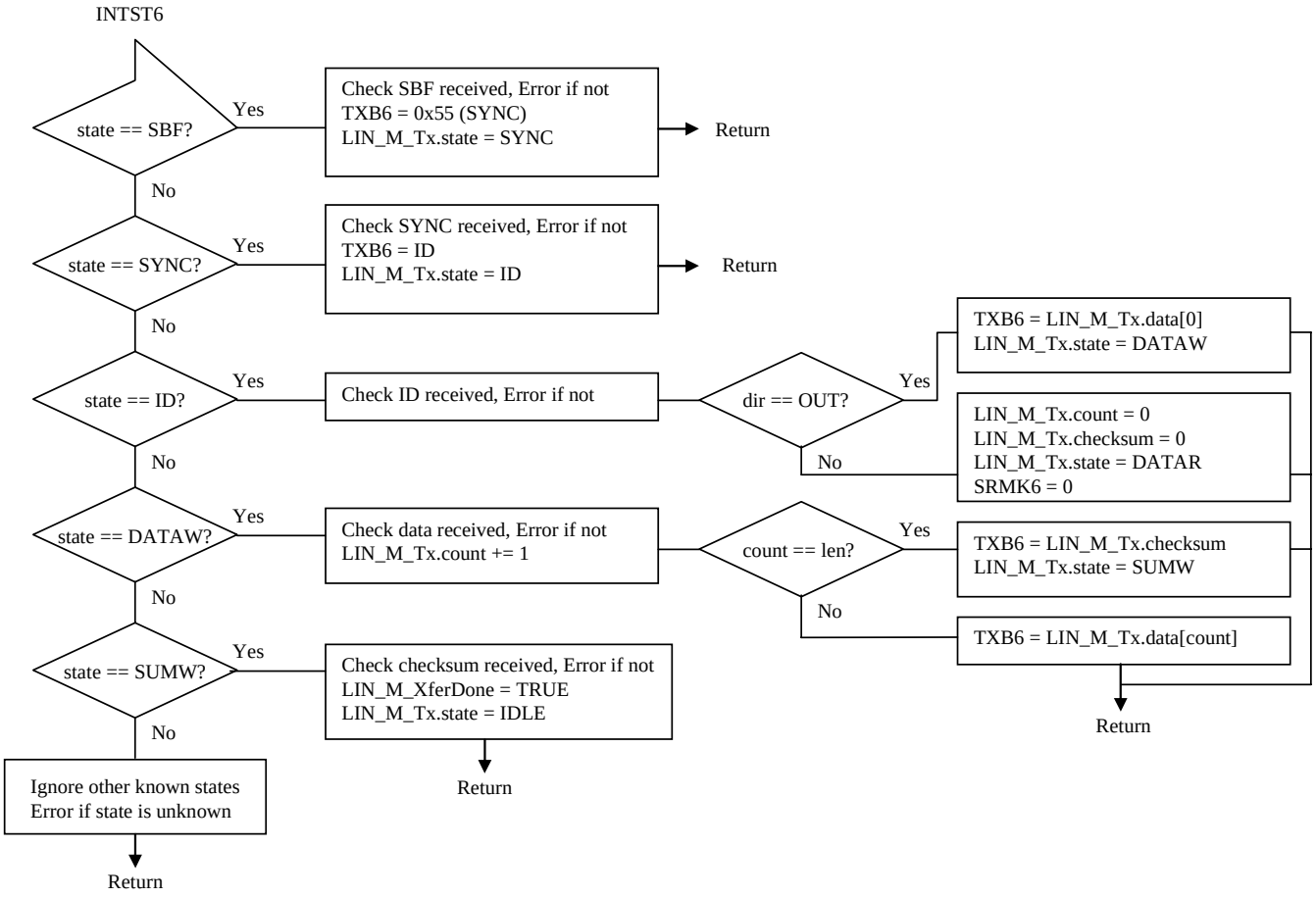


The LIN_M_ReceiveData() routine causes the master to request data from the slave. Very similar to the LIN_M_SendData() routine in structure, LIN_M_SendData copies txbuf to data[] and sets the checksum before the transfer. On the other hand, LIN_M_ReceiveData() sets the checksum to zero before the transfer, and copies received data from data[] to rxbuf after the transfer, if there was no error.

Note that both the LIN_M_SendData() and LIN_M_ReceiveData() routines should check for timeout conditions. This check is done by setting a timer that activates after the maximum time for a message-frame transmit or receive, and sets the LIN_M_Error to a code indicating timeout. This timer is not implemented in the code for this application note.

3.3.11 Master: LIN_M_Tx_Handler() – UART6 Transmit Done Interrupt-Service Routine

Figure 22. Flowchart for UART6 Transmit Done Interrupt-Service Routine



When the INTST6 interrupt occurs at the end of a character transmission, the interrupt invokes the LIN_M_Tx_Handler() interrupt-service routine. This routine either signals an error, by setting the LIN_M_Error variable to a value other than OK, or the successful end of transmission, by setting LIN_M_XferDone to true.

Because the LIN transceiver causes transmitted data to be received as well, you can check for proper data transmission by reading the receive interrupt flags and register.

The LIN_M_SendData() or LIN_M_ReceiveData() routines start Sync Break Field transmission by setting the SBTT6 bit true, shifting the system into the SBF state. The routine checks to see if the UART6 receiver received the SBF. If not, the routine sets LIN_M_Error and returns. If the UART6 received the SBF, the routine sends the **SYNC** byte, sets the state to SYNC, and returns.

The next interrupt occurs at the end of **SYNC** byte transmission. The routine sees the SYNC state, checks if the **SYNC** byte was received as sent, and gives an error message if the byte is incorrect. Otherwise, the routine sends the **ID** byte, sets the state to ID, and returns.

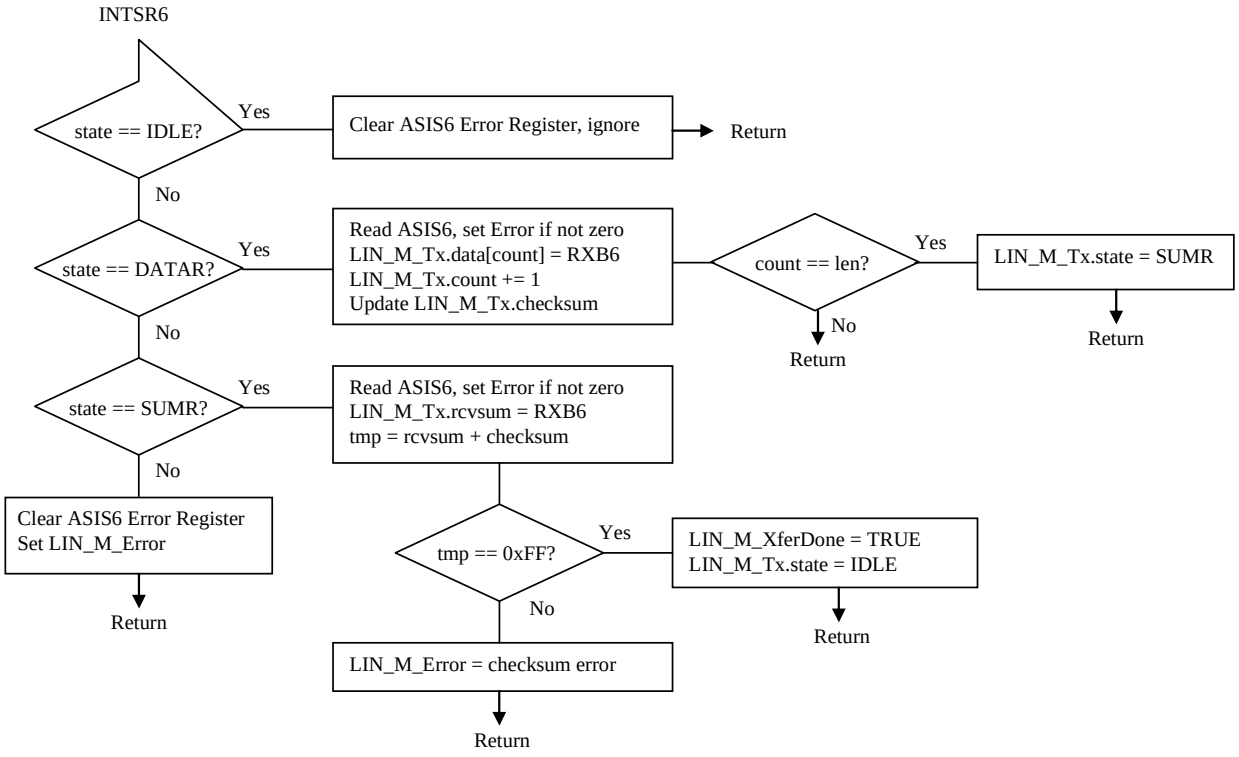
The next interrupt occurs at the end of the **ID** byte transmission. The routine sees the ID state, checks if the **ID** byte was received as sent, and gives an error if the byte is incorrect. If the **ID** byte is correct, the next action depends on the direction of data transfer. If OUT (sending data from master to slave), the routine sends the first data byte, and the state becomes DATAW (writing data). If the direction is IN (master receiving data from slave), the routine clears the count and checksum, sets the state to DATAR (reading data), and enables the receive interrupt. The LIN_M_Rx_Handler() interrupt-service routine takes care of the received data.

The next transmit interrupt occurs after sending the first data byte. LIN_M_Tx_Handler() sees the DATAW state, checks if the data byte was received as sent, and gives an error message if the byte is incorrect. If the data is correct, the routine increments the **count** member. When **count** equals **len**, all data bytes have been sent, and the routine sends the checksum value, sets the state to SUMW, and returns. If all data bytes have not been sent, the routine sends the next data byte and returns, keeping the state as DATAW. The next transmit interrupt occurs after the next data byte transmission.

After sending the checksum byte, the next transmit interrupt occurs. The routine sees the SUMW state, checks that the checksum was received as sent, returning an error if it is incorrect. If the checksum is correct, then the message frame has been sent with no error, so the routine sets the LIN_M_XferDone flag to true, and the state to IDLE.

3.3.12 Master: LIN_M_Rx_Handler() – UART6 Receive Character Interrupt-Service Routine

Figure 23. Flow chart for UART6 Receive Character Interrupt-Service Routine



When the INTSR6 interrupt occurs due to a received character or a reception error, the interrupt invokes the LIN_M_Rx_Handler() interrupt-service routine. This routine either signals an error, by setting the **LIN_M_Error** variable to a value other than OK, or the successful end of transmission, by setting LIN_M_XferDone to true.

Note that this interrupt should only occur after the master has already transmitted the Sync Break Field, the **SYNC** byte, and an **ID** value indicating that the slave should send data to the master. The state should be DATAR.

The receive interrupt should occur after the master receives the first data byte from the slave. The routine will see the DATAR state, and check the ASIS6 reception-error status register; if the value is non-zero, a reception error has occurred. The routine sets LIN_M_Error to a value indicating the error type. Possible errors are framing (low data past expected stop bit), parity (parity bit not correct), or overrun (new data received before RXD6 is read).

If there is no error, the routine stores the received data in the data[] array, increments the **count**, and updates the checksum. If **count** equals **len**, the routine knows it has received all of the expected data. The routine sets the state to SUMR, expecting to receive the checksum. If **count** is still less than **len**, the routine returns, leaving the state as DATAR. The next receive interrupt is handled as a data byte.

If the state has been set to SUMR, then the routine treats the next receive interrupt as the reception of the checksum byte. The routine checks the ASIS6 register and indicates an error if this register is non-zero. If there is no receive error, the routine stores the checksum byte in **rcvsum**. The routine adds the received checksum and the calculated checksum values together. If the sum is 0xFF, the checksum is correct.

If the checksum is correct, then the routine has received data from the slave correctly, and the message frame is complete. The routine sets **LIN_M_XferDone** flag to true, and the state to IDLE.

If the sum of **rcvsum** and the checksum is not 0xFF, the routine sets **LIN_M_Error** to indicate a checksum error.

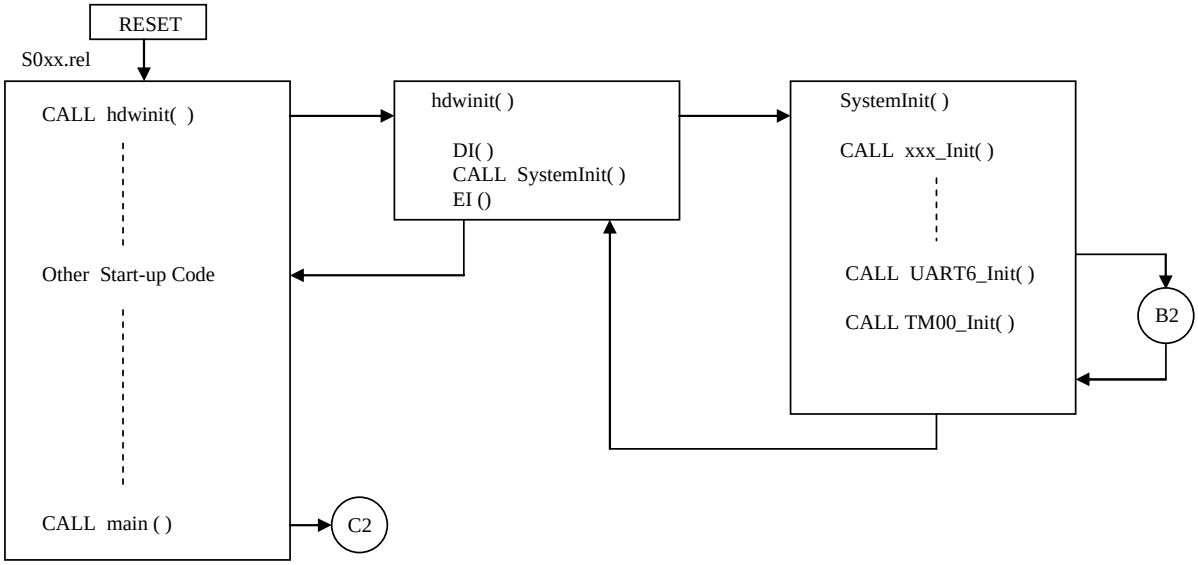
If the state is other than IDLE, DATAR, or SUMR, an unexpected receive interrupt has occurred, and the routine sets **LIN_M_Error** to indicate a generic error.

3.3.13 Slave: Program Startup and Initialization

For 78K0 microcontroller programs written in the C language, the startup code for the C program is supplied by an object code file such as s0l.rel, which is linked into the user program. This startup code calls a function named hdwinit(); you can place any hardware initialization code here.

When you use the Applilet to generate a C program for the 78K0 microcontroller, the hdwinit() function is automatically added to the user program. This function calls the function SystemInit(). The SystemInit() function in turn calls initialization routines for each peripheral.

Figure 24. Startup Code for Slave Program



After the hdwinit() function finishes, the startup code calls the program's main() function. When the main() function starts, all peripheral initialization is complete, and the main() function does not need to call peripheral-initialization routines.

3.3.14 Slave: UART6_Init() – Initialization of UART6 for LIN

For the slave program, UART6_Init() initializes the UART6 peripheral. This routine is identical to the routine used for the master, except for the setting of baud-rate registers. Please see the section above showing the UART6_Init() flowchart.

UART6_Init() sets the baud rate for UART6 by initializing the CKSR6 register, which sets a division of the main peripheral clock, and by initializing BRGC6, which counts the selected clock to specify a bit rate.

The slave program runs on an internal 8-MHz clock. The program divides that clock to produce the 19200 baud transfer rate used for LIN communication. For the slave, this rate results in initial values of CKSR6 = 0 (8 MHz clock), and BRGC6 = 0xD0 (208 decimal). With these settings, the bit time for serial data is determined by:

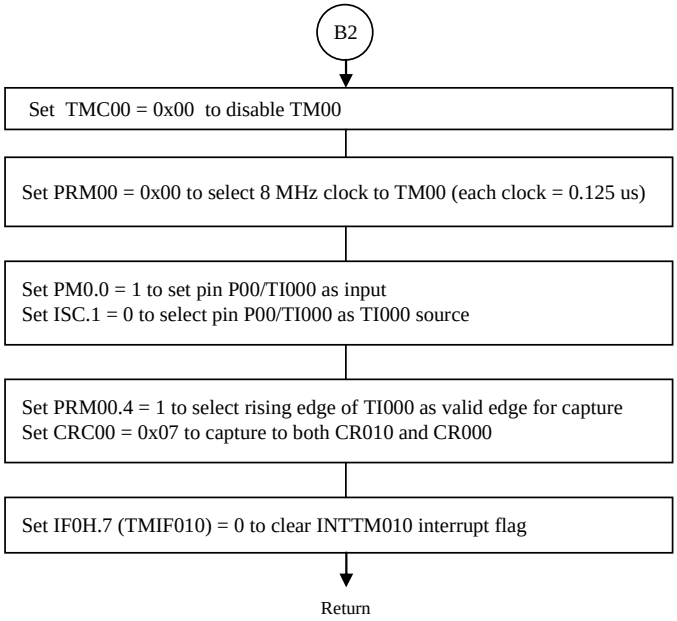
CKSR6 clock / BRGC6 * 2 = 8,000,000 / 208 * 2

= 19231 bits per second

You can modify the initial value of BRGC6 for the slave based on the measurement of the bit width of the **SYNC** byte sent from the master.

3.3.15 Slave: TM00_Init() – Initialization of Timer 00 for Measuring SYNC Bit Times

Figure 25. Initialization of Timer 00 for Measuring SYNC Bit Times



In the slave program, SystemInit() calls TM00_Init() to set up Timer 00 (TM00) for pulse-width measurement. TM00 measures the bit times of the **SYNC** byte sent from the master. The slave program calculates a new baud rate from this time measurement.

First the routine clears TMC00 (the control register for TM00) to zero to stop and disable the timer while writing other control registers.

PRM00 bits 1 and 0 control the clock used for TM00. Clearing these bits to 00 selects the peripheral clock, f_{PRS}, as the TM00 clock. For a slave using the 8-MHz high-speed internal oscillator for its peripheral clock, this setting results in an 8-MHz clock for TM00, with each clock taking 0.125 microseconds.

The initialization routine sets the port-mode register (PM0) and input-switch control register (ISC) to configure pin P00/TI000 as an input. The TI000 input to the timer comes from this pin. Later, the slave program will change the ISC register, routing the P14/RXD6 pin to TI000, allowing TM00 to measure received-data input on this pin.

The upper bits of PRM00 define the valid edges of the inputs used. The routine sets PRM00.4 to 1 to specify that the rising edge of TI000 is the valid edge. TM00_Init() sets CRC00 to capture the TM00 value to CR010 on the valid (rising) edge of TI000, and the TM00 value to CR000 on the non-valid (falling) edge of TI000.

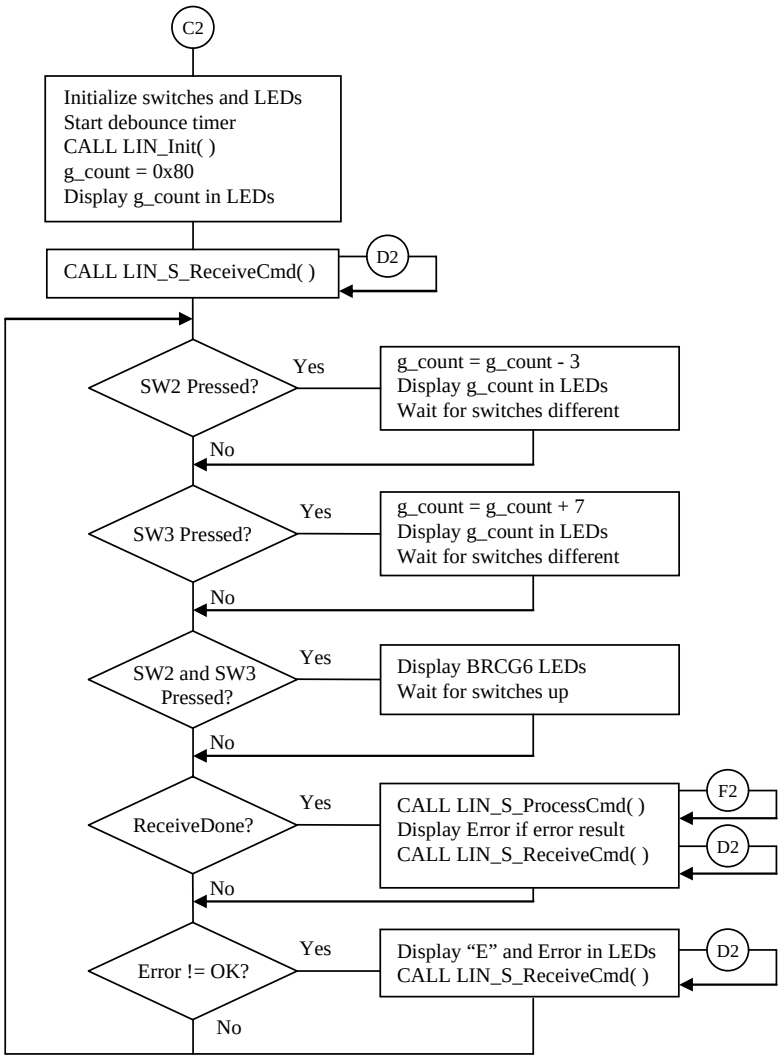
The initialization routine clears the TMIF010 interrupt flag and returns.

Once the TM00_Start() routine enables the timer in the clear-and-start mode, TM00 will count up once every 0.125 microseconds. On the falling edge of TI000, the microcontroller writes the TM00 value to CR000. On the rising edge of TI000, the microcontroller writes the TM00 value to CR010 and clears TM00. Then the interrupt INTTM010 occurs.

After the first rising edge clears TM00, on subsequent rising edges CR010 contains the number of clocks since the previous rising edge (cycle time), and CR000 contains the number of clocks from rising edge to falling edge (high pulse width). Knowing the count clock is 0.125 microseconds, the routine can calculate the width of the high, low, and full cycle of a pulse.

3.3.16 Slave: Main() – Main Program

Figure 26. Flowchart for Main Program



When the program startup code completes, it calls the slave program main() routine. Main() initializes the switch and LED display routines, and calls LIN_Init() to do any additional LIN hardware or software initialization. In the demonstration program, LIN_Init() sets the global variable **LIN_Active** to true.

Main sets the global count variable **g_count** to 0x80 and displays it on the LED display. After initialization, the display shows “80”.

The slave main() routine then calls LIN_M_ReceiveCmd() to prepare to receive a command from the master on the LIN bus. This routine prepares the UART6 peripheral to receive a Sync Break Field, sets the global error variable **LIN_S_Error** to OK, and sets the global flag **LIN_S_ReceiveDone** to false.

The slave main() routine then enters a loop, waiting for a switch to be pressed, or for the **LIN_S_Error** variable or **LIN_S_ReceiveDone** flag to signal a LIN event.

If you press SW2, main() decrements the **g_count** variable by three and displays the new value. Because the initial value is 0x80, repeated presses of SW2 show 7D, 7A, 77, etc. Main() waits for the switch positions to change before continuing, allowing it to recognize the state of both switches being pressed.

If you press SW3, main() increments **g_count** by seven and displays the new value; from the initial 0x80, repeated presses of SW3 show 87, 8E, 95, etc. Main() waits for the switch positions to change, then continues.

If the routine sees both SW2 and SW3 closed, it reads the BRGC6 register and displays the value on the LED display. The BRGC6 register controls UART6’s baud rate. Initially the rate is set to 0xD0, but the routine adjusts the rate after receiving **SYNC** bytes from the master. Pressing both SW2 and SW3 allows the value of BRGC6 to be examined. Main() waits for both switches to be up before continuing.

If the **LIN_S_ReceiveDone** flag is true, it indicates that the slave received a command from the master. This communication may be a full message frame containing **SYNC** and **ID** bytes, indicating data from master to slave, data bytes, and checksum; or the communication may be just **SYNC** and **ID** bytes, indicating data should be sent from the slave to the master. In the case of **LIN_S_ReceiveDone** set true, main() calls the LIN_S_ProcessCmd() routine to handle the data received or to send requested data.

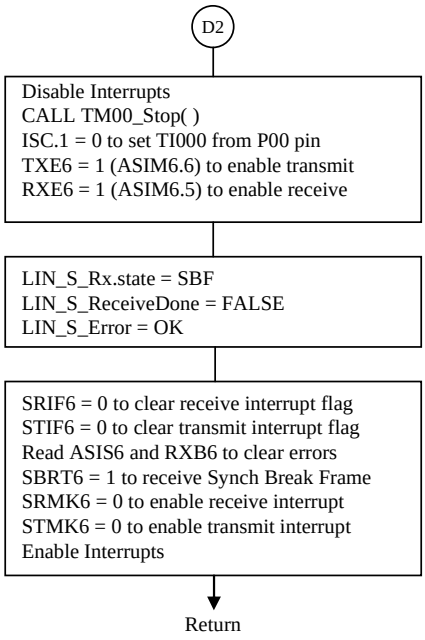
LIN_S_ProcessCmd() returns a status code. If this code indicates an error, LIN_S_ReceiveDone displays an “E” on the left LED and a four-bit error-code type on the right LED. Main() then calls LIN_S_ReceiveCmd() again to prepare for the next LIN command.

A value other than OK in the **LIN_S_Error** variable indicates that an error occurred during LIN reception or transmission. The routine displays an “E” plus the error code and calls LIN_S_ReceiveCmd() to prepare to receive the next command.

If no switch is pressed, and there is no indication of a LIN command or error, main() returns to the top of the loop and checks again.

3.3.17 Slave: LIN_S_ReceiveCmd() – Prepare to Receive Command/Data

Figure 27. Flowchart for Preparing to Receive Command/Data



The LIN_S_ReceiveCmd() routine prepares the slave to receive a message-frame header from the master. All message frames begin with the Sync Break Field (SBF), so the routine configures UART6 to receive the SBF.

The routine disables interrupts during this setup. It stops Timer 00 (TM00) and sets its TI000 input to the P00/TI000 pin by setting bit 1 of the input-switch control register (ISC) to zero.

The routine sets the TXE6 and RXE6 bits in the ASIM6 register to 1 to enable UART6 transmit and receive operation, in case baud-rate reprogramming disabled them.

The LIN_S_Rx structure, of type LIN_XFER_STRUCT, controls message-frame handling. The routine sets the state member to SBF, to indicate that the slave is waiting to receive the Sync Break Field. The routine also sets the global flag **LIN_S_ReceiveDone** false, to indicate that the routine has not received a command, and the **LIN_S_Error** variable to OK.

To configure UART6 to receive an SBF, LIN_S_ReceiveCmd() clears the transmit and receive interrupt flags, and reads the ASIS6 and RXB6 registers to clear any reception errors. The routine sets the SBRT6 bit in the ASICL6 register to 1. This value instructs the UART6 receiver to ignore any received data, and only

give an INTSR6 interrupt when a Sync Break Field is received. UART6 recognizes an SBF when the RXD6 input goes low and stays low for more than 10.5 bit times—longer than a standard 8-bit character with parity, which would consist of 1 low start bit, 8 data bits, and 1 high stop bit, for a maximum low time of nine bit times if the data were 0x00.

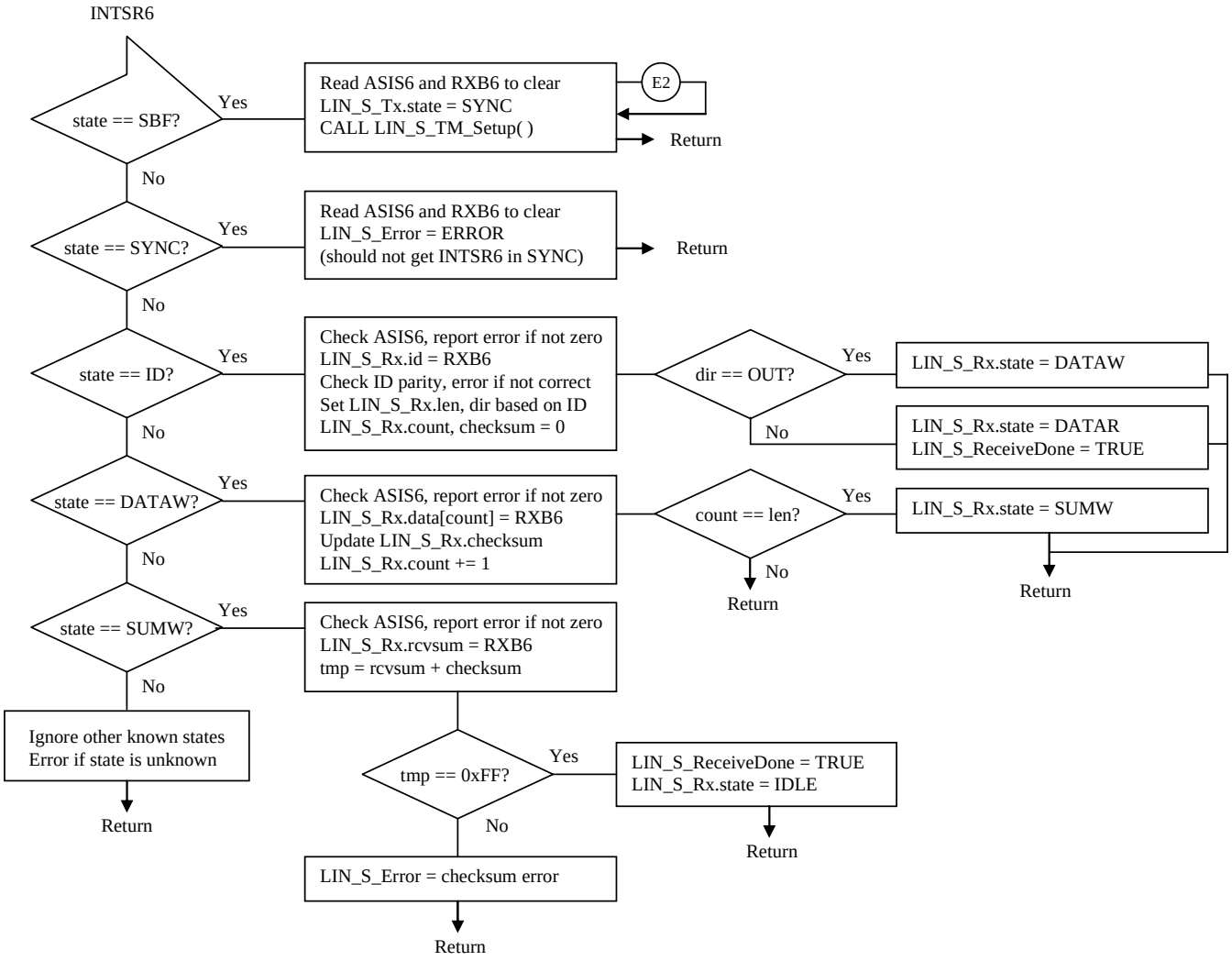
The LIN_S_ReceiveCmd() clears the receive and transmit interrupt masks to enable interrupts INTSR6 and INTST6. The routine also sets the master interrupt enable bit (IE, or PSW.7) to enable interrupts.

Now the UART6 responds to the SBF with an INTSR6 interrupt and ignores any other received characters. Thus, if an error is detected mid-frame, the slave can resynchronize to the beginning of the next message frame from the master.

The LIN_S_Rx_Handler() routine, the interrupt-service routine for INTSR6, takes care of further message-handling chores.

3.3.18 Slave: LIN_S_Rx_Handler() – UART6 Receive Character Interrupt-Service Routine

Figure 28. Flowchart for UART6 Receive Character Interrupt-Service Routine



The INTSR6 interrupt—which occurs when a Sync Break Field is received, a character is received, or a reception error occurs—invokes the LIN_S_Rx_Handler() interrupt-service routine. This interrupt-service routine either signals an error, by setting the **LIN_S_Error** variable to a value other than OK, or the successful end of reception, by setting **LIN_S_ReceiveDone** to true.

Once the program has called LIN_S_ReceiveCmd(), this interrupt should only occur when UART6 receives an SBF. The routine recognizes the SBF state, sets the state to SYNC, and calls LIN_S_TM_Setup() to set up the timer to measure the coming **SYNC**-byte bit widths.

LIN_S_TM_Setup() disables the receive interrupt; the routine now expects to see ID, not SYNC. The routine reports an error if it sees the SYNC state.

The next receive interrupt occurs after reception of the **ID** byte. The routine checks the ASIS6 register and sets an appropriate error if ASIS6 is non-zero. If there is no error, the routine stores the received data in the **id** member, and checks the **ID** byte's parity bits. If they are correct, the routine sets the **dir** and **len** structure members from the appropriate bits and sets the count and checksum members to zero. If the parity bits are incorrect, the routine reports an error.

If the data direction specified by the **ID** byte is IN (from slave to master), the message-frame header is complete, and the slave sends data. The routine sets the state to DATAR and the **LIN_S_ReceiveDone** flag to true. The main() routine then recognizes that the header has been received and prepares data to send.

If the direction specified by the **ID** byte is OUT (data from master to slave), more characters will be received. The next bytes expected will be data, so the state is set to DATAW.

The next receive interrupt should occur after receiving the first data byte. The routine sees the DATAW state, checks ASIS6 for errors, and stores the received character in the data[] array, indexed by the count member. The routine updates the checksum to reflect the received data and increments **count**. When **count** equals **len**, the slave has all the data bytes, and the routine sets the state to SUMW, expecting to receive the checksum. If **count** is still less than **len**, the routine just returns, leaving the state as DATAW. The next receive interrupt will indicate a data byte.

Once LIN_S_Rx_Handler() sets the state to SUMW, the routine expects the next bye it receives to be the checksum. The routine checks the ASIS6 register and reports an error if this register is non-zero. If there is no error, the routine stores the received checksum byte in **rcvsum**. The routine then adds the received checksum and the calculated checksum values together. If the sum is 0xFF, the checksum is correct.

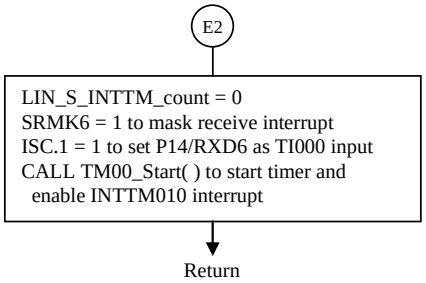
The correct checksum indicates that the slave received the data from the master correctly, and the message frame is complete. LIN_S_Rx_Handler() sets the **LIN_S_ReceiveDone** flag to true and the state to IDLE.

If the sum of **rcvsum** and the checksum is not 0xFF, LIN_S_Rx_Handler() sets **LIN_S_Error** to indicate a checksum error.

If the state is other than the above states, an unexpected receive interrupt has occurred. The routine ignores the data if it knows the state. If the state is invalid, the routine sets **LIN_S_Error** to indicate a generic error.

3.3.19 Slave: LIN_S_TM_Setup() – Timer Setup to Measure SYNC Bit Times

Figure 29. Flowchart for Timer Setup to Measure SYNC Bit Times



When the LIN_S_Rx_Handler() interrupt-service routine sees and SBF state, the routine calls the LIN_S_TM_Setup() routine to set up TM00 to measure received-character bit widths and to calculate the baud rate based on the **SYNC** byte.

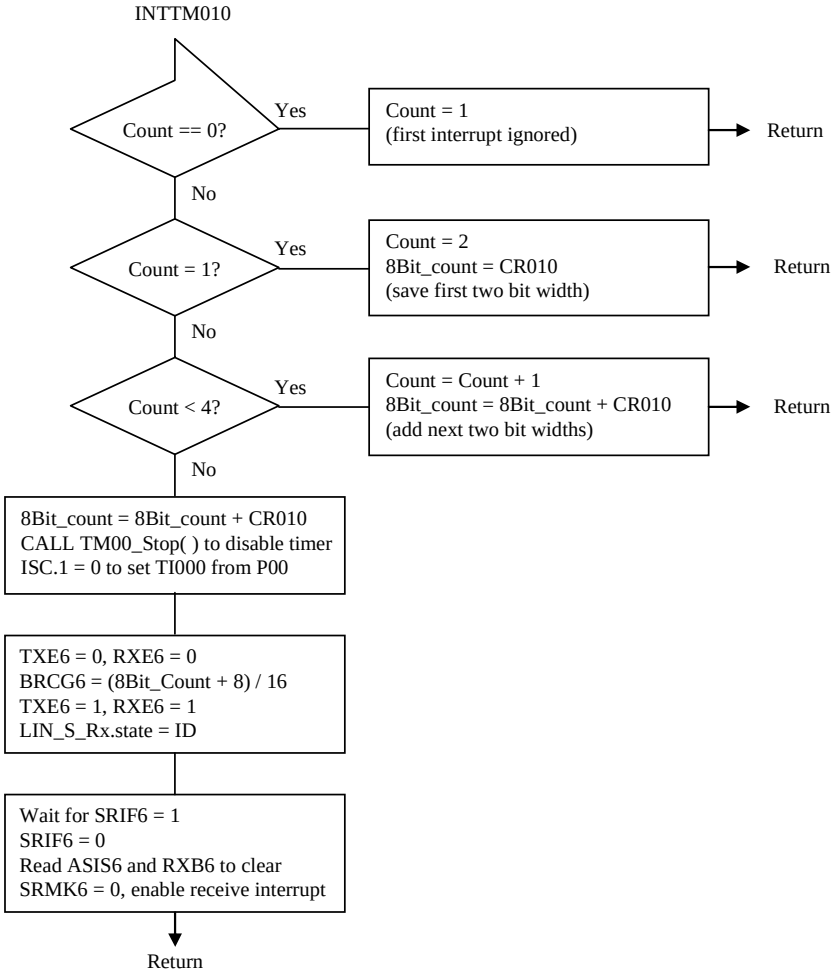
The LIN_S_TM_Setup() routine sets a counter for INTTM010 interrupts to zero, masks the serial-receive interrupt, and sets the input-switch control register (ISC) to route the P14/RXD6 pin to the TI000 timer input. The routine then calls TM00_Start(), which puts TM00 into the clear-and-start mode and enables the INTTM010 interrupt.

After this setup, TM00 counts up at the peripheral-clock rate (8 MHz). When a falling edge of RXD6 occurs (at the start bit of the **SYNC** byte), the MCU copies TM00 to CR000. When a rising edge of RXD6 occurs (at the beginning of bits 0, 2, 4, 6 of the **SYNC** byte, and at the stop bit), the MCU copies TM00 to CR010, clears TM00 to zero, and invokes the INTTM010 interrupt.

The LIN_S_INTTM_isr() routine, which handles the INTTM010, measures the width of **SYNC** byte bits and adjusts the slave's baud rate to match that of the master.

3.3.20 Slave: LIN_S_INTTM_isr() – Timer 00 Interrupt-Service Routine

Figure 30. Timer 00 Interrupt-Service Routine



The LIN_S_INTTM_isr() routine is invoked when the INTTM010 interrupt occurs on the rising edge of TI000 (coming from RXD6). The routine measures the bit width of the **SYNC** byte and adjusts the slave's baud rate to match that of the master.

The **LIN_S_INTTM**_count variable stores the number of occurrences of the INTTM010 interrupt. On the first interrupt, the count is zero (as set in the LIN_S_TM_Setup() routine). This interrupt occurs on the rising edge of RXD6, when the start bit (zero) changes to the LSB of the **SYNC** byte (0x55, so bit 0 is a logic one). This event happens at some undefined time after TM00 starts and the routine ignores it, except for setting the count to 1.

On the second interrupt, the count is 1. This time the rising edge of RXD6 occurs when bit 1 of the 0x55 **SYNC** byte (logic zero) changes to bit 2 (logic one). CR010 contains the value TM00, which LIN_S_INTTM_isr() cleared at the previous rising edge, but the routine now stores in CR010 the number of TM00 counts (0.125 microseconds each) from the start of bit 0 to the start of bit 2. CR010 therefore now contains two bits worth of counts. The routine stores this value in the **8Bit_count** variable.

On the third and fourth interrupts, the count is two and three, respectively. CR010 again has two bits worth of counts. LIN_S_INTTM_isr() adds these counts to the **8Bit_count** variable.

On the fifth interrupt, the count is four. This interrupt occurs on the rising edge of RXD6 when bit 7 of the 0x55 **SYNC** byte (logic zero) changes to the stop bit (logic one). This bit marks the end of the **SYNC** byte, and CR010 contains the two-bit width of bits 6 and 7. Adding this value to **8Bit_count** means that the variable now contains the number of counts in 8 bits of the **SYNC** byte.

To adjust the slave's baud rate to match the master, LIN_S_INTTM_isr() uses the number of counts to change the value in the BRGC6 baud-rate register. The BRGC6 register should be set to the number of counts in one bit, divided by two. The routine has collected the time for 8 bits, in order to average bit times. So the routine adds 8 to the total to round the result, then shifts right by four bits to divide by 16.

For example, the original value written to BRGC6 was 0xD0, or 208. The main 8-MHz frequency, divided by this value and divided by two again, produces the bit rate: 8,000,000 / (208 * 2) = 19230.7, or about 19,200 baud.

If the **SYNC** byte from the master is sent at exactly 19,200 baud, 8 bits take 416.666 microseconds. If the slave clock runs at exactly 8 MHz, each TM00 clock takes 0.125 microseconds, and during transmission of 8 bits count up 416.666 / 0.125 (or 416.666 * 8) = 3333 counts. By adding 8 to this count and shifting right four bits, you end up with 208 (0xD0) for BRGC6—the same value you started with.

But if the slave clock runs at 8.8 MHz (10% fast), each TM00 count takes only 1/8,800.00 = 0.1136 microseconds. During an 8-bit **SYNC** byte, this value results in 416.666 / 0.1136 = 3666 counts. Adding 8 and shifting right four bits produces the new BRGC6 value of 229, or 0xE5. Writing this value to BRGC6 produces a new baud rate of 8,800,000 / (229 * 2) = 19214 baud.

LIN_S_INTTM_isr() writes the new value for BRGC6 after turning off the UART6 transmit and receive functions by setting TXE6 and RXE6 in the ASIM6 register to zero. This disabling of transmit and receive is recommended when changing BRGC6. After changing the BRGC6 value, resetting TXE6 and RXE6 to one re-enables transmit and receive.

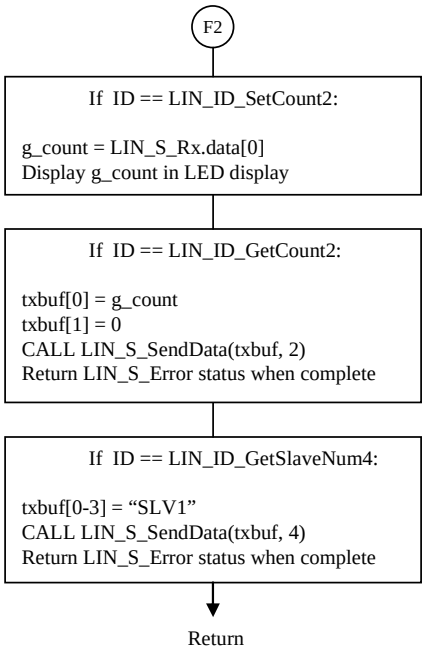
The handler sets the LIN_S_Rx.state variable to **ID**, to indicate the end of the SYNC state, and waits for the **ID** byte.

Because UART6 has received the **SYNC** byte, LIN_S_INTTM_isr() sets the **SRIF6** interrupt flag at the end of the stop bit. When the handler sees the interrupt flag, it clears the flag to zero and reads ASIS6 and RXB6 to clear any errors or characters. The routine then sets the SRMK6 interrupt mask to zero to enable INTSR6 interrupts on received characters.

The next character received will be the **ID** byte, and the LIN_S_Rx_Handler() routine handles the receive interrupt INTSR6 for that byte.

3.3.21 Slave: LIN_S_ProcessCmd() – Handle Received Data, Prepare Data to Send

Figure 31. Handle Received Data, Prepare Data to Send



Main () calls LIN_S_ProcessCmd() when the LIN_S_ReceiveDone flag is true, which indicates the slave received a command (or possibly data) from the master. The LIN_S_ProcessCmd() routine handles data sent to the slave, or prepares data and sends it to the master.

The action in this routine is application-dependant. Specifically, the action depends on the value of the **ID** byte received from the master. The demonstration application program defines three ID commands.

On the LIN_ID_SetCount2 command, the master sends its local count, **m_count**. The slave receives this data, and sets its local count, **g_count**, to the received value and displays the new **g_count** on the LED display.

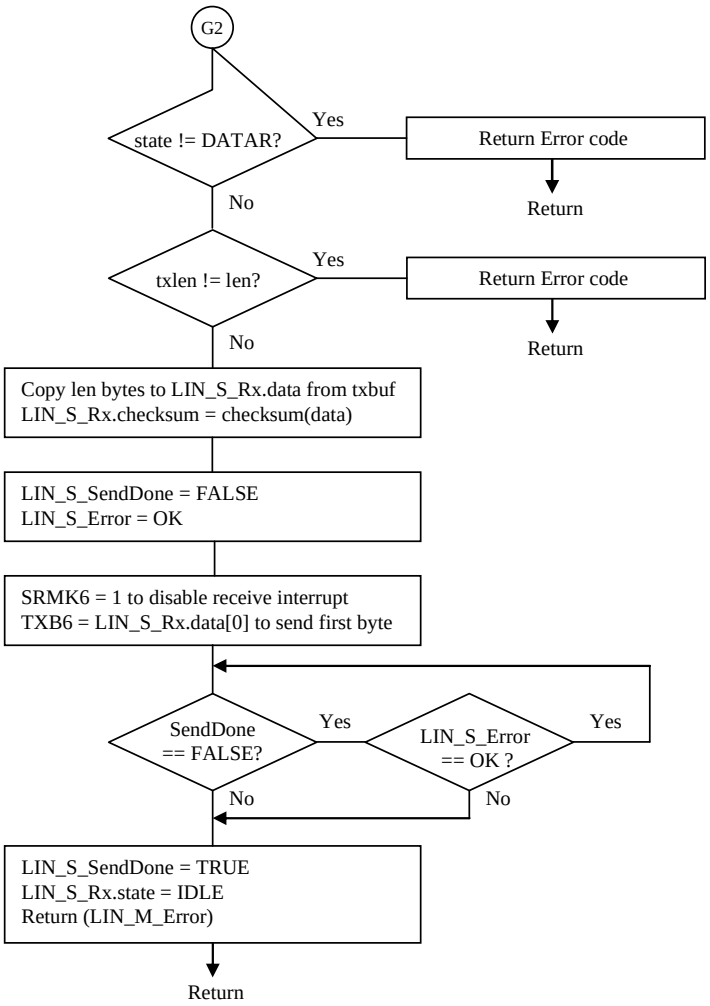
On the LIN_ID_GetCount2 command, the master sends a command requesting the slave's count value and waits for data from the slave. The slave program puts **g_count** into a two-byte transmit buffer and calls LIN_S_SendData(txbuf, 2) to send the two bytes of data to the master. This routine returns a status of OK or an error indicator, depending on whether the data was properly transmitted to the master.

On the LIN_ID_GetSlaveNum4 command, the master sends a command requesting the slave's four-byte identifier. The slave places the string "SLV1" in the txbuf array and calls LIN_S_SendData(txbuf, 4) to transmit this data to the master. The command returns the error status to main().

The slave ignores any other values of the **ID** byte than these three. In an actual application, there could be other slaves on the bus which process more commands.

3.3.22 Slave: LIN_S_SendData(*txbuf, txlen) – Send Data to Master

Figure 32. Send Data to Master



LIN_S_ProcessCmd() calls the LIN_S_SendData(*txbuf, txlen) routine to send requested data from the slave to the master. The **txbuf** parameter points to a buffer holding the data, and **txlen** is the buffer length in bytes. The routine starts sending the data, waits for the transmission to complete, and returns the status of the send.

The routine first checks for the proper state (DATAR) and that the length specified by **txlen** matches the **len** length set by the **ID** byte received. If these values are correct, the routine copies the data pointed to into the data[] array in the LIN_S_Rx structure. The routine calculates the checksum member from the data bytes to send.

For the LIN protocol, the routine creates the checksum by adding the data bytes together, incrementing the sum if a carry is generated for an addition. The routine inverts the resulting sum for sending. The same algorithm is used for received data. Adding the received and calculated checksums together should result in a value of 0xFF.

LIN_S_SendData() sets the global flag **LIN_S_SendDone** to false and the global variable **LIN_S_Error** to OK.

The routine masks the interrupt for UART6 received data or errors. When the slave transmits characters, they are echoed by the LIN transceiver and received by UART6. The transmit-done interrupt-service routine handles the receive-character interrupt flag.

To begin sending data, the routine writes the first data byte to the TXB6 register. After transmission of the first byte, the INTST6 interrupt occurs, and LIN_S_Tx_Handler() handles the sending of subsequent data.

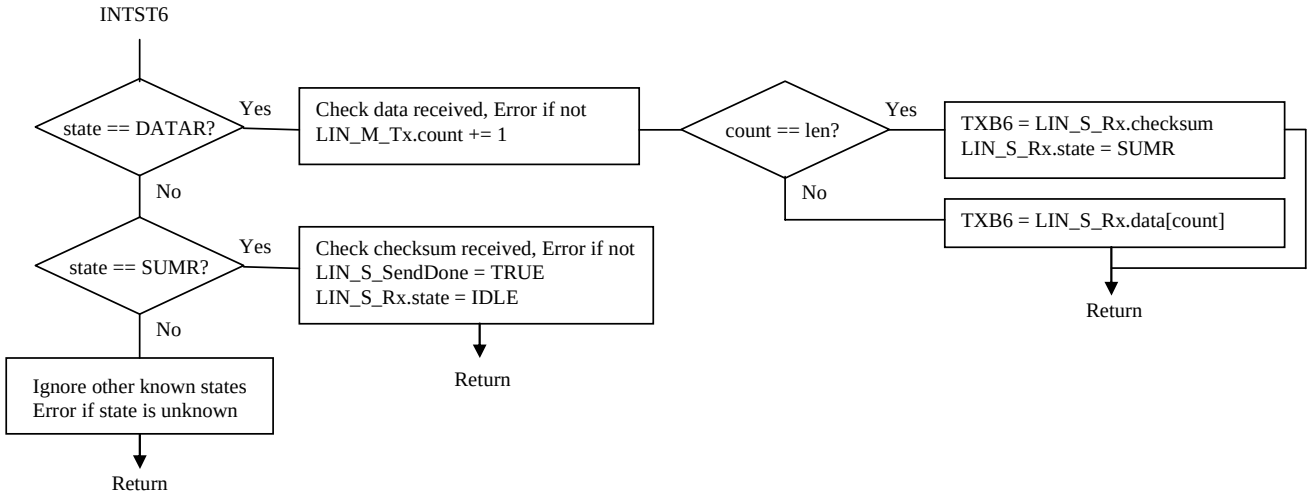
LIN_S_SendData() then loops, waiting for either **LIN_S_SendDone** to be true or **LIN_S_Error** to be set to a value other than OK. The LIN_S_Tx_Handler() interrupt-service routine sets these flags. When one of these flags is set, **LIN_S_SendDone** is forced to true, and the routine returns the value in the **LIN_S_Error** global variable.

If **LIN_S_Error** is still set as OK, the transfer completed successfully, and the routine ended because **LIN_S_SendDone** was true. In this case, LIN_S_SendData() returns OK.

If **LIN_S_Error** is not OK, an error occurred somewhere in the transfer, and the LIN_S_SendData() routine returns the error code.

3.3.23 Slave: LIN_S_Tx_Handler() – UART6 Transmit-Done Interrupt-Service Routine

Figure 33. Transmit-Done Interrupt-Service Routine



At the end of a character transmission, the INTST6 interrupt occurs and invokes the LIN_S_Tx_Handler() interrupt-service routine. This routine signals either an error, by setting **LIN_S_Error** to a value other than OK, or the successful end of transmission, by setting **LIN_S_SendDone** to true.

Because the LIN transceiver also receives the transmitted data, reading the transceiver's receive interrupt flags and register indicates if the data was transmitted properly.

The LIN_S_SendData() routine puts the system in the DATAR state by writing the first data byte to the TXB6 register.

The first transmit interrupt occurs after sending the first data byte. LIN_S_Tx_Handler() sees the DATAR state, checks if the data byte was received as sent, and returns an error if the byte is incorrect. If the data is correct, the routine increments the **count** member. If **count** equals **len**, all data bytes have been sent, and the routine sends the checksum, sets the state to SUMR, and returns. If there is more data to send, the routine sends the next data byte and returns, keeping the state as DATAR. The next transmit interrupt occurs at the end of that data byte, and LIN_S_Tx_Handler() sees the DATAR state again.

After transmission of the checksum byte, the next transmit interrupt occurs. The routine sees the SUMR state, checks if the checksum was received as sent, and returns an error if the checksum is incorrect. If the checksum is correct, the LIN_S_Tx_Handler() sets the **LIN_S_SendDone** flag to true and the state to IDLE.

3.4 Applilet's Reference Drivers

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

The Applilet produces the basic initialization code, the program's main function, driver code for the UART6 and TM00 peripherals, as well as code to manage the I/O ports used for switch input and display output, and timers that provide switch debouncing and delays. After the Applilet produces the basic code, you can add code to customize the program.

The Applilet does not produce specific routines to support the LIN protocol. You must add the code for LIN master and LIN slave support.

This section describes how to set up the Applilet to produce code for UART6 and TM00 (used by the slave), and lists the routines produced, as well as some additional files.

The LIN demonstration uses two separate NEC Electronics microcontrollers and thus requires two separate Applilet projects. One project handles the LIN master processor—command transmission. The other, for the LIN slave processor, handles command reception. Both master and slave use UART6 to transmit and send data.

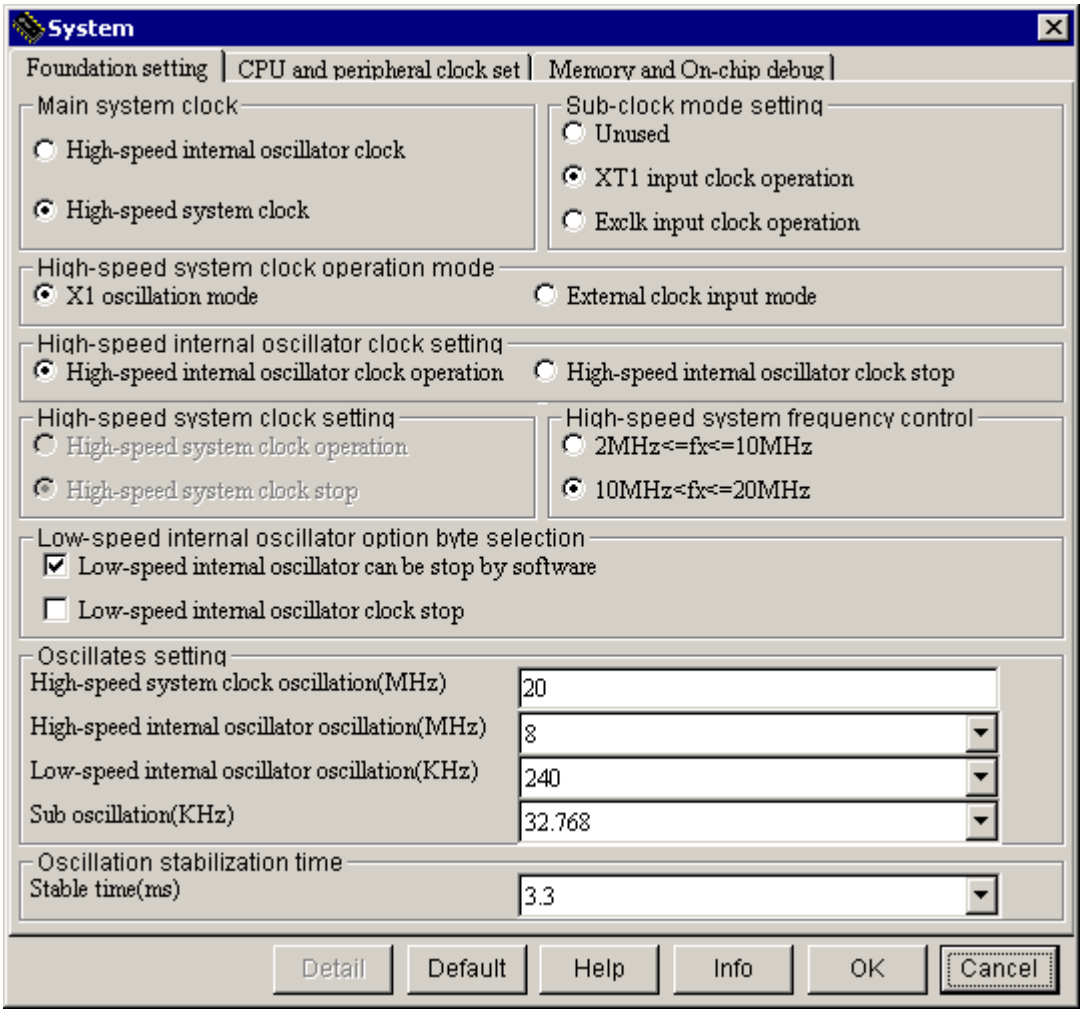
The master and slave operate on the same hardware platform. To demonstrate how a LIN slave adapts its baud rate to that of the master, these microcontrollers use different clock speeds and sources. The master program uses an external 20-MHz crystal oscillator, and the slave uses an internal 8-MHz oscillator, with some possible frequency variations.

When you start the Applilet and select the target device, save your settings to a new project (.prx) file. The Applilet displays a dialog box that lets you select the peripheral blocks you want to set up.

3.4.1 Master: Configuring the Applilet for 20-MHz External Clock

The master program uses a 20-MHz external crystal clock. Running the Applilet and selecting **System** brings up a clock-configuration dialog.

Figure 34. Configuring External Clock



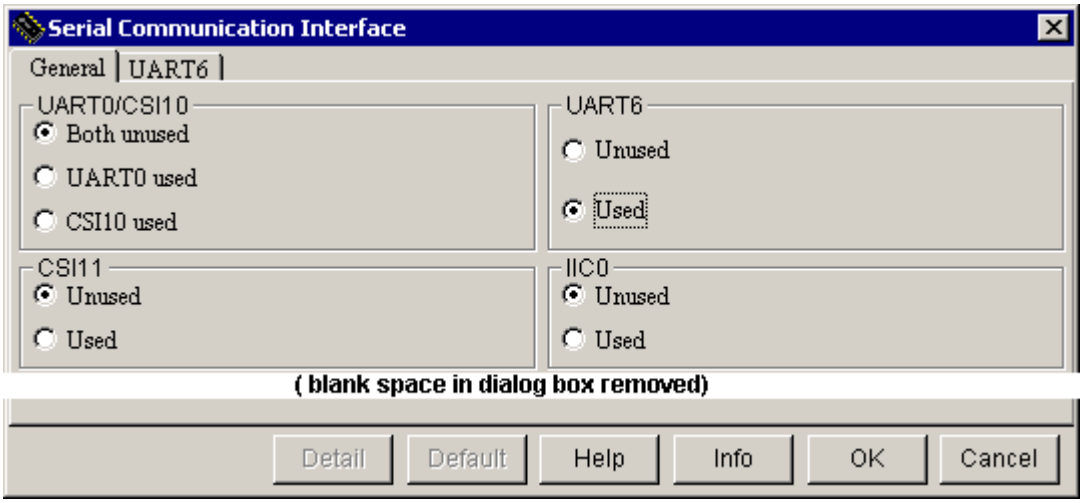
Under **Main system clock** select **High-speed system clock** to choose an external clock source rather than the internal oscillator. Set **High-speed system clock operation mode** to **X1 oscillation mode** to accommodate a crystal attached to the X1/X2 pins, rather than a driven external clock.

Under **Oscillation setting** set **High-speed system clock oscillation(MHz)** to 20, to specify a 20-MHz external crystal frequency. The Applilet uses this setting to calculate the divider values for baud-rate generation and timer periods.

3.4.2 Master: Configuring Applilet for UART6

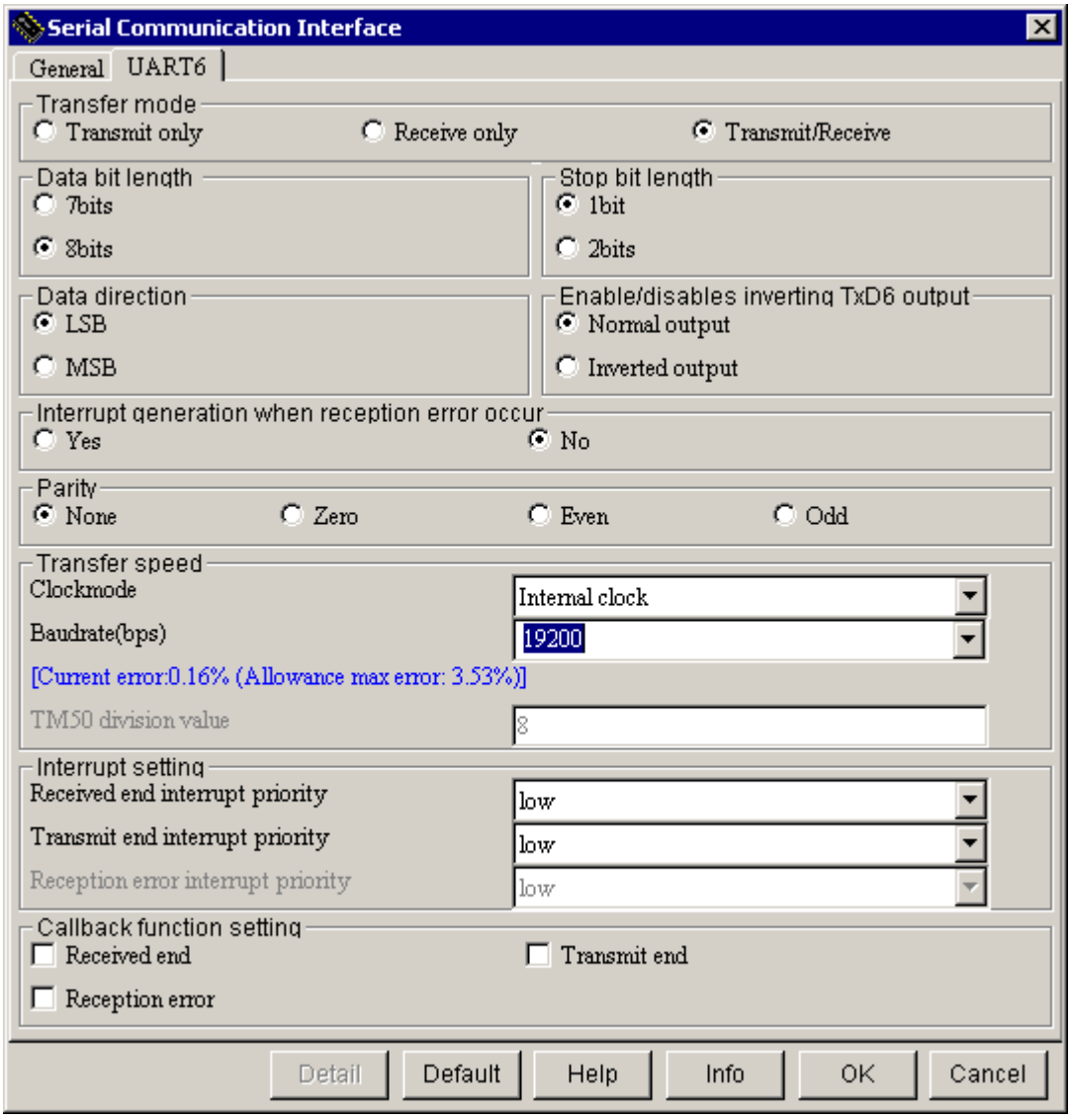
The master processor uses the UART6 peripheral for LIN communication. In the Applilet master project select **Serial Communication Interface** to bring up a dialog showing the various serial blocks. Select UART6 as used.

Figure 35. Configuring Applilet for UART6



Once you select UART6, clicking on **Detail** brings up the UART6 detail dialog.

Figure 36. UART6 Detail Dialog



Select the transmit/receive **Transfer mode**, since the master needs to send LIN commands and data, and receive data from the slave. Set **Data-bit length** to 8 bits, with 1 stop bit, LSB transmitted first, normal (non-inverted) output, and no parity. The clock source (**Clockmode**) is the internal clock, and the baud rate is 19200. The Applilet generates values for the CKSR6 and BRGC6 registers, which control the baud rate, based on the baud-rate selection and the frequency of the system clock.

UART6 generates interrupts after transmitting characters, receiving characters, and on a receive error. The Applilet generates interrupt-service routines for these interrupts. You can have receive errors generate a separate interrupt or generate the same interrupt as received characters. Selecting No under **Interrupt generation when reception error occur** causes receive errors to generate the same INTSR6 interrupt as received characters. Note that for this program you need to replace the Applilet-generated interrupt-service routines with LIN functions.

Select low priority for both receive and transmit interrupts. Within this group, receive interrupts have higher priority than transmit.

Do not select any of the **Callback function settings.** These settings provide hooks for user code when interrupts occur. This demonstration uses LIN functions for interrupt handling, so the callback option is not needed.

3.4.3 Master: Generating Code With Applilet

Once you have set up the UART6 and other dialog boxes, click **Generate code.** The Applilet shows the peripherals and functions to be generated, and you can select a directory for storing the source code.

When you click **Generate**, 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 UART6 serial device, 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.

3.4.4 Master: Applilet-Generated UART6 Files and Functions

The files serial.h, serial.c, and serial_user.c contain the code generated for UART6 support.

3.4.4.1 Serial.h

The header file serial.h contains definitions for the UART6 functions and values for baud rate setting. 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.

3.4.4.2 Serial.c

The source file serial.c contains the following functions generated by the Applilet.

void UART6_Init(void)
This routine initializes the UART6 peripheral as specified in the Applilet UART6 dialog.

MD_STATUS UART6_ReceiveData(UCHAR* rxbuf, UCHAR rxnum)
This routine sets up a receive operation, requesting rxnum characters be received to the rxbuf buffer. The LIN application program uses LIN replacement functions rather that this routine.

MD_STATUS UART6_SendData(UCHAR* txbuf, UCHAR txnum)

This routine sets up a transmit operation of txnum characters from the txbuf buffer. The routine also starts the transmission operation by sending the first one or two characters. The LIN application program uses LIN replacement functions rather that this routine.

__interrupt void MD_INTSR6(void)

As the interrupt-service routine for the receive interrupt INTSR6, this routine checks for reception errors (in case INTSRE6 is using the same vector), puts the received character in the rxbuf buffer, and decrements the receive count.

You need to add code to intercept this interrupt for when the LIN_M_Active variable is true, and redirect the INTSR6 interrupt to LIN_M_Rx_Handler() instead.

__interrupt void MD_INTST6(void)

This is the interrupt-service routine for the transmit done interrupt INTST6. If the transmit is done, the interrupt is ignored. Otherwise, this routine sends the next character and decrements the transmit count.

You must add code to intercept this interrupt for when the LIN_M_Active variable is true, and redirect the INTST6 interrupt to LIN_M_Tx_Handler() instead.

3.4.4.3 Serial_user.c

The source file Serial_user.c contains stub functions for user code. These functions are empty on code generation so that you can add application-specific code. Since you do not select any callback functions, this file is empty.

3.4.5 Master: Other Applilet-Generated Files

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

Table 7. Other Applilet-Generated 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
Port.h	Definitions for default I/O port states
Port.c	PORT_Init() function
Timer.h	Definitions for timer settings and functions
Timer.c	Timer management functions
Timer_user.c	Timer interrupt ISR (timer is used to debounce pushbuttons)
Option.inc	Option-byte, POC, and security definitions
Option.asm	Option-byte, POC, and security data

3.4.6 Master: Demonstration Program LIN Files and Functions

For LIN master support, the files LIN.h, LIN_M.h and LIN_M.c are added to the demonstration program.

Table 8. Demonstration Program LIN Files and Functions

File	Function
LIN.h	General header file for LIN structure and command definitions
LIN_M.h	Header file for LIN_M.c, declaring data and functions for master
LIN_M.c	LIN master functions and data

3.4.6.1 LIN.h

The header file LIN.h contains definitions for values used for LIN ID commands, as well as bit and mask definitions for **ID** fields. The file defines LIN error codes and states, and defines the LIN_XFER_STRUCT structure. Both the LIN master and slave applications use this header file.

3.4.6.2 LIN_M.h

The header file LIN_M.h contains declarations for functions and global variables defined in LIN_M.c.

3.4.6.3 LIN_M.c

The source file LIN_M.c contains the following functions for LIN master support.

void LIN_M_Init(void)
This routine initializes LIN hardware. UART6_Init() has already initialized the UART6. In the demonstration program, LIN_M_Init() just sets the **LIN_Active** variable true.

void LIN_M_WaitHalfBit(unsigned int halftimes)
During data transmission, this routine uses a timer to wait for half of one bit time. This wait ensures that the stop bit has had time to transmit before checking for a received character. The wait also allows time to add inter-character delays at some points in the message frame.

UCHAR LIN_M_CalcIdParity(UCHAR ID)
This routine calculates the double parity bits for the **ID** byte.

void LIN_M_SendByte(UCHAR data)
This routine writes data to TXB6 for transmission. A separate routine provides a convenient place to intercept transmitted bytes for debugging.

MD_STATUS LIN_M_SendData(UCHAR ID, UCHAR *txbuf)
This routine initiates transmission of command and data from the master to the slave. The **ID** parameter contains the command to send, and the txbuf parameter points to a buffer containing the data to send. The length of the transfer is specified by bits in the ID command.

MD_STATUS LIN_M_ReceiveData(UCHAR ID, UCHAR *rxbuf)
This routine starts transmission of a command to the slave and waits for data reception from the slave. The **ID** parameter contains the command to send, and the rxbuf parameter points to a received-data buffer. Bits in the ID command specify the transfer length.

void LIN_M_Tx_Handler(void)
This is the interrupt-service routine for the INTST6 interrupt, generated at the end of character transmission or the end of SBF transmission.

void LIN_M_Rx_Handler(void)

This is the interrupt-service routine for the INTSR6 interrupt, generated on character received, reception error, or SBF reception.

3.4.7 Master: Other Added Demonstration Program Files

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

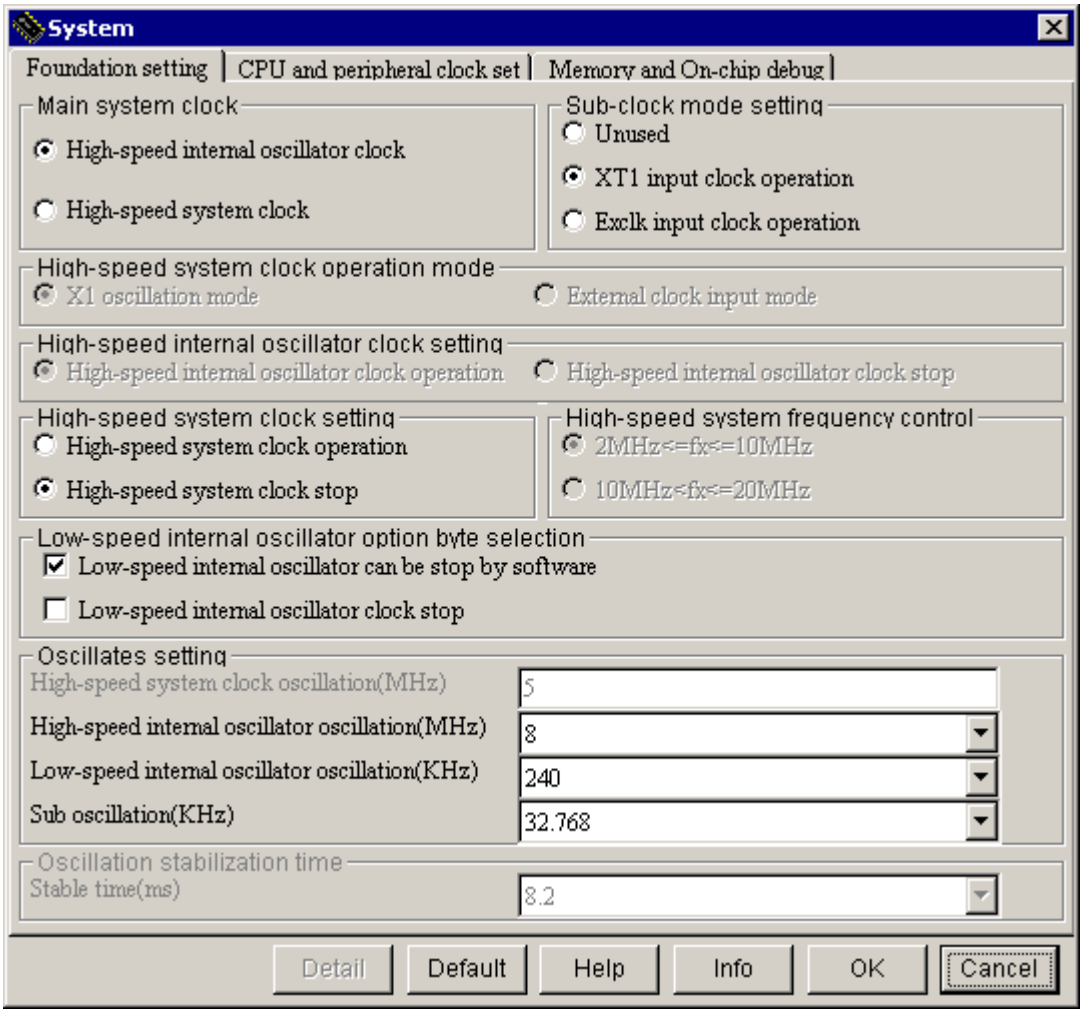
Table 9. Files Not Generated by Applilet

File	Function
Sw_0537.h	Header file for pushbutton switch input
Sw_0537.c	Code to read and debounce pushbutton switches
Led_0537.h	Header file for seven-segment LED patterns and functions
Led_0537.c	Code to display data in seven-segment LED displays

3.4.8 Slave: Configuring Applilet for 8-MHz Clock from Internal Oscillator

The slave program uses the microcontroller's 8-MHz internal oscillator clock. To configure this clock, select the Applilet's **System** block.

Figure 37. Configuring Applilet for 8-MHz Clock



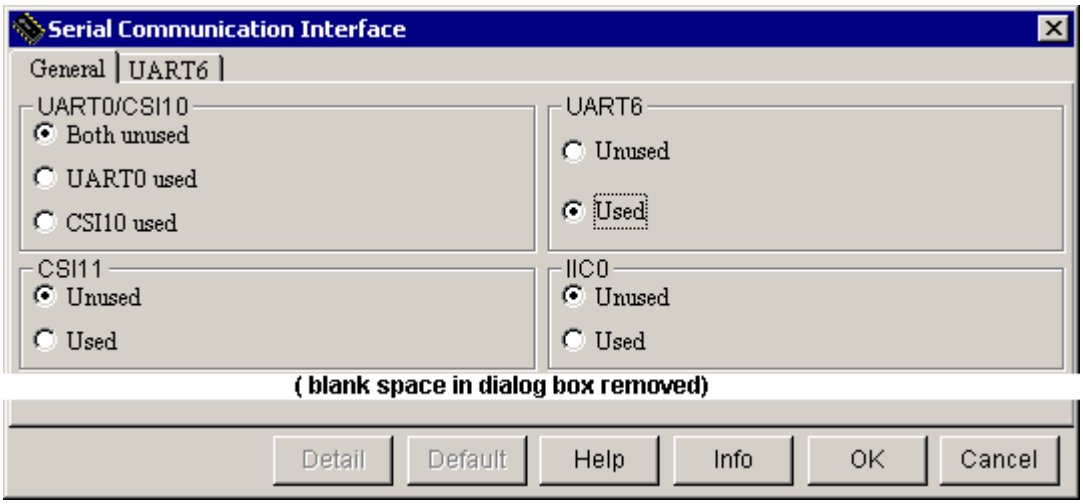
Set **Main system clock** for **High-speed internal oscillator clock**, which selects the high-speed internal-oscillator source rather than an external source. Choosing **High-speed system stop clock** as the **High-speed system clock setting** option frees the X1 and X2 pins for I/O ports. The other external-clock settings are grayed out.

Under **Oscillates setting**, the **High-speed system clock oscillation(MHz)** setting is a default 5 MHz, but is grayed out. Set the **High-speed internal oscillator oscillation(MHz)** option to 8 MHz, which is the nominal frequency of the high-speed internal-oscillator. The Applilet uses this setting to calculate the divider values for baud-rate generation and timer periods.

3.4.9 Slave: Configuring Applilet for UART6

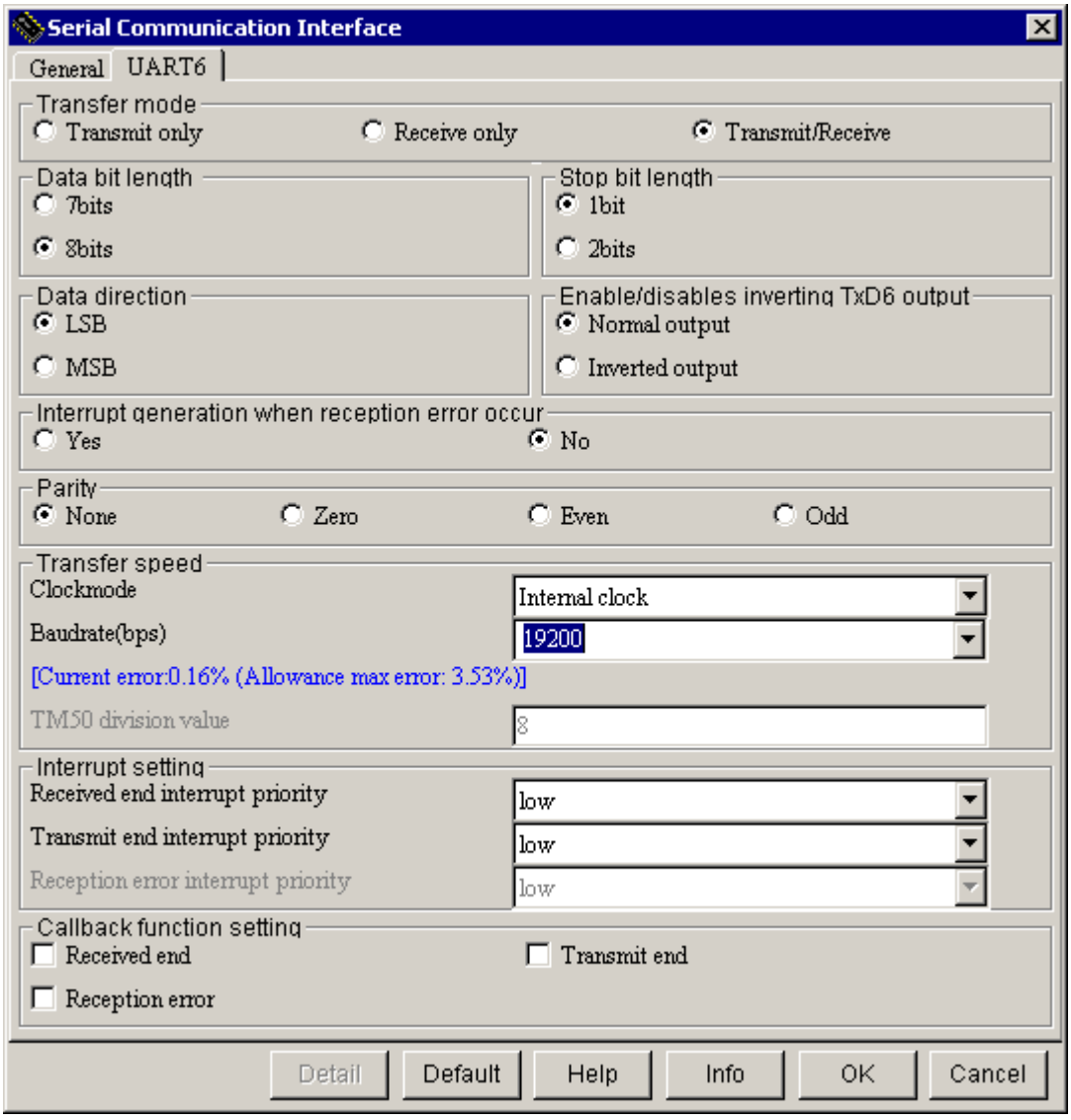
The slave processor uses the UART6 peripheral for LIN communication. In the Applilet project for the slave processor, select the **Serial** block to bring up a dialog showing the various serial blocks.

Figure 38. Configuring the Applilet for the Slave UART6



Select **UART6** as **Used**, then click **Detail** to bring up the UART6 detail dialog box.

Figure 39. UART6 Detail Dialog



Select **Transmit/Receive** for the **Transfer mode**, as the slave must receive LIN commands and data, and send data back to the master on request. Set **Data bit length** for 8 bits, with 1 stop bit, LSB transmitted first, normal (non-inverted) output, and no parity. The clock source (**Clockmode**) is the internal clock, and the baud rate is 19200 baud. The Applilet generates values for the CKSR6 and BRGC6 registers, which control the baud rate, based on this selection and the system clock's frequency.

The UART6 generates interrupts at the end of character transmission, when it receives a character, and when there is a receive error. The Applilet creates interrupt-service routines for these interrupts. You can choose to have receive errors generate a separate interrupt, or use the same interrupt as for received characters. Under **Interrupt generation when reception error occur**, selecting **No** causes receive errors

to generate the same INTSR6 interrupt as received characters. Note that LIN functions will replace the Applilet-generated interrupt-service routines.

Select low priority for receive and transmit interrupts. Within this group, the receive interrupts have a higher priority than transmit interrupts.

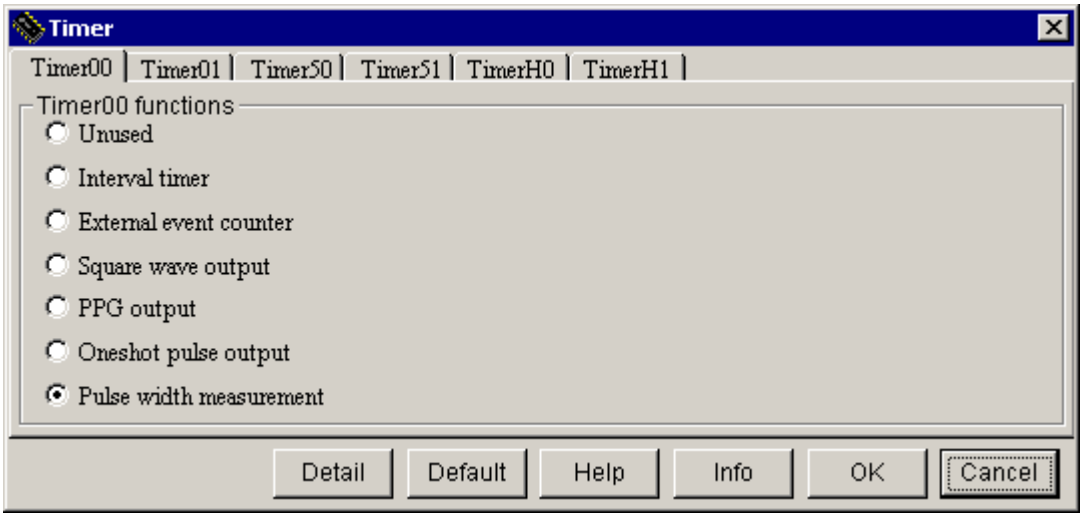
Do not select any callback functions. These functions provide hooks for user code when interrupts occur. The demonstration program uses LIN functions to handle interrupts, so the callback option is not needed.

3.4.10 Slave: Configuring Applilet for TM00

The slave processor measures the bit widths of the **SYNC** byte with Timer 00 (TM00).

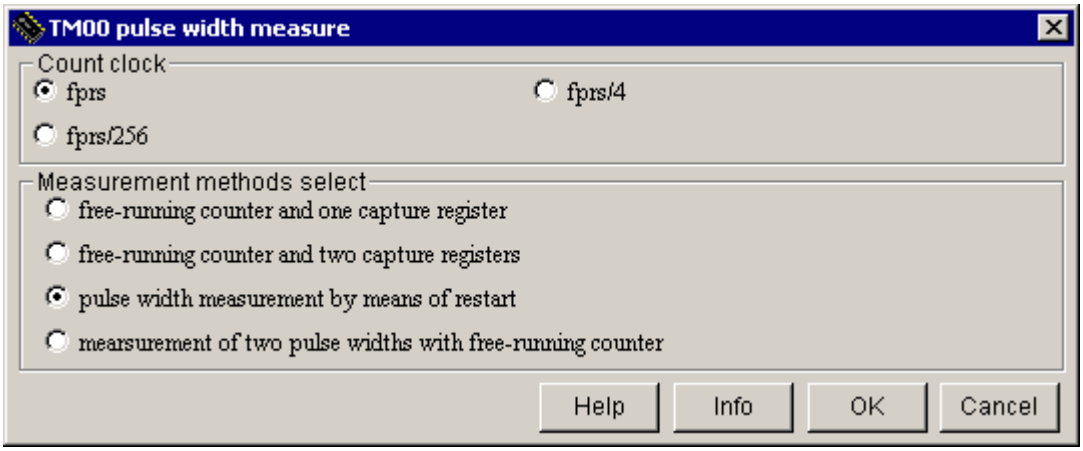
When running the Applilet, selecting **Timer** brings up a timer-block dialog. Select **Timer00** and set it for **Pulse-width measurement**.

Figure 40. Timer Dialog



Clicking **Detail** brings up the detail dialog for TIMER00 pulse-width measurement settings.

Figure 41. TIMER00 Pulse-Width Measurement Settings



Set **Count clock** to **fprs** to obtain the fastest available system clock. This setting provides the greatest timing accuracy and also matches the clock selected for UART6 baud-rate generation. Select **pulse width measurement by means of restart** to make the timer run in its clear-and-start mode.

The Applilet automatically creates the interrupt-service routine for the INTTM010 interrupt, which occurs at the end of timer pulse-width measurement. There is no option to select the interrupt or set its priority group.

3.4.11 Slave: Generating Code With Applilet

Once you have set up the UART6, TM00, and other dialog boxes, select **Generate code**. The Applilet displays a list of the peripherals and functions to be generated, and allows you to select a directory for storing the source code.

When you click **Generate**, 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 UART6 serial device, the Applilet generates the files serial.h, serial.c, and serial_user.c.

The Applilet generates timer.h, timer.c, and timer_user.c to support the TM00 timer, and other timers the program uses.

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

3.4.12 Slave: Applilet-Generated UART6 Files and Functions

The files serial.h, serial.c, and serial_user.c contain the code generated for UART6 support.

3.4.12.1 Serial.h

The header file serial.h contains definitions for the UART6 functions and values for the baud-rate setting. 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.

3.4.12.2 Serial.c

The source file serial.c contains the following functions.

void UART6_Init(void)
This routine initializes the UART6 peripheral as specified in the Applilet UART6 dialog.

MD_STATUS UART6_ReceiveData(UCHAR* rxbuf, UCHAR rxnum)
This routine sets up a receive operation, requesting rxnum characters be received in the rxbuf buffer. The LIN application program does not use this routine, but uses LIN replacement functions instead.

MD_STATUS UART6_SendData(UCHAR* txbuf, UCHAR txnum)
This routine sets up a transmit operation of txnum characters from the txbuf buffer and initiates transmission by sending the first one or two characters. The LIN application program does not use this routine, but uses LIN replacement functions instead.

__interrupt void MD_INTSR6(void)
This is the interrupt-service routine for the receive interrupt INTSR6. The routine checks for reception errors (in case INTSRE6 is using the same vector), puts the received character in the rxbuf buffer, and decrements the receive count.

You must add code to intercept this interrupt for when the **LIN_S_Active** variable is true, and redirect the INTSR6 interrupt to LIN_M_Rx_Handler().

__interrupt void MD_INTST6(void)
This is the interrupt-service routine for the transmit-done interrupt INTST6. If the transmit is complete, the routine ignores the interrupt. Otherwise the routine sends the next character and decrements the transmit count.

You must add code to intercept this interrupt for when the LIN_S_Active variable is true, and redirect the INTST6 interrupt to LIN_M_Tx_Handler().

3.4.12.3 Serial_user.c

The source file Serial_user.c contains stub functions for user code. The Applilet generates empty functions, and you can add application-specific code. Because you set up the Applilet without any callback functions, this file is empty.

3.4.13 Slave: Applilet-Generated Files and Functions for TM00 and Other Timers

The code generated for TM00 for pulse-width measurement is in files timer.h, timer.c and timer_user.c. These files also contain code for timers TM50 and TM51, which provide switch debouncing and interval measurement. This description covers only the TM00-related functions.

3.4.13.1 Timer.h

The header file timer.h contains declarations for the functions controlling the timers 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.

3.4.13.2 Timer.c

The source file timer.c contains the following functions.

void TM00_Init(void)
The TM00_Init() routine initializes Timer 00 for pulse-width measurement.

void TM00_Start(void)
The TM00_Start() routine starts Timer 00 by enabling the timer and the interrupt INTTM010 (used in pulse-width measurement).

void TM00_Stop(void)
The TM00_Stop() routine stops Timer 00 by disabling the timer and the timer interrupt.

MD_STATUS TM00_GetPulseWidth(*activewidth, *inactivewidth, channel)
The Applilet-generated TM00_GetPulseWIdth() routine obtains a measured pulse width, which is used by the interrupt-service routine for INTTM010. The demonstration program does not use this routine, but rather uses specialized code in the LIN slave to measure **SYNC** bit width.

3.4.13.3 Timer_user.c

The source file timer_user.c contains stub functions for user code.

__interrupt void MD_INTTM010(void)
This is the interrupt-service routine for Timer 00 interrupt INTTM010, generated when a valid edge occurs on the TI000 input for pulse-width measurement. The code uses the values in CR010 and CR000 to calculate pulse widths.

The demonstration program uses specialized code in the LIN slave to measure **SYNC** bit width. You must add code to this routine to intercept the INTTM010 interrupt for when the **LIN_Active flag** is true, and redirect interrupt service to the LIN_S_INTTM_isr() routine.

3.4.14 Slave: Other Applilet-Generated Files

For the demonstration program, the Applilet generates several other source files. The files and their functions are shown below.

Table 10. Additional Applilet-Generated 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
Port.h	Definitions for default I/O port states
Port.c	PORT_Init() function
Option.inc	Option-byte, POC, and security definitions
Option.asm	Option-byte, POC, and security data

3.4.15 Slave: Demonstration Program LIN Files and Functions

For LIN slave support, you must add the files LIN.h, LIN_S.h and LIN_S.c to the demonstration program.

Table 11. Demonstration Program LIN Files and Functions

File	Function
LIN.h	General header file for LIN structure and command definitions
LIN_S.h	Header file for LIN_S.c, declaring data and functions for slave
LIN_S.c	LIN slave functions and data

3.4.15.1 LIN.h

The header file LIN.h contains definitions for values used for LIN ID commands, as well as bit and mask definitions for **ID** fields. LIN.h defines LIN error codes and states, and the LIN_XFER_STRUCT structure. This header file is used by both the LIN master and slave applications.

3.4.15.2 LIN_S.h

The header file LIN_S.h contains declarations for functions and global variables defined in LIN_S.c.

3.4.15.3 LIN_S.c

The source file LIN_S.c contains the following functions for LIN slave support.

void LIN_S_Init(void)
This routine initializes LIN hardware. UART6_Init() initializes the UART6. In the demonstration program, LIN_S_Init() just sets the **LIN_Active** variable true.

void LIN_S_WaitHalfBit(unsigned int halftimes)
This routine uses a timer to wait for half of one bit time. This wait is performed during a data transmission to ensure that the stop bit has had time to transmit before the device checks for a received character.

UCHAR LIN_S_CalcIdParity(UCHAR ID)
This routine calculates the double parity bits for the **ID** byte.

MD_STATUS LIN_S_ReceiveCmd(void)
This routine sets up the UART6 peripheral to prepare the slave for Sync Byte Field reception. The routine sets the **LIN_S_ReceiveDone** flag to false. This flag is set to true once a command and data are received from the master.

void LIN_S_SendByte(UCHAR data)
This routine writes data to TXB6 for transmission. A separate routine provides a convenient place to intercept transmitted bytes for debugging.

MD_STATUS LIN_S_SendData(UCHAR *txbuf, UCHAR txlen)

This routine starts the sending of data from the slave to the master. The txbuf parameter points to a buffer containing the data to send. The txlen parameter specifies the number of bytes to send; the routine compares this parameter with the length of the transfer as specified by the ID command.

void LIN_S_Tx_Handler(void)

This is the interrupt-service routine for the INTST6 interrupt, generated at the end of character transmission.

void LIN_S_Rx_Handler(void)

This is the interrupt-service routine for the INTSR6 interrupt, generated on character received, reception error, or SBF reception.

void LIN_S_TM_Setup(void)

After an SBF is received, this routine sets up the TM00 timer to measure **SYNC** byte bit widths.

void LIN_S_INTTM_isr(void)

This is the interrupt-service routine for the INTTM010 interrupt, generated on rising edges at the P14/RXB6 pin (sent to the TI000 input). After the **SYNC** byte is completed, the program uses the measured value to modify the slave's baud rate.

3.4.16 Slave: Other Added Demonstration-Program Files

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

Table 12. Other Added Demonstration-Program Files

File	Function
Sw_0537.h	Header file for pushbutton-switch input
Sw_0537.c	Code to read and debounce pushbutton switches
Led_0537.h	Header file for seven-segment LED patterns and functions
Led_0537.c	Code to display data in seven-segment LED displays

3.5 Demonstration Platform

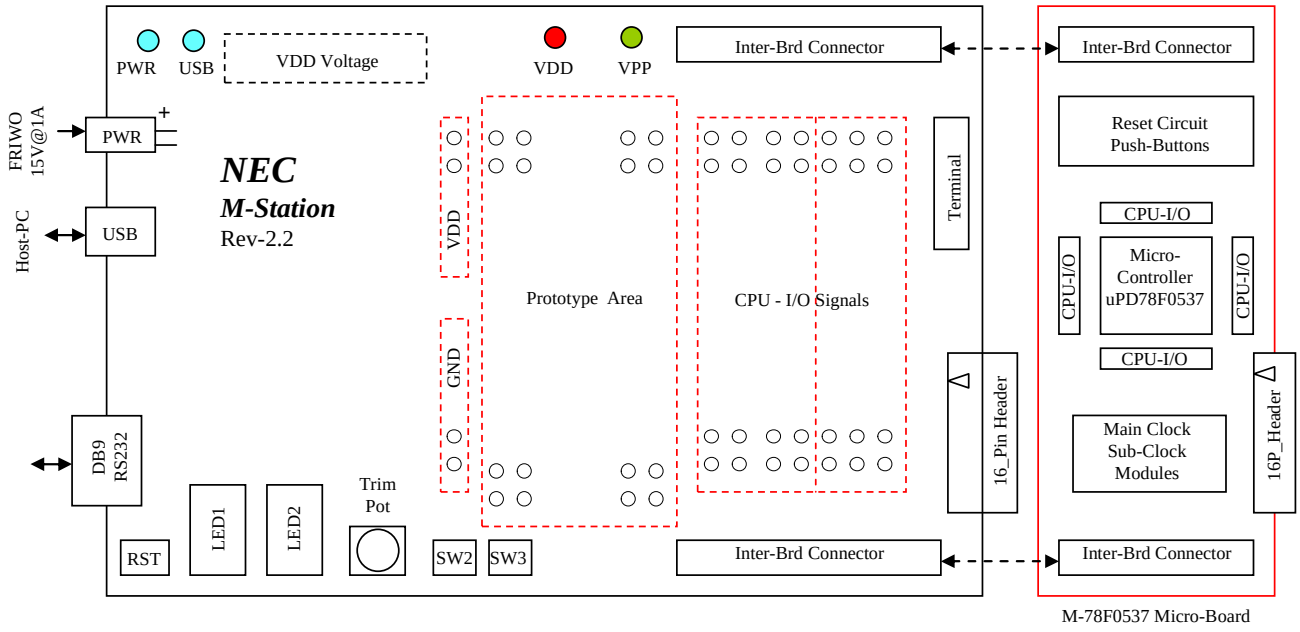
The demonstration uses a development board from NEC Electronics. You may be able to duplicate the hardware using off-the-shelf components along with the NEC Electronics' microcontroller of interest.

3.5.1 Resources For LIN Master

The LIN master program demonstration uses the following resources:

- M-78F0537 Micro-Board, with uPD78F0537 8-bit microcontroller mounted
- uPD78F0537 resources:
  - UART6 peripheral, using P13/TXD6 and P14/RXD6
  - External 20-MHz crystal connected to X1 and X2 pins
- M-Station II Evaluation System, using M-Station II resources:
  - 7-Segment LED displays LED1 and LED2
  - Pushbutton switches SW2 and SW3
  - LIN transceiver for LIN bus interface
  - TB1 terminal block for connection of LIN and GND signals

Figure 42. LIN Master Demonstration Platform



3.5.2 Resources For LIN Slave

To demonstrate the LIN slave program, a similar hardware configuration is used:

- M-78F0537 Micro-Board, with uPD78F0537 8-bit microcontroller mounted
- uPD78F0537 resources:
  - UART6 peripheral, using P13/TXD6 and P14/RXD6
  - TM00 peripheral, using P14/RXD6 as input
  - 8-MHz high-speed internal-oscillator
- M-Station II Evaluation System, using M-Station II resources:
  - 7-Segment LED displays LED1 and LED2
  - Pushbutton switches SW2 and SW3
  - LIN transceiver for LIN bus interface
  - TB1 terminal block for connection of LIN and GND signals

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

3.5.3 Program Demonstration

With the hardware configured and the uPD78F0537 microcontroller programmed with the demonstration code, the demonstration is as follows:

On initial power-up, the LED displays on the master M-Station II show "=1" to indicate that test 1 will be performed if you press SW3. The LED display on the slave M-Station II shows "80" for the **g_count** value. For the following inputs, the displays are as shown, with independent modification of local count values on both the master and slave.

Table 13. Master and Slave Displays with Local Counts Modified

Input	Master M-Station	Slave M-Station	Comment
Master SW3	00	80	Display **m_count** on master, value is 00
Master SW3	0F	80	Increase **m_count** on master, value is 0F
Master SW3	1E	80	Increase **m_count**, value is 1E
Slave SW3	1E	87	Increase **g_count** on slave, value is 87

The following table shows commands and data sent from the master to the slave, with the slave acting on the received data.

Table 14. Master and Slave Displays with Slave Acting on Received Data

Input	Master M-Station	Slave M-Station	Comment
Master SW2	=1	87	Return to main() on master
Master SW2	=2	87	Select test = 2 on master
Master SW3	1E	87	Display **m_count** (1E) before sending to slave
Master SW3	1E	1E	LIN command from master, sending **m_count**; slave sets **g_count** to received data, now 1E

The next table shows the slave adapting its baud rate to match the master.

Table 15. Master and Slave Displays with Slave Adapting Baud Rate to Match Master

Input	Master M-Station	Slave M-Station	Comment
Slave RESET	1E	80	Slave is reset, original count restored
Slave SW2 and SW3		D0	See initial slave BRGC6 register
Master SW3	1E	1E	LIN command, master sends count 1E to slave
Slave SW2 and SW3		D2*	Slave BRGC6 value modified from **SYNC** byte

*Note: The value shown for the slave varies depending on the exact clock rate. The value shown indicates that the 8-MHz slave clock was slightly high and the routine increased the value in BRGC6 to lower the baud rate.

The next table shows a command sent from the master to the slave. The slave responds by sending data back to the master, and the master acts on the received data.

Table 16. Master and Slave Displays for Command Sent From Master to Slave

Input	Master M-Station	Slave M-Station	Comment
Slave SW3	1E	25	Add 7 to **g_count** on slave
Slave SW3	1E	2C	Add 7 to **g_count** on slave
Slave SW3	1E	33	Add 7 to **g_count** on slave
Master SW2	=2	33	Return to main() on master
Master SW2	=3	33	Select test = 3 on master
Master SW3	3_	33	In Test 3 on master, before command to slave
Master SW3	33	33	LIN command from master, requesting **g_count**; slave sends **g_count** data (33); master sets **m_count** (33)
Slave SW3	33	3A	Add 7 to **g_count** on slave, now 3A
Master SW3	3A	3A	LIN command to get **g_count**, set **m_count**

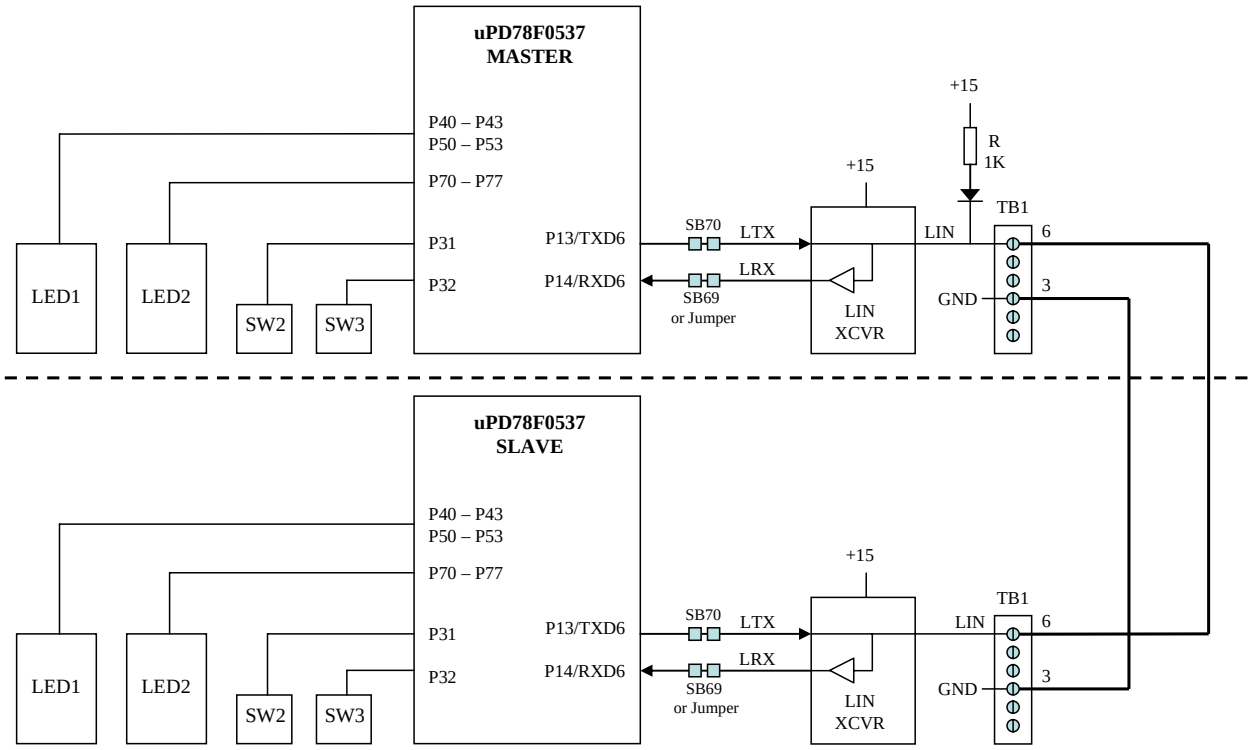
The final table of display values shows the master sending a command requesting four bytes, the slave responding with the data, and the master acting on the received data. This test (=4) is similar to test=3 in the mechanics of the LIN transfer, but the number of bytes is different. The slave and master handle the data sent and received differently based on the **ID** value.

Table 17. Master and Slave Displays for Test 4

Input	Master M-Station	Slave M-Station	Comment
Master SW2	=3	3A	Return to main() on master
Master SW2	=4	3A	Select test = 4 on master
Master SW3	4_	3A	In Test 4 on master, before command to slave
Master SW3	4_ then 53, 4C, 56, 31 4_	3A 3A	LIN command from master, requesting 4 byte **ID** Slave sends "SLV1" (53, 4C, 56, 31) Master displays bytes received, returns to Test 4

3.6 Hardware Block Diagram

Figure 43. Demonstration Hardware Block Diagram



The M-78F0537 Micro-Boards are mounted on two M-Station II base boards, and the demonstration uses the M-Station default connection of ports to SW2, SW3 and seven-segment LED displays.

For LIN connection, connect P13/TXD6 to the LTX (LIN Transmit) input pin of the LIN transceiver, shorting through solder block SB70 on both M-Stations.

To avoid conflicts on some Micro-boards, the LIN transceiver's received-data line LRX (LIN Receive) of M-Station 2 is not connected to the processor by default. You can connect the transceiver LRX to RXD6 with shorting solder block SB69. However, for the M-78F0537 Micro-Board, this connection interferes with using UART6 for flash programming. The actual demonstration system has an added jumper in the prototyping area for both the master and slave. The jumper allows you to make or break the SB69 connection by inserting or removing the jumper.

For LIN bus systems, attach a 1K pull-up resistor through a diode to the LIN line on the master only. On the M-Station II V2.1E board used, this 1K resistor and diode are added in the prototyping area. On M-Station II V2.2 and later systems, an SB shorting block is available for this purpose.

3.7 Software Modules

The following files make up the demonstration program software modules. The tables show which files were generated by the Applilet, and which need modification to create the demonstration programs.

The listings for these files are located in the Appendix.

3.7.1 Software Modules for LIN Master Program

Table 18. LIN Master 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
Serial.h	UART6-related definitions	Yes	No
Serial.c	UART6-related functions	Yes	Yes[Note 1]
Serial_user.c	User code for CSI callback routine	Yes	No
Port.h	Definitions for default I/O port states	Yes	No
Port.c	PORT_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
Timer_user.c	User code for timer-interrupt handling	Yes	Yes[Note 2]
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No
LIN.h	General header file for LIN	No	--
LIN_M.h	LIN master header file for LIN_M.c	No	--
LIN_M.c	LIN master functions and data	No	--
Led_0537.h	LED-display definitions	No	--
Led_0537.c	LED-display functions	No	--
Sw_0537.h	Switch-input definitions	No	--
Sw_0537.c	Switch-input functions	No	--

Note 1: Modify Serial.c to redirect the INTST6 interrupt to the LIN_M_Tx_Handler() routine and to redirect the INTSR6 interrupt to the LIN_M_Rx_Handler() routine.

Note 2: Modify Timer_user.c to have the MD_INTTM50() interrupt-service routine call the sw_isr() routine for switch debouncing.

3.7.2 Software Modules for LIN Slave Program

Table 19. LIN Master 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
Serial.h	UART6-related definitions	Yes	No
Serial.c	UART6-related functions	Yes	Yes[Note 1]
Serial_user.c	User code for CSI callback routine	Yes	No
Port.h	Definitions for default I/O port states	Yes	No
Port.c	PORT_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	Yes[Note 2]
Timer_user.c	User code for timer-interrupt handling	Yes	Yes[Note 2]
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No
LIN.h	General header file for LIN	No	--
LIN_S.h	LIN slave header file for LIN_M.c	No	--
LIN_S.c	LIN slave functions and data	No	--
Led_0537.h	LED-display definitions	No	--
Led_0537.c	LED-display functions	No	--
Sw_0537.h	Switch-input definitions	No	--
Sw_0537.c	Switch-input functions	No	--

Note 1: Modify Serial.c to redirect the INTST6 interrupt to the LIN_S_Tx_Handler() routine and to redirect the INTSR6 interrupt to the LIN_S_Rx_Handler() routine.

Note 2: Modify Timer_user.c to have the MD_INTTM50() interrupt-service routine call the sw_isr() routine for switch debouncing; redirect interrupt INTTM010 to the LIN_S_TMINT_isr() routine for pulse-width measurement. Modify Timer.c to correct an error in the code produced by the Applilet for TM00_Init() for ISC-register initialization.

Additionally, note that although the LIN master and LIN slave share the same set of Applilet-generated files, the code and values in these files are not the same, reflecting their different system clocks, clock frequencies, and differing use of timer peripherals.

4. Serial Communication Using CSI (Clocked Serial I/O)

The Clocked-Serial I/O (CSI) peripheral, also known as 3-wire serial I/O, uses three lines for Serial Clock (SCK), Serial-Data Input (SI) and Serial-Data Output (SO). Most NEC Electronics microcontrollers incorporate one or more of CSI peripheral channels for easy device interconnection. You can also configure this interface to connect to other devices supporting a 3-wire clocked-serial interface.

In 3-wire serial I/O communication, data is transmitted or received in 8-bit units. Each data bit is transmitted or received in synchronization with the serial clock. One side of the communication controls the clock line, so this is a master-slave configuration, typically with one master and one slave. If only one slave device drives data back to the master, you can have multiple slaves receiving data.

When your application's data transmission is bidirectional, using CSI can shorten transmission time, since you can transmit and receive simultaneously.

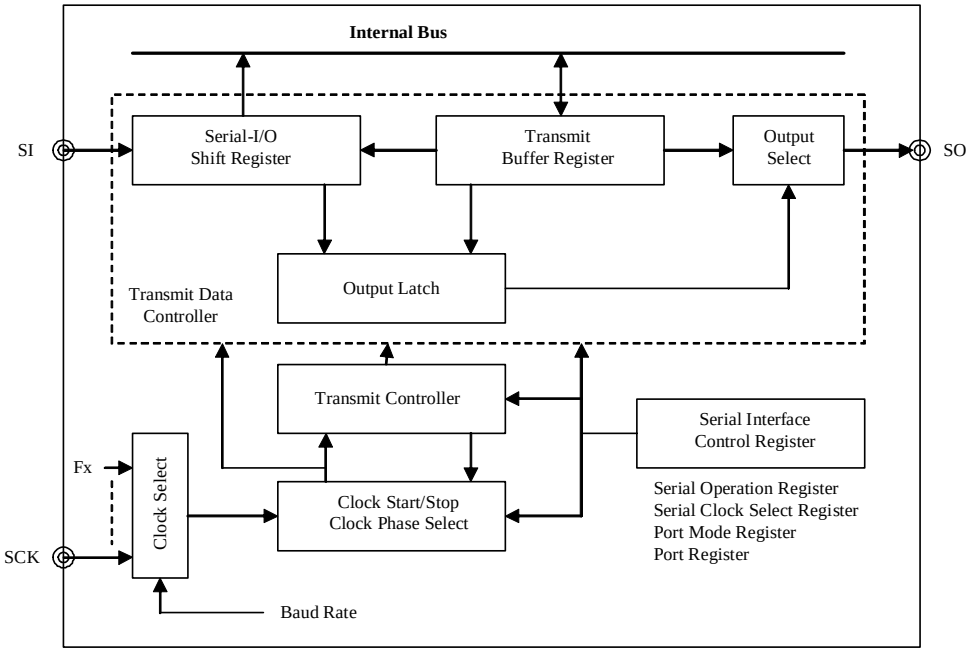
4.1 CSI Features

The clocked-serial I/O peripheral offers the following features:

- Automatic shifting of 8-bit data
- Selection of MSB-first or LSB-first serialization
- Simultaneous data transmit and receive
- Selectable internal-clock frequency for master mode, enabling up to 10 Mbps for some devices
- External clock for slave mode
- Selectable serial-clock phase (normally low or normally high)
- Selectable data-change point for transmission (before or after initial clock edge)
- Selectable data clock-in point for reception (clock's leading or trailing edge)
- External chip-select input to enable/disable transmission (on some CSI units)
- Interrupt on end of transmission or data-byte reception

When not using the CSI peripheral, you can use the SCK, SO and SI I/O pins as port pins.

Figure 44. Block Diagram of Typical CSI 3-Wire Serial I/O Peripheral



The registers controlling 3-wire serial-communication operations are shown below.

Table 20. CSI Control Registers

Registers	Functions
Serial-Operation Mode Register	Enable/Disable 3-wire serial communication Set transmit or receive mode Set MSB-first or LSB-first Indicate communication stop or communication in-progress
Serial Clock Selection Register	Specify timing of transmit/receive and sets serial clock
Port Mode Register	Set port mode (input or output)
Port Register	Select port functions

To configure the CSI peripheral for transmit/receive, use the following steps. The registers and bits listed are for the CSI11 peripheral:

- Set port-mode register bits to set input/output mode on appropriate pins, and set port-output latches to appropriate states.
- Set the clock source as external (slave mode), or select internal-clock frequency (master mode) by setting the serial-clock selection register CSIC11.
- Select clock and data polarity using the CCP11 and DAP11 bits in the CSIC11 register.

- Set MSB-first or LSB-first with the DIR11 bit in the serial-operation mode register CSIM11.
- Select whether to use external enable with the SSE11 bit in the CSIM11 register.
- Set the TRMD11 bit to one to select transmit/receive mode.
- Set the interrupt priority, clear the interrupt flag, and unmask the interrupt.
- Set the CSIE11 bit to one in the CSIM11 register to enable communication.

To configure the CSI peripheral for receive-only operation, use the same steps, except set the TRMD11 bit to zero for receive-only mode.

To begin data transmission, write a data byte to the transmit-buffer register (SOTB11 for this example).

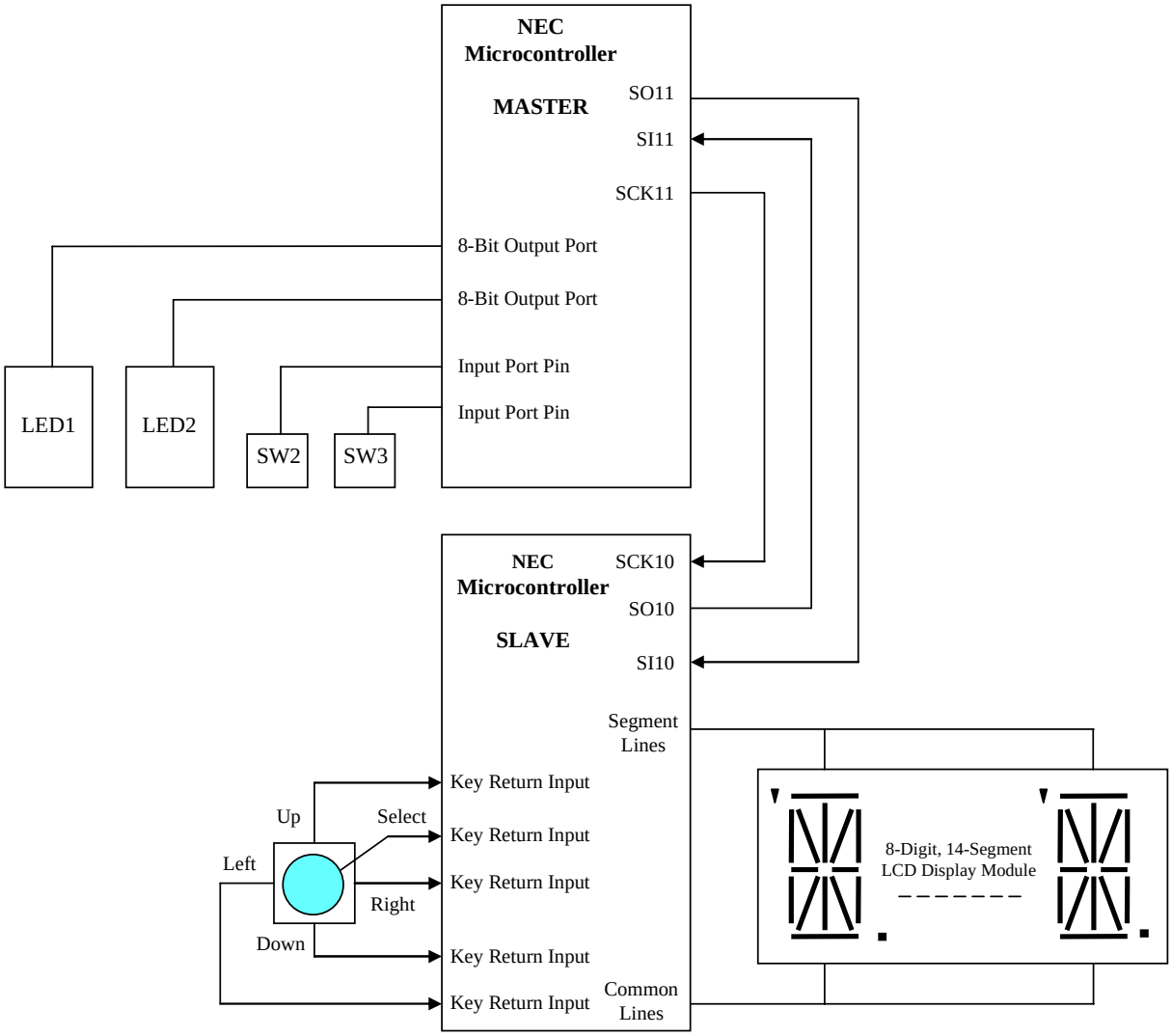
To begin data reception, read a dummy byte from the serial-I/O shift register (SIO11 for this example). After the data shifts, read the actual data from SIO11.

4.2 Program Description and Specification

The demonstration of CSI-peripheral data transmission using 3-wire serial I/O uses two NEC Electronics microcontrollers—a master device sending data and a slave device receiving data.

The master has a 2-digit LED display for showing data values and two switches to control operation. The slave has an 8-digit LCD for messages, and a 5-position switch for input. The master and slave are connected by 3-wire serial interface between their CSI peripherals. Power and ground interconnect to ensure correct voltage levels.

Figure 45. Block Diagram of CSI Demonstration Hardware



In operation, the master device displays a count on the LED displays. When you press SW2, the master increments the count. When you press SW3, the master sends the count to the slave. Initially, the slave displays the message “WAITING” on the LCD. When the slave receives data via CSI, the slave displays the message “RX nn” with “nn” being the value of the received data.

Specifications:

- The master uses CSI11 in transmit/receive mode, while the slave uses CSI10 in receive-only mode.
- The data transmission rate is 1 Mbps (clock cycle time is 1 microsecond).
- Data is sent MSB first, Type 3 (clock normally low, data clocked on clock trailing edge).

4.3 Software Flowcharts

The CSI demonstration requires two programs: one transmitting data (master) and one receiving data (slave).

The demonstration programs consist of the following major sections related to CSI operation:

- Program-initialization code, including CSI initialization, called before the main() program starts
- The main program loop, which checks switches and transmits data through the CSI, checks for received data and displays values
- Subroutines for sending and receiving CSI data, and interrupt-service routines related to CSI operation

The programs also include sections not related directly to CSI operation:

- Subroutines generated by the Applilet for port initialization for switches and LED display (master)
- Subroutines generated by the Applilet to handle timer starting, stopping, and interval setting (master and slave)
- Subroutines with user code for handling timer interrupts, which are used for switch debouncing (master and slave)
- Subroutines for switch input and LED-display output (master)
- Subroutines for key-return interrupt handling (slave)
- Subroutines for LCD-controller operation and IIC0 peripheral operation used to communicate with the LCD controller (slave)

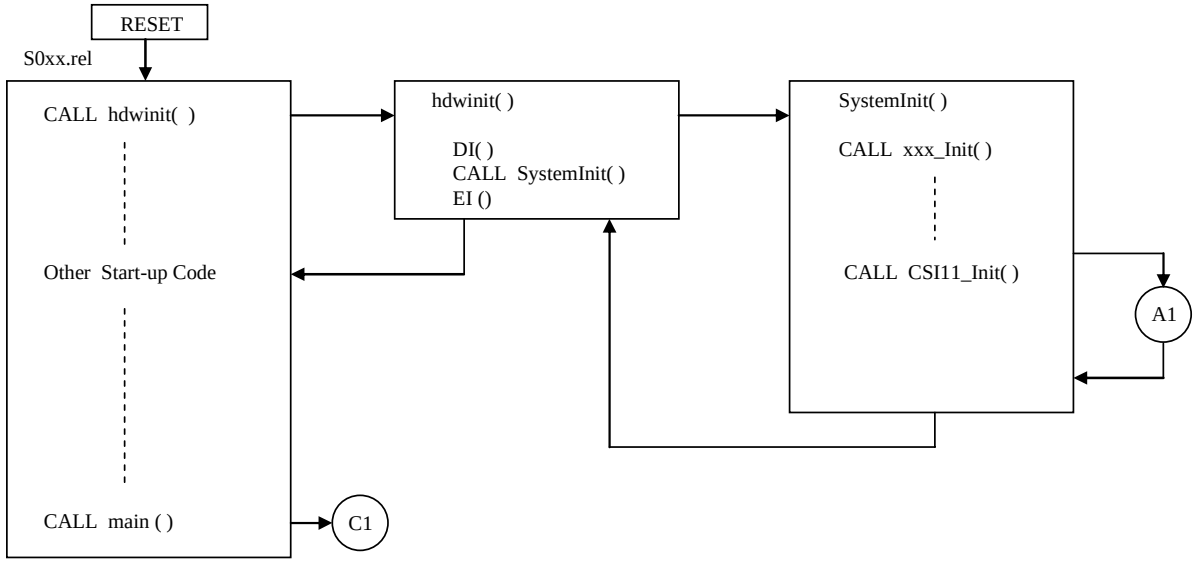
The flowcharts describe initialization, the main program, and interrupt routines related to CSI operation. Flowcharts are not included for other initializations, other peripheral-operation subroutines, and other peripheral interrupt-service routines. The software listings include this code.

4.3.1 Master: Program Startup and Initialization

For 78K0 programs written in the C language, an object code file such as s0l.rel is linked into the user program and provides the startup code for the C program. This startup code calls a function named hdwinit(); you can place hardware-initialization code here.

When the Applilet generates a C program for the 78K0, it 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.

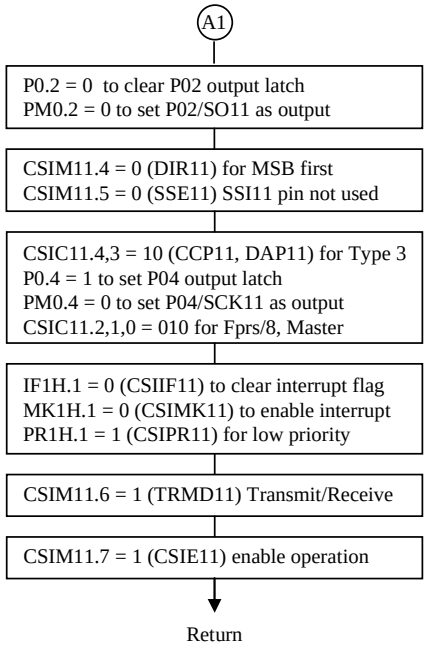
Figure 46. Flowchart for Program Startup and Initialization



After the hdwinit() function finishes, the startup code calls the main() function. When the main() function starts, all peripheral initialization is done, and the function does not need to call individual peripheral-initialization routines.

4.3.2 Master: CSI11_Init( ) – CSI11 Initialization For Transmit

Figure 47. Flowchart for Initializing CSI11 for Transmit



SystemInit() calls CSI11_Init() to initialize the CSI11 peripheral for transmit/receive operation.

First this routine clears the P02 output latch to zero and sets the port-mode register PM0 bit 2 to zero. These settings configure the pin P02/SO11 as an output. The peripheral transmits serial data from the master to the slave using this pin.

Next the routine sets the DIR11 bit in the CSIM11 register to 0 to select MSB-first transmission. The CSI11 unit can use an external pin (SSI11) to enable or disable output. This pin is not used in the demonstration, so the routine sets the SSE11 bit in the CSIM11 register to zero.

The CSIC11 register controls the clocking of CSI data for CSI11. The initialization routine sets the CCP11 bit to 1 to select clock polarity as normally low; sets the DAP11 bit to 0 to select data changed after the clock's leading edge (clocked on falling edge by slave). This combination of the CCP11 and DAP11 bits selects Type 3 transmission. The routine sets P0 bit 4 to one and port-mode register PM0 bit 4 to zero to set the P04/SCK11 pin as an output. Note that in this case, the routine sets the output latch to one, rather than to zero as with P02. Bits 2 through 0 in the CSIC11 register select the clock used to drive the SCK11 output. This initialization routine sets the bits to 010, which selects f_{PRS} divided by 8 as the clock. With an 8-MHz peripheral clock, this setting results in an SCK11 rate of 1 Mbps.

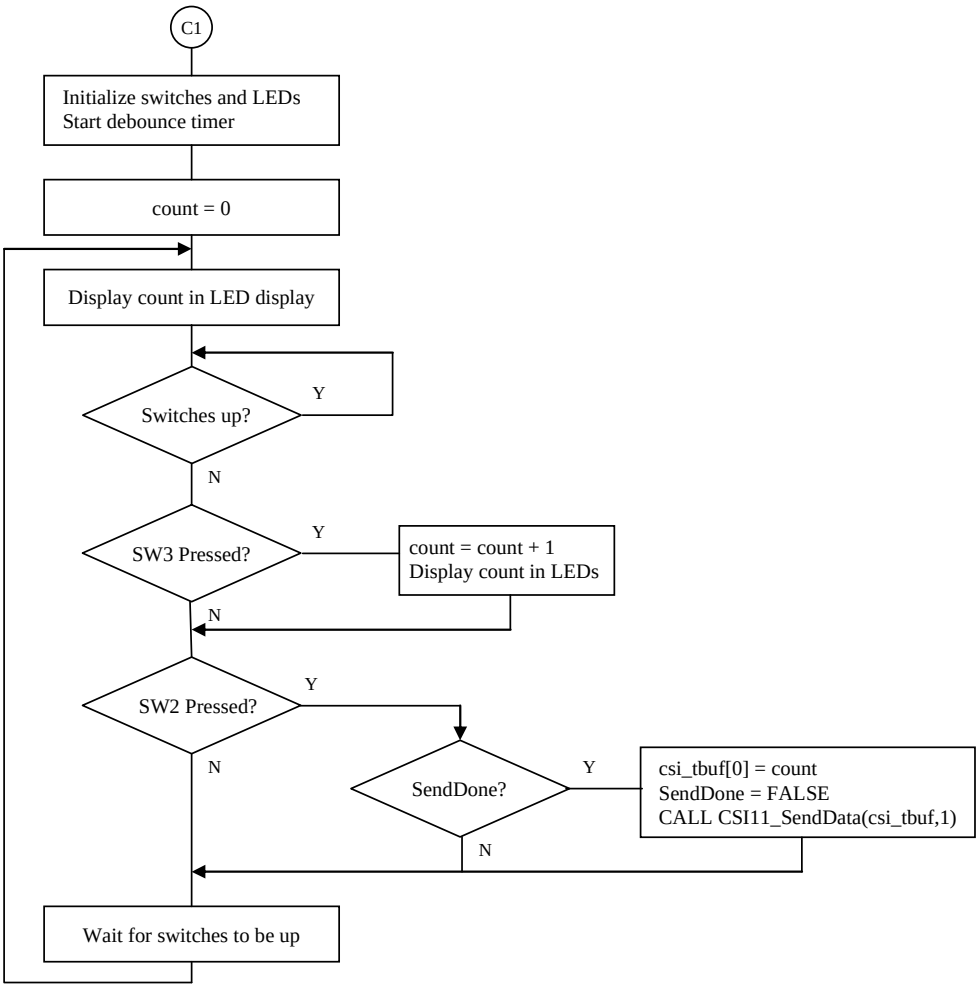
After shifting the 8 data bits, the CSI11 peripheral generates an interrupt, INTCSI11. This interrupt sets the CSIIF11 interrupt flag (interrupt register IF1H bit 0). The routine clears the interrupt flag and the corresponding mask bit (CSIMK11 in the MK1H register) to enable the interrupt. The routine then sets the CSIPR11 bit in the PR1H register to one for a low-priority interrupt.

CSI11_Init() sets the TRMD11 bit in the CSIM11 register to 1, to select the transmit/receive mode of operation, and the CSIE11 bit in the CSIM11 register to 1 to enable operation.

At this point, the CSI11 peripheral is ready. Writing a byte to the SOTB11 register starts data transmission, and reading a dummy byte from the SIO11 register begins data reception.

4.3.3 Master: Main( ) – Main Program – CSI11 Transmission

Figure 48. Flowchart for Main Program – CSI11 Transmission



The master main() program initializes handling for the switches and LED display, and starts the timer used for debouncing the switches. The program sets the count variable to zero and enters the main program loop.

Main() displays the current value of **count** in the LED display, and then checks the state of the switches. If no switch is closed, main() loops, waiting for a switch closure. If you press SW3, the routine increments the count variable and updates the LED display.

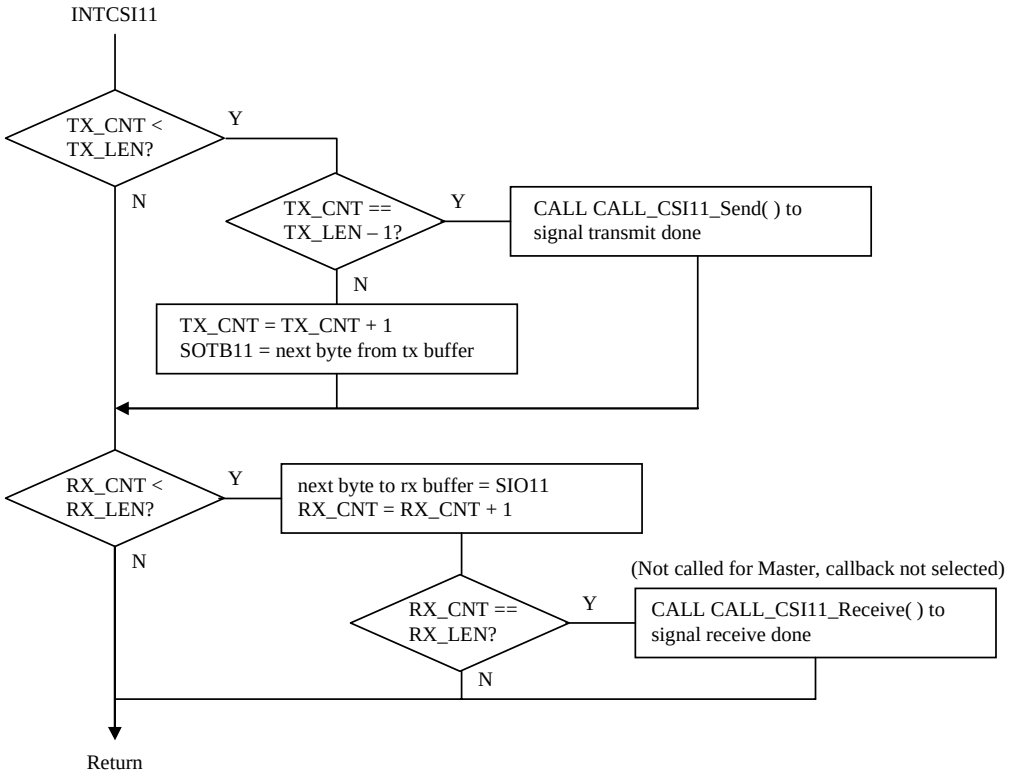
If you press SW2, main() checks the state of the **CSI11_SendDone** flag (which is true on program startup). If **CSI11_SendDone** is true, then the routine stores the value of count in the first location of the transmit buffer, csi_tbuf[0]. Main() sets the **CSI11_SendDone** flag to false and calls CSI11_SendData(csi_tbuf, 1). This routine passes the address of the data buffer and the number of bytes to transmit.

CSI11_SendData() stores a local copy of the address of data to send and the number of bytes. The routine also sets a local count to zero and writes the first byte to the SOTB11 register, which starts data shifting out. The routine then returns to main(). Main() waits for the switches to be open, then returns to the top of the loop.

When the data finishes shifting, the INTCSI11 interrupt occurs, invoking the MD_INTCSI11() interrupt-service routine. After all data is sent, the routine sets **CSI11_SendDone**. When you press SW3 again, the next transmission take place.

4.3.4 Master: MD_INTCSI11( ) – Interrupt-Service Routine for INTCSI11

Figure 49. Flowchart for Interrupt-Service Routine for INTCSI11



When 8 data bits have been shifted, the INTCSI11 interrupt occurs, setting the CSIIF11 interrupt flag and invoking the MD_INTCSI11() interrupt-service routine.

MD_INTCSI11() checks the count of bytes transmitted (**CSI11_TX_CNT**) against the number to send (**CSI11_TX_LEN**). If the count is one less than the length, then the last byte has been sent, and the current interrupt is caused by the end of transmission. Otherwise, there are more bytes to send. If all bytes have been sent, MD_INTCSI11() calls the callback routine CALL_CSI11_Send(). User-inserted code in this routine sets the global flag **CSI11_SendDone** to signal the main() program that transmission is finished.

If the count is less than the length minus one, there is data to transmit. The interrupt-service routine increments the count and writes the next data byte to the SOTB11 register from the transmit buffer.

Once transmit-side handling is done, the MD_INTCSI11() routine checks whether it should receive data by checking whether the receive count is less than the receive length. The demonstration program does not receive data, therefore both counts are zero and there is no reception.

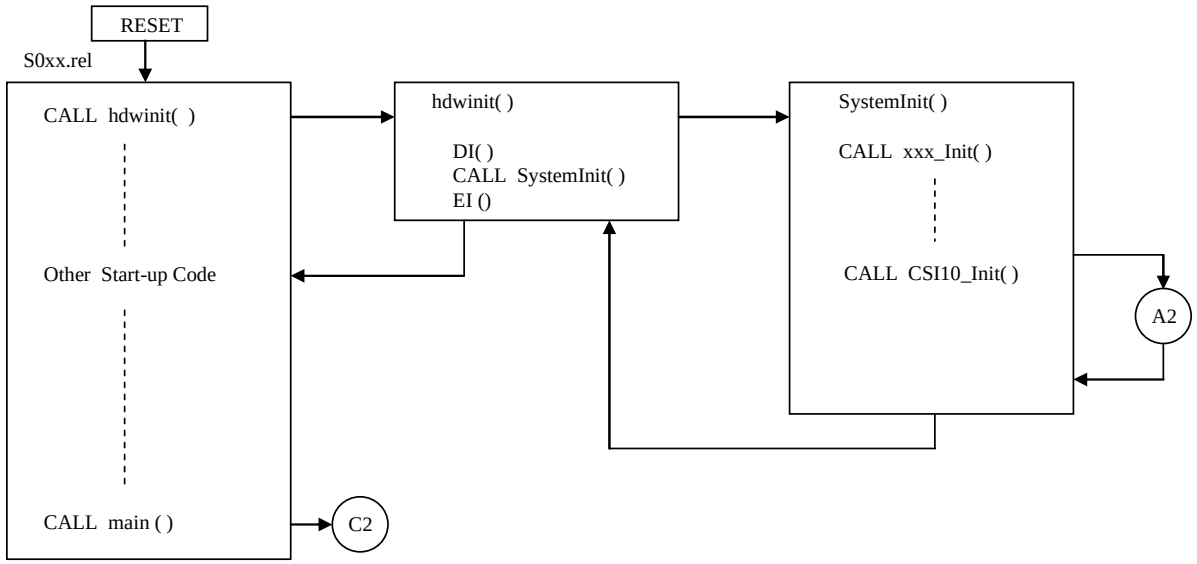
When setting up the CSI11 peripheral in the Applilet, you select a callback routine for transmit. Thus, when the transmission completes, the routine calls CALL_CSI11_Send(). You do not select a callback for receive in the Applilet, so the demonstration does not call CALL_CSI11_Receive(). However, the flowchart shows where this routine would be called if used.

When interrupt handling for transmitted or received data is complete, MD_INTCSI11() returns to the point in the program where the interrupt occurred.

4.3.5 Slave: Program Startup and Initialization

The demonstration program for the slave starts the same as the master program, except that the slave program calls CSI10_Init() instead of CSI11_Init().

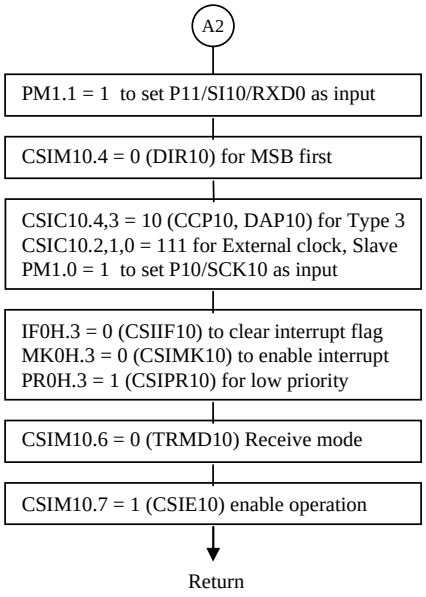
Figure 50. Flowchart for Slave Program Startup and Initialization



After initialization, the startup code calls the main() routine to start the slave demonstration program.

4.3.6 Slave: CSI10_Init( ) – CSI10 Initialization for Receive

Figure 51. Flowchart for Initializing CSI10 for Receive



SystemInit() calls CSI10_Init() to initialize the CSI10 peripheral for receive-only operation.

First CSI10_Init() sets the port-mode register (PM1) bit 1 to one, to set the pin P11/SI10/RXD0 as an input. The slave receives serial data from the master on this pin.

The initialization routine sets the DIR10 bit in the CSIM10 register to 0 to select MSB-first transmission. Note that the CSI10 peripheral does not have an external enable as the CSI11 peripheral does.

The CSIC10 register controls the clocking of CSI data for CSI10, so the routine sets the CCP10 bit to 1 to select clock polarity as normally low and the DAP10 bit to 0 to select data clocked on the falling edge. This combination of CCP10 and DAP10 bits selects Type 3 reception. The routine sets bits 2 through 0 of CSIC10 to 111, to select the clock for CSI10 as an external-clock input on SCK10; this setting specifies slave mode. CSI10_Init() sets port-mode register (PM1) bit 0 to one to set pin P10/SCK10 as an input.

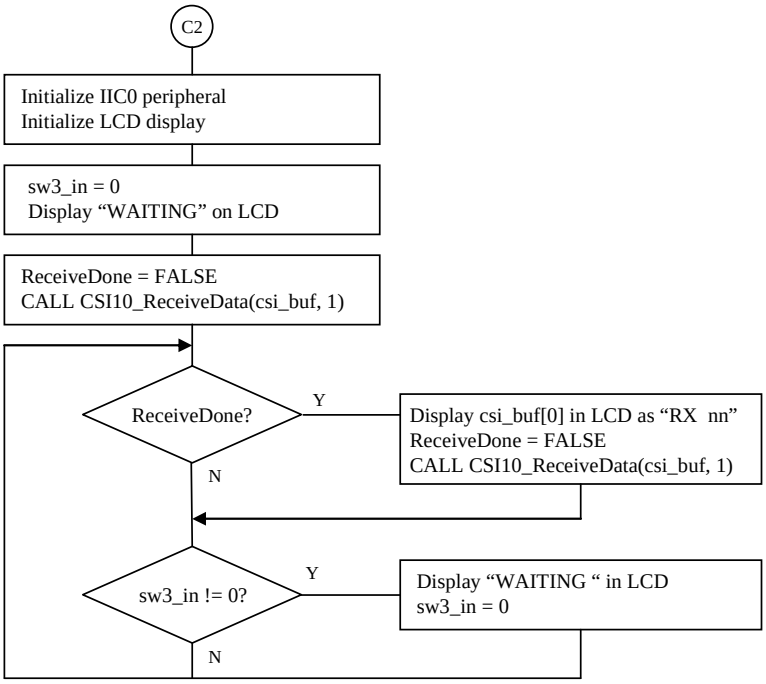
When 8 data bits shift out, the CSI10 peripheral generates interrupt INTCSI10. This event sets the CSIIF10 interrupt flag (interrupt register IF0H bit 3). The initialization routine clears the interrupt flag and the corresponding mask bit CSIMK10 in the MK0H register to enable the interrupt. The routine sets the CSIPR10 bit in the PR0H register to one for the default low-priority interrupt.

The routine sets the TRMD10 bit in the CSIM10 register to 0 to select receive-only operation. In the receive-only mode, the SO10 output is fixed at zero. You can use the P12/SO10 pin as an I/O port. The routine sets the CSIE10 bit in the CSIM10 register to 1 to enable operation.

At this point, the CSI10 peripheral is ready. Reading a dummy byte from the SIO10 register prepares for data reception. Data is available after receiving 8 external clocks.

4.3.7 Slave: Main( ) – Main Program – CSI10 Reception

Figure 52. Flowchart for Slave Main Program



The slave main() program initializes the IIC0 peripheral and the LCD controller, sets the **sw3_in** global variable (which shows the state of the input switches) to zero, and displays "WAITING" on the LCD.

Main() then sets the global flag **CSI10_ReceiveDone** to false and calls CSI10_ReceiveData(csi_buf,1), passing the address of a buffer for storing received data and a desired length of one byte.

CSI10_ReceiveData() stores a local copy of the data-buffer address and the number of bytes, sets a local count to zero, and reads a dummy byte from the SIO10 register, which prepares this register to receive data when clocked in by the master. CSI10_ReceiveData() then returns to main(). At this point, the slave is ready to receive one data byte.

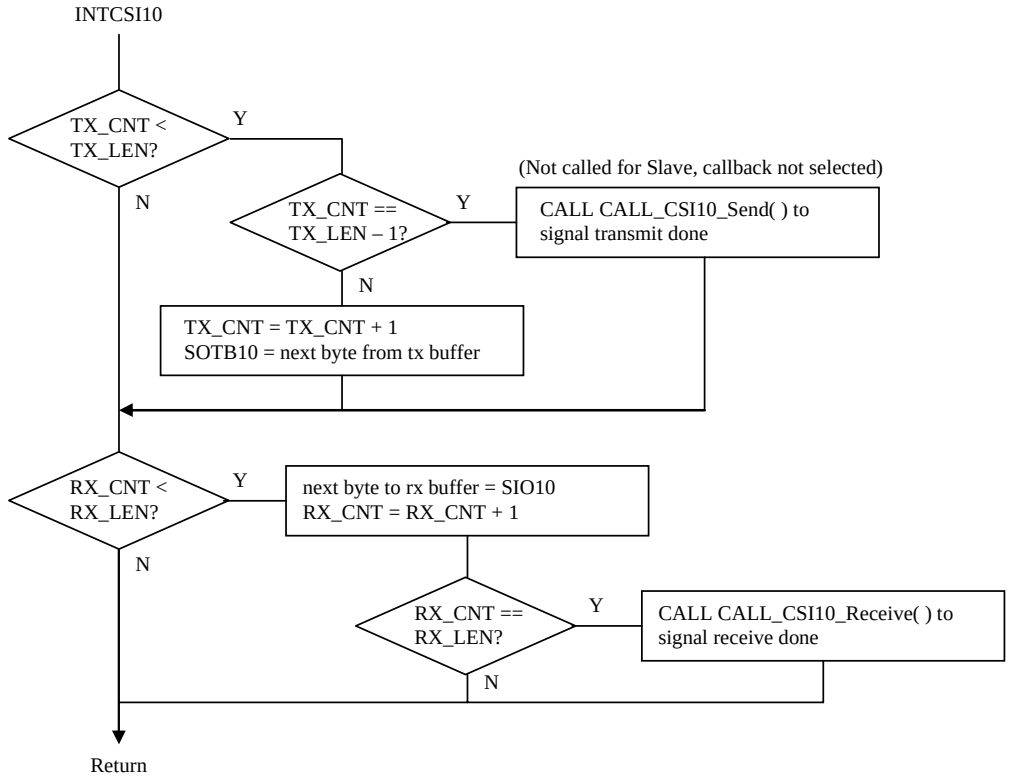
Main() then enters a loop. In this loop, it checks the **CSI10_ReceiveDone** flag. If the flag is true, main() writes a string to the LCD in the format "RX nn", where nn is the hexadecimal value of the data byte received. Main() then sets the **CSI10_ReceiveDone** flag false again and calls CSI10_ReceiveData() to prepare for the next data reception.

Main() then checks the **sw3_in** variable. If this variable is non-zero, indicating a switch has been pressed, main() writes the string “WAITING” to the LCD. Main() continues in this loop, processing received data and switch closures.

When the master transmits a data byte, the SCK10 input clocks the data into the SIO10 register. After 8 data bits have shifted in, the INTCSI10 interrupt occurs, invoking the MD_INTCSI10() interrupt-service routine. MD_INTCSI10() stores the received data in the receive buffer and increments the count. When the count reaches the desired length, the routine calls CALL_CSI10_Receive(), which sets the **CSI10_ReceiveDone** flag.

4.3.8 Slave: MD_INTCSI10( ) – Interrupt-Service Routine for INTCSI0

Figure 53. Flowchart for INTCSI0 Interrupt-Service Routine



After 8 data bits shift in, the INTCSI10 interrupt occurs, sets the CSIIF10 interrupt flag, and invokes the MD_INTCSI10() interrupt-service routine.

MD_INTCSI10() first checks the transmitted byte count against the number to send. The slave demonstration program operates in receive-only mode, so the transmit count and length are always zero.

The MD_INTCSI10() routine checks whether the received-data byte count (**CSI10_RX_CNT**) is less than the length of data to receive (**CSI10_RX_LEN**). If so, the routine reads the SIO10 register to shift the data in, stores the data in the next location in the receive buffer, and increments the count.

If the count equals the specified data length, MD_INTCSI10() calls the callback routine CALL_CSI10_Receive() to indicate reception is complete. User-inserted code in this routine sets the global flag **CSI10_ReceiveDone** to inform the main program that data has been received.

Note that if data is received while **CSI10_RX_CNT** and **CSI10_RX_LEN** are equal, the data is not read from SIO10 and hence is dropped. The program must call CSI10_ReceiveData() to prepare for a receive operation before receiving data.

The setup of the CSI10 peripheral in the Applilet program specifies a callback routine for receive. Therefore, when reception completes, the program calls CALL_CSI10_Receive(). Transmission is not set up, and the demonstration program does not call the callback function for transmit, CALL_CSI10_Send(). The flowchart shows where this routine would be called if needed.

When interrupt handling for received data completes, MD_INTCSI10() returns to the point in the program where the interrupt occurred.

4.4 Applilet's Reference Drivers

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

The Applilet produces the basic initialization code and main function for the program, driver code for the CSI, as well as code to manage the I/O ports used for switch input and display output. After the Applilet produces the basic code, you can add additional code to customize the program.

This section describes how to set up the Applilet to produce code for the CSI and lists the routines produced. Additional files added are also listed.

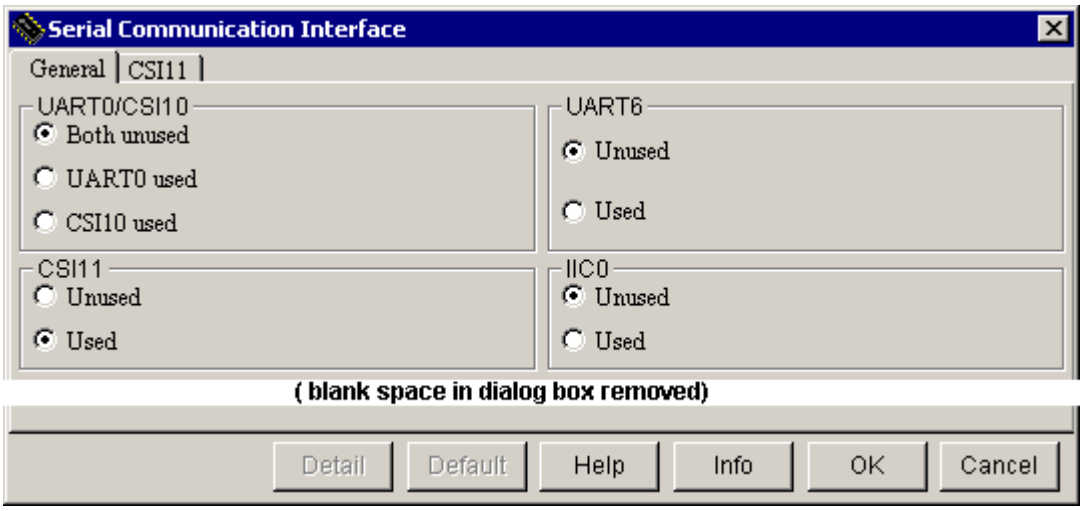
When you start the Applilet and select the target device, save your settings to a new project (.prx) file. The Applilet displays a dialog box that lets you select the peripheral blocks you want to set up.

The CSI demonstration runs on two separate NEC Electronics microcontrollers and therefore requires two separate Applilet projects. One project is for the master processor, doing data transmission, and the other project is for the slave processor, doing data reception.

4.4.1 Master: Configuring Applilet for CSI Transmission

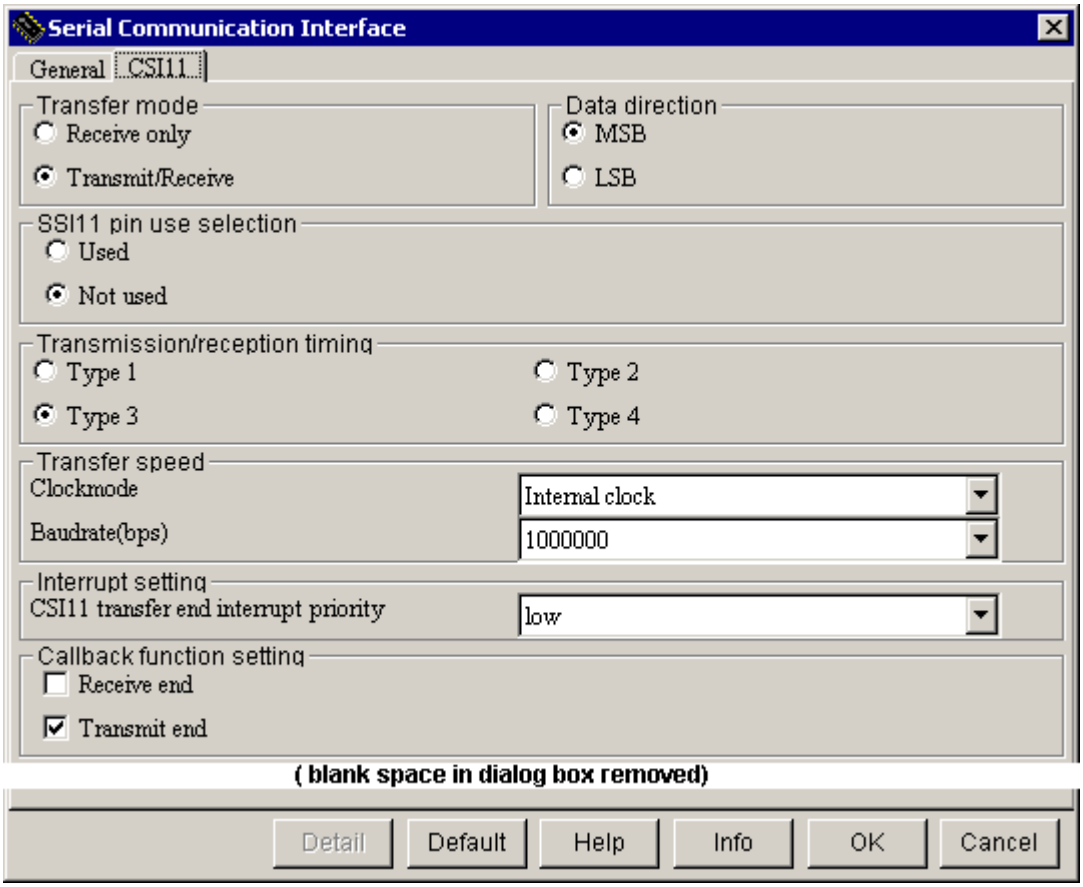
The master processor uses the CSI11 peripheral. In the Applilet project for the master processor, select **Serial** to brings up a dialog box for selecting the various serial blocks. Select **CSI11** as **Used**.

Figure 54. Selecting CSI11



Now click **Detail** to bring up the CSI11 detail dialog box.

Figure 55. CSI11 Detail Dialog



CSI11 has two **transfer mode** options: **Receive only** or **Transmit/Receive**. Select **Transmit/Receive**, and under **Data direction** select **MSB** so the serial byte clocks out with the most-significant bit first.

The external pin SSI11 to enable transmission is **Not used**.

The **Transmission/reception timing** sets the polarity of the clock (normally high for Types 1 and 2, and normally low for Types 3 and 4). This choice also sets where the data is changed (after initial clock edge for Types 1 and 3, or before clock initial edge for Types 2 and 4). Select **Type 3**, for a normally-low clock, and data changed after initial clock edge.

For a master device, the SCK (serial clock) pin is an output, driven to clock the serial data. An **Internal clock** serves as the clock source for the CSI. Select the value **1000000** (1 Mbps) for the baud rate.

CSI11 generates an interrupt on transfer end, and the Applilet creates an interrupt-service routine for this interrupt. You can choose the priority for the INTCSI11 interrupt. Select **low** priority.

Under **Callback function setting**, select **Transmit end** to provide hooks for user code when the transmission completes.

4.4.2 Master: Generating Code With Applilet

Once you have set up the CSI11 dialog box, click **Generate code.** The Applilet displays a list of the peripherals and functions to be generated, and allows you to select a directory for storing the source code.

When your click **Generate**, 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 CSI11 serial device, the Applilet generates the files serial.h, serial.c, and serial_user.c.

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

4.4.3 Master: Applilet-Generated CSI Files and Functions

The files serial.h, serial.c, and serial_user.c contain the code generated for CSI support.

4.4.3.1 Serial.h

The header file serial.h contains definitions for the CSI11 functions. 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.

4.4.3.2 Serial.c

The source file serial.c contains the following functions generated by the Applilet.

void CSI11_Init(void)
This routine initializes the CSI11 peripheral as specified in the Applilet CSI11 dialog.

MD_STATUS CSI11_SendData(UCHAR* txbuf, UCHAR txnum)
This routine sets up a transmit operation of txnum characters from the txbuf buffer and initiates data transmission by sending the first byte to the SOTB11 register.

MD_STATUS CSI11_ReceiveData(UCHAR* rxbuf, UCHAR rxnum)
This routine sets up a receive operation, requesting rxnum characters be received to the rxbuf buffer. The routine starts data reception by reading a dummy value from the SIO11 register. The master demonstration program does not use this routine.

__interrupt void MD_INTCSI11(void)
This is the interrupt-service routine for the CSI11 interrupt INTCSI11. For transmit operations, the routine sends the next data and increments the count. When done, the routine calls the

CALL_CSI11_Send() callback routine. For reception operations, the routine stores the received data to the receive buffer and increments the count.

4.4.3.3 Serial_user.c

The source file serial_user.c contains stub functions for user code. The Applilet generates these functions empty so that you can add application-specific code.

void CALL_CSI11_Send(void)
This routine executes when a transmission completes. You add code here to set the flag **CSI11_SendDone** to indicate to the main program that transmit is complete.

4.4.4 Master: Other Applilet-Generated Files

For the demonstration program, the Applilet generates several other source files. The files and their functions are shown below.

Table 21. Other Applilet-Generated 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
Port.h	Definitions for default I/O port states
Port.c	PORT_Init() function
Option.inc	Option-byte, POC, and security definitions
Option.asm	Option-byte, POC, and security data

4.4.5 Master: Demonstration Program Files Not Generated by Applilet

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

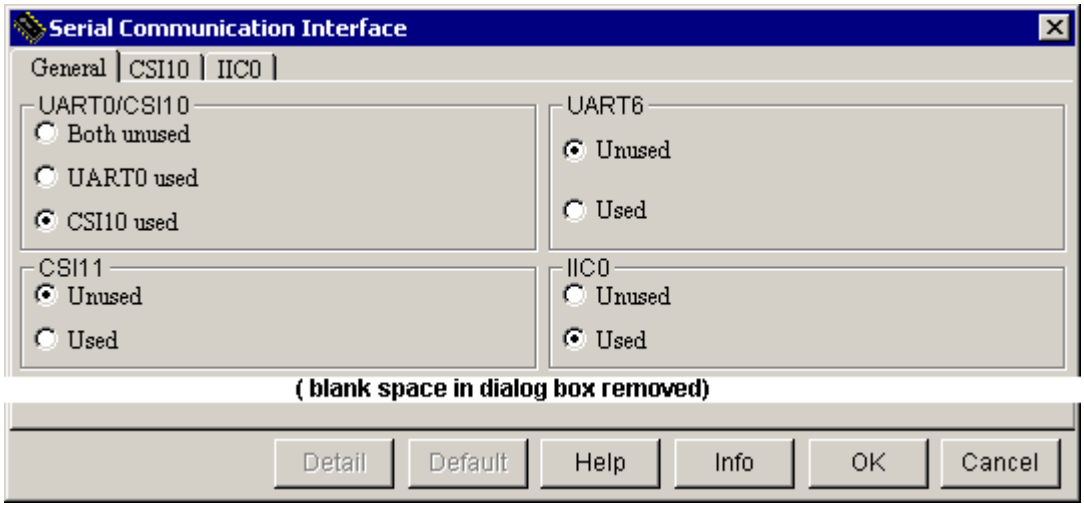
Table 22. Demonstration Program Files Not Generated by Applilet

File	Function
Sw_0537.h	Header file for pushbutton switch input
Sw_0537.c	Code to read and debounce pushbutton switches
Led_0537.h	Header file for seven-segment LED patterns and functions
Led_0537.c	Code to display data in seven-segment LED displays

4.4.6 Slave: Configuring Applilet for CSI Reception

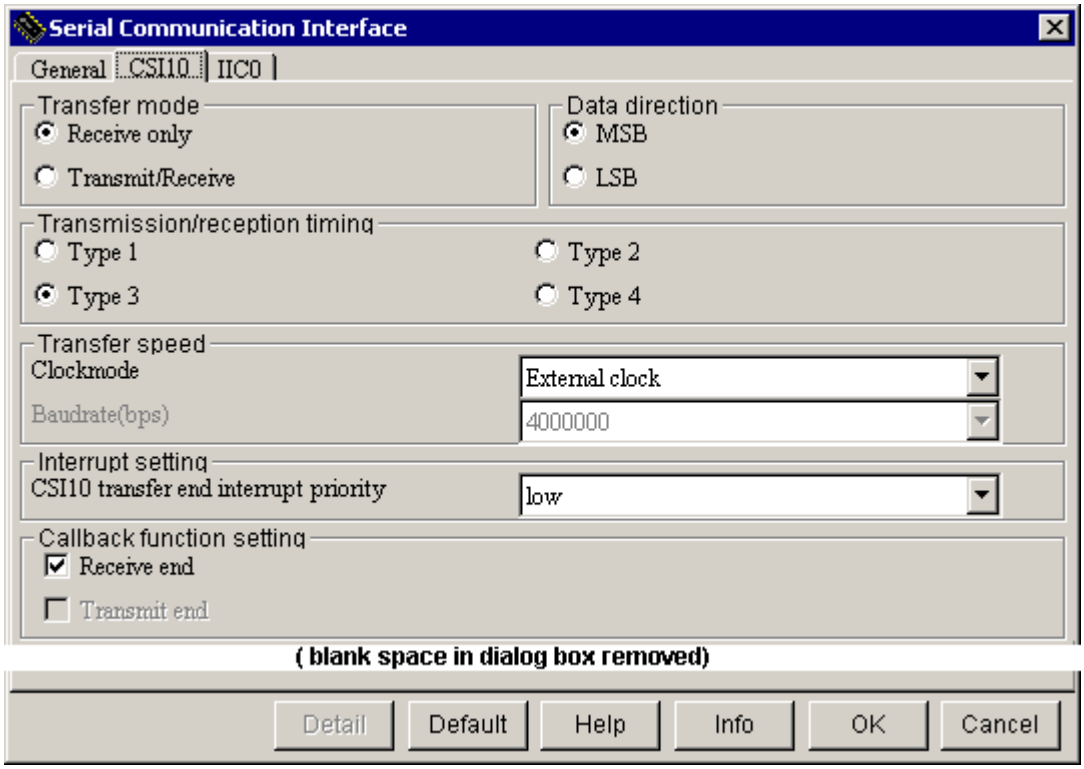
The slave processor uses the CSI10 peripheral. In the Applilet project for the slave program, selecting **Serial Communication Interface** brings up a dialog showing the various serial blocks. Select **CSI10 used**. Note that CSI10 uses the same pins as UART0, so you can only use one of these peripherals at a time.

Figure 56. Selecting CSI10



Once you have selected CSI10, clicking **Detail** brings up the CSI10 detail dialog box.

Figure 57. CSI10 Detail Dialog



For the slave, select **Receive only** for the **Transfer mode**. Select **MSB** for the **Data direction** to clock out the serial byte with the most-significant bit first, as for the master.

Under **Transmission/reception timing** select **Type 3** to match the master. This setting provides a normally-low clock and data changed after initial clock edge. The slave thus samples data on the clock's trailing edge.

For a slave device, the SCK (serial clock) pin is an input driven by the master to clock the serial data. Thus, select **External clock** for the slave configuration. With this choice, the baud rate setting is not available.

CSI10 generates interrupts on receive end. The Applilet creates an interrupt-service routine for this interrupt. Select **low** priority for the INTCSI10 interrupt.

Select **Receive end** for the **Callback function setting** to provide hooks for user code for receiving data. For receive-only mode, there is no option to select a transmit-end callback function.

4.4.7 Slave: Generating Code With Applilet

Once you have set up the CSI10 dialog box, click **Generate code**. The Applilet displays a list of peripherals and functions to be generated, and allows you to select a directory for storing the source code.

When you click **Generate**, 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 CSI10 serial device, the Applilet generates serial.h, serial.c, and serial_user.c.

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

4.4.8 Slave: Applilet-Generated CSI Files and Functions

The files serial.h, serial.c, and serial_user.c code contain the code generated for CSI support.

The slave processor uses the IIC0 serial peripheral. The code to support this peripheral is also in the serial files generated by the Applilet.

4.4.8.1 Serial.h

The header file serial.h contains definitions for the CSI10 functions. 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.

4.4.8.2 Serial.c

The source file serial.c contains the following functions generated by the Applilet.

void CSI10_Init(void)
This routine initializes the CSI10 peripheral as specified in the Applilet CSI10 dialog.

MD_STATUS CSI10_ReceiveData(UCHAR* rxbuf, UCHAR rxnum)
This routine sets up a receive operation, requesting rxnum characters be received to the rxbuf buffer. This routine starts reception by reading a dummy value from the SIO11 register.

__interrupt void MD_INTCSI10(void)
This is the interrupt-service routine for the CSI10 interrupt INTCSI10. For transmit operations (not used in the slave), the routine sends the next data and increments the count. For reception operations, the routine stores the received data in the receive buffer and increments the count.

When the count reaches the desired number of bytes, the routine calls the CALL_CSI10_Receive() callback function.

4.4.8.3 Serial_user.c

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

void CALL_CSI10_Receive(void)
The program calls this routine when reception finishes. Add code to set the flag **CSI10_ReceiveDone** to indicate to the main program that reception is complete.

4.4.9 Slave: Other Applilet-Generated Files

For the demonstration program, the Applilet generates several other source files. The files and their functions are shown below.

Table 23. Other Applilet-Generated 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
Int.h	Interrupt-related definitions
Int.c	Key-return interrupt-related functions
Int_user.h	User code for key-return interrupt
Timer.h	Timer-related definitions
Timer.c	Timer functions
Timer_user.c	User code for timer-interrupt handling
Option.inc	Option-byte, POC, and security definitions
Option.asm	Option-byte, POC, and security data

4.4.10 Slave: Demonstration Program Files Not Generated by Applilet

The demonstration program also uses files not generated by the Applilet.

Table 24. Demonstration 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
LcdDrvApp.h	Header file for LCD-driver functions
LcdDrvApp.c	Code to initialize, write and read the LCD controller; These functions call Applilet-generated IIC0 routines

4.5 Demonstration Platform

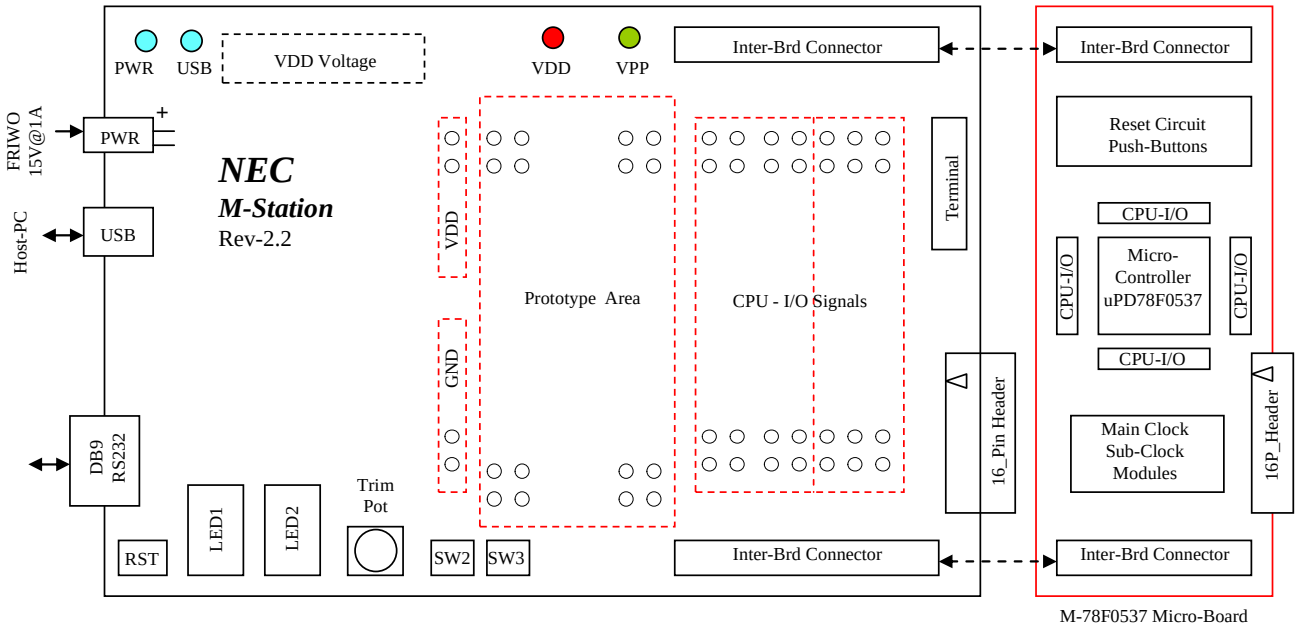
This demonstration uses a development board from NEC Electronics. You may be able to duplicate the hardware with off-the-shelf components along with the NEC Electronics microcontroller of interest.

4.5.1 Resources For Master

The master program demonstration uses the following resources:

- M-78F0537 Micro-Board, with uPD78F0537 8-bit microcontroller mounted
- uPD78F0537 CSI11 peripheral, using P02/SO11, P03/SI11, and P04/SCK10
- M-Station II Evaluation System, using M-Station II resources:
  - 7-Segment LED displays LED1 and LED2
  - Pushbutton switches SW2 and SW3
  - Connections for uPD78F0537 CSI11 signals

Figure 58. Demonstration System



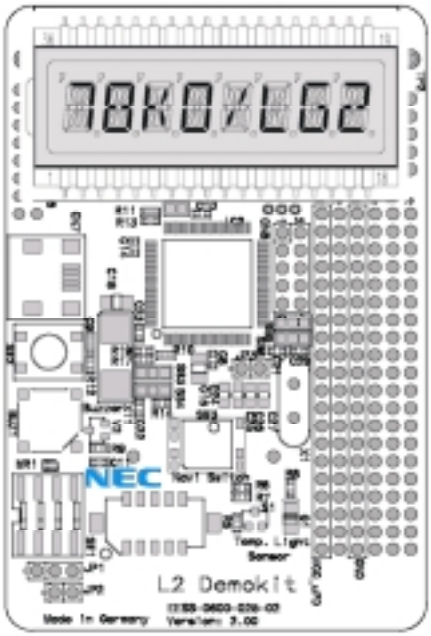
4.5.2 Resources For Slave

The slave demonstration program uses the following resources:

- DemoKit-LG2 demonstration board, with uPD78F0397 8-bit microcontroller mounted
- DemoKit-LG2 resources:
  - 5-position navigation switch SW3
  - 8-character LCD
  - CSI10 interface using P10/SCK10, P11/SI10/RXD0, and P12/SO10

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

Figure 59. Slave Program Demonstration Hardware



4.5.3 Demonstration of Program

With the hardware configured and the uPD78F0397 microcontroller programmed with the demonstration code, the demonstration is as follows.

On initial power-up, the LED displays on the M-Station II show “00” for the count value. The LCD on the DemoKit-LG2 shows “WAITING”. The table shows the demonstration steps.

Table 25. CSI Demonstration Steps

Input	M-Station Display	DemoKit-LG2 Display	Comment
SW3 on M-Station	01	WAITING	Count incremented
SW3 on M-Station	02	WAITING	Count incremented
SW2 on M-Station	02	RX 02	Data value 02 sent via CSI
SW3 on M-Station	03	RX 02	Count incremented
SW2 on M-Station	03	RX 03	Data value 03 sent via CSI
Any switch on DemoKit-LG2	03	WAITING	Display changed
SW2 on M-Station	03	RX 03	Data value 03 sent via CSI

4.6 Hardware Block Diagram

Figure 60. Demonstration Hardware Block Diagram

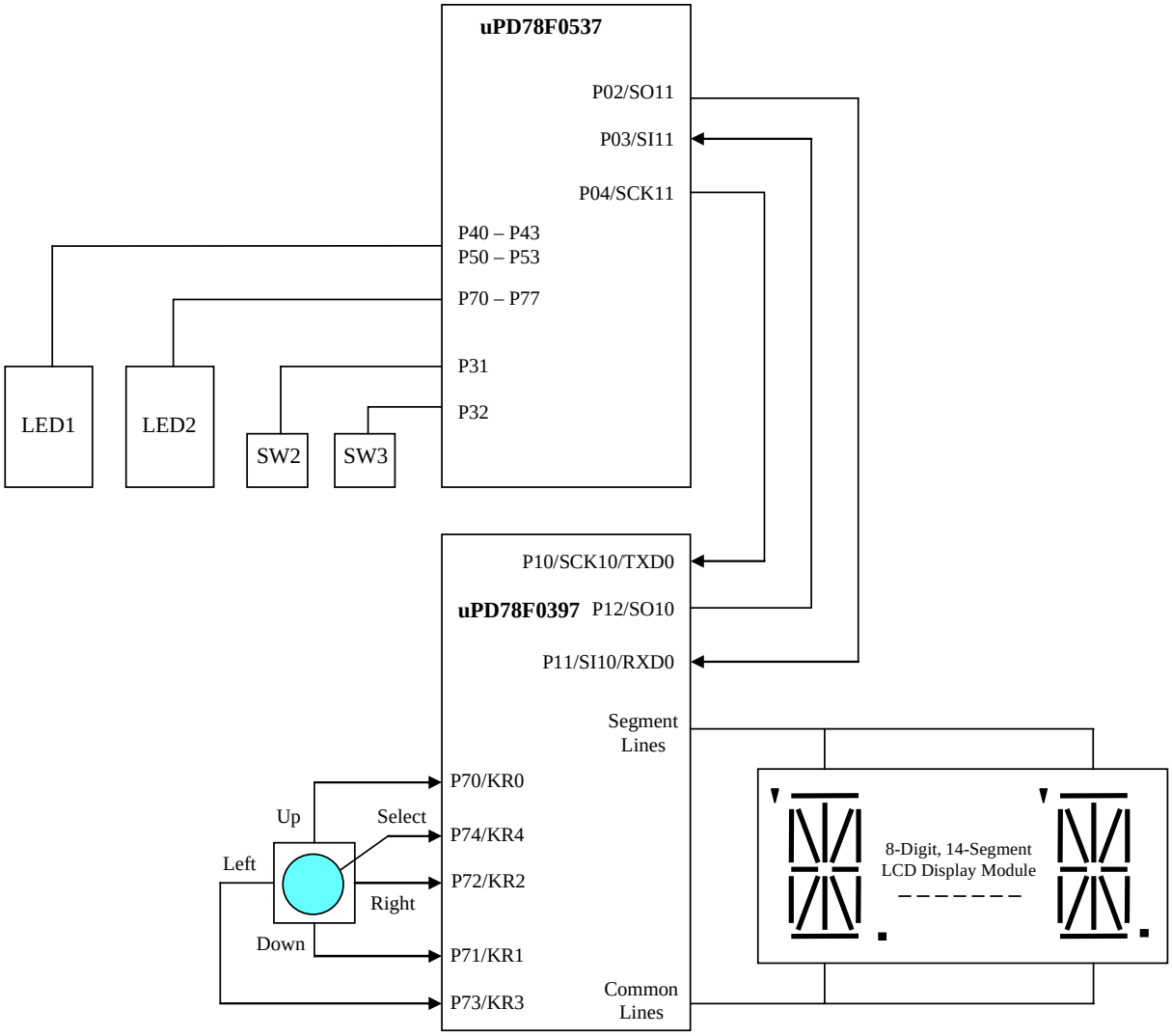


Table 26. Signal Interconnects for CSI signals

uPD78F0537 Signal	uPD78F0537 Pin	M-Station Location	DemoKit-LG2 Location	uPD78F0397 Signal	uPD7850397 Pin
P02/SO11	60	J1_C.3	CN4.18	P11/SI10/RXD0	75
P03/SI11	59	J1_C.4	CN4.20	P12/SO10	74
P04/SCK11	58	J1_C.5	CN4.16	P10/SCK10/TXD0	76

Ground connects between the two systems. VDD power comes from the M-Station/M-78F0537 system to the DemoKit-LG2 by connecting VDD on the M-Station to DemoKit-LG2 JP1 pin 2. (Leave JP1 open).

4.7 Software Modules

The following files make up the software modules for the demonstration programs. The tables below shows which files are generated by the Applilet, and which need modification to create the demonstration programs.

The listings for these files are located in the Appendix.

4.7.1 Software Modules for CSI Master Program

Table 27. CSI Master 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
Serial.h	CSI-related definitions	Yes	Yes[Note 1]
Serial.c	CSI-related functions	Yes	No
Serial_user.c	User code for CSI callback routine	Yes	Yes[Note 1]
Port.h	Definitions for default I/O port states	Yes	No
Port.c	PORT_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
Timer_user.c	User code for timer-interrupt handling	Yes	Yes[Note 2]
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No
Led_0537.h	LED-display definitions	No	--
Led_0537.c	LED-display functions	No	--
Sw_0537.h	Switch-input definitions	No	--
Sw_0537.c	Switch-input functions	No	--

Note 1: Modify serial.h to add the declarations of the CSI-related global variable **CSI11_SendDone**. Modify serial_user.c to define this variable, and to have CALL_CSI11_Send() set the variable.

Note 2: Modify timer_user.c to have the MD_INTTM000() interrupt-service routine call the sw_isr() routine for switch debouncing.

4.7.2 Software Modules for CSI Slave Program

Table 28. CSI Slave 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	CSI and IIC0-related definitions	Yes	Yes[Note 2]
Serial.c	CSI and IIC0-related functions	Yes	Yes[Note 2]
Serial_user.c	User code for CSI and 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	No
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	No	--
Lcd.h	LCD high-level function definition	No	--
Lcd.c	LCD high-level functions	No	--
LcdDrvApp.h	LCD-controller driver definitions	No	--
LcdDrvApp.c	LCD-controller driver functions	No	--

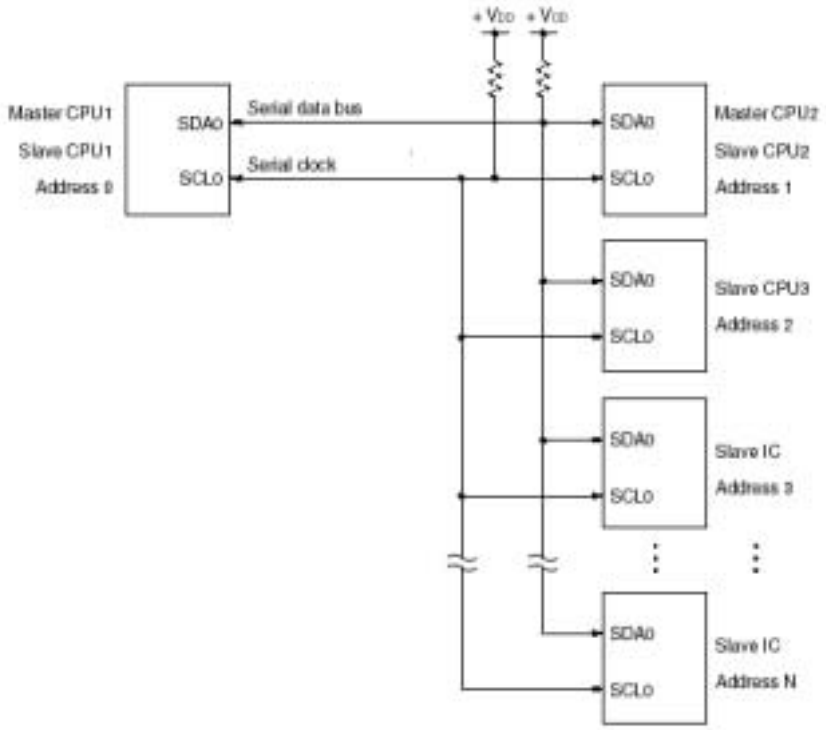
Note 1: Modify int.h to add the declaration of **sw3_in**, the variable used to store the navigation-switch debounced input. Modify int_user.c to add the variable **sw3_in** and the code for handling the key-return interrupt.

Note 2: Modify serial.h to add the declarations of CSI- and IIC0-related global variables, defined in serial_user.c. Also modify serial.h to declare the functions IIC0_Init() (not declared by default) and IIC0_MasterStartAndSend(). Modify serial.c slightly for CSI to correct an error in the MD_INTCSI10 code, and for IIC0 to remove settings for P6 not supported for 78K0/LG2, and to replace "asm" versions of DI and EI instructions with NEC Electronics C Compiler equivalents. Modify serial_user.c to have CALL_CSI10_Receive() set the **CSI10_ReceiveDone** variable, and to have IIC0 callback functions set the IIC0-related global variables, and to add the IIC0_MasterStartAndSend() function.

5. Serial Communication Using IIC

The Inter-Integrated Communication Interface (IIC) provides 8-bit data transfers between two or more devices. The IIC bus uses two signaling lines, Serial Data Bus (SDA) and Serial Clock (SCL). Any node can drive these lines, or they can be pulled up by external resistors. The communication protocol supports multiple devices attached to the bus. A device may be a master, slave, or operate as both a master and a slave. You can attach multiple masters and slaves to the bus.

Figure 61. Block Diagram for IIC Communications



A master device can initiate transfers. The master specifies an address for a slave device as part of the transfer. Master nodes may send data to slave nodes or read data from slave nodes. Slave devices detect the slave address specified at the beginning of the transfer. The LSB of the slave address specifies whether the transfer is a read or a write by the master. The slave device whose locally set slave address matches the address sent by the master responds by receiving or sending data.

Many NEC Electronics microcontrollers provide an internal IIC0 peripheral, which handles many of the functions of IIC communication. Other devices support IIC communication by using two port pins connected to SCL and SDA, and controlling the drive of these pins.

The application described in this section shows two devices communicating over the IIC bus, with one device using port pins to support IIC functions, and one using the IIC0 peripheral for both slave- and master-mode communication.

5.1 IIC0 Features

The IIC0 peripheral in NEC Electronics microcontrollers supports IIC communication with the following features:

- Hardware serialization of IIC input and output, with interrupt on completion, for low overhead
- Simple generation of Start, Ack, and Stop states
- Automatic sensing of IIC bus states: Busy, Start, Ack, Stop
- Arbitration with other masters for bus access
- Communication-reservation function, to obtain bus access when the current transfer ends
- Automatic generation and recognition of wait conditions in sending or receiving
- Selectable clock to the IIC0 peripheral, supporting multiple data rates
- Standard IIC rate (100,000 bps) and high-speed data rates (to 400,000 bps)
- Digital filter for noise suppression in the high-speed mode
- Support for IIC extension codes
- Settable slave-address register for response to different slave addresses

Figure 62. Simplified Block Diagram of IIC0 Serial Communication Circuit

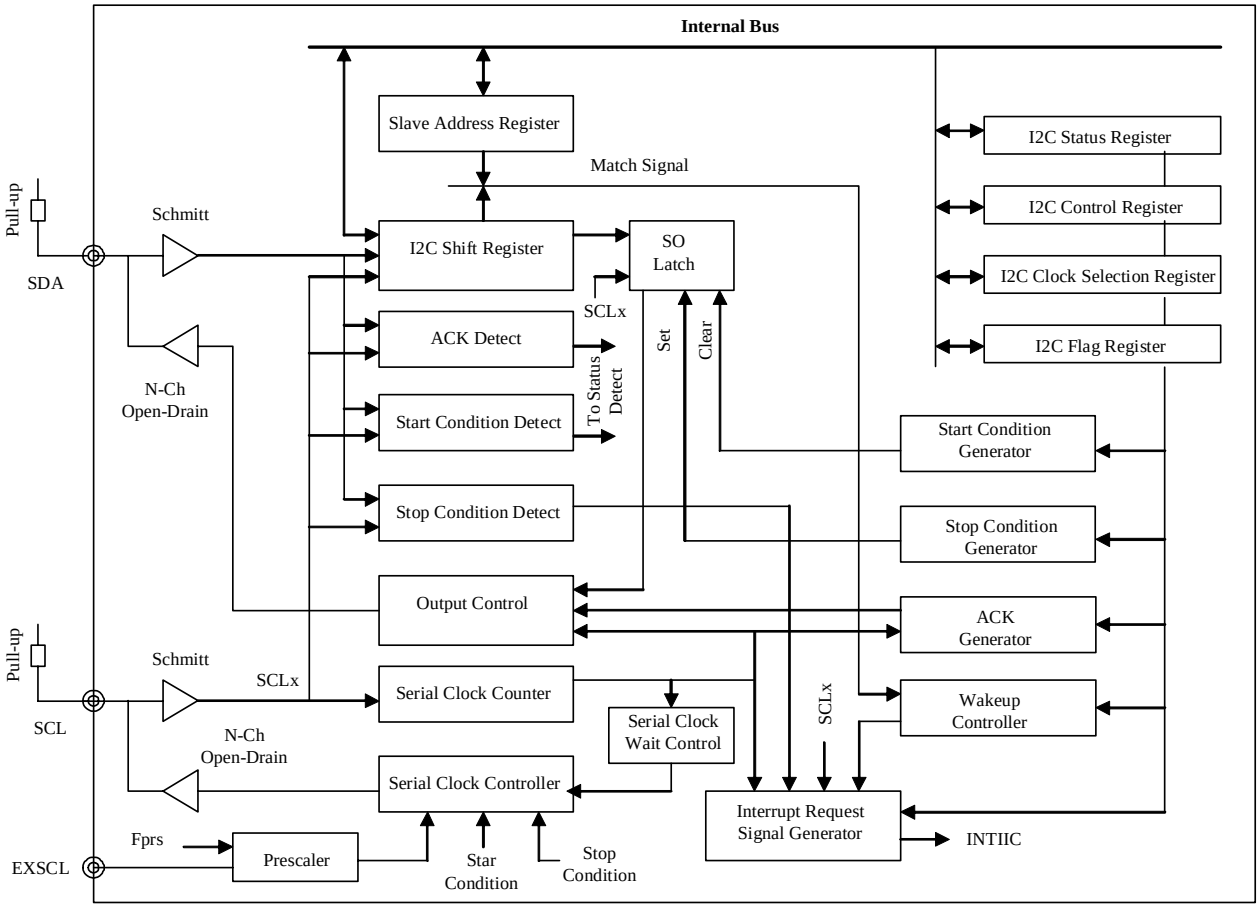


Table 29. IIC Functional Blocks

Functional Blocks	Description of Functions
IIC Shift Register	Converts 8-bit serial data to 8-bit parallel data and vice versa
Slave Address Register	Store local address when in slave mode
SO Latch	The SO latch used to retain SDA output level
Wakeup Controller	Generates an interrupt when the address received matches with the address value stored in slave address register
Prescaler	Selects sampling clock
Serial Clock Counter	Counts the serial clocks that are output or input during transmit/receive. Verifies that 8-bit data has been transmitted or received
Interrupt Request Signal Generator	Generates interrupt-request signal for the following conditions - Falling Edge of 8th or 9th clock of the serial clock - When a stop condition is detected
Serial Clock Controller	Generates clock output (SCL) from sampling clock when in master mode
Serial Clock Wait Controller	Controls waiting time
ACK Generator/ACK Detector Start/Stop Condition Detector	Generates and detects each condition
Start Condition Generator	Generates start condition
Stop Condition Generator	Generates stop condition

Table 30. Registers Controlling IIC Operations

Register	Description of Functions
IIC Control Register	Enable/Disable IIC Operations, set wait-timing Enable/Disable interrupt request when stop condition is detected Control wait and interrupt-request generation ACK Control Start condition or stop condition trigger
IIC Status Register	Device status - master or slave status Detection of status - matching address, transmit and receive, ack, start/stop
IIC Flag Register	Indicates the operation mode and the status of IIC bus
IIC Clock Selection Register	Sets the transfer clock for the IIC bus
IIC Function Expansion Register	Sets function expansion of IIC
Port Mode and Port Registers	Sets port mode and functions

To configure the IIC0 peripheral for use in IIC protocol communication, use the following steps:

- Clear the IICE0 bit in the IICC0 control register to disable the peripheral while setting other registers.
- Set communication parameters in the IICF0 flag register and IICC0 control register.
- Select clock source and speed using the IICCL0 clock register and IICX0 extension register.

- Set interrupt priority using the appropriate interrupt-priority register bit.
- Set slave address for slave mode in the SVA0 slave-address register.
- Enable the INTIIC0 interrupt by clearing the appropriate interrupt-mask bit.
- Set the IICE0 bit in the IICC0 control register to enable the IIC0 peripheral.
- Set the PM6 register bit to output mode to enable drive of the SCL0 and SDA0 pins.

To start master-mode communication:

- Use the communication-reservation function to acquire the bus and generate a start condition.
- Or check the bus-busy status and generate a start condition when the bus is not busy.
- Write the desired slave address (with LSB set for direction) to the IIC0 register.

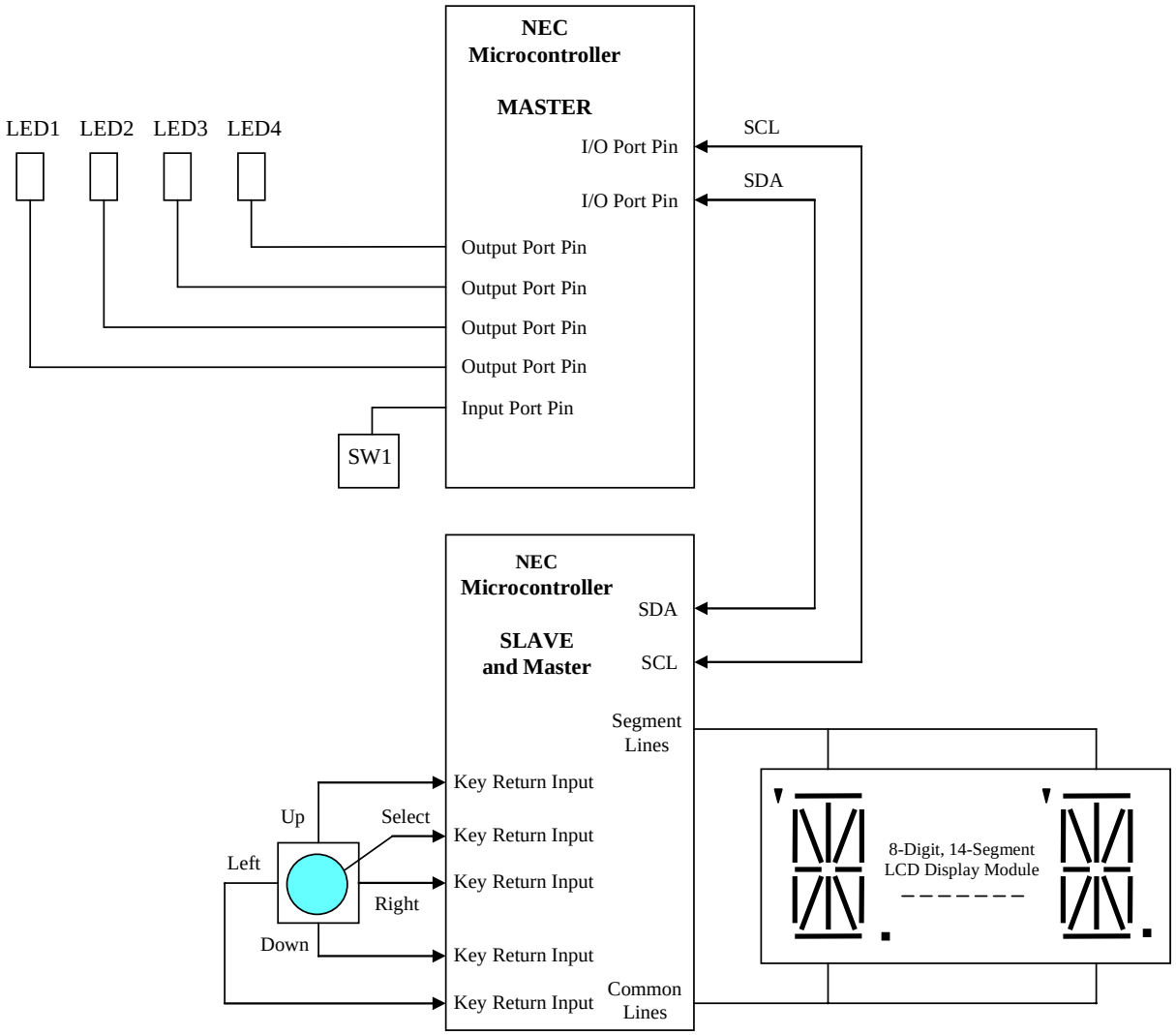
Further master-mode communication is handled in response to the INTIIC0 interrupt. Slave-mode communication is also handled in response to an INTIIC0 interrupt received after a master sends a matching slave address.

5.2 Program Description and Specification

The IIC data-communication demonstration uses two NEC Electronics microcontrollers. One microcontroller serves as the master, initiating transfers to send or receive data. The master does not use an internal IIC peripheral, but emulates IIC operation with two bidirectional I/O port pins. This demonstration thus illustrates IIC operation without a dedicated peripheral.

The slave microcontroller responds to IIC transfers from the master. The slave uses an internal IIC peripheral, which operates in slave mode for communication with the master microcontroller and in master mode for communication with an internal LCD controller.

Figure 63. IIC Data-Communication Demonstration



The master has four individual LEDs indicating data and status, and an input switch. The slave has an 8-digit LCD for displaying messages and a five-position switch for input. The master and slave connect through the IIC interface lines SCL (serial clock) and SDA (serial data). Power and ground also interconnect to ensure correct voltage levels.

The master maintains a local count value which it can send to the slave. The slave changes its display in response to switch inputs and retains the last switch-input value for transfer to the master on request.

The master device alternates between sending data to the slave and receiving data from the slave, as you press switch SW1:

Press 1: Show low four bits of count value; initially 0xFF, shown as on-on-on-on
Press 2: Send count to Slave. Blank displays if no error; show 8 (on-off-off-off) if error.

	Count is decremented by 0x11 (to 0xEE, 0xDD, etc).		
Press 3:	Read last switch input value from Slave, display depending on data:		
	UP switch (0x01)	display 1	(off-off-off-on)
	DOWN switch (0x02)	display 2	(off-off-on-off)
	RIGHT switch (0x04)	display 3	(off-off-on-on)
	LEFT switch (0x08)	display 4	(off-on-off-off)
	SELECT switch (0x10)	display 5	(off-on-off-on)
	Multiple bits	display 7	(off-on-on-on)
	Error in reading:	display 8	(on-off-off-off)
Press 4:	Blank display		

This cycle repeats as you press the input switch.

The slave displays a string on its LED corresponding to the switch pressed (for instance, “UP” for the UP switch) and saves the input-switch value. When the master sends the slave a value, the slave displays “RCV xx”, where xx is a two-digit hexadecimal value corresponding to the received data. When the master reads IIC data, the slave provides the last switch-input value. When the slave displays data on the LCD, it uses IIC communication with the LCD controller, operating as an IIC master.

Specifications:

- SDA and SCL lines have external pull-up resistors, allowing either side to drive or float.
- IIC communication speed is 100,000 bps or lower (standard IIC data rate).
- Master uses two port pins for SCL and SDA connection, driving low or allowing lines to be pulled up.
- Slave uses IIC0 peripheral, with SCL0 and SDA0 pins controlled by IIC0 peripheral.
- Slave responds to IIC slave address 0x20 for slave-mode communication.
- Slave communicates with the LCD controller as master, using IIC slave addresses 0x70 and 0x72.
- Master writes one-byte transfers of count data to the slave.
- Master reads one-byte transfers of last switch value from the slave.

5.3 Software Flowcharts

The IIC demonstration consists of two programs, one initiating transfers (master) and one responding (slave). Both master and slave can send or receive IIC data, depending on the direction specified by the master. The slave can also initiate master-mode IIC transfers to its LCD controller.

The demonstration programs consist of the following major sections related to IIC operation:

- Program-initialization code, including port initialization called before the main() program starts
- The main program loop, which checks switches, transmits data through the IIC bus, checks for received data, and displays values
- Subroutines for sending and receiving IIC data, and interrupt-service routines related to IIC communication

The programs also include sections not related directly to IIC operation:

- Subroutines generated by the Applilet for port initialization for switches and LED display (master)
- Subroutines for switch input and LED display output (master)
- Subroutines for key-return interrupt handling (slave)
- Subroutines for displaying data on LCD controller, other than IIC communication (slave)

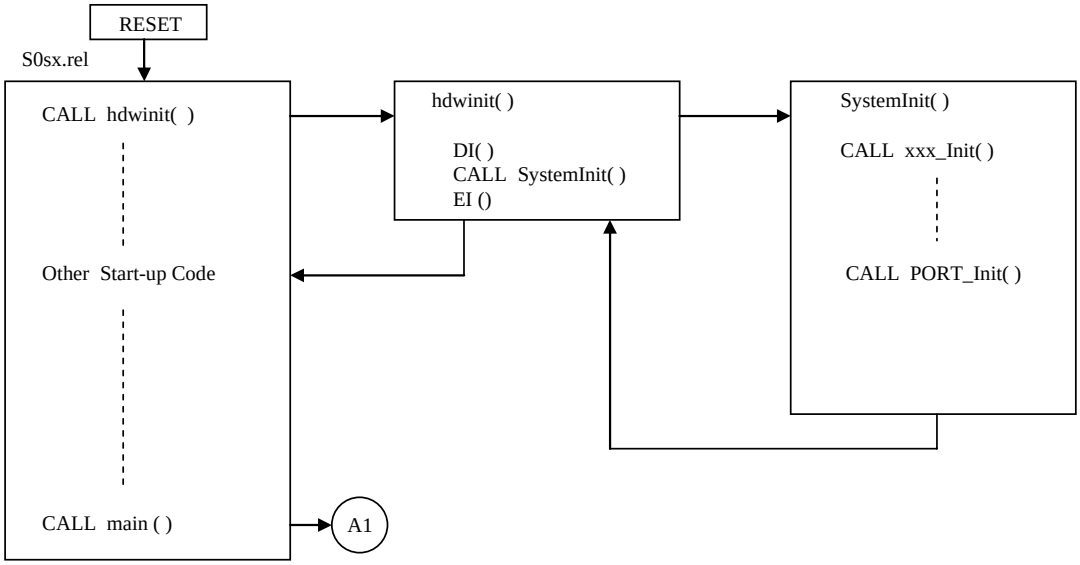
The flowcharts describe initialization, the main programs, and subroutines and interrupt routines related to IIC operation. Flowcharts are not included for other initialization, other peripheral operation subroutines, and other peripheral interrupt-service routines, but the software listings include this code.

5.3.1 Master: Program Startup and Initialization

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

When the Applilet generates a C program for the 78K0, the tool 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.

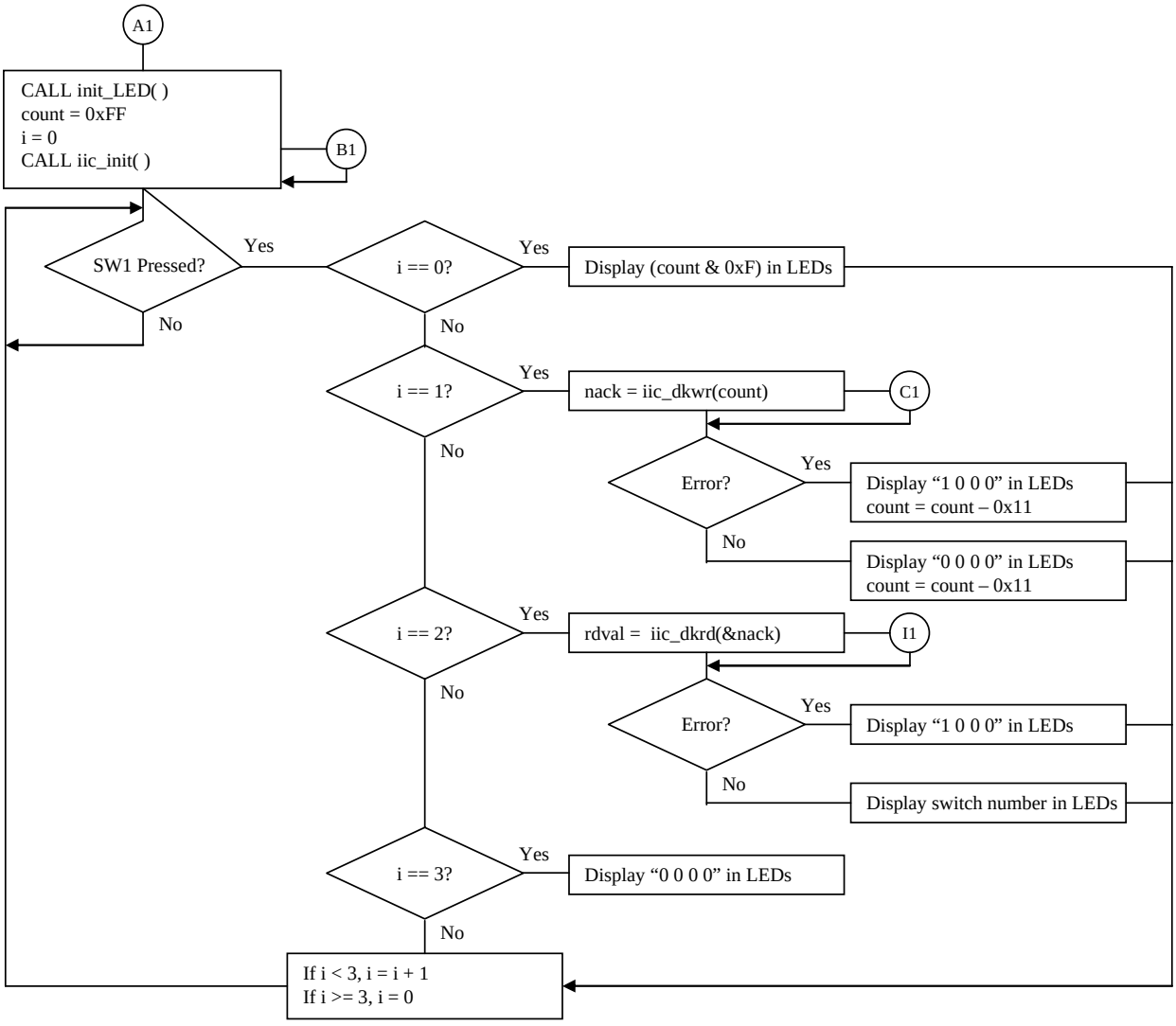
Figure 64. Master Program Startup and Initialization



After the hdwinit() function finishes, the startup code calls the main() function. When main() starts, all peripheral initialization is complete, and there is no need to call individual peripheral-initialization routines.

5.3.2 Master: Main() –Main Program for IIC Master

Figure 65. Flowchart for Main Program



The main() routine begins once the program startup code finishes. Main() initializes the LED display, sets the local count variable to 0xFF, sets the local **i** variable to zero, and calls iic_init() to initialize the ports used for IIC communication.

Main() then waits for SW3 to be pressed. When the routine detects a closure of SW3, the reaction depends on the value of the **i** variable, with different actions taken for **i** equal to 0, 1, 2, or 3. After it takes the indicated action, main()increments **i**, if **i** is less than three, or sets **i** to zero if greater than or equal to three. Thus, main() cycles through four different actions for each set of four closures of SW1.

On the first switch closure (**i** equal to zero), the routine displays the lower four bits of the count variable on the four LED displays, indicating the value written in the next step.

On the second switch closure, main() calls the iic_dkwr(count) routine to write the count value to the slave using IIC communication. The routine returns an error status, which is stored in the **nack** variable. If the write is successful, the routine blanks the LED display. If an error occurs, the display has the high bit on. In either case, the routine modifies the count value by subtracting 0x11. Given the initial value of 0xFF, this results in successive count values of 0xEE, 0xDD, 0xCC, etc.

On the third switch closure, main() calls the iic_dkrd(&nack) routine to read the last switch value from the slave with an IIC read operation, and places the value in the **rdval** variable. The parameter &nack points to the **nack** variable used to hold error status on return. If there is an error, the high bit of the LED display is on and the others are off.

If the read is successful, the routine converts the returned value to a number indicating which switch was pressed on the slave. A value of 0x01, for example, indicates the UP switch was pressed, and the routine displays the value one (0 0 0 1). The correspondence between returned values and display is shown in the table below.

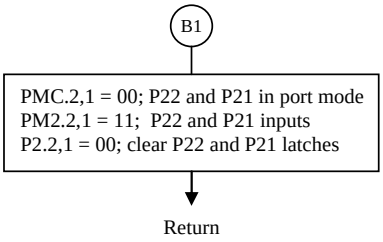
Table 31. IIC Demonstration Display Values

Return value	Display	Comment
0x00	0 0 0 0	No switch pressed yet
0x01	0 0 0 1	UP switch was last switch pressed
0x02	0 0 1 0	DOWN switch was last switch pressed
0x04	0 0 1 1	RIGHT switch was last switch pressed
0x08	0 1 0 0	LEFT switch was last switch pressed
0x10	0 1 0 1	SELECT switch was last switch pressed
Other	0 1 1 1	Multiple switches down at the same time
On error:	1 0 0 0	Error on read

On the fourth switch closure, the LED display is blank to provide a pause before showing the next value of the count, where **i** will be zero again.

5.3.3 Master: iic_init() – Initialize Port Pins for IIC

Figure 66. Initializing Port Pins



Main() calls the iic_init() routine to set up the port pins used for IIC communication—pin P22 for SDA, and P21 for SCL.

These port pins operate in a pseudo-open-collector mode. Setting the pin as an output with the output latch set to zero causes the pin to drive low. Setting the pin back to an input turns off the drive and thus allows the pin to float so that an external resistor can pull the line high.

The PMC2 register controls whether P2 pins are used as I/O ports or analog inputs. Setting PMC2 bits 2 and 1 to zero selects port mode on P22 and P21.

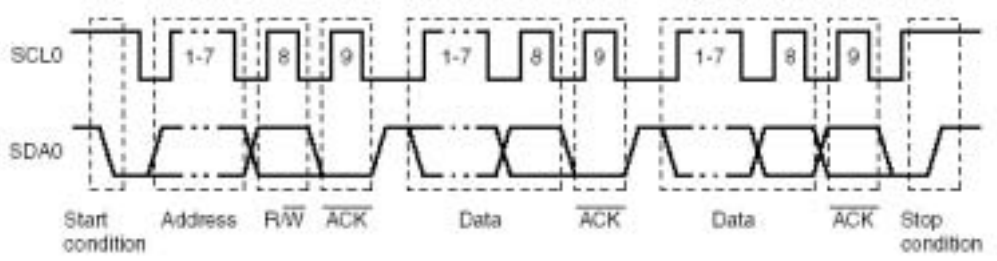
The port-mode register PM2 controls whether P2 pins are inputs or outputs. Setting PM2 bits 2 and 1 to one selects input mode for P22 and P21.

Writing a value to P2 sets or clears the output latches of the P2 pins. The routine clears P2 bits 2 and 1 to zero to clear the output latches. This setting has no effect on the P22 and P21 pins. When the program modifies PM2 to clear the appropriate bit, the code sets P22 and/or P21 to output mode, driving their SDA or SCL lines low.

5.3.4 Master: Low-Level IIC SCL and SDA Transitions

The master program uses a set of subroutines to manipulate the port pins used for SCL and SDA. The signals on these pins indicate various IIC states.

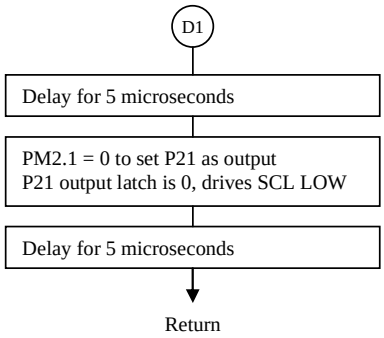
Figure 67. Typical IIC Transmission



Both the master and slave can drive the SCL or SDA lines, or release their drive so that another device can drive the lines. Before a transfer starts, external pull-up resistors pull both SCL and SDA high.

5.3.4.1 Master: iic_SCL_0() – Drive IIC SCL Line Low

Figure 68. Drive IIC SCL Line Low

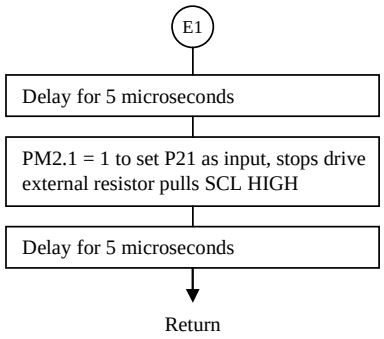


The master routine iic_SCL_0() drives the SCL line low by changing the port-mode register PM2 bit 1 to zero. This bit setting makes pin P21 an output. The P21 output latch is zero, causing P21 to drive SCL low.

There is a delay of 5 microseconds before and after the clock transition, to ensure a minimum of 10 microseconds between clock transitions. As a result, the maximum IIC clock rate is 100,000 bps. Since the SCL and SDA lines are pulled up by external resistors, and there may be capacitance on the lines, this arrangement allows time for the line voltage to rise before other transitions occur. Since a slave may change SDA on the falling or rising edge of SCL, the delay also provides time for the SDA line voltage to rise before being sampled as an input.

5.3.4.2 Master: iic_SCL_1() – Release Drive On IIC SCL Line To Set High

Figure 69. Release Drive On IIC SCL Line To Set High



The master routine iic_SCL_1() sets the SCL line high by changing the port-mode register PM2 bit 1 to one. This bit setting makes the P21 pin an input, no longer driving SCL low. The external resistor pulls the SCL line up, or a slave may hold the line low to signal a wait condition.

5.3.4.3 Master: iic_istart() – Generate IIC Start Condition

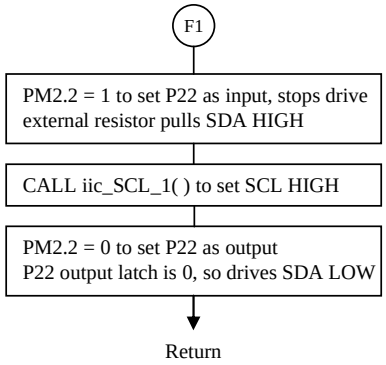


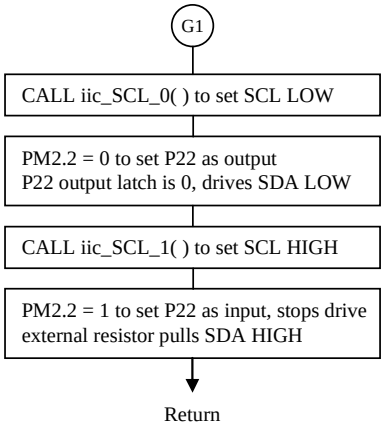
Figure 70. IIC Start

The master routine iic_istart() sets a start condition on the IIC bus, defined as a high-to-low transition on SDA while SCL is high. The routine first sets the port-mode register PM2 bit 2 to a one, which makes P22 an input. An external resistor pulls SDA up to a high. The routine then calls iic_SCL_1() to set a high level on SCL. Note that both SDA and SCL may have been high already, and normally would be prior to a start condition.

The routine then sets port-mode register PM2 bit 2 to a zero, which makes P22 an output. The P22 output latch is zero, causing P22 to drive SDA low. This transition of SDA from high to low while SCL is high creates the start condition.

5.3.4.4 Master: iic_istop() – Generate IIC Stop Condition

Figure 71. Generate IIC Stop Condition

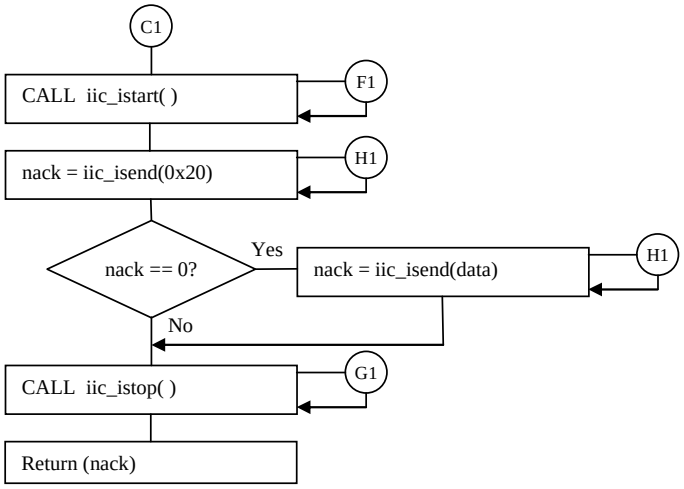


The master routine iic_istop() sets a stop condition on the IIC bus, which is defined as a low-to-high transition on SDA while SCL is high. The routine first calls iic_SCL_0() to set SCL low, and then sets port-mode register PM2 bit 2 to zero, making P22 an output. The P22 output latch is zero and drives SDA low. SDA is set low to prepare for the transition from low to high. SCL is set low before this, to avoid causing a start condition by creating an SDA transition to low.

Calling iic_SCL_1() sets SCL high by and then sets PM2 bit 2 to a one, making P22 an input, no longer driving SDA low. The external resistor pulls SDA up, and this transition from low to high creates the stop condition on the IIC bus.

5.3.5 Master: UCHAR iic_dkwr(data) – Write Byte to Slave Using IIC Transfer

Figure 72. Writing Byte to Slave



The iic_dkwr(data) routine performs an IIC transfer to write a byte to the slave. The data parameter is the byte to write. The routine returns a value of zero if there is no error, or a value of one if there is an error.

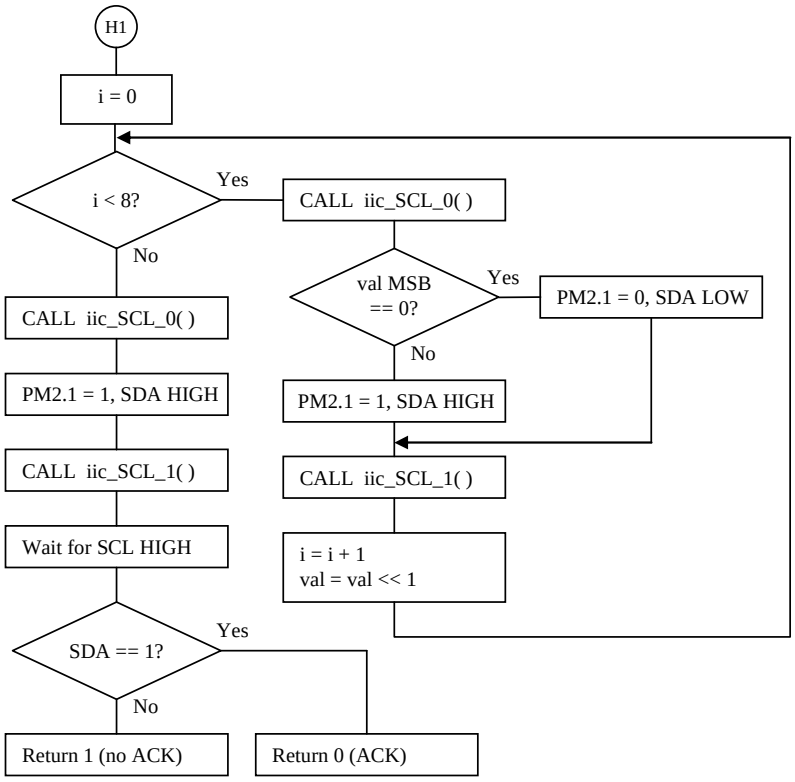
The routine first calls iic_istart() to create an IIC start condition. Then iic_dkwr(data) calls iic_isend(0x20) to serialize 8 bits. These bits consist of the slave address (0x20) with the LSB cleared to zero to indicate a write. Iic_isend(0x20) also returns the acknowledge status from the slave on the next clock.

If iic_isend(0x20) returns a non-zero value, the slave at that address did not acknowledge, and the routine skips data transfer. If iic_isend() returns zero, the slave acknowledged, and data transfer can begin. In this case, iic_isend(data) serializes the 8 data bits, and returns the acknowledge status after the data transfer.

Whether or not the slave acknowledged the slave address or the data, iic_istop() is called to end the transfer. The **nack** variable is returned indicating whether the routine succeeded.

5.3.6 Master: UCHAR iic_isend(val) – Write Serial Byte to IIC Pins

Figure 73. Writing Serial Byte to IIC Pins



The iic_isend() routine serializes 8 bits of data on the IIC bus, checks the acknowledge from the slave, and returns the acknowledge status. The **val** parameter indicates the data byte to send. The return value from the routine is zero if the slave acknowledged the transfer, and the value is one if the slave did not acknowledge.

For 8 data bits, the routine first uses the iic_SCL_0() routine to drive the line low with delays before and after the transition.

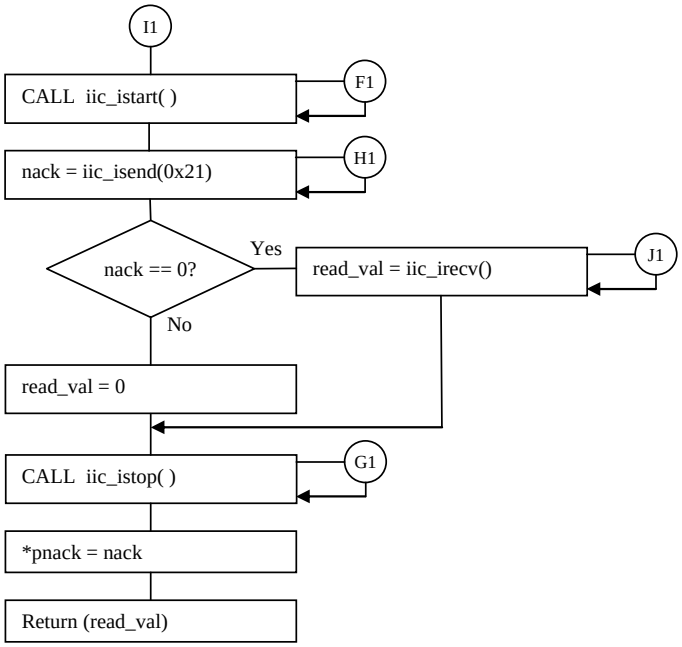
Then iic_isend() puts the most significant bit of **val** on the SDA line. If the MSB is zero, the routine clears the port mode-register PM2 bit 1 to zero, making P22 an output and driving the SDA line low. If the MSB is high, the routine sets PM2.2 to one, making P22 an input and allowing the external pull-up resistor to bring SDA high. After SDA has been set, iic_isend() calls iic_SCL_1() to set the SCL line high, clocking the SDA bit into the slave.

After each bit, iic_SCL_1() increments the bit counter and shifts the **val** value one bit to the left, making the next lower bit the new MSB, in preparation for sending that bit.

After shifting all 8 bits, the routine sets SCL low and SDA high, to release the drive on the SDA line. The program then releases SCL. At this point, the slave may assert a wait by holding SCL low, so the routine waits for the SCL line to go high, reading the P21 input. Once SCL goes high, the slave may be asserting an acknowledge by driving SDA low; otherwise SDA is high. The routine returns a value of zero to indicate an acknowledge, or a value of one to indicate no acknowledge.

5.3.7 Master: UCHAR iic_dkrd(*pnack) – Read Byte from Slave Using IIC Transfer

Figure 74. Reading Byte from Slave



The iic_dkrd(data) routine performs an IIC transfer to read a byte from the slave. The ***pnack** parameter points to a location to return the error status. The routine writes a zero into that location if there is no error, or write a value of one if there is an error. If the read is successful, the routine returns the value of the byte read, otherwise the value is zero.

The routine first calls iic_istart() to create an IIC start condition. Then iic_dkrd(data) calls iic_isend(0x21) to clock out 8 bits. These bits consist of the slave address (0x20) with the LSB set to one (to indicate a read). Iic_isend(0x21) also returns the acknowledge status from the slave on the next clock.

If iic_isend(0x21) returns a non-zero value, there has been no acknowledge from the slave at that address, and iic_dkrd(data) skips the data transfer. If iic_isend() returns zero, the slave has issued an acknowledge, and data transfer can begin. In this case, iic_dkrd(data) calls iic_irecv() to serialize 8 bits of data and store the data in the **read_val** variable.

Whether or not the slave acknowledged the slave address, iic_istop() ends the transfer. The routine writes the **nack** status at ***pnack** and returns the **read_val** variable indicating whether the routine was successful.

5.3.8 Master: UCHAR iic_irecv() – Read Serial Byte from IIC Pins

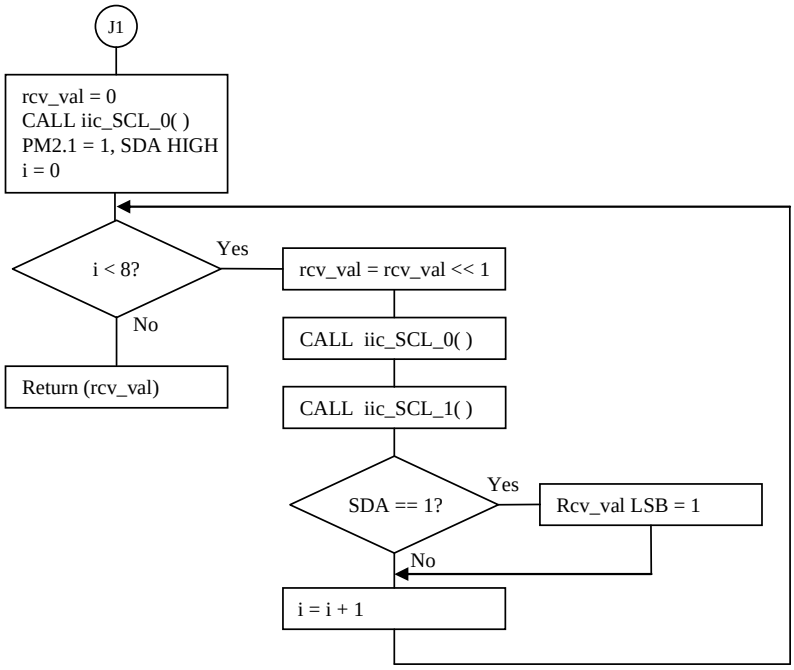


Figure 75. Reading Serial Bytes from IIC Pins

The iic_irecv() routine reads a serial IIC byte and returns the value of the byte read.

The routine sets an initial value of zero in **rcv_val**, where the routine accumulates the read bits. Iic_irecv() sets SCL low. The SDA line is not driven, so the routine can read the input status. Setting SCL low before SDA is allowed to rise avoids creating a stop condition on the bus.

For 8 data bits, a sequence of clocks and read operations is used to read the serial byte. First iic_irecv() shifts the **rcv_val** variable left one bit, to shift previous bits up by one. The routine sets the SCL line low, and then high. At this point, the slave can drive the SDA line low to set a zero, or keep the line high to indicate a one. The iic_irecv() routine reads the SDA line. If the line is at a value of one, the routine sets the least significant bit of **rcv_val** to one.

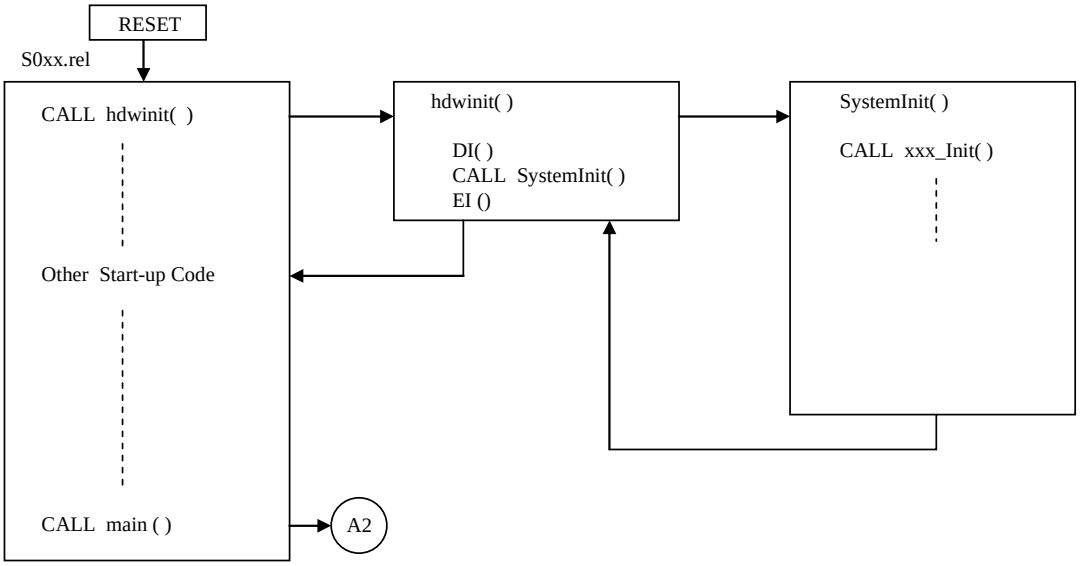
After eight shifts and bit reads, bit 7 of **rcv_val** contains the first bit read (MSB) from the slave, and bit 0 of **rcv_val** has the eighth bit read (LSB). Iic_irecv() returns the **rcv_val** value to the caller.

5.3.9 Slave: Program Startup and Initialization

For 78K0 programs written in the C language, an object code file such as s0l.rel is linked to the program and provides the startup code for the C program. This startup code calls a function named hdwinit(). You can place hardware-initialization code here.

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

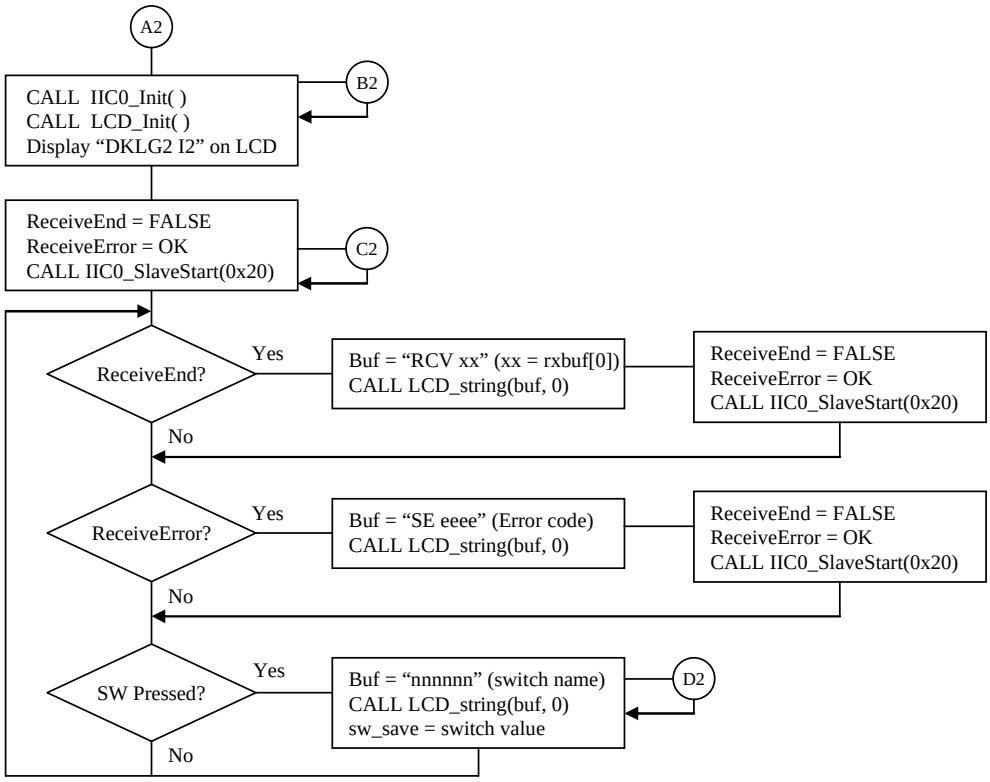
Figure 76. Slave Program Startup and Initialization



After the hdwinit() function finishes, the startup code calls the main() function. Thus, when main() begins, all peripheral initialization has been done, and main() does not need to call individual peripheral-initialization routines.

5.3.10 Slave: Main() –Main Program for IIC Slave

Figure 77. Slave Main Program



The slave program main() routine is called after the program startup code completes. Main() initializes the IIC0 controller by calling IIC0_Init(), and then calls LCD_Init() to initialize the LCD controller. The LCD_Init() routine requires IIC0 operation in master mode to set values to the LCD-controller control registers. The routine displays a startup string on the LCD, which also requires IIC0 operation. An example of master mode IIC0 operation is shown below for the LCD_string() function.

To allow both slave- and master-mode operation, main() sets flags for **SlaveReceiveEnd** and **SlaveError** to inactive values, and calls IIC0_SlaveStart(0x20) to set up for slave response to slave address 0x20.

Main() then loops, waiting for one of three things to occur: a **SlaveReceiveEnd**, indicating that the master has sent a data value to be displayed; **SlaveError** set true, indicating that an IIC0 error has occurred in slave mode; or a switch closure.

If **SlaveReceiveEnd** is true, the received data is in the rxbuf[0] location. Main() displays “RCV xx” on the LCD, indicating the value received, and sets up for slave operation again..

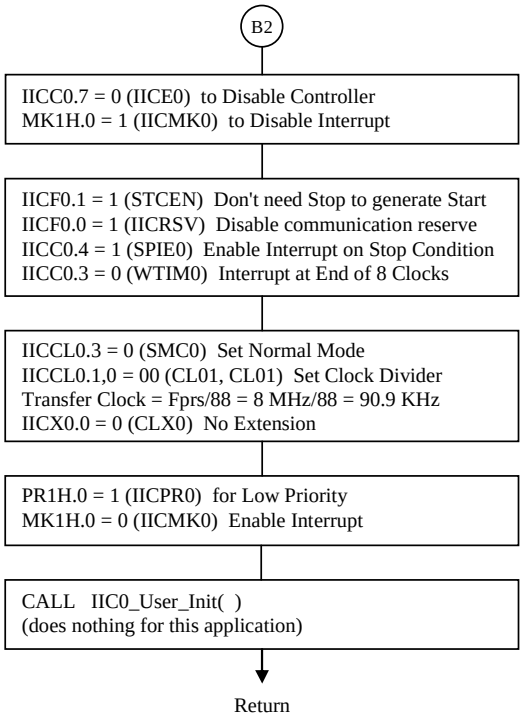
If **SlaveError** is set, main() displays the error code on the LCD as “SE eeee”, and sets up for slave operation again.

If you have pressed a switch, the routine displays a string corresponding to the switch pressed (“UP” for the UP switch, etc.) and stores the value of the switch input in the **sw_save** location.

Note that the last switch value does not transfer to the master in the main() routine. In slave mode, the interrupt-service routine for IIC0 takes care of this transfer.

5.3.11 Slave: IIC0_Init( ) – IIC0 Controller Initialization

Figure 78. IIC0 Controller Initialization



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

The initialization routine sets IICF0 (IIC0 flag register) bits STCEN and IICRSV to one. STCEN=1 allows the controller to generate 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 one, to enable the INTIIC0 interrupt on a stop condition. Setting bit WTIM0 to 0 sets up for an IIC0 interrupt on the eighth data bit—a normal practice for slave mode. After initialization, the IIC0 controller is operating in slave mode, and enters master mode by starting a master mode function. At that point, setting the WTIM0 bit to one enables interrupts on the ninth bit—standard practice for master mode.

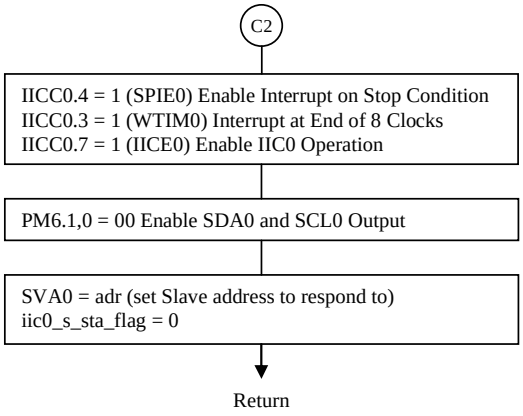
IIC0_Init() sets the IICCL0 (IIC0 Clock register) and ICCX0 (IIC0 Extension register) to specify the IIC communication speed. These registers are set for normal mode, with the clock set at $f_{PRS}/88$. Using the 8-MHz peripheral clock, this setting results in an IIC clock of 8,000,000/88 = 90.9 KHz, which is within the standard IIC maximum of 100 KHz for normal-speed devices.

The routine readies the IIC0 peripheral for operation by setting the IIC0 interrupt priority to low and the interrupt mask to enable the IIC0 interrupt INTIIC0. However, the peripheral is not enabled until IICC0 bit 7 (IICE0) is set to 1.

Finally, the initialization routine calls the IIC0_User_Init( ) function to perform user-specified initialization. For this application, this routine is empty and just returns.

5.3.12 Slave: IIC0_SlaveStart(adr) – Set Up Slave Operation at Specified Slave Address

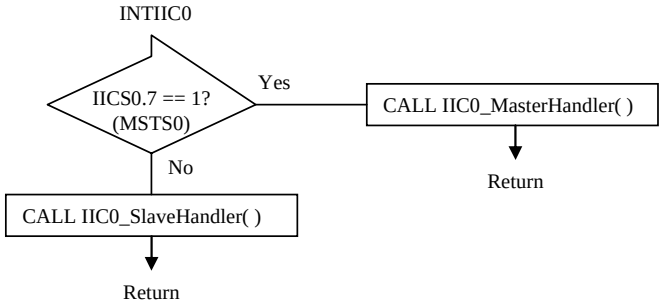
Figure 79. Setting Up Slave Operation at Specified Address



The IIC0_SlaveStart() routine sets the IIC0 peripheral for slave mode operation, first setting the IICC0 control register bits SPIE0, WTIM0, and IICE0 to one.

The SPIE0 bit allows an INTIIC0 interrupt when a stop condition is detected. This arrangement allows the slave to recognize when the master has ended transmission by setting the stop condition.

Setting the WTIM0 bit to one causes the INTIIC0 interrupt to be generated after shifting nine data bits. In slave mode, this means that the INTIIC0 interrupt does not occur until after shifting the 8-bit slave address



and ensuring that it matches the local slave address in the SVA0 register. Note that once the slave address is recognized, the WTIM0 bit setting depends on whether the slave is receiving data (WTIM0 = 0) or sending data (WTIM0 = 1).

5.3.13 Slave: MD_INTIIC0( ) - IIC0 Interrupt-service routine

Figure 80. IIC0 Interrupt-service routine

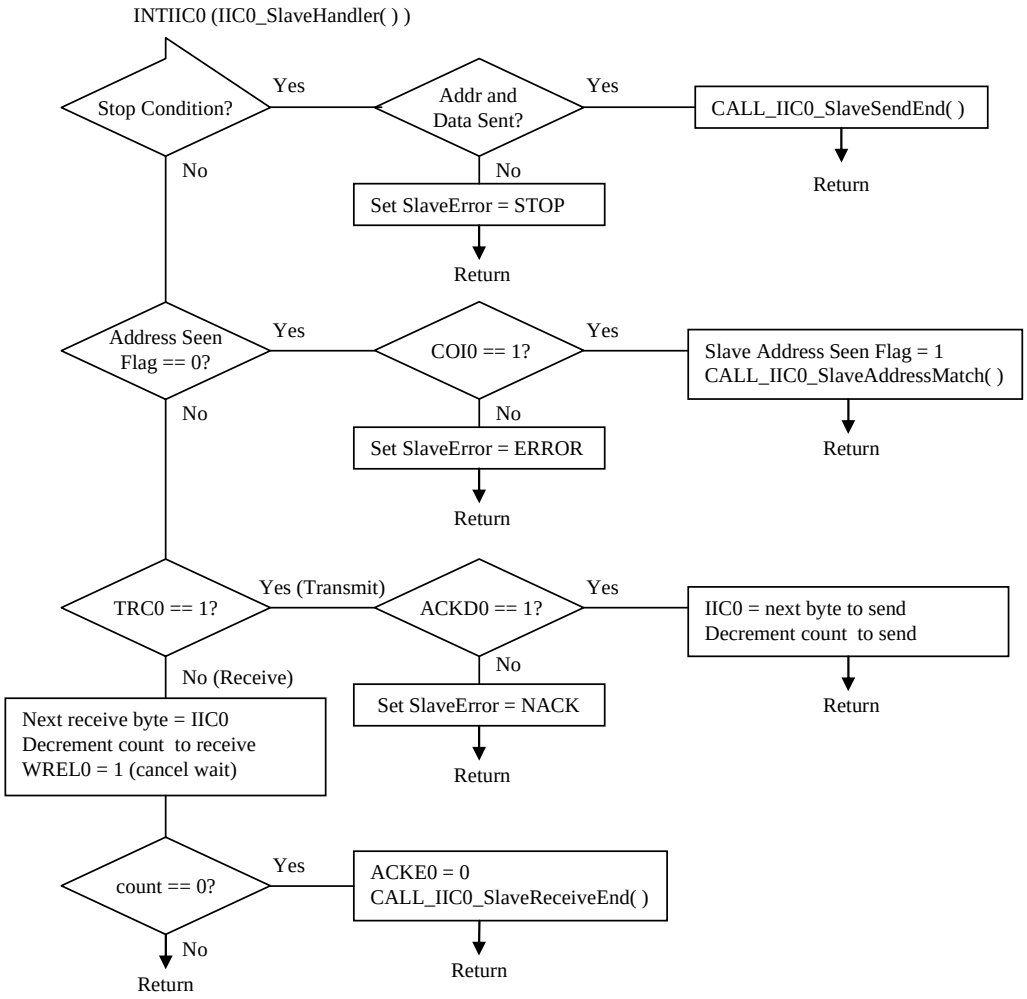
An IIC0 event asserts the INTIIC0 interrupt, invoking the MD_INTIIC0() interrupt-service routine. Because operations differ depending on the operating mode (slave or master) the MD_INTIIC0() routine just checks the state of the MSTS0 bit in the IICS0 status register.

If the MSTS0 bit is one, the IIC0 peripheral is in master mode after having started a transmission, and the IIC0_MasterHandler() routine handles the interrupt.

If MSTS0 is zero, the IIC0 peripheral has not started an operation, and is in slave mode. In this case, the IIC0_SlaveHandler() routine handles the interrupt.

5.3.14 Slave: IIC0_SlaveHandler() – IIC0 Interrupt-service routine (Slave Mode)

Figure 81. IIC0 Interrupt-service routine (Slave Mode)



MD_INTIIC0() calls the IIC0_SlaveHandler() routine after an INTIIC0 interrupt when the device is in slave mode. The routine handles IIC0 interrupts from a number of possible causes.

The routine checks on whether a stop condition has occurred. If so, and if a slave address has been recognized and data has been sent, the stop has occurred at the end of sending data to the master. In that case, IIC0_SlaveHandler() calls CALL_IIC0_SlaveSendEnd(). In the demonstration, CALL_IIC0_SlaveSendEnd() calls IIC0_SlaveStart() to prepare for another slave operation. The end of sending does not require any action from the main() routine. If a stop condition has occurred, but no slave address has been seen, or data has not been sent, the routine sets a stop error. In this case, the master may have stopped the transfer.

If the INTIIC0 interrupt was not caused by a stop condition, the slave checks to see if a slave address has been sent. If so, and if the COI0 bit (IICS0.4) is set (indicating an address match with the SVA0 register), then the slave address from the master matches the local address. The slave sets its address-seen flag and calls CALL_IIC0_SlaveAddressMatch() to deal with data sending or receiving. If the address has not been seen, and the COI0 bit is not set, an error has occurred.

If the INTIIC0 interrupt was not caused by a stop condition or by receiving a slave address, then the interrupt was caused by the end of a data byte being received or sent. The TRC0 bit (IICS0.3) should equal one if sending to the master, or zero if receiving.

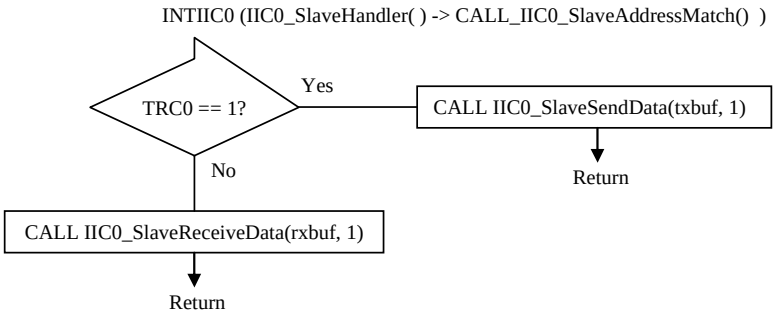
If the slave is sending, IIC0_SlaveHandler() checks the ACKD0 flag for an acknowledge from the master. If the acknowledge has been received, the routine writes the next byte to send to the IIC0 register to transmit, decrements the count of bytes to send, and returns. If the master has not acknowledged the previously sent byte, the routine sends an error.

If the slave is receiving, the routine reads the data from the IIC0 register and stores it in the receive buffer, decrements the count of bytes to be received, and then sets the WREL0 bit (IICC0.5) to cancel the wait condition. The wait is automatically set once the eighth data bit is received, to keep the master from continuing to send data while the slave reads the current byte.

When the count of bytes to receive reaches zero, the receive operation is done. IIC0_SlaveHandler() clears the ACKE0 bit (IICC0.2) to prevent acknowledgement of further data bytes, and calls the CALL_IIC0_SlaveReceiveEnd() callback routine to deal with the end of reception. In the demonstration, this routine sets the **SlaveReceiveEnd flag**, to allow main() to deal with the received data.

5.3.15 Slave: CALL_IIC0_SlaveAddressMatch() – Callback Routine for Slave Address Match

Figure 82. Callback Routine for Slave Address Match



When the IIC0_SlaveHandler() interrupt service sees an INTIIC0 interrupt and the slave address sent by the master matches the local slave address set in the SVA0 register, IIC0_SlaveHandler() calls the

CALL_IIC0_SlaveAddressMatch() routine. At this point, the TRC0 bit (IICS0.3) is set to the value of the LSB of the slave address, indicating a transmission from the master if TRC0 is zero, or a read by the master if TRC0 is one.

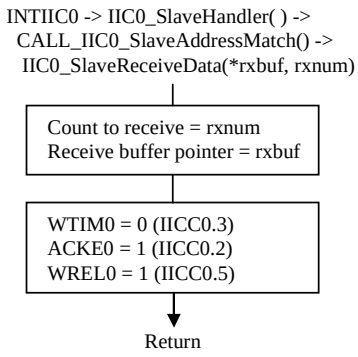
How the program handles writes or reads is application dependant, and the callback function allows the application to specify what actions to take. In the demonstration, if the master is writing data, the slave should receive a count value. The routine IIC0_SlaveReceiveData(rxbuf, 1) configures the system to receive one byte of data into the receive buffer.

If the master is reading data, the slave sends the last switch value, which has been put in txbuf[0]. The IIC0_SlaveSendData(txbuf, 1) routine sends one byte of data from this buffer.

Note that this action takes place inside the interrupt service of INTIIC0, and so should not produce a delay. The slave asserts a wait condition by holding SCL0 low, releasing the wait later in one of the routines to send or receive data.

5.3.16 Slave: IIC0_SlaveReceiveData(*rxbuf, rxnum) – Prepare for IIC0 Data Reception (Slave Mode)

Figure 83. Preparing for IIC0 Data Reception



The IIC0_SlaveReceiveData(*rxbuf, rxnum) routine starts receiving data from the master in response to a slave address with the LSB set to zero, indicating a write from the master.

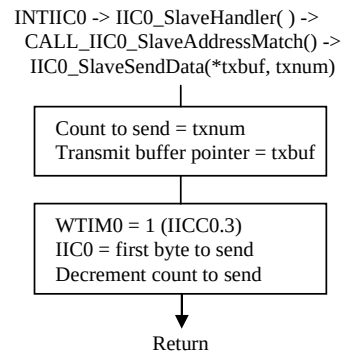
The **rxnum** parameter indicates the number of bytes to receive (one, in this application) and the routine saves the value to a local counter. The ***rxbuf** parameter specifies the location for storing data. This value is written to the local receive-data pointer.

IIC0_SlaveReceiveData(*rxbuf, rxnum) sets the WTIM0 bit (IICC0.3) to zero. This setting specifies that the INTIIC0 interrupt occurs after the eighth data bit, rather than the ninth. This is the normal setting for the receiving side of a transfer, allowing the slave to assert a wait after the eighth bit and deal with the data received.

The routine sets the ACKE0 bit to acknowledge subsequent data bytes received, and sets the WREL0 bit to one to release the wait condition after receiving the slave address.

5.3.17 Slave: IIC0_SlaveSendData(*txbuf, txnum) – Queue Data for IIC0 Transmission (Slave Mode)

Figure 84. Queue Data for IIC0 Transmission (Slave Mode)



The IIC0_SlaveSendData(*txbuf, txnum) routine starts data transmission from slave to master in response to a slave address in which the LSB is set to one.

The **txnum** parameter indicates the number of bytes to send (one, in this application). The routine saves this value to the local counter. The ***txbuf** parameter specifies the location of data to send. This value is written to the local transmit data pointer.

The routine sets the WTIM0 bit (IICC0.3) to one. This setting specifies that the INTIIC0 interrupt occurs after the ninth data bit, rather than the eighth. This is the normal setting for the sending side of a transfer, allowing the receiver to assert a wait after the eighth bit.

IIC0_SlaveSendData(*txbuf, txnum) writes the first data byte to the IIC0 register (which releases the wait condition) and decrements the data count. Handling of the next data byte (when more than one is sent) takes place on the next INTIIC0 interrupt.

5.3.18 Slave: LCD Functions (Master Mode IIC Transfers)

The slave uses the IIC0 peripheral in master mode to communicate with the on-chip LCD controller. A brief summary of the LCD functions is provided below.

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

The LCD_string() function writes a string to the LCD at a specified display position. This routine calls LCD_putc(dpos, data) to write individual digits.

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

The LCD_putc() routine takes one data character and displays it at the selected display position.

The routine's data value addresses the character-segment data array, which holds character bit-map data in 16-bit (four 4-bit values) format. The single 16-bit value is written into a two-byte array (transmit_buffer[]). The routine then calls LcdDrvSegWrite(&transmit_buffer[0], digit * 2, 2), to write the two bytes into four successive four-bit locations addressed by digit * 2.

5.3.18.3 Slave: LcdDrvSegWrite(*src, addr, size) - Write Character Data to LCD Segment Memory

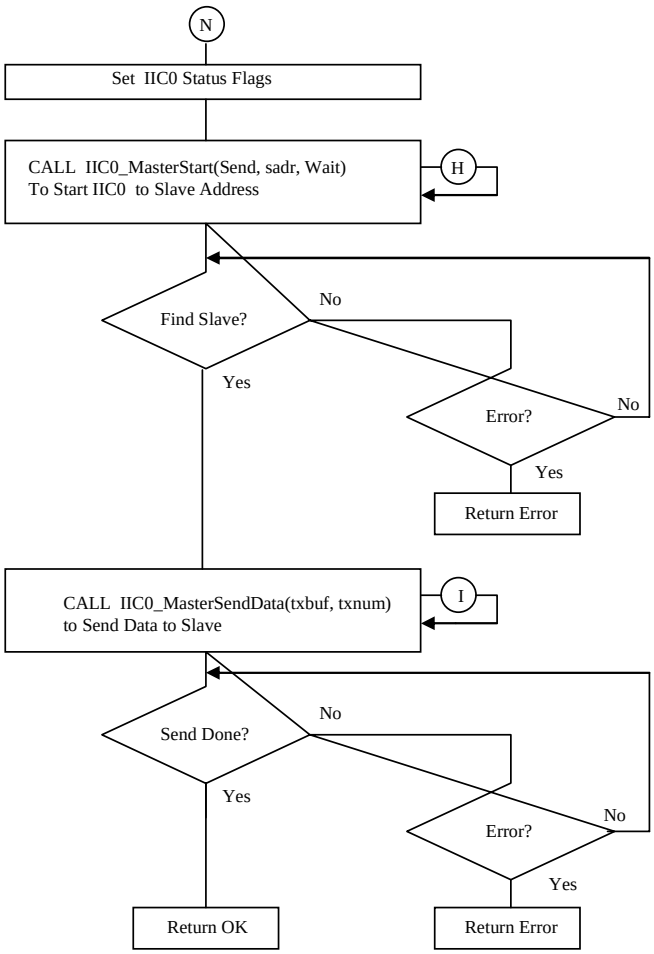
The LcdDrvSegWrite() routine writes data bytes to the LCD controller's LCDSEG memory. The ***src** parameter specifies a pointer to a byte array. The **addr** parameter specifies the byte address in LCDSEG memory at which to write the data, and **size** indicates the number of bytes to write.

The routine sets up a buffer of data 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 routine multiplies the byte address by two to get the LCDSEG location. The routine divides the bytes in the input **src** array into two bytes (each with four bits of data) in the succeeding buf[] array locations.

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

5.3.19 Slave: IIC0_MasterStartAndSend(sadr, *txbuf, txnum) - Start and Send IIC0 Data

Figure 85. Start and Send IIC0 Data



Though not created by the Applilet, the IIC0_MasterStartAndSend(sadr, *txbuf, txnum) routine is a simple combination function of IIC0_MasterStart() and IIC0_MasterSendData() for sending IIC data as a master.

IIC0_MasterStartAndSend(sadr, *txbuf, txnum) first sets IIC communication flags to their default settings. During the IIC communications, the IIC0 alters 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 IIC0_MasterStart(Send, sadr, 10) to begin IIC communications as a master for sending data, and specifies the slave address sadr. On return, either the routine has enabled the IIC0 peripheral and written the slave address to the IIC0 register for transmission, or the IIC communication channel is busy, which should not occur in the demonstration system.

The routine then waits for either **UI_MasterFindSlave** to be OK, or for **UI_MasterError** to indicate an error condition.

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

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

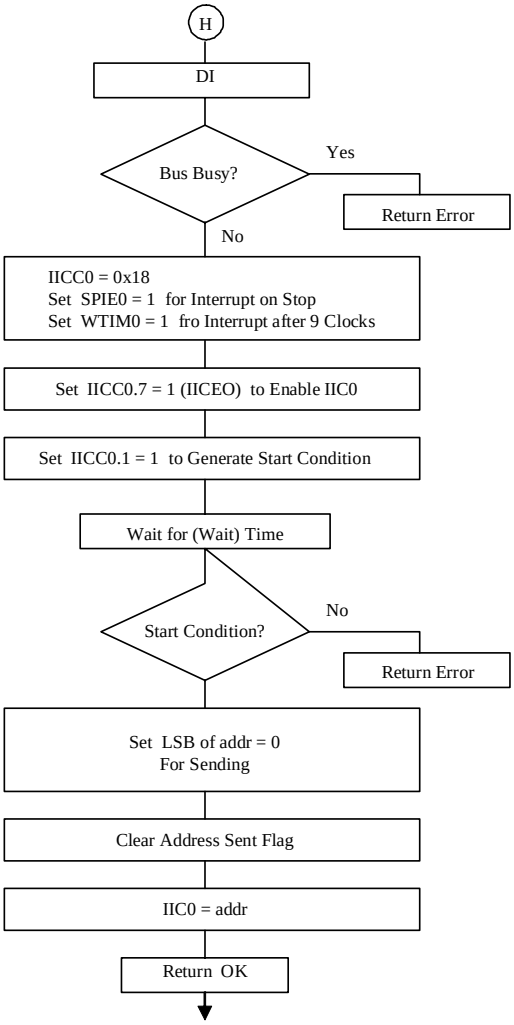
IIC0_MasterSendData() stores the buffer address and count, writes the first byte of data to the IIC0 register, and returns.

An INTIIC0 interrupt occurs after every data byte transmission, invoking the MD_INTIIC0 interrupt-service routine. This routine sends the next data byte, setting **UI_MasterSendDone** if all bytes have been sent and received. The routine sets 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 indicate an error. The routine then returns the operation status to the caller.

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

Figure 86. Start IIC0 Sending To Slave Address



The IIC0_MasterStart(Send, addr, wait) function starts master-mode IIC communication.

The routine checks to see if the IIC bus is busy, and if so returns an error. IIC0_MasterStart(Send, addr, wait) sets the IICC0 register bits SPIE0 and WTIM0 to enable interrupts on stop conditions, and to interrupt after 9 clocks (master mode). Then the routine sets IICC0 bit 7 (IICE0) to 1 to enable the IIC0 peripheral.

Setting IICC0.1 (STT0) to 1 generates a start condition, and the routine waits a short time to allow the start condition to be registered. IICS0.1 (STD0) indicates that a start condition exists. If a start condition is not detected, the routine returns an error.

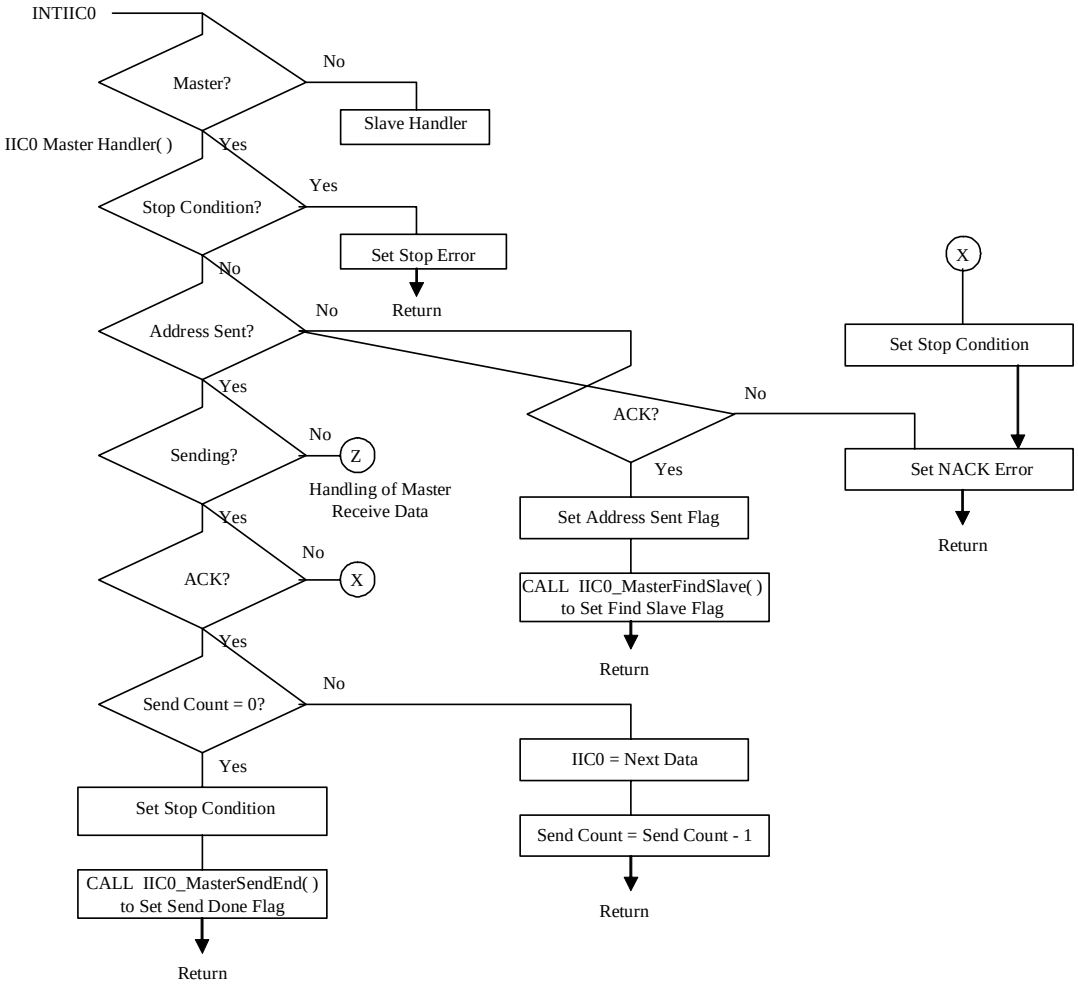
Setting the LSB of the slave address to 0 indicates a master-to-slave transmission of data, and the routine writes the slave address to the IIC0 register to start the IIC transfer. The IIC0_MasterStart(Send, addr, wait) routine clears the address-sent flag, which is used by the MD_INTIIC0 interrupt-service routine to determine the action to take on interrupt.

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

Note that if the first parameter is “receive” instead of “send,” the IIC0_MasterStart() function initiates a receive operation, transferring data from slave to master.

5.3.21 Slave: IIC0_MasterHandler( ) - IIC0 Interrupt-service routine (Master Mode)

Figure 87. IIC0 Interrupt-service routine (Master Mode)



The MD_INTIIC0() interrupt-service routine, and the functions it calls, do most of the IIC-transfer work. The interrupt-service routine determines whether a master or slave operation is in progress, and calls either IIC0_MasterHandler() or IIC0_SlaveHandler(). Communications with the LCD controller use master communication, so the flowchart above shows the flow of IIC0_MasterHandler().

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

The handler checks to see if a stop condition has been detected. This condition should not occur in normal master-mode communication, so if it does, the routine calls UI_MasterError().

The handler then checks the slave address-sent flag. If this flag has not been set, the interrupt occurred after sending the slave address. In that case, the routine checks to see if it received an ACK from the slave. If not, the routine sets the NACK error condition. If the ACK has been received, the slave responded to the address properly. The handler sets the **UI_MasterFindSlave** flag, sets the address-sent flag true, and returns.

After the first interrupt, further interrupts occur after sending each data byte, or after a receive operation has started. The handler checks the transfer direction to see if it is sending (master to slave) or receiving (slave to master).

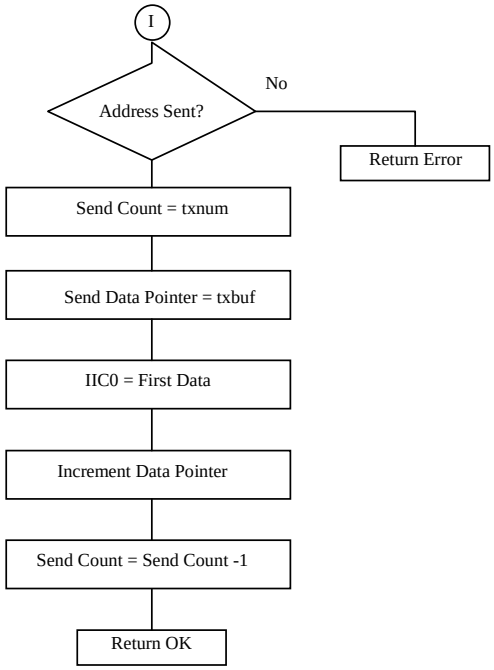
If it is sending, the handler expects an ACK from the slave after sending each data byte. If the handler does not detect an ACK, it sets the NACK error condition and returns.

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

If there are no more bytes to be sent, the transfer is done and the handler sets a stop condition to inform the slave, sets the **UI_MasterSendDone** flag, and returns.

5.3.22 Slave: IIC0_MasterSendData(*txbuf, txnum) – Queue Data For IIC0 Transmission (Master Mode)

Figure 88. Queue Data For IIC0 Transmission (Master Mode)



After IIC0_MasterStart() starts, and the **UI_MasterFindSlave** flag indicates that the slave is there and ready for data, IIC0_MasterStart() calls.

IIC0_MasterSendData(*txbuf, txnum) checks to see if the address-sent flag has been set, and returns an error if not. This error would result if the routine were called before the INTIIC0 interrupt occurs at the end of a slave-address transmission.

The routine sets the data-buffer address to the location pointed to by txbuf, sends the count of bytes to send to txnum, writes the first byte to the IIC0 register, and increments the pointer and decrements the count. The MD_INTIIC0 interrupt-service routine sends the remaining bytes.

5.4 Applilet's Reference Drivers

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 versions of the Applilet used.

The Applilet produces the basic initialization code and main function for the programs, driver code for the IIC0 peripheral, as well as code to manage the I/O ports used for switch input and display output. After the Applilet produces the basic code, you can add code to customize the program.

This section describes how to set up the Applilet to produce code to initialize the ports for the master program, and to produce code for the IIC0 peripheral for the slave program.

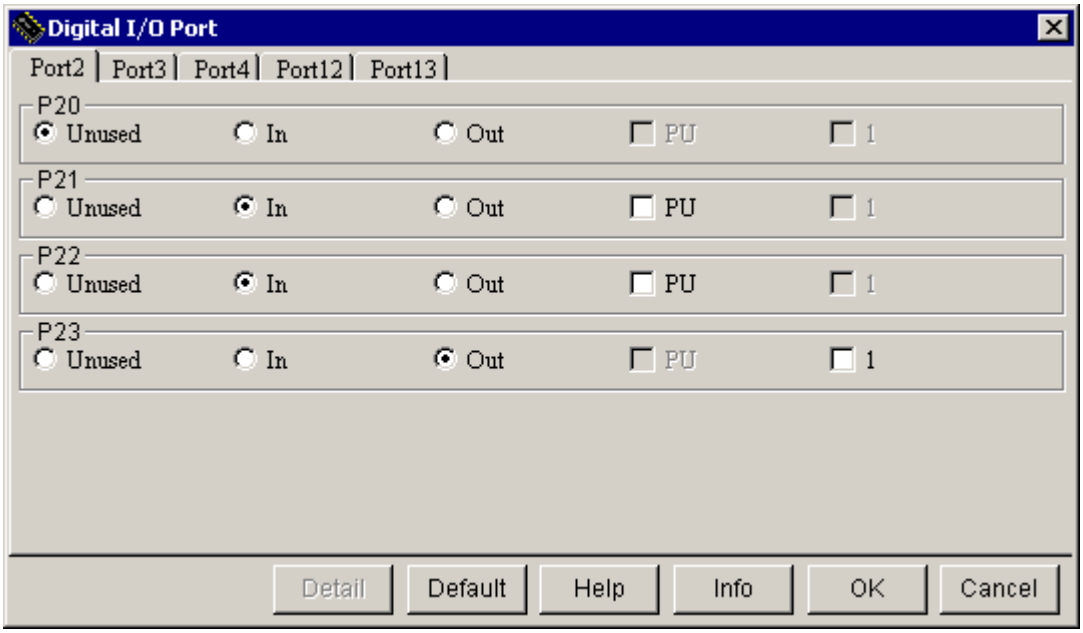
When you start the Applilet and select the target device, save your settings to a new project (.prx) file. The Applilet displays a dialog box that lets you select the peripheral blocks you want to set up.

The IIC demonstration uses two separate NEC Electronics microcontrollers, and requires two separate Applilet projects. One project is for the master processor, initiating IIC transfers using port pins. The other project is for the slave processor, using the IIC0 peripheral for slave-mode response to the master, and master-mode transfers to the LCD controller.

5.4.1 Master: Configuring Applilet for IIC Port Pin Use

The master processor uses the I/O port pins P21 (SCL) and P22 (SDA) for IIC communication. In the Applilet project for the master processor, select **Digital I/O Port** to bring up a dialog box showing the various I/O ports. Select **Port2**.

Figure 89. Selecting Digital I/O Port



Select **P21** and **P22** as inputs without internal pull-up resistors (**PU** not checked). These settings initialize port-mode register PM2 and pull-up register (PU2) bits controlling these pins

5.4.2 Master: Generating Code With Applilet

Once you have set up the Port2 dialog box, click **Generate code**. The Applilet shows the peripherals and functions to generate, and allows you to select a directory for storing the source code.

When you click **Generate**, 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 initializing the I/O ports, the Applilet generates port.h and port.c.

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

5.4.3 Master: Applilet-Generated Port Files and Functions

The files port.h and port.c contain the code generated for I/O port initialization.

5.4.3.1 Port.h

The header file port.h contains definitions of values to write into the PMn, Pun, and Pn registers for each port, as well as a declaration of the PORT_Init() function.

5.4.3.2 Port.c

The source file port.c contains the following function generated by the Applilet.

void PORT_Init(void)
This routine initializes each I/O port by writing values to the appropriate registers. Writing to the associated port-output latch register Pn sets the initial output state. Writing to the associated pull-up-resistor register Pun sets whether an internal pull-up resistor is used or not. And writing to the associated port-mode register PMn controls the input or output status of each port pin.

5.4.4 Master: Other Applilet-Generated Files

For the demonstration program, the Applilet generates several other source files. The files and their functions are shown below.

Table 32. Other Applilet-Generated 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
Int.h	Interrupt-related definitions
Int.c	INT_Init() function
Int_user.c	User code for INTP0 interrupt

5.4.5 Master: Demonstration Program IIC Emulation Files

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

Table 33. Demonstration-Program Files not generated by Applilet

File	Function
Iic_dkka1.h	Header file for IIC emulation for DemoKit-KA1+
Iic_dkka1.c	Data and functions for IIC emulation for DemoKit-KA1+

These files contain the following items.

5.4.5.1 Iic_dkka1.h

The header file iic_dkka1.h contains declarations of the functions in the iic_dkka1.c source file.

5.4.5.2 Iic_dkka1.c

The source file iic_dkka1.c contains the following functions for IIC emulation on the DemoKit-KA1+.

void iic_init(void)
This routine initializes P21 and P22 states for IIC operation.

void iic_SCL_0(void)
This routine sets the P21 output state to drive SCL low, with 5-microsecond delays before and after transition that limit the transfer rate to 100,000 bps maximum.

void iic_SCL_1(void)

This routine sets the P21 input state to let SCL be pulled high, with 5-microsecond delays before and after transition that limit the transfer rate to 100,000 bps maximum.

void iic_iack(void)

This routine sends an ACK (acknowledge) after a data transfer but is not used in this application. The routine would be used in multiple-byte transfers.

void iic_inack(void)

This routine sends a NACK (not acknowledge) after a data transfer but is not used on this application. The routine would be used in multiple-byte transfers.

void iic_istart(void)

This routine sets a start condition on the IIC bus.

unsigned char iic_irecv(void)

This routine receives a byte over the IIC bus by clocking in 8 data bits.

unsigned char iic_isend(unsigned char val)

This routine sends a byte over the IIC bus by clocking out 8 data bits and checking for acknowledge. The data to send is parameter **val**. The return value is zero for an acknowledge, one if no acknowledge.

void iic_istop(void)

This routine sets a stop condition on the IIC bus.

unsigned char iic_dkrd(unsigned char *pnack)

This routine performs an IIC transfer to read one byte from slave address 0x20. The returned value is the byte read. Parameter ***pnack** is a pointer to a location at which to store error status.

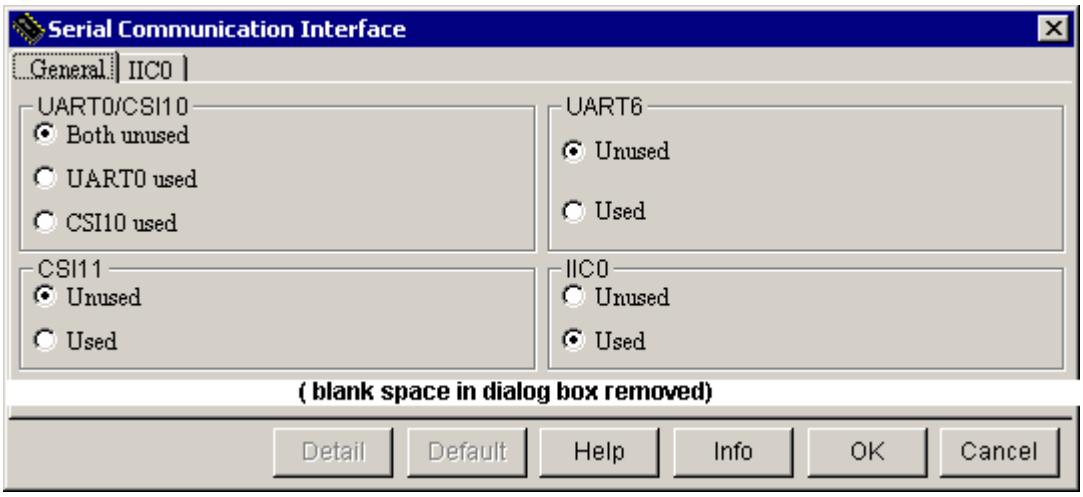
unsigned char iic_dkwr(unsigned char idata)

This routine performs an IIC transfer to write one byte to slave address 0x20. The parameter **idata** is the data to write. The return value is the error status.

5.4.6 Slave: Configuring Applilet for IIC0 Communication

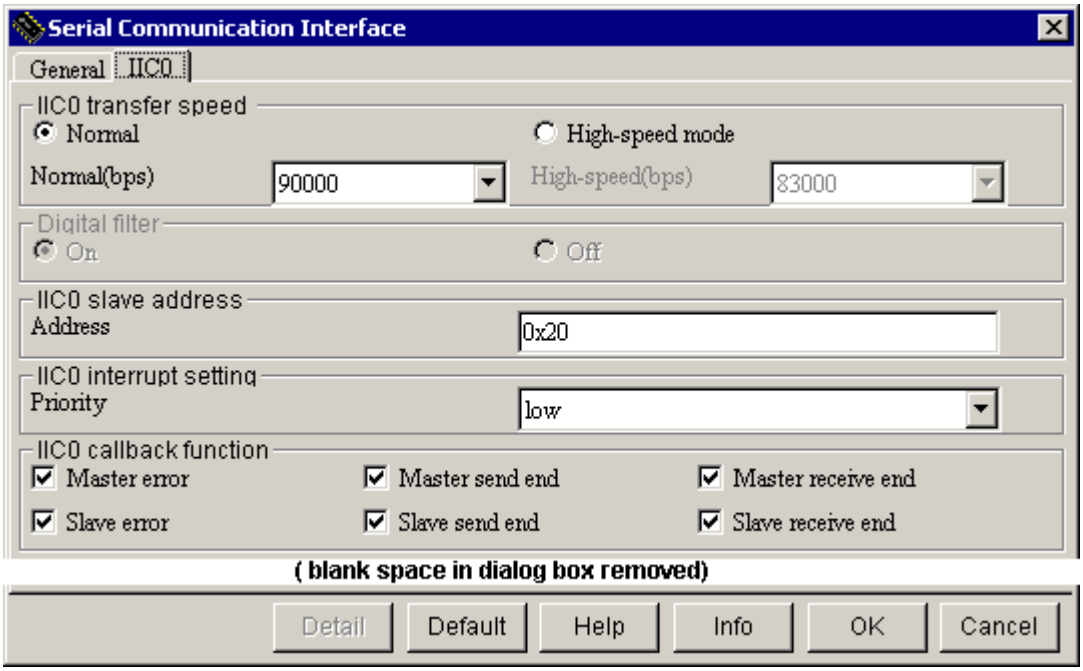
In the Applilet slave project, select **Serial Communication Interface** to bring up the serial-communications dialog box. Then select the **IIC0** peripheral as **Used**.

Figure 90. Serial-Communication Interface Dialog



Once you have selected IIC0, click **Detail** to bring up the detail dialog box.

Figure 91. IIC Detail Dialog



Select the **IIC0 transfer speed** for **Normal**. From the drop-down list that gives a selection of communication speeds based on divisions of the peripheral clock, select the highest available speed—90000 bps.

The IIC0 peripheral operates in either slave or master mode and can switch between the two modes. The peripheral is in slave mode unless it is in the middle of a transfer it initiated as a master. When in slave mode, the peripheral responds to IIC transmissions from an external master which match the local slave address. This demonstration uses both slave and master mode. Set the **IICO slave address** to **0x20**. Set the **IIC0 interrupt setting** to **low** priority.

The Applilet generates several standard operating routines and can generate blank callback routines, which allow you to insert application-specific code. The demonstration uses all of these callback routines to support master and slave communication, so under **IIC0 callback function** check all of the options.

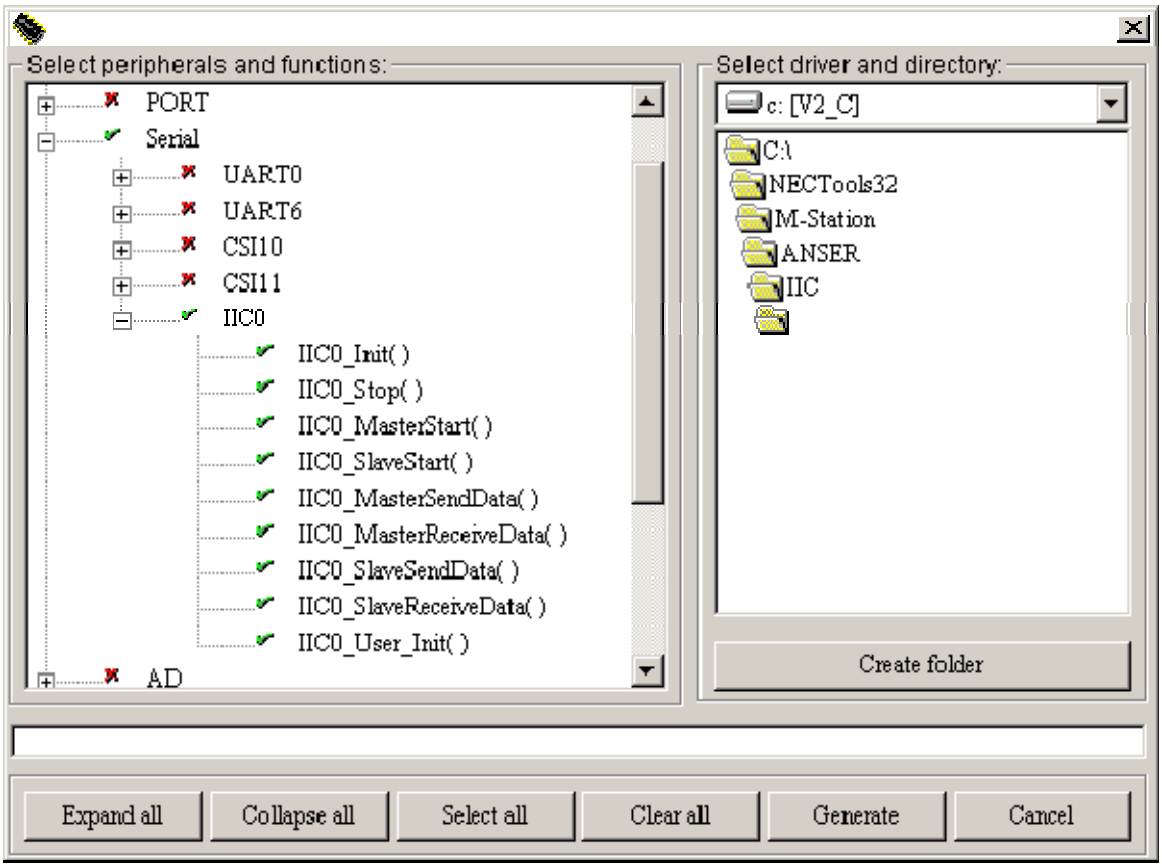
There are optional Applilet functions for the IIC0 peripheral, and you can select these functions either in the Applilet function view or during code generation.

5.4.7 Generating Code with Applilet, Selecting Optional IIC0 Routines

Once you have set up the IIC0 dialog box, click **Generate code**. The Applilet shows the peripherals and functions to be generated, and allows you to select a directory for storing the source code.

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

Figure 92. Selecting Optional IIC0 Routines



For a new project, the Applilet may only show IIC0_Init(). Select all of the additional functions:

- IIC0_Stop()
- IIC0_MasterStart(), IIC0_MasterSendData(), and IIC0_MasterReceiveData() for operation in master mode
- IIC0_SlaveStart(), IIC0_SlaveSendData(), and IIC0_SlaveReceiveData() for operation in slave mode
- IIC0_User_Init() user callback function

When you click **Generate,** 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 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.

5.4.8 Applilet-Generated Files and Functions for IIC0 Communication

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

5.4.8.1 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 Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

You must add external declarations for the variables used to signal IIC0 states (such as **UI_SlaveReceiveEnd**), for buffers to hold data to send and receive, and for the functions IIC0_Init( ) and IIC0_MasterStartAndSend( ).

5.4.8.2 Serial.c

The source file serial.c contains the following functions generated by the Applilet for IIC0:

void IIC0_Init(void)
This routine initializes the IIC0 peripheral as specified in the Applilet IIC0 detail dialog.

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 the number of loop times to wait for a start condition to occur. The routine creates a start condition and sends the slave address.

MD_STATUS IIC0_SlaveStart(UCHAR adr)
This routine prepares the IIC0 to respond to slave-mode operation at the specified slave address.

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

MD_STATUS IIC0_MasterSendData(UCHAR* txbuf , UINT txnum)
This routine is called after IIC0_MasterStart( ) starts 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. The

routine sets the buffer pointer and count, and sends the first byte in the buffer. The MD_INTIIC0( ) interrupt-service routine sends further bytes.

MD_STATUS IIC0_MasterReceiveData(UCHAR* rxbuf, UINT rxnum)
This routine is called after IIC0_MasterStart( ) starts 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 us this routine.

MD_STATUS IIC0_SlaveSendData(UCHAR* txbuf, UINT txnum)
After the slave sees an address that matches its local address, and the LSB of the slave address specifies a read by the external master, the slave calls this routine. 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. The MD_INTIIC0( ) interrupt-service routine sends further bytes.

MD_STATUS IIC0_SlaveReceiveData(UCHAR* rxbuf, UINT rxnum)
When a slave sees an address matching its local address, and the LSB of the slave address specifies a write from the external master, the slave calls this routine. The **rxbuf** parameter specifies the address of a buffer to store received characters, and **rxnum** specifies the number of characters to receive. The routine sets the buffer pointer and count, and prepares for data reception. Actual data reception is handled in the MD_INTIIC0( ) interrupt-service routine.

__interrupt void MD_INTIIC0(void)
This IIC0 interrupt-service routine is invoked by the INTIIC0 interrupt. The 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 slave mode.

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

5.4.8.3 Serial_user.c

The source file serial_user.c contains stub functions for user code, described below. These functions are empty on code generation, and you can add application-specific code.

You will need to 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 master sending is done
- MD_STATUS UI_MasterReceiveEnd — Reports that master receiving is done (not used)
- MD_STATUS UI_MasterFindSlave — Reports that slave has responded, send/receive can start
- MD_STATUS UI_SlaveError — Stores slave errors
- MD_STATUS UI_SlaveReceiveEnd — Reports that a slave-receive transfer has ended

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

void CALL_IIC0_MasterError(MD_STATUS flag)
Called by the IIC0_MasterHandler( ) routine when an error is detected, this routine requires additional code to store the **flag** parameter in **UI_MasterError**.

void CALL_IIC0_SlaveError(MD_STATUS flag)
Called by the IIC0_SlaveHandler( ) routine when an error is detected, this routine requires additional code to store the **flag** parameter in **UI_SlaveError**. Note that STOP errors are not signaled, since a stop may be the normal end of a transfer from the master.

void CALL_IIC0_MasterReceiveEnd(void)
IIC0_MasterHandler( ) calls this routine 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)
IIC0_MasterHandler( ) calls this routine after the last transmit byte has been sent. You must add code to set the **UI_MasterSendEnd flag** to MD_OK.

void CALL_IIC0_SlaveReceiveEnd(void)
IIC0_SlaveHandler( ) calls this routine when the receive count reaches zero. You must add code to set the **UI_SlaveReceiveEnd flag** to MD_OK.

void CALL_IIC0_SlaveSendEnd(void)
IIC0_SlaveHandler( ) calls this routine when the last transmit byte has been sent and it sees a stop condition. You must add code to call IIC0_SlaveStart(), to restart slave mode operation.

void CALL_IIC0_SlaveAddressMatch(void)
IIC0_SlaveHandler( ) calls this routine when the slave address received in slave mode matches the set slave address for the IIC0 peripheral. You must add code to call IIC0_SlaveSendData() to transmit the last switch value if the master is reading, or to call IIC0_SlaveReceiveData() to receive a count value if the master is sending.

void CALL_IIC0_MasterFindSlave(void)
IIC0_MasterHandler( ) calls this routine after sending the slave address and receiving an ACK, indicating that a slave has responded to the address. CALL_IIC0_MasterFindSlave() sets the **UI_MasterFindSlave** flag to MD_OK.

The following routine is not generated by the Applilet, but was written 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. The routine then waits for **UI_MasterFindSlave**, indicating that the slave has responded.

Then IIC0_MasterSendData(txbuf, txnum) sends the specified data. The routine then waits for the **UI_MasterSendDone** flag to be set, indicating that the transfer has completed. The routine monitors the **UI_MasterError** flag while waiting, to check for error conditions in the sending of the slave address or data transfer.

5.4.9 Slave: Other Applilet-Generated Files

For the demonstration program, the Applilet generates several other source files. The files and their functions are shown below.

Table 34. Additional Applilet-Generated 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
Int.h	Interrupt-related definitions
Int.c	Key-return interrupt-related functions
Int_user.h	User code for key-return interrupt
Timer.h	Timer-related definitions
Timer.c	Timer functions
Timer_user.c	User code for timer-interrupt handling
Option.inc	Option-byte, POC, and security definitions
Option.asm	Option-byte, POC, and security data

5.4.10 Slave: Demonstration Program Files Not Generated by Applilet

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

Table 35. Demonstration 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
LcdDrvApp.h	Header file for LCD-driver functions
LcdDrvApp.c	Code to initialize, write and read the LCD controller; These functions call Applilet-generated IIC0 routines

5.5 Demonstration Platform

This demonstration uses a development board from NEC Electronics. You may be able to duplicate the hardware using off-the-shelf components along with the NEC Electronics microcontroller of interest.

5.5.1 Resources for IIC Master

The master-program demonstration uses the following resources:

- DemoKit-KA1+ demonstration board, with uPD78F9222 8-bit microcontroller mounted
- DemoKit-KA1+ resources:
  - User input switch SW1 attached to P30/INTP0
  - LED displays D1 (P23), D2 (P130), D3 (P45), and D4 (P123)
  - IIC lines connected to I/O ports: SCL to P21, SDA to P22

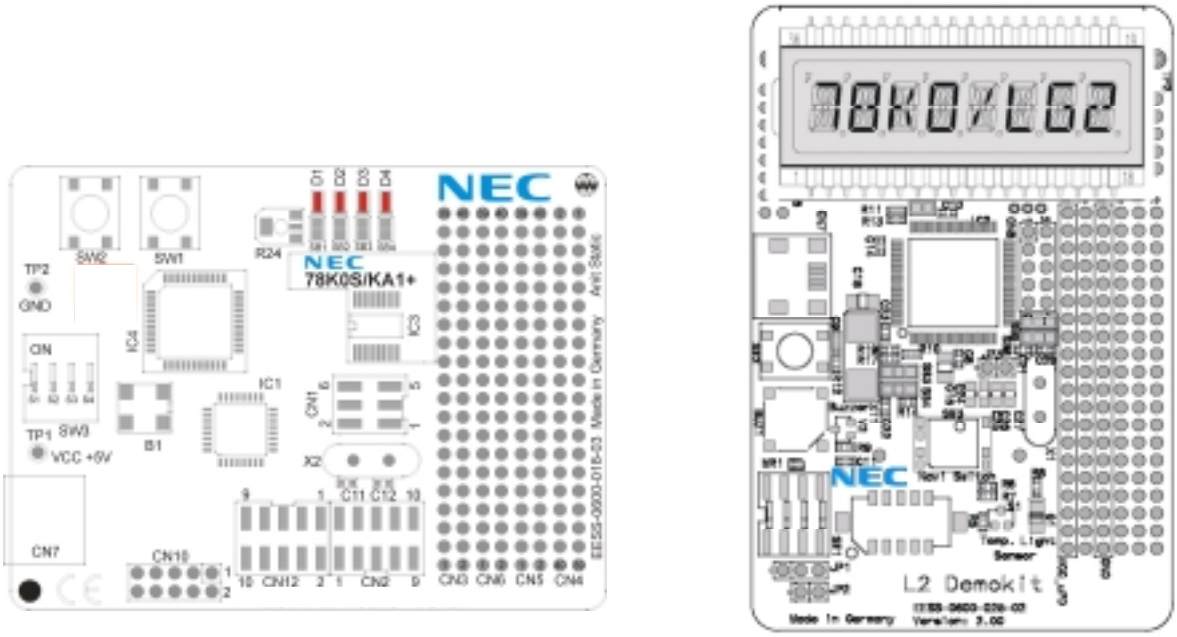
5.5.2 Resources For IIC Slave

The slave-program demonstration uses the following resources:

- DemoKit-LG2 demonstration board, with uPD78F0397 8-bit microcontroller mounted
- DemoKit-LG2 resources:
  - 5-position navigation switch SW3 connected to P70/KR0 through P74/KR4
  - 8-character LCD, using on-chip LCD controller
  - IIC interface using SCL0 and SDA0

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

Figure 93. Demonstration Hardware



5.5.3 Demonstration of Program

With the hardware configured and the uPD78F0397 microcontroller programmed with the demonstration code, the demonstration is as follows:

On initial power-up, the LED displays on the DemoKit-KA1+ are all off. The LCD on the DemoKit-LG2 shows “DKLG2 I2”. The switch inputs and corresponding program displays are shown below.

Table 36. IIC Demonstration Inputs and Display Values—Set 1

Input	DemoKit-KA1+				DemoKit-LG2	Comment
	D1	D2	D3	D4		
SW1 on KA1+	1	1	1	1	DKLG2 I2	Low nibble of count (0xFF) shown
SW1 on KA1+	0	0	0	0	RCV FF	Master writes count to slave
UP on LG2	0	0	0	0	UP	Slave stores 0x01 for last switch value
SW1 on KA1+	0	0	0	1	UP	Master reads 0x01 from slave, shows 1
SW1 on KA1+	0	0	0	0	UP	Idle state on master, blank display
SW1 on KA1+	1	1	1	0	UP	Low nibble of count (0xEE) shown
SW1 on KA1+	0	0	0	0	RCV EE	Master writes count to slave
DOWN on LG2	0	0	0	0	DOWN	Slave stores 0x02 for last switch value
SW1 on KA1+	0	0	1	0	DOWN	Master reads 0x02 from slave, shows 2
SW1 on KA1+	0	0	0	0	DOWN	Idle state on master, blank display

Table 37. IIC Demonstration Inputs and Display Values—Set 2: Order Changed

Input	DemoKit-KA1+				DemoKit-LG2	Comment
	D1	D2	D3	D4		
RIGHT on LG2	0	0	0	0	RIGHT	Slave stores 0x04 for last switch value
SW1 on KA1+	1	1	0	1	RIGHT	Low nibble of count (0xDD) shown
SW1 on KA1+	0	0	0	0	RCV DD	Master writes count to slave
SW1 on KA1+	0	0	1	1	RCV DD	Master reads 0x04 from slave, shows 3
SW1 on KA1+	0	0	0	0	RCV DD	Idle state on master, blank display

Table 38. IIC Demonstration Inputs and Display Values—Set 3: Original Order

Input	DemoKit-KA1+				DemoKit-LG2	Comment
	D1	D2	D3	D4		
SW1 on KA1+	1	1	0	0	RCV DD	Low nibble of count (0xCC) shown
SW1 on KA1+	0	0	0	0	RCV CC	Master writes count to slave
LEFT on LG2	0	0	0	0	LEFT	Slave stores 0x08 for last switch value
SW1 on KA1+	0	1	0	0	LEFT	Master reads 0x08 from slave, shows 4
SW1 on KA1+	0	0	0	0	LEFT	Idle state on master, blank display
SW1 on KA1+	1	0	1	1	LEFT	Low nibble of count (0xBB) shown
SW1 on KA1+	0	0	0	0	RCV BB	Master writes count to slave
SELECT on LG2	0	0	0	0	SELECT	Slave stores 0x10 for last switch value
SW1 on KA1+	0	1	0	1	SELECT	Master reads 0x10 from slave, shows 5
SW1 on KA1+	0	0	0	0	SELECT	Idle state on master, blank display

If you run through the cycle on the KA1+ without changing the switch on the LG2, the same value is read for the last switch input, as shown below.

Table 39. IIC Demonstration Inputs and Display Values—Set 4: LG2 Switch Unchanged

Input	DemoKit-KA1+				DemoKit-LG2	Comment
	D1	D2	D3	D4		
SW1 on KA1+	1	0	1	0	SELECT	Low nibble of count (0xAA) shown
SW1 on KA1+	0	0	0	0	RCV AA	Master writes count to slave
SW1 on KA1+	0	1	0	1	RCV AA	Master reads 0x10 from slave, shows 5
SW1 on KA1+	0	0	0	0	RCV AA	Idle state on master, blank display

5.6 Hardware Block Diagram

Figure 94. Hardware Block Diagram

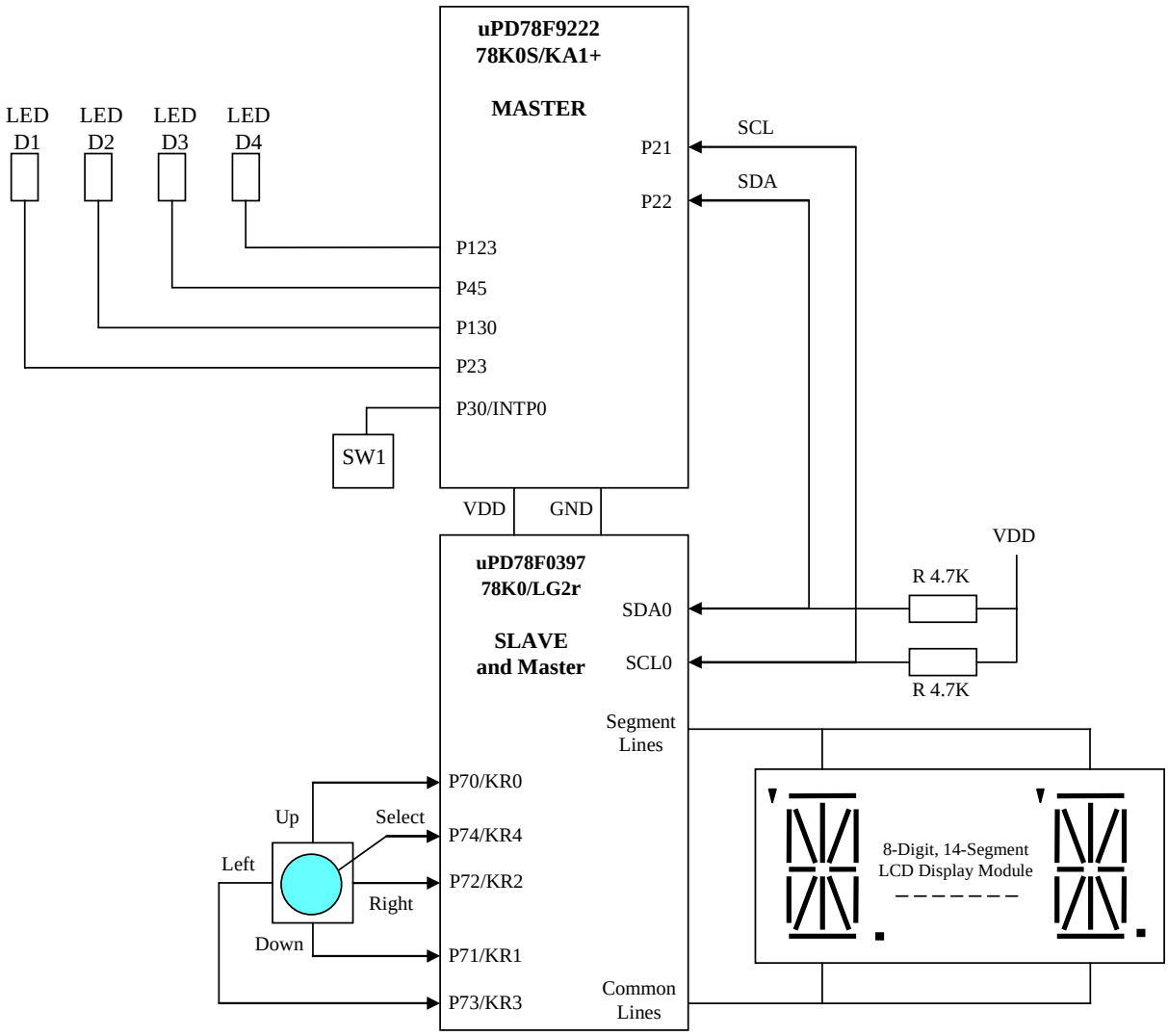


Table 40. SCL and SDA connections Between DemoKit-KA1+ and DemoKit-LG2

uPD78F9222 Pin	uPD78F9222 Port	DemoKit-KA1+ Connector	IIC Signal	DemoKit-LG2 Connector	uPD78F0397 Port	uPD78F0397 Pin
18	P21	CN3.36	SCL	CN3.16	SCL0	5
17	P22	CN3.34	SDA	CN3.18	SDA0	4

External 4700-ohm resistors, mounted as standard pull-ups on the DemoKit-LG2, pull up the SCL0 and SDA0 lines.

5.7 Software Modules

The following files make up the software modules for the demonstration programs. The tables below shows which files were generated by the Applilet, and which need modification to create the demonstration programs.

The listings for these files are located in the Appendix.

5.7.1 Software Modules for IIC Master Program

Table 41. Master 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	INT_Init() function	Yes	No
Int_user.h	User code for INTP0 interrupt	Yes	Yes[Note 1]
Port.h	Definitions for default I/O port states	Yes	No
Port.c	PORT_Init() function	Yes	No
Iic_dkka1.h	IIC definitions for DemoKit-KA1+	No	--
Iic_dkka1.c	IIC functions for DemoKit-KA1+	No	--

Note 1: Modify Int.h to add the declarations of the global variable **IntP0Flag**, used to signal the occurrence of SW1 closure. Modify Int_user.c to define this variable and to have MD_INTP0() set the variable.

5.7.2 Software Modules for IIC Slave Program

Table 42. Slave 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	No
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	No	--
Lcd.h	LCD high-level function definition	No	--
Lcd.c	LCD high-level functions	No	--
LcdDrvApp.h	LCD-controller driver definitions	No	--
LcdDrvApp.c	LCD-controller driver functions	No	--

Note 1: Modify int.h to add the declaration of **sw3_in**, the variable used to store the navigation-switch debounced input. Modify int_user.c to add the variable **sw3_in** and to add the code for handling the key-return interrupt.

Note 2: Modify serial.h 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(). Modify serial.c slightly for IIC0 to remove settings for P6 not supported for 78K0/LG2, to replace "asm" versions of DI and EI instructions with NEC Electronics C Compiler equivalents, and to move the clearing of PM6 bits to after IICE0 set. Modify serial_user.c to have IIC0 callback functions set the IIC0-related global variables and to add the IIC0_MasterStartAndSend() function.

6. Appendix A - Development Tools

Developing this application requires the following software and hardware tools.

6.1 Software Tools

Table 43. Software Tools for 78K0 Devices

Tool	Version	Comments
Applilet for 78K0KE2	V1.51	Source-code generation tool for 78K0/KE2 devices
PM Plus	V5.21	Project manager for program compilation and linking
CC78K0	V3.70	C Compiler for NEC Electronics 78K0 devices
RA78K0	V3.80	Assembler for NEC Electronics 78K0 devices
DF0397.78K	V1.01	Device file for uPD78F0397 device
DF053764.78K	V2.00	Device file for uPD78F0537_64 device

Table 44. Software Tools for 78K0S Devices

Tool	Version	Comments
Applilet	V1.45	Source code generation tool for 78K0S devices
NEC Electronics 78K0SKA1H MCU	V1.46	Applilet source code library and configuration for uPD78F9222
PM Plus	V5.21	Project manager for program compilation and linking
CC78K0S	V1.50	C Compiler for NEC Electronics 78K0S devices
RA78K0S	V1.40	Assembler for NEC Electronics 78K0S devices
DF9222.78K	V1.10	Device file for uPD78F9222 device

6.2 Hardware Tools

Table 45. Hardware Tools

Tool	Version	Comments
DemoKit-LG2	V2.00	Demonstration Kit for 78F0397 (78K0/LG2)
M-Station 2	V2.1E	Base platform for NEC Electronics Micro-Board demonstration
M-78F0537	V1.0	NEC Electronics Micro-Board for uPD78F0537; CPU chip is uPD78F0537DGB
DemoKit-KA1+	V3.00	Demonstration Kit for 78F9222 (78K0S/KA1+)

7. Appendix B – Software Listings

This Appendix contains the software listings for the UART, LIN, CSI, and IIC demonstration programs described in this application note. Because several of the programs run on the same hardware platform, many of the files are identical in multiple-demonstration programs.

The sections for each demonstration program contain the listing files unique to that program. The common files for multiple demonstration programs are provided in separate sections.

The M-78F0537 micro-board on the M-Station II platform is used for the following programs:

- UART demonstration program
- LIN master demonstration program
- LIN slave demonstration program
- CSI master demonstration program

The common files for this platform are collected in one common file section.

The DemoKit-LG2 demonstration kit platform is used for the following programs:

- CSI slave demonstration program
- IIC slave demonstration program

The common files for this platform are collected in the other common file section.

The DemoKit-KA1+ demonstration kit platform was used for the following program:

- IIC master demonstration program

All files for this program are in the IIC master demonstration program listing section.

Note that since the Applilet program generation tool was used for all programs, there are many instances of the same filename, such as macrodriver.h, serial.h, serial.c, or serial_user.c. At first glance, many of these files may seem to be identical, because the Applilet may place a large amount of similar code in each version of the file. However, the files contain differences in initialization values for registers, or differences in generated code, depending on the options selected in the Applilet. The files listed in the sections for each demonstration program may be different from the same-named files in other sections.

7.1 Listings for UART Demonstration Program

This demonstration program runs on the M-78F0537 plus M-Station II platform. The files listed in this section are unique for this demonstration program. The non-unique files are listed in the M-78F0537 common-files section below.

7.1.1 UART Main.c

```
/* main.c for NEC Electronics Serial Application Note, UART Section */
/*
*****************************************************************************
**
** 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
*****************************************************************************
*/
/*
*****************************************************************************
** Include files
*****************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "port.h"
#include "timer.h"
#include "serial.h"

#include "string.h"        /* added for strcpy and strlen functions */
#include "led_0537.h"      /* M-Station LED code */
#include "sw_0537.h"       /* M-Station switch code */

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

```
**     main function
**
** Parameters:
**     None
**
** Returns:
**     None
**
**------------------------------------------------------------------------
*/

/* global receive and transmit buffer */
UCHAR m_rxbuf[256];
UCHAR m_txbuf[256];

void main( void )
{
UCHAR rxchar;
UCHAR txcount;
UCHAR swval;

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;
       /* TODO. add user code */
       /* not necessary to add led_init, ports inited in Port_Init() */
       /* show two dashes in LEDs at start of program */
       led_out_left(LED_PAT_DASH);
       led_out_right(LED_PAT_DASH);

       /* do initialize switches, to set local variables */
       sw_init();
       sw_set_debounce(10);       /* set debounce counter to 10 for 10 msec stable time */

       TM00_Start();              /* start 10ms timer for switch debouncing */

       /* queue a receive operation for one character */
       UART6_ReceiveData(m_rxbuf, 1);

       while(1){
              /* check for left switch down */
              swval = sw_get();
              if (swval == SW_LD_RU) {
                     if (UART6_TX_CNT == 0) {
                            strcpy( (char*)m_txbuf, "SW2 Pressed\r\n");
                            txcount = (UCHAR) strlen( (char*)m_txbuf);
                            UART6_SendData(m_txbuf, txcount);
                     }
                     /* wait for switch different */
                     while ( (swval = sw_get()) == SW_LD_RU )
                            ;
              }
              /* check for right switch down */
              swval = sw_get();
              if (swval == SW_LU_RD) {
                     if (UART6_TX_CNT == 0) {
                            /* add space to start of string, to make it easier to see the
difference */
                            strcpy( (char*)m_txbuf, " SW3 Pressed\r\n");
                            txcount = (UCHAR) strlen( (char*)m_txbuf);
                            UART6_SendData(m_txbuf, txcount);
                     }
                     /* wait for switch different */
                     while ( (swval = sw_get()) == SW_LU_RD )
```

```
                        ;
            }
            /* check for character received */
            if (UART6_RX_CNT == 0) {
                  /* get the character, display in LEDs */
                  rxchar = *m_rxbuf;
                  led_dig(rxchar);
                  /* start the next receive operation */
                  UART6_ReceiveData(m_rxbuf, 1);
            }
            /* check for reception error */
            if (Receive_Error == TRUE) {
                  led_out_left(LED_PAT_E);   /* signal error */
                  led_dig_right(Last_Error & 0x0F); /* show error code (low nibble) */
                  Receive_Error = FALSE;           /* rearm error detect */
            }
      }
}
```

7.1.2 UART 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#ifndef _MDSERIAL_
#define _MDSERIAL_
/*
************************************************************************
** Global variables
************************************************************************
*/

/* added global variables */
extern UCHAR  UART6_RX_CNT; /* make available for main program (READ ONLY!) */
extern UCHAR  UART6_TX_CNT; /* make available for main program (READ ONLY!) */
```

```
extern UCHAR  Receive_Error;      /* flag for reception error */
extern UCHAR  Last_Error;         /* storage for last reception error code */

/* Definition to fix Applilet-generated code for UART6 bugs */
#define APPLILET_FIX_UART6 1

#define      UART_BAUDRATE_M6    0x3
#define      UART_BAUDRATE_K6    0x82

void UART6_Init( void );
MD_STATUS UART6_SendData( UCHAR* txbuf, UCHAR txnum );
MD_STATUS UART6_ReceiveData( UCHAR* rxbuf, UCHAR rxnum );
__interrupt void MD_INTSR6( void );
__interrupt void MD_INTST6( void );
__interrupt void MD_INTSRE6( void );
void CALL_UART6_Error( UCHAR err_type );

enum TransferMode { Send, Receive };

#endif
```

7.1.3 UART 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/

#pragma      interrupt    INTSR6       MD_INTSR6
#pragma      interrupt    INTST6       MD_INTST6
#pragma      interrupt    INTSRE6             MD_INTSRE6

/*
************************************************************************
** Include files
```

```
************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"
/* uart6 transmission */
UCHAR* UART6_TX_ADDRESS;                /* uart transfer buffer address */
UCHAR UART6_TX_CNT;                     /* uart transfer number */

/* uart6 reception */
UCHAR* UART6_RX_ADDRESS;                /* uart receive buffer address */
UCHAR UART6_RX_CNT;                     /* uart receive data number */

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     This function initializes UART6 module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void UART6_Init( void )
{
       ASIM6 = 0;

       SetIORBit(P1, 0x08);                    /* port setting in transmit/receive mode */
       ClrIORBit(PM1, 0x08);

       SetIORBit(ASIM6, 0x04);                 /* data length 8 bits */
       ClrIORBit(ASIM6, 0x02);                 /* stop length 1 bits */
       ClrIORBit(ASIM6, 0x01);
       ClrIORBit(ASIM6, 0x18);                 /* none parity mode */
       SetIORBit(ASICL6, 0x02);          /* LSB first transfer */
       ClrIORBit(ASICL6, 0x01);          /* normal output of TxD6 */
       CKSR6 = UART_BAUDRATE_M6;         /* baudrate selection */
       BRGC6 = UART_BAUDRATE_K6;
       ClrIORBit(IF0H, 0x02);
       ClrIORBit(MK0H, 0x02);                  /* transfer end interrupt enable */
       ClrIORBit(IF0H, 0x01);
       ClrIORBit(MK0H, 0x01);                  /* receive end interrupt enable */
       ClrIORBit(PR0H, 0x01);                  /* interrupt high */
       SetIORBit(PR0H, 0x02);                  /* interrupt low */
#if (APPLILET_FIX_UART6 == 1)
       /* this is corrected code */
       /* SREIF6 is IF0L.7, SREMK6 is MK0L.7, SREPR6 is PR0L.7 */
       ClrIORBit(IF0L, 0x80);
       ClrIORBit(MK0L, 0x80);                  /* error interrupt enable */
       ClrIORBit(PR0L, 0x80);                  /* interrupt high */
#else
       /* this is Applilet generated code, enable and flag wrong */
       /* IF0H.7 is TMIF010, not SREIF6 */
       /* MK0H.7 is TMMK010, not SREMK6 */
       /* PR0H.7 is TMPR010, not SREPR6 */
       ClrIORBit(IF0H, 0x80);
       ClrIORBit(MK0H, 0x80);                  /* error interrupt enable */
       ClrIORBit(PR0H, 0x80);                  /* interrupt high */
#endif

       SetIORBit(ASIM6, 0x80);
```

```
      SetIORBit(ASIM6, 0x60);                /* transmit/receive mode */
}

/*
**-------------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for UART6 data receiving.
**
** Parameters:
**     rxbuf: receive buffer pointer
**     rxnum: buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------------
*/
MD_STATUS UART6_ReceiveData( UCHAR* rxbuf, UCHAR rxnum )
{
      if( rxnum < 1 ){
      return MD_ERROR;
 }

      UART6_RX_CNT = rxnum;
      UART6_RX_ADDRESS = rxbuf;

      return MD_OK;
}

/*
**-------------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for UART6 data transfering.
**
** Parameters:
**     txbuf: transfer buffer pointer
**     txnum: buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------------
*/
MD_STATUS UART6_SendData( UCHAR* txbuf, UCHAR txnum )
{
      if( txnum < 1 ){
            return MD_ERROR;    /* 1 frame data at least */
      }

      UART6_TX_ADDRESS = txbuf;
      UART6_TX_CNT = txnum;
      if( txnum == 1) {          /* only 1 frame data to transfer doesn't need contiously
transfer */
            if((ASIF6 & 0x02) == 0){
                  TXB6 = *UART6_TX_ADDRESS++;
            }
            else{
                  return MD_DATAEXISTS;
            }
```

```
	}
	else {
		if((ASIF6 & 0x02) == 0){
			TXB6 = *UART6_TX_ADDRESS++;
			while((ASIF6 & 0x02) == 1);		/* wait */
			UART6_TX_CNT--;
			TXB6 = *UART6_TX_ADDRESS++;
		}
		else{
			return MD_DATAEXISTS;
		}
	}
	return MD_OK;
}

/*
**-------------------------------------------------------------------------------
**
** Abstract:
**    This function is the UART6 receive end interrupt handler.
**
** Parameters:
**    None
**
** Returns:
**    None
**
**-------------------------------------------------------------------------------
*/
__interrupt void MD_INTSR6( void )
{
	UCHAR err_type;
	UCHAR tmp;

	err_type = ASIS6;
	tmp = RXB6;

	if( err_type != 0 ){
		if( err_type & 0x1 ){
			err_type = MD_OVERRUNERROR;		/* overrun error */
		}
		else if( err_type & 0x2 ){
			err_type = MD_FRAMEERROR;	/* framing error */
		}
		else{
			err_type = MD_PARITYERROR;	/* parity error */
		}

		CALL_UART6_Error( err_type );
		return;
	}

	if( UART6_RX_CNT == 0 ){
		/* receive data end */
		return;
	}

	/* the interrut generated by receive end */
	*UART6_RX_ADDRESS ++= tmp;
	UART6_RX_CNT--;
}

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

```
**
** Abstract:
**     This function is the UART6 transfer end interrupt handler.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTST6( void )
{
       if( UART6_TX_CNT == 0 ){
              /* send data complete */
              return;
       }

       if(((ASIF6 & 0x02) == 0) && ((ASIM6 & 0x80) == 0x80)){
              UART6_TX_CNT--;
              if( UART6_TX_CNT != 0 ){
                     TXB6 = *UART6_TX_ADDRESS++;
              }
       }
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is the UART6 receive error interrupt handler.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTSRE6( void )
{
       UCHAR err_type;

       *UART6_RX_ADDRESS = RXB6;               /* one-time read */
       err_type = ASIS6;

       if( err_type & 0x1 ){
              err_type = MD_OVERRUNERROR;             /* overrun error */
       }
       else if( err_type & 0x2 ){
              err_type = MD_FRAMEERROR;        /* framing error */
       }
       else{
              err_type = MD_PARITYERROR;       /* parity error */
       }

       CALL_UART6_Error( err_type );
}
```

7.1.4 UART 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     This function is a call back function to deal with data process
**     after an error occurred of UART6 interface.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
UCHAR Receive_Error = FALSE;     /* flag for serial error occurred */
UCHAR Last_Error = 0;                  /* storage for last error code */

void CALL_UART6_Error( UCHAR err_type )
{
       /* uart6 error process */
       Receive_Error = TRUE;
       Last_Error = err_type;
}
```

7.1.5 UART 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/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   20000000
#define       SUBCLOCK      32768
#define       MAINCLOCK     20000000
#define       FRCLOCK       8000000
#define       FRCLOCKLOW    240000


#endif
```

7.1.6 UART 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#ifndef      _MDSYSTEM_
#define      _MDSYSTEM_
/*
************************************************************************
** MacroDefine
************************************************************************
*/
#define      CG_X1STAB_SEL 0x5
#define      CG_X1STAB_STA 0x1f
#define      CG_CPU_CLOCKSEL     0x0
#define      CG_Mainosc    0x14

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
```

7.1.7 UART 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
*****************************************************************************
*/
/*
*****************************************************************************
** Include files
*****************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "port.h"
#include "timer.h"
#include "serial.h"
/*
*****************************************************************************
** MacroDefine
*****************************************************************************
*/

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     Init every Macro
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void SystemInit( void )
{
      /* Clock generator initiate */
      Clock_Init();
      /* Port initiate */
      PORT_Init();
      /* UART6 initiate */
      UART6_Init();
      /* TM00 initiate */
      TM00_Init();
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     Init hardware setting
**
** Parameters:
**     None
```

```
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void hdwinit( void )
{
       DI( );
       SystemInit( );
       EI( );
}
```

7.1.8 UART 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 :   NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC Electronics/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 )
{
       USHORT i;
       UCHAR temp_stabset, temp_stabwait;
       SetIORBit(PM12, 0x06);                  /* P121/122 input mode */
       ClrIORBit(OSCCTL, 0x80);          /* X1/X2 input mode */
       SetIORBit(OSCCTL, 0x40);
       ClrIORBit(MOC, 0x80);
       /* OSC stabilization time */
       temp_stabset = CG_X1STAB_STA;
       do{
              temp_stabwait = OSTC;
              temp_stabwait &= temp_stabset;
       }while(temp_stabwait != temp_stabset);
       OSTS = CG_X1STAB_SEL;
       for(i = 0; i <= 20; i++){         /* wait 5us */
              NOP();
       }
       SetIORBit(OSCCTL, 0x01);          /* 10MHz<fx<=20MHz */
       SetIORBit(MCM, 0x05);                   /* X1 operate for CPU */
       SetIORBit(PM12, 0x18);                  /* P123/124 input mode */
       ClrIORBit(OSCCTL, 0x20);          /* XT1 input mode */
       SetIORBit(OSCCTL, 0x10);
       ClrIORBit(RCM, 0x01);
       PCC = CG_CPU_CLOCKSEL;
}
```

7.1.9 UART 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
```

```
**
** Compiler: NEC Electronics/CC78K0
**
************************************************************************************
*/

#ifndef _MDTIMER_
#define _MDTIMER_
/*
************************************************************************************
** MacroDefine
************************************************************************************
*/
#define      REGVALUE_MAX  0xff
#define      TM_TM00_CLOCK 0x0
#define      TM_TM00_INTERVALVALUE      0x4e1f
#define      TM_TM00_SQUAREWIDTH 0x4e1f
#define      TM_TM00_PPGCYCLE    0x4e1f
#define      TM_TM00_PPGWIDTH    0x00
#define      TM_TM00_ONESHOTCYCLE       0x4e1f
#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 0x2
#define      TM_TM50_INTERVALVALUE      0x00
#define      TM_TM50_SQUAREWIDTH 0x00
#define      TM_TM50_PWMACTIVEVALUE     0x00
#define      TM_TM51_CLOCK 0x2
#define      TM_TM51_INTERVALVALUE      0x00
#define      TM_TM51_SQUAREWIDTH 0x00
#define      TM_TM51_PWMACTIVEVALUE     0x00
#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 TM00_Init(void);

/*timer start*/
void TM00_Start(void);

/*timer stop*/
void TM00_Stop(void);
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
__interrupt void MD_INTTM000(void);

#endif       /* _MDTIMER_*/
```

7.1.10 UART 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
************************************************************************
*/

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


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

/*TM01 pulse width measure*/

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     This function initializes TM00_module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
```

```
*/
void TM00_Init( )
{
       TMC00=0x00;
                          /* internal count clock */
       PRM00 |= TM_TM00_CLOCK;
       SetIORBit(PR0H, 0x40);                  /* low priority level */
       ClrIORBit(IF0H, 0x40);
       /* TM00 interval */
       ClrIORBit(CRC00,0x01);
       CR000 = TM_TM00_INTERVALVALUE;
}
/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function starts the TM00 counter.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM00_Start()
{
       TMC00 = 0x0c;                     /* interval timer start */
       ClrIORBit(MK0H, 0x40);                  /* INTTM000 enable */
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function stops the TM00 counter and clear the count register.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM00_Stop()
{
       TMC00=0x0;
       SetIORBit(MK0H, 0x40);                  /* INTTM000 stop */
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function changes TM00 condition.
**
** Parameters:
**     USHORT* :     array_reg
**     USHORT :      array_num
** Returns:
**     MD_OK
```

```
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num)
{
 switch (array_num){
 case 2:
 CR010=*(array_reg + 1);
 case 1:
 CR000=*(array_reg + 0);
 break;
 default:
 return MD_ERROR;
 }
        return MD_OK;
}
```

7.1.11 UART 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
*****************************************************************************
*/

#pragma sfr
#pragma interrupt INTTM000 MD_INTTM000
/*
*****************************************************************************
** Include files
*****************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"

#include "sw_0537.h"       /* for sw_isr() definition */

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

```
** MacroDefine
*****************************************************************************
*/

/* Timer00, Timer01 pulse width measure */
/*
**-------------------------------------------------------------------------
**
** Abstract:
**     TM00 INTTM000 interrupt-service routine
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTTM000( )
{
       /*TODO*/
       sw_isr();     /* call function to check switch input */
}
```

7.2 Listings for LIN Demonstration Programs

Note: The file lin.h is common to both the LIN master and the LIN slave demonstration programs. This file is listed here only.

7.2.1 LIN.h

```
/*
*******************************************************************************
**
** This file was created for the NEC Electronics Serial Application Note, LIN Section
** 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 : LIN.h
** Abstract : Header for LIN definitions, both Master and Slave
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
*******************************************************************************
*/
#ifndef _LIN_H
#define _LIN_H

/* LIN Error codes, based on MD_ERROR (0x80) */
#define MD_LIN_ERROR             (MD_ERROR)                      /* 0x80 generic error -
base */
#define MD_LIN_ERROR_STATE ((MD_LIN_ERROR) + 1)      /* 0x81 wrong state */
#define MD_LIN_ERROR_PARAM ((MD_LIN_ERROR) + 2)      /* 0x82 incorrect parameter */
#define MD_LIN_ERROR_NOREC ((MD_LIN_ERROR) + 3)      /* 0x83 no Rx interrupt on send */
#define MD_LIN_ERROR_DTNEQ ((MD_LIN_ERROR) + 4)      /* 0x84 Rx data != Tx data */
#define MD_LIN_ERROR_SYNCB ((MD_LIN_ERROR) + 5)      /* 0x85 SYNC not correct */
#define MD_LIN_ERROR_RCVOV ((MD_LIN_ERROR) + 6)      /* 0x86 receive error - overflow */
#define MD_LIN_ERROR_RCVFE ((MD_LIN_ERROR) + 7)      /* 0x87 receive error - framing */
#define MD_LIN_ERROR_RCVPE ((MD_LIN_ERROR) + 8)      /* 0x88 receive error - parity */
#define MD_LIN_ERROR_RCVMU ((MD_LIN_ERROR) + 9)      /* 0x89 receive error - multiple bits
*/
#define MD_LIN_ERROR_IDPAR ((MD_LIN_ERROR) + 10)     /* 0x8A ID double parity incorrect */
#define MD_LIN_ERROR_CKSUM ((MD_LIN_ERROR) + 11)     /* 0x8B Data Checksum error */
#define MD_LIN_ERROR_NOSBF ((MD_LIN_ERROR) + 12)     /* 0x8C No SBF receipt on send (Master
only) */
/* note - timeout not implemented at this time, but error code reserved */
#define MD_LIN_ERROR_TMOUT ((MD_LIN_ERROR) + 13)     /* 0x8D Timeout waiting for data */

/* LIN ID field definitions */
/* ID Bits 6 and 7 are double parity bits, calculated by routine */
#define LIN_ID_PARITYMASK  0xC0
```

```
/* ID Bits 4 and 5 define length of data (optional in LIN spec) */
#define LIN_ID_2BYTE       0x00   /* xx00xxxx is 2 bytes */
#define LIN_ID_2BYTE_B     0x10   /* xx01xxxx is also 2 bytes */
#define LIN_ID_4BYTE       0x20   /* xx10xxxx is 4 bytes */
#define LIN_ID_8BYTE       0x30   /* xx11xxxx is 8 bytes */
#define LIN_ID_LENMASK     0x30   /* mask to length bits */

/* ID Bits 3 - 0 are available to define command */
/* Locally define bit 3 of command as direction: 0=out, 1=in */
/* This is not LIN spec; demo application will use this convention */
#define LIN_ID_DIR_OUT           0x00   /* out from Master, in to Slave */
#define LIN_ID_DIR_IN            0x08   /* in to Master, out from Slave */
#define LIN_ID_DIRMASK           0x08   /* mask for direction bit */

/* define local LIN commands using bits 0-2 */
#define LIN_ID_GetSlaveNum4            (0x00 | LIN_ID_4BYTE | LIN_ID_DIR_IN)
#define LIN_ID_SetCount2         (0x01 | LIN_ID_2BYTE | LIN_ID_DIR_OUT)
#define LIN_ID_GetCount2         (0x01 | LIN_ID_2BYTE | LIN_ID_DIR_IN)

/* define states for LIN transmission */
#define LIN_STATE_IDLE     0x00   /* not in transmission/receive */
#define LIN_STATE_SBF      0x01   /* send/receive Synch Break Field */
#define LIN_STATE_SYNC     0x02   /* send/receive SYNC byte */
#define LIN_STATE_ID       0x03   /* send/receive ID field */
#define       LIN_STATE_DATAW     0x04   /* writing data */
#define LIN_STATE_SUMW     0x05   /* writing checksum */
#define       LIN_STATE_DATAR     0x06   /* reading data */
#define LIN_STATE_SUMR     0x07   /* reading checksum */

/* define structure to control LIN transfers */
typedef struct {
       UCHAR state;        /* one of LIN_STATE_xxx above */
       UCHAR id;                /* ID byte (will define dir and len */
       UCHAR dir;               /* direction, either LIN_ID_DIR_OUT or LIN_ID_DIR_IN */
       UCHAR len;               /* length of transfer: 2, 4, or 8 bytes */
       UCHAR count;        /* count of data transferred */
       UCHAR data[8];           /* buffer for data to send/received */
       UCHAR checksum;          /* checksum to send, or expected to receive */
       UCHAR rcvsum;       /* actual received checksum byte */
} LIN_XFER_STRUCT;

#endif /* _LIN_H */
```

7.2.2 Listings for LIN Master Program

This demonstration program was run on the M-78F0537 plus M-Station II platform. The files listed in this section are the unique files for this particular demonstration program. The other files are listed in the M-78F0537 common files section below.

7.2.2.1 LIN Master Main.c

```
/* main.c for NEC Electronics Serial Application Note, LIN Master */
/*
************************************************************************
**
** 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "port.h"
#include "timer.h"
#include "serial.h"

#include "LIN.h"          /* include for LIN Application Note, generic LIN */
#include "LIN_M.h"        /* include for LIN Application Note, LIN Master */
#include "led_0537.h"     /* M-Station LED code */
```

```
#include "sw_0537.h"       /* M-Station switch code */
/*
*****************************************************************************
** MacroDefine
*****************************************************************************
*/
/* global variables to be accessed in send/get functions */
UCHAR m_count = 0;         /* global master count */

/* definitions used in main() menu processing */
#define      TEST_MAX     4

/* test function definitions */
void Change_Count(void);
void LIN_Send_Count_2Byte(void);
void LIN_Get_Count_2Byte(void);
void LIN_Get_Num_4Byte(void);

/* define PF_VF_RV as pointer to a function, void parameter, returning void */
typedef void (*PF_VF_RV)(void);

const PF_VF_RV menu[TEST_MAX] = {
       Change_Count,
       LIN_Send_Count_2Byte,
       LIN_Get_Count_2Byte,
       LIN_Get_Num_4Byte };

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     main function
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void main( void )
{
unsigned char test;        /* selection of test to run */
unsigned char sw_val;      /* value of switches */

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;

       led_init();         /* initialize LEDs */
       sw_init();          /* initialize switch variables */
       sw_set_debounce(10);       /* set debounce counter to 10 for 10 msec stable time */


       TM50_Start();/* start timer for switch debouncing */

       LIN_M_Init();/* initialize LIN */

       test = 1;                  /* set initial test number */
       while(1){
              led_out_left(LED_PAT_EQUAL);     /* display an equal sign */
              led_dig_right(test);                    /* display number of test to execute
*/
```

```
            while (SW_LU_RU == sw_get())
                  ;                                                  /* wait for switch
pressed */
            sw_val = sw_get();                        /* get value of switch */

            if (sw_val == SW_LD_RU) {
                  /* SW2 (left) is down, right (SW3) is up */
                  test = test + 1;                    /* increment the test number */
                  if (test > TEST_MAX)
                        test = 1;                           /* wrap around if hit max */
                  led_dig_right(test);                /* update display right away */
            }

            if (sw_val == SW_LU_RD) {
                  /* SW2 up, SW3 down */
                  while (SW_LU_RU != sw_get())
                        ;                                            /* wait for switch up
again */
                  led_dig_left(test);                 /* show test in left digit */
                  led_out_right(LED_PAT_BLANK);       /* show blank in right digit */

                  (menu[test-1])();                          /* execute function */

            }

            while (SW_LU_RU != sw_get())
                  ;                                                  /* wait for switch up */
      } /* end of while (1) loop */
} /* end of main() */

/* function to display and change local count */
void Change_Count(void)
{
unsigned char sw_val;

      /* display local count */
      led_dig(m_count);

      while (1) {
            /* get non-zero switch */
            while (SW_LU_RU == sw_get())
                  ;
            sw_val = sw_get();
            switch (sw_val) {
            case SW_LU_RD:
                  /* SW3 pressed, increment and display count */
                  m_count = m_count + 15;
                  led_dig(m_count);
                  /* wait for switches to be different */
                  while (SW_LU_RD == sw_get())
                        ;
                  break;
            case SW_LD_RU:
                  /* SW2 pressed, return to main */
                  /* wait for switches to be up */
                  while (SW_LU_RU != sw_get())
                        ;
                  return;
                  break;
            } /* end of switch */
      } /* end of while (1) */
}
```

```
/* LIN Test Functions */

/* send our count to the slave */
void LIN_Send_Count_2Byte(void)
{
unsigned char sw_val;
unsigned char txbuf[2];
MD_STATUS status;

       led_dig(m_count);   /* show the count to send */

       while (1) {
              /* get non-zero switch */
              while (SW_LU_RU == sw_get())
                     ;
              sw_val = sw_get();
              switch (sw_val) {
              case SW_LU_RD:
                     /* SW3 pressed, send the count to slave */
                     /* wait for switches to be up */
                     while (SW_LU_RU != sw_get())
                            ;
                     txbuf[0] = m_count;
                     txbuf[1] = 0;
                     status = LIN_M_SendData(LIN_ID_SetCount2, txbuf);
                     if (status != MD_OK) {
                            /* if error, show E plus low four bits of error code */
                            led_out_left(LED_PAT_E);
                            led_dig_right((UCHAR)status & 0x0F);
                     }
                     break;
              case SW_LD_RU:
                     /* SW2 pressed, return from test */
                     /* wait for switches to be up */
                     while (SW_LU_RU != sw_get())
                            ;
                     return;
              } /* end of switch */
       } /* end of while (1) */
}

/* get the slave's count, set our count and display */
void LIN_Get_Count_2Byte(void)
{
unsigned char sw_val;
unsigned char rxbuf[2];
MD_STATUS status;

       while (1) {
              /* get non-zero switch */
              while (SW_LU_RU == sw_get())
                     ;
              sw_val = sw_get();
              switch (sw_val) {
              case SW_LU_RD:
                     /* SW3 pressed, get the count from the slave */
                     /* wait for switches to be up */
                     while (SW_LU_RU != sw_get())
                            ;
                     status = LIN_M_ReceiveData(LIN_ID_GetCount2, rxbuf);
                     if (status == MD_OK) {
                            /* no error, update count and display it */
                            m_count = rxbuf[0];
```

```
                        led_dig(m_count);
                    } else {
                        /* if error, show E plus low four bits of error code */
                        led_out_left(LED_PAT_E);
                        led_dig_right((UCHAR)status & 0x0F);
                    }
                    break;
                case SW_LD_RU:
                    /* SW2 pressed, return from test */
                    /* wait for switches to be up */
                    while (SW_LU_RU != sw_get())
                        ;
                    return;
                } /* end of switch */
        } /* end of while (1) */
}

/* get four bytes of slave identifier, display serially in LED display */
void LIN_Get_Num_4Byte(void)
{
unsigned char sw_val;
unsigned char rxbuf[4];
unsigned char dig;
unsigned int i;
MD_STATUS status;

        while (1) {
                /* get non-zero switch */
                while (SW_LU_RU == sw_get())
                    ;
                sw_val = sw_get();
                switch (sw_val) {
                case SW_LU_RD:
                    /* SW3 pressed, get number from the slave */
                    /* wait for switches to be up */
                    while (SW_LU_RU != sw_get())
                        ;
                    status = LIN_M_ReceiveData(LIN_ID_GetSlaveNum4, rxbuf);
                    if (status == MD_OK) {
                        /* no error, show number in LEDs */
                        for (dig = 0; dig < 4; dig++) {
                            led_dig(rxbuf[dig]);
                            /* delay about a second */
                            for (i=0; i < 1000; i++) {
                                LIN_M_WaitHalfBit(40);     /* 40 * 26 usec = about 1
millisecond */
                            }
                        }
                        /* restore display to show "4 " */
                        led_dig_left(4);
                        led_out_right(LED_PAT_BLANK);
                    } else {
                        /* if error, show E plus low four bits of error code */
                        led_out_left(LED_PAT_E);
                        led_dig_right((UCHAR)status & 0x0F);
                    }
                    break;
                case SW_LD_RU:
                    /* SW2 pressed, return from test */
                    /* wait for switches to be up */
                    while (SW_LU_RU != sw_get())
                        ;
                    return;
                } /* end of switch */
```

```
      } /* end of while (1) */
}
```

7.2.2.2 LIN Master LIN_M.h

```
/*
*******************************************************************************
**
** This file was created for the NEC Electronics Serial Application Note, LIN Section
** 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 : LIN_M.h
** Abstract : Header for LIN Master functions and variables in LIN_M.c
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
*******************************************************************************
*/
#ifndef _LIN_M_H
#define _LIN_M_H

/* global variables for LIN Master operation */
extern BOOL LIN_M_Active;               /* turn on to bypass Applilet ISR */
extern MD_STATUS LIN_M_Error;           /* result error codes for LIN operation */

 /* initialize LIN for Master */
void LIN_M_Init(void);
 /* routine to wait for N * 1/2 bit times */
void LIN_M_WaitHalfBit(unsigned int halftimes);
 /* routine to calculate LIN ID parity */
UCHAR LIN_M_CalcIdParity(UCHAR ID);
 /* LIN Function to Send Data */
MD_STATUS LIN_M_SendData(UCHAR ID, UCHAR *txbuf);
 /* LIN Function to Receive Data */
MD_STATUS LIN_M_ReceiveData(UCHAR ID, UCHAR *rxbuf);

void LIN_M_Tx_Handler(void);            /* transmit ISR handler */
void LIN_M_Rx_Handler(void);            /* receive ISR handler */

#endif /* _LIN_M_H */
```

7.2.2.3 LIN Master LIN_M.c

```
/*
************************************************************************
**
** This file was created for the NEC Electronics Serial Application Note, LIN Section
**
** 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 : LIN_M.c
** Abstract : This file implements functions for LIN Master
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "LIN.h"           /* include for LIN Application Note, generic LIN */
#include "LIN_M.h"         /* include for LIN Application Note, LIN Master */
#include "timer.h"         /* Timer used for LIN_M_WaitHalfBit() routine */
#include "sw_0537.h"       /* for switch check in DebugWaitBeforeTx() */

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

/* set following define TRUE to wait for switch press before transmit */
/* set FALSE for normal operation */
#define LIN_M_DEBUG_TX     FALSE

/*
************************************************************************
** LIN Global Variables
************************************************************************
*/

BOOL LIN_M_Active = FALSE;              /* intercept serial interrupts */

MD_STATUS LIN_M_Error = MD_OK;          /* error flag for LIN send/receive */

/*
************************************************************************
** LIN Local Variables
************************************************************************
*/

BOOL LIN_M_XferDone = TRUE;             /* flag for transfer done */

LIN_XFER_STRUCT LIN_M_Tx;         /* master structure to control transfer */

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

```
** Abstract:
**     void LIN_M_Init(void) - Initialize LIN
** UART6 initialization has already been done in UART6_Init()
** For this application, just sets LIN_M_Active
**      Depending on hardware, could do other hardware/software initialization
**
** Parameters: None
** Returns: None
**--------------------------------------------------------------------------
*/
void LIN_M_Init(void)
{
      LIN_M_Active = TRUE;       /* set flag to intercept UART6 ISRs */
      /* other initialization here as required */
}

/*
**--------------------------------------------------------------------------
** Abstract:
**     void LIN_M_WaitHalfBit(unsigned int halftimes) waits for (halfbits * 1/2 bit)
** Uses TM51 for 1/2 bit time timer
**
** Parameters: unsigned int halftimes - number of half bits to wait
** Returns: None
**--------------------------------------------------------------------------
*/
void LIN_M_WaitHalfBit(unsigned int halftimes)
{
      /* user Timer 51, interval set for 26 microseconds = 1/2 bit at 19200 baud */
      if (halftimes == 0)
            return;
      TMIF51 = 0;               /* make sure flag is clear */
      TM51_Start();/* start the timer */
      while (halftimes > 0) {
            /* wait for timer interrupt flag set */
            while (TMIF51 != 1)
                  ;
            TMIF51 = 0;         /* clear flag */
            halftimes--; /* count down */
      }
      TM51_Stop();
}

/*
**--------------------------------------------------------------------------
** Abstract:
**     LIN_M_CalcIdParity(UCHAR ID) calculates LIN double parity bits 7 and 6
**
** Parameters: UCHAR ID - current ID value (bits 7 and 6 ignored)
** Returns: UCHAR - ID with bits 7 and 6 set to LIN parity
**--------------------------------------------------------------------------
*/
UCHAR LIN_M_CalcIdParity(UCHAR ID)
{
UCHAR p0,p1;

      p0 = p1 = 0;  /* starting state is zero for both bits */
      ID &= 0x3F;         /* strip existing parity bits if any */
      if (ID & 0x01) {    /* check ID0 */
            p0 = 0x40;          /* P0 = ID0 */
      }
      if (ID & 0x02) {    /* check ID1 */
            p0 ^= 0x40;         /* P0 = ID0 xor ID1 */
```

```
            p1 = 0x80;          /* P1 = ID1 */
      }
      if (ID & 0x04) {    /* check ID2 */
            p0 ^= 0x40;         /* P0 = ID0 xor ID1 xor ID2 */
      }
      if (ID & 0x08) {    /* check ID3 */
            p1 ^= 0x80;         /* P1 = ID1 xor ID3 */
      }
      if (ID & 0x10) {    /* check ID4 */
            p0 ^= 0x40;         /* P0 = ID0 xor ID1 xor ID2 xor ID4, done */
            p1 ^= 0x80;         /* P1 = ID1 xor ID3 xor ID4 */
      }
      if (ID & 0x20) {    /* check ID5 */
            p1 ^= 0x80;         /* P1 = ID1 xor ID3 xor ID4 xor ID5 */
      }
      p1 ^= 0x80;               /* P1 = NOT(ID1 xor ID2 xor ID4 xor ID5), done */
      ID = ID | p0 | p1;  /* form result */
      return (ID);        /* and return */
}

/*
**-------------------------------------------------------------------------
** Abstract:
**     void LIN_M_SendByte(UCHAR data)
**      sends a byte of data by writing to TXB6
** optional spot to wait for keypress if debugging
**
** Parameters: UCHAR data - byte to send
** Returns:  None
**-------------------------------------------------------------------------
*/
void LIN_M_SendByte(UCHAR data)
{
#if (LIN_M_DEBUG_TX == TRUE)
      /* if debugging, wait for switch down */
      while (SW_LU_RU == sw_get()) {
            /* check debounce timer interrupt flag, so switches debounced */
            /* even if interrupts are disabled */
            if (TMIF50) {
                  TMIF50 = 0;
                  sw_isr();
            }
      }
      while (SW_LU_RU != sw_get()) {
            if (TMIF50) {
                  TMIF50 = 0;
                  sw_isr();
            }
      }
#endif
      TXB6 = data;  /* send data byte */
}

/*
**-------------------------------------------------------------------------
** Abstract:
**     MD_STATUS LIN_M_SendData(UCHAR ID, UCHAR *txbuf)
** Sets up and starts operation to send data to slave
**
** Parameters:
**           UCHAR ID - LIN ID byte, command to slave (includes length)
**           UCHAR *txbuf - pointer to data to send (copied to local buffer)
** Returns: Error code
**           MD_LIN_ERROR_PARAM - ID not correct, send not done
```

```
**           MD_OK  no error sending
**           other  - error set during sending
**-------------------------------------------------------------------------
*/
MD_STATUS LIN_M_SendData(UCHAR ID, UCHAR *txbuf)
{
UCHAR tmp;
unsigned int i;

       if (ID > 0x3F)
              return MD_LIN_ERROR_PARAM; /* parity bits not zero, ID error */
       tmp = ID & LIN_ID_DIRMASK;
       if (tmp != LIN_ID_DIR_OUT)
              return MD_LIN_ERROR_PARAM; /* not a write operation, ID error */

       /* set up structure for transmission */
       LIN_M_Tx.dir = tmp;        /* set direction */
       LIN_M_Tx.state = LIN_STATE_SBF;  /* at point of sending sync break field */
       LIN_M_Tx.id = LIN_M_CalcIdParity(ID);
       tmp = ID & LIN_ID_LENMASK; /* mask to length bits */

       if ((tmp == LIN_ID_2BYTE) || (tmp == LIN_ID_2BYTE_B)) {
              LIN_M_Tx.len = 2;
       } else {
              if (tmp == LIN_ID_4BYTE) {
                     LIN_M_Tx.len = 4;
              } else {
                     LIN_M_Tx.len = 8;
              }
       }
       LIN_M_Tx.count = 0;
       /* copy data to local buffer, calculate checksum to send */
       i = 0;
       for (tmp = 0; tmp < LIN_M_Tx.len; tmp++) {
              LIN_M_Tx.data[tmp] = txbuf[tmp];
              i += (unsigned int) txbuf[tmp];
              if (i > 0xFF) {
                     i = (i + 1) & 0xFF;
              }
       }
       i = (~i) & (unsigned int) 0xFF;  /* invert checksum and mask */
       LIN_M_Tx.checksum = (UCHAR) i;

       /* ok to start transmission, set flags */
       LIN_M_XferDone = FALSE;
       LIN_M_Error = MD_OK;

       /* Set UART6 to send and receive sync break */
       SRMK6 = 1;    /* mask receive interrupt */
       SBRT6 = 1;    /* set receive in SBF mode */
/*     SBTT6 = 1; */ /* CC78K0 rejects the name SBTT6 for ASICL6.5 */
       ASICL6.5 = 1;

       /* wait for transmission to be done or error to occur */
       while ((LIN_M_XferDone == FALSE) && (LIN_M_Error == MD_OK))
              ;
       LIN_M_XferDone = TRUE;                          /* done if error or not */
       LIN_M_Tx.state = LIN_STATE_IDLE; /* mark as idle */
       return (LIN_M_Error);                           /* return error code */
}

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

```
** Abstract:
**     MD_STATUS LIN_M_ReceiveData(UCHAR ID, UCHAR *rxbuf)
** Sets up and starts operation to send command, receive data from slave
**
** Parameters:
**           UCHAR ID - LIN ID byte, command to slave (includes length)
**           UCHAR *rxbuf - pointer to buffer to hold received data (copied from local
buffer)
** Returns: Error code
**           MD_LIN_ERROR_PARAM - ID not correct, command not sent
**           MD_OK  no error sending/receiving
**           other  - error set during sending/receive
**-------------------------------------------------------------------------
*/
MD_STATUS LIN_M_ReceiveData(UCHAR ID, UCHAR *rxbuf)
{
UCHAR tmp;

      if (ID > 0x3F)
            return MD_LIN_ERROR_PARAM; /* parity bits not zero, ID error */
      tmp = ID & LIN_ID_DIRMASK;
      if (tmp != LIN_ID_DIR_IN)
            return MD_LIN_ERROR_PARAM; /* not a read operation, ID error */

      /* set up structure for transmission */
      LIN_M_Tx.dir = tmp;        /* set direction */
      LIN_M_Tx.state = LIN_STATE_SBF;  /* at point of sending sync break field */
      LIN_M_Tx.id = LIN_M_CalcIdParity(ID);
      tmp = ID & LIN_ID_LENMASK; /* mask to length bits */

      if ((tmp == LIN_ID_2BYTE) || (tmp == LIN_ID_2BYTE_B)) {
            LIN_M_Tx.len = 2;
      } else {
            if (tmp == LIN_ID_4BYTE) {
                  LIN_M_Tx.len = 4;
            } else {
                  LIN_M_Tx.len = 8;
            }
      }
      LIN_M_Tx.count = 0;
      LIN_M_Tx.checksum = 0;

      /* ok to start transmission, set flags */
      LIN_M_XferDone = FALSE;
      LIN_M_Error = MD_OK;

      /* Set UART6 to send and receive sync break */
      SRMK6 = 1;    /* mask receive interrupt */
      SBRT6 = 1;    /* set receive in SBF mode */
/*    SBTT6 = 1; */ /* CC78K0 rejects the name SBTT6 for ASICL6.5 */
      ASICL6.5 = 1;

      /* wait for transmission to be done or error to occur */
      while ((LIN_M_XferDone == FALSE) && (LIN_M_Error == MD_OK))
            ;
      LIN_M_XferDone = TRUE;           /* done if error or not */
      LIN_M_Tx.state = LIN_STATE_IDLE; /* mark as idle */
      /* if receive was good, copy data back */
      if (LIN_M_Error == MD_OK) {
            for (tmp = 0; tmp < LIN_M_Tx.len; tmp++) {
                  *rxbuf++ = LIN_M_Tx.data[tmp];
            }
      }
      return (LIN_M_Error);            /* return error code */
```

```
}

/*
**-------------------------------------------------------------------------
** Abstract:
**     void LIN_M_Tx_Handler(void) is LIN master transmit done ISR
**      in this application, called from MD_INTST6
**
** Parameters: None
** Returns: None
**           Changes LIN_S_Tx structure in sending data
**           Sets LIN_M_Error on errors
**-------------------------------------------------------------------------
*/
void LIN_M_Tx_Handler(void)
{
UCHAR tmp;

       if (LIN_M_XferDone) {
              /* interrupt should not happen at this point - ignore it */
              /* could report or handle error here */
              return;
       }

       /* handle transmit interrupt based on state */
       switch (LIN_M_Tx.state) {
       case LIN_STATE_IDLE:
              /* interrupt should not happen in this state- ignore it */
              /* could report or handle error here */
              break;

       case LIN_STATE_SBF:
              /* interrupt at end of synch break transmission */
              /* make sure it was also received */
              LIN_M_WaitHalfBit(2);      /* wait for one bit time */
              if ((SRIF6 == 0) || (SBRF6 == 1)) {
                     /* no interrupt, or still SBF reception state */
                     LIN_M_Error = MD_LIN_ERROR_NOSBF;
                     break;
              }
              /* received SBF ok */
              SRIF6 = 0;          /* clear receive interrupt flag */
              LIN_M_SendByte(0x55);      /* send SYNC byte */
              LIN_M_Tx.state = LIN_STATE_SYNC; /* change to SYNC state */
              break;

       case LIN_STATE_SYNC:
              /* interrupt at end of sending SYNC */
              /* make sure it was also received */
              LIN_M_WaitHalfBit(3);            /* wait 1.5 bit times for receive check */
              if (SRIF6 == 0) {
                     LIN_M_Error = MD_LIN_ERROR_NOREC;/* data not received */
                     break;
              }
              SRIF6 = 0;          /* clear receive interrupt flag */
              tmp = ASIS6;  /* clear and ignore any receive errors */
              tmp = RXB6;         /* get character received */
              if (tmp != 0x55) {
                     LIN_M_Error = MD_LIN_ERROR_DTNEQ;/* sync not received as sent */
                     break;
              }
              LIN_M_WaitHalfBit(6);            /* wait 3 bit times for slave to process sync
*/
```

```
            LIN_M_SendByte(LIN_M_Tx.id);      /* send ID */
            LIN_M_Tx.state = LIN_STATE_ID;
            break;

      case LIN_STATE_ID:
            /* interrupt at end of sending ID */
            /* make sure it was also received */
            LIN_M_WaitHalfBit(3);             /* wait 1.5 bit times for receive check */
            if (SRIF6 == 0) {
                  LIN_M_Error = MD_LIN_ERROR_NOREC; /* data not received */
                  break;
            }
            SRIF6 = 0;          /* clear receive interrupt flag */
            tmp = ASIS6;  /* clear and ignore any receive errors */
            tmp = RXB6;         /* get character received */
            if (tmp != LIN_M_Tx.id) {
                  LIN_M_Error = MD_LIN_ERROR_DTNEQ; /* sync not received as sent */
                  break;
            }
            /* after ID send and received, action depends on data direction */
            if (LIN_ID_DIR_OUT == (LIN_M_Tx.id & LIN_ID_DIRMASK)) {
                  /* transmitting, send the first data byte */
                  LIN_M_SendByte(LIN_M_Tx.data[0]);
                  LIN_M_Tx.state = LIN_STATE_DATAW;
            } else {
                  /* receiving, enable receive interrupt, rest of operation handled in rx
ISR */
                  LIN_M_Tx.count = 0;         /* clear count to be sure */
                  LIN_M_Tx.checksum = 0;      /* clear checksum of received data */
                  LIN_M_Tx.state = LIN_STATE_DATAR; /* move to receive data state */
                  SRMK6 = 0;                        /* enable serial receive interrupt */
                  /* TODO - set timer for transfer timeout */
            }
            break;

      case LIN_STATE_DATAW:
            /* interrupt after sending data bytes */
            /* should receive same data as sent */
            LIN_M_WaitHalfBit(3);             /* wait 1.5 bit times for receive check */
            if (SRIF6 == 0) {
                  LIN_M_Error = MD_LIN_ERROR_NOREC; /* data not received */
                  break;
            }
            SRIF6 = 0;          /* clear receive interrupt flag */
            tmp = ASIS6;  /* clear and ignore any receive errors */
            tmp = RXB6;         /* get character received */
            if (tmp != LIN_M_Tx.data[LIN_M_Tx.count]) {
                  LIN_M_Error = MD_LIN_ERROR_DTNEQ; /* data not received as sent */
                  break;
            }
            /* data received as sent, send next, or move to checksum if data done */
            LIN_M_Tx.count = LIN_M_Tx.count + 1;
            if (LIN_M_Tx.count < LIN_M_Tx.len) {
                  /* send next data byte */
                  LIN_M_SendByte(LIN_M_Tx.data[LIN_M_Tx.count]);
            } else {
                  /* data sent, send checksum */
                  LIN_M_SendByte(LIN_M_Tx.checksum);
                  LIN_M_Tx.state = LIN_STATE_SUMW;
            }
            break;

      case LIN_STATE_SUMW:
            /* interrupt after sending checksum */
```

```
            /* should receive same checksum as sent */
            LIN_M_WaitHalfBit(3);            /* wait 1.5 bit times for receive check */
            if (SRIF6 == 0) {
                  LIN_M_Error = MD_LIN_ERROR_NOREC;/* data not received */
                  break;
            }
            SRIF6 = 0;          /* clear receive interrupt flag */
            tmp = ASIS6;  /* clear and ignore any receive errors */
            tmp = RXB6;         /* get character received */
            if (tmp != LIN_M_Tx.checksum) {
                  LIN_M_Error = MD_LIN_ERROR_DTNEQ;/* data not received as sent */
                  break;
            } else {
                  /* checksum received as sent, transmission is done */
                  LIN_M_XferDone = TRUE;
                  LIN_M_Tx.state = LIN_STATE_IDLE;
            }
            break;

      case LIN_STATE_DATAR:
      case LIN_STATE_SUMR:
            /* interrupt should not happen in these states- ignore it */
            /* could report or handle error here */
            break;
      default:
            /* unknown state, should not happen - data frame corrupted? */
            LIN_M_Error = MD_LIN_ERROR;             /* report generic error */
            break;
      } /* end of switch (LIN_M_Tx.state) */
/* end of LIN_M_Tx_Handler */
}

/*
**-----------------------------------------------------------------------------
** Abstract:
**     void LIN_M_Rx_Handler(void) is LIN master receive character ISR
**      in this application, called from MD_INTSR6
**
** Parameters: None
** Returns: None
**            Changes LIN_S_Tx structure in receiving data
**            Sets LIN_M_Error on errors
**-----------------------------------------------------------------------------
*/
void LIN_M_Rx_Handler(void)
{
UCHAR tmp,tmp2;
unsigned int i;

      switch (LIN_M_Tx.state) {
      case LIN_STATE_IDLE:
            /* should not happen, ignore data or error */
            tmp2 = ASIS6;/* read error status (clears ASIS6) */
            tmp = RXB6;         /* read data byte */
            /* if desired, could report by setting LIN_M_Error here, or handle in place */

            break;
      case LIN_STATE_SBF:
      case LIN_STATE_SYNC:
      case LIN_STATE_ID:
      case LIN_STATE_DATAW:
      case LIN_STATE_SUMW:
            /* all above states should not happen */
```

```
            tmp2 = ASIS6;/* read error status (clears ASIS6) */
            tmp = RXB6;         /* read data byte */
            LIN_M_Error = MD_LIN_ERROR_STATE;       /* report state error */
            break;

      case LIN_STATE_DATAR:
            /* receive data */
            tmp2 = ASIS6;/* read error status (clears ASIS6) */
            tmp = RXB6;         /* read data byte */
            if (tmp2 != 0) {
                  /* report error based on bits set in ASIS6 */
                  switch (tmp2 & 0x07) {
                   case 0x01:  LIN_M_Error = MD_LIN_ERROR_RCVOV;break; /* overflow */
                   case 0x02:  LIN_M_Error = MD_LIN_ERROR_RCVFE;break;       /* framing
*/
                   case 0x04:  LIN_M_Error = MD_LIN_ERROR_RCVPE;break; /* parity */
                   default:          LIN_M_Error = MD_LIN_ERROR_RCVMU;break; /* multiple
bits */
                  }
                  return;
            }
            /* store data */
            LIN_M_Tx.data[LIN_M_Tx.count] = tmp;
            /* update checksum */
            i = (unsigned int) LIN_M_Tx.checksum;   /* get to 16-bit value to maintain
carry */
            i += (unsigned int) tmp;
            if (i > 0xFF) {
                  i = (i + 1) & (unsigned int) 0xFF;
            }
            LIN_M_Tx.checksum = (UCHAR) i;
            LIN_M_Tx.count = LIN_M_Tx.count + 1;    /* increment count */
            if (LIN_M_Tx.count == LIN_M_Tx.len) {
                  /* have received all data, wait for checksum */
                  LIN_M_Tx.state = LIN_STATE_SUMR;
            }
            break;

      case LIN_STATE_SUMR:
            /* receive checksum */
            tmp2 = ASIS6;/* read error status (clears ASIS6) */
            tmp = RXB6;         /* get byte */
            if (tmp2 != 0x00) {
                  /* report error based on bits set in ASIS6 */
                  switch (tmp2 & 0x07) {
                   case 0x01:  LIN_M_Error = MD_LIN_ERROR_RCVOV;break; /* overflow */
                   case 0x02:  LIN_M_Error = MD_LIN_ERROR_RCVFE;break;       /* framing
*/
                   case 0x04:  LIN_M_Error = MD_LIN_ERROR_RCVPE;break; /* parity */
                   default:          LIN_M_Error = MD_LIN_ERROR_RCVMU;break; /* multiple
bits */
                  }
                  return;
            }
            LIN_M_Tx.rcvsum = tmp;                  /* store received checksum */
            i = LIN_M_Tx.checksum + tmp;            /* add received inverted sum to
checksum */
            if (i != 0xFF) {
                  LIN_M_Error = MD_LIN_ERROR_CKSUM;/* checksum error */
                  return;
            }
            LIN_M_Tx.state = LIN_STATE_IDLE;
            LIN_M_XferDone = TRUE;
            break;
```

```
      default:
            /* unknown state, should not happen */
            tmp2 = ASIS6;/* read error status (clears ASIS6) */
            tmp = RXB6;         /* read data byte */
            LIN_M_Error = MD_LIN_ERROR;             /* report generic error */
            break;
      } /* end of switch (LIN_Tx.state) */
/* end of LIN_M_Rx_Handler */
}
```

7.2.2.4 LIN Master 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/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  20000000
#define      SUBCLOCK     32768
#define      MAINCLOCK    20000000
#define      FRCLOCK      8000000
#define      FRCLOCKLOW   240000


#endif
```

7.2.2.5 LIN Master 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#ifndef      _MDSYSTEM_
#define      _MDSYSTEM_
/*
************************************************************************
** MacroDefine
************************************************************************
*/
#define      CG_X1STAB_SEL 0x5
#define      CG_X1STAB_STA 0x1f
#define      CG_CPU_CLOCKSEL     0x0
#define      CG_Mainosc    0x14

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
```

7.2.2.6 LIN Master 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************************
*/
/*
************************************************************************************
** Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "port.h"
#include "timer.h"
#include "serial.h"
/*
************************************************************************************
** MacroDefine
************************************************************************************
*/

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     Init every Macro
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void SystemInit( void )
{
      /* Clock generator initiate */
      Clock_Init();
      /* Port initiate */
      PORT_Init();
      /* UART6 initiate */
      UART6_Init();
      /* TM50 initiate */
      TM50_Init();
      /* TM51 initiate */
      TM51_Init();
```

```
}

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

7.2.2.7 LIN Master 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 :   NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC Electronics/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 )
{
       USHORT i;
       UCHAR temp_stabset, temp_stabwait;
       SetIORBit(PM12, 0x06);                  /* P121/122 input mode */
       ClrIORBit(OSCCTL, 0x80);          /* X1/X2 input mode */
       SetIORBit(OSCCTL, 0x40);
       ClrIORBit(MOC, 0x80);
       /* OSC stabilization time */
       temp_stabset = CG_X1STAB_STA;
       do{
              temp_stabwait = OSTC;
              temp_stabwait &= temp_stabset;
       }while(temp_stabwait != temp_stabset);
       OSTS = CG_X1STAB_SEL;
       for(i = 0; i <= 20; i++){         /* wait 5us */
              NOP();
       }
       SetIORBit(OSCCTL, 0x01);          /* 10MHz<fx<=20MHz */
       SetIORBit(MCM, 0x05);                  /* X1 operate for CPU */
       SetIORBit(PM12, 0x18);                 /* P123/124 input mode */
       ClrIORBit(OSCCTL, 0x20);          /* XT1 input mode */
       SetIORBit(OSCCTL, 0x10);
       ClrIORBit(RCM, 0x01);
       PCC = CG_CPU_CLOCKSEL;
}
```

7.2.2.8 LIN Master 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#ifndef _MDSERIAL_
#define _MDSERIAL_
/*
************************************************************************
** Global variables
************************************************************************
*/
#define      UART_BAUDRATE_M6    0x2
#define      UART_BAUDRATE_K6    0x82

void UART6_Init( void );
MD_STATUS UART6_SendData( UCHAR* txbuf, UCHAR txnum );
MD_STATUS UART6_ReceiveData( UCHAR* rxbuf, UCHAR rxnum );
__interrupt void MD_INTSR6( void );
__interrupt void MD_INTST6( void );

enum TransferMode { Send, Receive };

#endif
```

7.2.2.9 LIN Master 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
```

```
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/

#pragma      interrupt    INTSR6       MD_INTSR6
#pragma      interrupt    INTST6       MD_INTST6

/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"

/* add includes for LIN */
#include "LIN.h"
#include "LIN_M.h"

/* uart6 transmission */
UCHAR* UART6_TX_ADDRESS;                /* uart transfer buffer address */
UCHAR UART6_TX_CNT;                     /* uart transfer number */

/* uart6 reception */
UCHAR* UART6_RX_ADDRESS;                /* uart receive buffer address */
UCHAR UART6_RX_CNT;                     /* uart receive data number */

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     This function initializes UART6 module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void UART6_Init( void )
{
      ASIM6 = 0;

      SetIORBit(P1, 0x08);                  /* port setting in transmit/receive mode */
      ClrIORBit(PM1, 0x08);

      SetIORBit(ASIM6, 0x04);               /* data length 8 bits */
      ClrIORBit(ASIM6, 0x02);               /* stop length 1 bits */
      SetIORBit(ASIM6, 0x01);
      ClrIORBit(ASIM6, 0x18);               /* none parity mode */
      SetIORBit(ASICL6, 0x02);        /* LSB first transfer */
      ClrIORBit(ASICL6, 0x01);        /* normal output of TxD6 */
      CKSR6 = UART_BAUDRATE_M6;       /* baudrate selection */
      BRGC6 = UART_BAUDRATE_K6;
      ClrIORBit(IF0H, 0x02);
      ClrIORBit(MK0H, 0x02);                /* transfer end interrupt enable */
      ClrIORBit(IF0H, 0x01);
      ClrIORBit(MK0H, 0x01);                /* receive end interrupt enable */
      SetIORBit(PR0H, 0x01);                /* interrupt low */
```

```
	SetIORBit(PR0H, 0x02);			/* interrupt low */

	SetIORBit(ASIM6, 0x80);
	SetIORBit(ASIM6, 0x60);			/* transmit/receive mode */
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for UART6 data receiving.
**
** Parameters:
**     rxbuf: receive buffer pointer
**     rxnum: buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS UART6_ReceiveData( UCHAR* rxbuf, UCHAR rxnum )
{
	if( rxnum < 1 ){
	return MD_ERROR;
 }

	UART6_RX_CNT = rxnum;
	UART6_RX_ADDRESS = rxbuf;

	return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for UART6 data transfering.
**
** Parameters:
**     txbuf: transfer buffer pointer
**     txnum: buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS UART6_SendData( UCHAR* txbuf, UCHAR txnum )
{
	if( txnum < 1 ){
		return MD_ERROR;	/* 1 frame data at least */
	}

	UART6_TX_ADDRESS = txbuf;
	UART6_TX_CNT = txnum;
	if( txnum == 1) {		/* only 1 frame data to transfer doesn't need contiously
transfer */
		if((ASIF6 & 0x02) == 0){
			TXB6 = *UART6_TX_ADDRESS++;
		}
```

```
		else{
			return MD_DATAEXISTS;
		}
	}
	else {
		if((ASIF6 & 0x02) == 0){
			TXB6 = *UART6_TX_ADDRESS++;
			while((ASIF6 & 0x02) == 1);		/* wait */
			UART6_TX_CNT--;
			TXB6 = *UART6_TX_ADDRESS++;
		}
		else{
			return MD_DATAEXISTS;
		}
	}
	return MD_OK;
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function is the UART6 receive end interrupt handler.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTSR6( void )
{
	UCHAR err_type;
	UCHAR tmp;

	/* intercept receive interrupt for LIN if active */
	if (LIN_M_Active) {
		LIN_M_Rx_Handler();
		return;
	}

	err_type = ASIS6;
	tmp = RXB6;

	if( err_type != 0 ){
		return;
	}

	if( UART6_RX_CNT == 0 ){
		/* receive data end */
		return;
	}

	/* the interrut generated by receive end */
	*UART6_RX_ADDRESS ++= tmp;
	UART6_RX_CNT--;
}

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

```
**     This function is the UART6 transfer end interrupt handler.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTST6( void )
{
       /* intercept transmit interrupt for LIN if active */
       if (LIN_M_Active) {
              LIN_M_Tx_Handler();
              return;
       }

       if( UART6_TX_CNT == 0 ){
              /* send data complete */
              return;
       }

       if(((ASIF6 & 0x02) == 0) && ((ASIM6 & 0x80) == 0x80)){
              UART6_TX_CNT--;
              if( UART6_TX_CNT != 0 ){
                     TXB6 = *UART6_TX_ADDRESS++;
              }
       }
}
```

7.2.2.10 LIN Master 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 :   NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
*******************************************************************************
```

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

7.2.2.11 LIN Master 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/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 0x6
#define       TM_TM50_INTERVALVALUE      0x4d
#define       TM_TM50_SQUAREWIDTH 0x4d
#define       TM_TM50_PWMACTIVEVALUE     0x4d
#define       TM_TM51_CLOCK 0x4
#define       TM_TM51_INTERVALVALUE      0x1f
#define       TM_TM51_SQUAREWIDTH 0x1f
#define       TM_TM51_PWMACTIVEVALUE     0x1f
#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);
__interrupt void MD_INTTM50(void);

#endif       /* _MDTIMER_*/
```

7.2.2.12 LIN Master 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/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/256 */
       SetIORBit(PR0H, 0x20);                  /* low priority level */
       ClrIORBit(IF0H, 0x20);
       /* 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);
      ClrIORBit(MK0H, 0x20);                  /* INTTM50 enable */
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function can stop the TM50 counter and clear the count register.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM50_Stop( void )
{
      ClrIORBit(TMC50, 0x80);
      SetIORBit(MK0H, 0x20);                  /* INTTM50 disable */
}

/*
**-------------------------------------------------------------------------
**
** 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/4 */
      /* 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;
}
```

7.2.2.13 LIN Master 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
************************************************************************
*/

#pragma sfr
#pragma interrupt INTTM50 MD_INTTM50
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"

#include "sw_0537.h"       /* for sw_isr() definition */


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

/* Timer00, Timer01 pulse width measure */
/*
**-----------------------------------------------------------------------
**
** Abstract:
**     TM50 INTTM50 interrupt-service routine
```

```
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTTM50( )
{
       sw_isr();     /* call function to check switch input */
}
```

7.2.3 Listings for LIN Slave Program

This demonstration program runs on the M-78F0537 plus M-Station II platform. The files listed are unique to this demonstration program. The non-unique files are listed in the M-78F0537 common-files section below.

7.2.3.1 LIN Slave Main.c

```
/* main.c for NEC Electronics Serial Application Note, LIN Slave */
/*
************************************************************************
**
** 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "port.h"
#include "timer.h"
#include "serial.h"

#include "LIN.h"          /* include for LIN Application Note, generic LIN */
#include "LIN_S.h"        /* include for LIN Application Note, LIN Slave */
#include "led_0537.h"     /* M-Station LED code */
#include "sw_0537.h"      /* M-Station switch code */
/*
************************************************************************
** MacroDefine
************************************************************************
*/
MD_STATUS LIN_S_ProcessCmd(void);
```

```
/*
*****************************************************************************
** Global variables
*****************************************************************************
*/
unsigned char g_count;     /* count to display, receive, send */

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     main function
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
extern UCHAR edge_count;
extern UCHAR edge_last;

void main( void )
{
unsigned char sw_val;      /* value of switches */
MD_STATUS status;

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;

       led_init();         /* initialize LEDs */
       sw_init();          /* initialize switch variables */
       sw_set_debounce(10);       /* set debounce counter to 10 for 10 msec stable time */


       TM50_Start();/* start timer for switch debouncing */

       LIN_S_Init();/* initialize LIN */

       g_count = 0x80;     /* starting value */
       led_dig(g_count);          /* show count in LEDs */

       LIN_S_ReceiveCmd(); /* prepare to receive LIN command/data */

       /* main loop for slave - handle switch presses and commands from Master */
       while(1){
              /* check for switch pressed */
              if (SW_LU_RU != (sw_val = sw_get())) {
                     if (sw_val == SW_LD_RU) {
                            /* SW2 pressed */
                            g_count = g_count - 3;           /* decrement count */
                            led_dig(g_count);                /* display count */
                            while (SW_LD_RU == sw_get())
                                   ;                                 /* wait for switches
differetn */
                     }
                     if (sw_val == SW_LU_RD) {
                            /* SW3 pressed */
                            g_count = g_count + 7;           /* increment count */
                            led_dig(g_count);                /* display count */
                            while (SW_LU_RD == sw_get())
```

```
                                ;                                   /* wait for switches
different */
                }
                if (sw_val == SW_LD_RD) {
                        /* SW2 and SW3 pressed */
                        led_dig(BRGC6);                             /* display current
baudrate setting */
                        while (SW_LU_RU != sw_get())
                                ;                                   /* wait for switches up
*/
                }
        }
        /* check for LIN data received, or error occurred */
        if ((LIN_S_ReceiveDone) || (LIN_S_Error != MD_OK)) {
                if (LIN_S_Error != MD_OK) {
                        /* show E plus error code */
                        led_out_left(LED_PAT_E);
                        led_dig_right((UCHAR)LIN_S_Error & 0x0F);
                } else {
                        status = LIN_S_ProcessCmd();     /* handle command */
                        if (status != MD_OK) {
                                /* show E plus error code from ProcessCmd */
                                led_out_left(LED_PAT_E);
                                led_dig_right((UCHAR)status & 0x0F);
                        }
                }
                /* set up for next receive */
                LIN_S_ReceiveCmd(); /* prepare to receive LIN data */
        }
    } /* end of while (1) loop */
} /* end of main() */

/*
**-------------------------------------------------------------------------
** Abstract:
**     MD_STATUS LIN_S_ProcessCmd(void) - Handle command from Master
**      Called after SBF, SYNC, and ID received from Master properly;
**      ID specifies command to perform and number of bytes;
**      For this application, also specifies direction
**
**     For data from master to slave, have also received data and checksum
**     Process command with sent data
**
**     For data to master from slave, prepare data to send,
**     then call LIN_S_SendData() to send to master
**
** Parameters: None (contained in LIN_S_Rx structure)
** Returns: Error code
**            MD_OK - command not supported (no action)
**            MD_OK - command recognized, data to slave, data ok
**            MD_OK - command recognized, data to master, send conpletes ok
**            other - command recognized, data to master, error on sending
**-------------------------------------------------------------------------
*/
MD_STATUS LIN_S_ProcessCmd(void)
{
UCHAR txbuf[8];
MD_STATUS status = MD_OK;

    /* handle known commands */
    switch (LIN_S_Rx.id & 0x3F) {
    case LIN_ID_SetCount2:
        /* set and display count sent from Master */
```

```
            g_count = LIN_S_Rx.data[0];
            led_dig(g_count);
            break;
      case LIN_ID_GetCount2:
            /* send count back to Master */
            txbuf[0] = g_count;
            txbuf[1] = 0;
            status = LIN_S_SendData(txbuf, 2);
            break;
      case LIN_ID_GetSlaveNum4:
            /* command to get four bytes of Slave ID */
            /* send back "SLV1" */
            txbuf[0] = 'S';
            txbuf[1] = 'L';
            txbuf[2] = 'V';
            txbuf[3] = '1';
            status = LIN_S_SendData(txbuf, 4);
            break;
      default:
            /* unknown commands are ignored */
            break;
      }
      return (status);
}
```

7.2.3.2 LIN Slave LIN_S.h

```
/*
************************************************************************
**
** This file was created for the NEC Electronics Serial Application Note, LIN Section
** 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 : LIN_S.h
** Abstract : Header for LIN Slave functions and variables in LIN_S.c
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#ifndef _LIN_S_H
#define _LIN_S_H

/* global variables for LIN operation */
extern BOOL LIN_S_Active;              /* turn on to bypass Applilet UART6 ISR */
extern MD_STATUS LIN_S_Error;          /* result error codes for LIN operation */
extern BOOL LIN_S_ReceiveDone;         /* LIN reception done flag */
extern BOOL LIN_S_SendDone;                  /* LIN transmission done flag */
extern LIN_XFER_STRUCT LIN_S_Rx;  /* LIN receive structure */
```

```
 /* Initialize LIN for Slave */
void LIN_S_Init(void);
 /* routine to wait for 1/2 bit time */
void LIN_S_WaitHalfBit(unsigned int halftimes);
 /* routine to calculate LIN ID parity */
UCHAR LIN_S_CalcIdParity(UCHAR ID);
 /* LIN Function to Receive Command */
MD_STATUS LIN_S_ReceiveCmd(void);
 /* LIN Function to send data to master in response to command */
MD_STATUS LIN_S_SendData(UCHAR *txbuf, UCHAR txlen);

void LIN_S_Tx_Handler(void);            /* transmit ISR handler */
void LIN_S_Rx_Handler(void);            /* receive ISR handler */

 /* setup and start timer for capturing bit width times */
void LIN_S_TM_Setup(void);
 /* isr to replace MD_INTTM010 timer isr */
void LIN_S_INTTM_isr(void);

#endif /* _LIN_S_H */
```

7.2.3.3 LIN Slave LIN_S.c

```
/*
************************************************************************
**
** This file was created for the NEC Electronics Serial Application Note, LIN Section
**
** 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 : LIN_S.c
** Abstract : This file implements functions for LIN Slave
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "LIN.h"           /* include for LIN Application Note, generic LIN */
#include "LIN_S.h"         /* include for LIN Application Note, LIN Slave */
#include "timer.h"         /* Timer used for LIN_S_WaitHalfBit() routine */
#include "sw_0537.h"       /* for switch check when debugging */
```

```
/*
*****************************************************************************
** MacroDefine
*****************************************************************************
*/

/* set following define TRUE for adaptive baudrate calculation */
/* set FALSE to use original baudrate */
#define LIN_S_CALC_BAUDRATE      TRUE

/* set following define TRUE to wait for switch press before transmit */
/* set FALSE for normal operation */
#define LIN_S_DEBUG_TX     FALSE

/*
*****************************************************************************
** LIN Global Variables
*****************************************************************************
*/
BOOL LIN_S_Active = FALSE;              /* intercept serial and timer interrupts */

MD_STATUS LIN_S_Error = MD_OK;          /* error flag for LIN send/receive */

BOOL LIN_S_ReceiveDone = FALSE;         /* flag for frame header received from master */

/*
*****************************************************************************
** LIN Local Variables
*****************************************************************************
*/
BOOL LIN_S_SendDone = FALSE;            /* flag for sending data to master */

LIN_XFER_STRUCT LIN_S_Rx;               /* structure to manage Slave receive/send */

/*
**-----------------------------------------------------------------------------
** Abstract:
**     void LIN_S_Init(void) - Initialize LIN for Slave
** UART6 initialization has already been done in UART6_Init()
** For this application, just sets LIN_S_Active
**      Depending on hardware, could do other hardware/software initialization
**
** Parameters: None
** Returns: None
**-----------------------------------------------------------------------------
*/
void LIN_S_Init(void)
{
       LIN_S_Active = TRUE;       /* set flag to intercept UART6 and timer ISRs */
       /* other initialization here as required */
}

/*
**-----------------------------------------------------------------------------
** Abstract:
**     LIN_S_WaitHalfBit(unsigned int halftimes) waits for (halfbits * 1/2 bit)
** Uses TM51 for 1/2 bit time timer
**
** Parameters: unsigned int halftimes - number of half bits to wait
** Returns: None
**-----------------------------------------------------------------------------
*/
void LIN_S_WaitHalfBit(unsigned int halftimes)
```

```
{
      /* user Timer 51, interval set for 26 microseconds = 1/2 bit at 19200 baud */
      if (halftimes == 0)
            return;
      TMIF51 = 0;               /* make sure flag is clear */
      TM51_Start();/* start the timer */
      while (halftimes > 0) {
            /* wait for timer interrupt flag set */
            while (TMIF51 != 1)
                  ;
            TMIF51 = 0;         /* clear flag */
            halftimes--; /* count down */
      }
      TM51_Stop();
}

/*
**-----------------------------------------------------------------------------
** Abstract:
**    LIN_S_CalcIdParity(UCHAR ID) calculates LIN double parity bits 7 and 6
**
** Parameters: UCHAR ID - current ID value (bits 7 and 6 ignored)
** Returns: UCHAR - ID with bits 7 and 6 set to LIN parity
**-----------------------------------------------------------------------------
*/
UCHAR LIN_S_CalcIdParity(UCHAR ID)
{
UCHAR p0,p1;

      p0 = p1 = 0;  /* starting state is zero for both bits */
      ID &= 0x3F;         /* strip existing parity bits if any */
      if (ID & 0x01) {    /* check ID0 */
            p0 = 0x40;          /* P0 = ID0 */
      }
      if (ID & 0x02) {    /* check ID1 */
            p0 ^= 0x40;         /* P0 = ID0 xor ID1 */
            p1 = 0x80;          /* P1 = ID1 */
      }
      if (ID & 0x04) {    /* check ID2 */
            p0 ^= 0x40;         /* P0 = ID0 xor ID1 xor ID2 */
      }
      if (ID & 0x08) {    /* check ID3 */
            p1 ^= 0x80;         /* P1 = ID1 xor ID3 */
      }
      if (ID & 0x10) {    /* check ID4 */
            p0 ^= 0x40;         /* P0 = ID0 xor ID1 xor ID2 xor ID4, done */
            p1 ^= 0x80;         /* P1 = ID1 xor ID3 xor ID4 */
      }
      if (ID & 0x20) {    /* check ID5 */
            p1 ^= 0x80;         /* P1 = ID1 xor ID3 xor ID4 xor ID5 */
      }
      p1 ^= 0x80;               /* P1 = NOT(ID1 xor ID2 xor ID4 xor ID5), done */
      ID = ID | p0 | p1;  /* form result */
      return (ID);        /* and return */
}

/*
**-----------------------------------------------------------------------------
** Abstract:
**    MD_STATUS LIN_S_ReceiveCmd(void) sets UART to receive Sync Break Field
**
** Parameters: None
** Returns:  Error code
```

```
**      MD_OK currently no error returned
**-------------------------------------------------------------------------------
*/
MD_STATUS LIN_S_ReceiveCmd(void)
{
UCHAR tmp;
       DI();               /* disable interrupts */
       TM00_Stop();  /* make sure timer stopped */
       ISC &= 0xFD;  /* set timer input TI000 to P00/TI000 */
       ASIM6 |= 0x60;      /* set bits 6 and 5 to restart TX/RX in case stopped */

       LIN_S_Rx.state = LIN_STATE_SBF;   /* set waiting for SBF */
       LIN_S_ReceiveDone = FALSE;        /* set receive flag false */
       LIN_S_Error = MD_OK;                    /* clear error flag */

       SRIF6 = 0;          /* clear any pending receive or transmit interrupts */
       STIF6 = 0;
       tmp = RXB6;         /* read receive register so no overruns */
       tmp = ASIS6;  /* clear receive errors */
       SBRT6 = 1;          /* set receive to wait for Sync Break Field */
       SRMK6 = 0;          /* make sure receive and transmit interrupts unmasked */
       STMK6 = 0;
       EI();               /* enable interrupts */
       return MD_OK;
}

/*
**-------------------------------------------------------------------------------
** Abstract:
**     void LIN_S_SendByte(UCHAR data)
**      sends a byte of data by writing to TXB6
** optional spot to wait for keypress if debugging
**
** Parameters: UCHAR data - byte to send
** Returns:  None
**-------------------------------------------------------------------------------
*/
void LIN_S_SendByte(UCHAR data)
{
#if (LIN_S_DEBUG_TX == TRUE)
       /* if debugging, wait for switch down */
       while (SW_LU_RU == sw_get()) {
              /* check debounce timer interrupt flag, so switches debounced */
              /* even if interrupts are disabled */
              if (TMIF50) {
                     TMIF50 = 0;
                     sw_isr();
              }
       }
       while (SW_LU_RU != sw_get()) {
              if (TMIF50) {
                     TMIF50 = 0;
                     sw_isr();
              }
       }
#endif
       TXB6 = data;  /* send data byte */
}

/*
**-------------------------------------------------------------------------------
** Abstract:
**     MD_STATUS LIN_S_SendData(UCHAR *txbuf, UCHAR txlen)
** Queues bytes of data for sending to master
```

```
**
** Parameters:
**           UCHAR *txbuf - pointer to data to send (copied to local buffer)
**           UCHAR txlen - count of bytes to send
** Returns: Error code
**           MD_OK  no error in queueing
**           MD_LIN_ERROR_STATE - not in DATAR (master data receive) state
**           MD_LIN_ERROR_PARAM - length doesn't match request from master
**           other  - error set during sending
**--------------------------------------------------------------------------
*/
MD_STATUS LIN_S_SendData(UCHAR *txbuf, UCHAR txlen)
{
UCHAR tmp;
unsigned int i;
       /* should be at the state of transfer after receipt of ID */
       /* LIN_S_Rx structure has been set up, including length */

       if (LIN_S_Rx.state != LIN_STATE_DATAR)
              return MD_LIN_ERROR_STATE;       /* error in state */
       if (LIN_S_Rx.len != txlen)
              return MD_LIN_ERROR_PARAM;       /* error in length */
       /* copy data to local buffer, calculate checksum to send */
       i = 0;
       for (tmp = 0; tmp < txlen; tmp++) {
              LIN_S_Rx.data[tmp] = txbuf[tmp];
              i += (unsigned int) txbuf[tmp];
              if (i > 0xFF) {
                     i = (i + 1) & 0xFF;
              }
       }
       i = (~i) & (unsigned int) 0xFF;  /* invert checksum and mask */
       LIN_S_Rx.checksum = (UCHAR) i;

       /* ok to start transmission, set flags */
       LIN_S_SendDone = FALSE;
       LIN_S_Error = MD_OK;

       /* Set UART6 to send and receive data */
       SRMK6 = 1;   /* mask receive interrupt */
       LIN_S_SendByte(LIN_S_Rx.data[0]);/* write the first data byte */

       /* rest of data and checksum transmission handled in Tx_Handler */

       /* wait for transmission to be done or error to occur */
       while ((LIN_S_SendDone == FALSE) && (LIN_S_Error == MD_OK))
              ;
       LIN_S_SendDone = TRUE;           /* done if error or not */
       LIN_S_Rx.state = LIN_STATE_IDLE; /* mark as idle */
       return (LIN_S_Error);
}

/*
**--------------------------------------------------------------------------
** Abstract:
**     void LIN_S_Tx_Handler(void) is LIN slave transmit done ISR
**      in this application, called from MD_INTST6
**
** Parameters: None
** Returns: None
**           Changes LIN_S_Rx structure in sending data
**           Sets LIN_S_Error on errors
**--------------------------------------------------------------------------
```

```
*/
void LIN_S_Tx_Handler(void)
{
UCHAR tmp;

	if (LIN_S_SendDone) {
		/* interrupt should not happen at this point - ignore it */
		/* could report or handle error here */
		return;
	}

	/* handle transmit interrupt based on state */
	switch (LIN_S_Rx.state) {
	case LIN_STATE_IDLE:
	case LIN_STATE_SBF:
	case LIN_STATE_SYNC:
	case LIN_STATE_ID:
	case LIN_STATE_DATAW:
	case LIN_STATE_SUMW:
		/* interrupt should not happen in these states - ignore it */
		/* could report or handle error here */
		break;
	case LIN_STATE_DATAR:
		/* interrupt after sending data bytes to master */
		/* should receive same data as sent */
		LIN_S_WaitHalfBit(3);		/* wait 1.5 bit times for receive check */
		if (SRIF6 == 0) {
			LIN_S_Error = MD_LIN_ERROR_NOREC;/* data not received when sent */
			break;
		}
		SRIF6 = 0;		/* clear receive interrupt flag */
		tmp = ASIS6;  /* clear and ignore any receive errors */
		tmp = RXB6;		/* get character received */
		if (tmp != LIN_S_Rx.data[LIN_S_Rx.count]) {
			LIN_S_Error = MD_LIN_ERROR_DTNEQ;/* data not received as sent */
			break;
		}
		/* data received as sent, send next, or move checksum if data done */
		LIN_S_Rx.count = LIN_S_Rx.count + 1;
		if (LIN_S_Rx.count < LIN_S_Rx.len) {
			/* send next data byte */
			LIN_S_SendByte(LIN_S_Rx.data[LIN_S_Rx.count]);
		} else {
			/* data sent, send checksum */
			LIN_S_SendByte(LIN_S_Rx.checksum);
			LIN_S_Rx.state = LIN_STATE_SUMR;
		}
		break;

	case LIN_STATE_SUMR:
		/* interrupt after sending checksum */
		/* should receive same checksum as sent */
		LIN_S_WaitHalfBit(3);		/* wait 1.5 bit times for receive check */
		if (SRIF6 == 0) {
			LIN_S_Error = MD_LIN_ERROR_NOREC;/* data not received */
			break;
		}
		SRIF6 = 0;		/* clear receive interrupt flag */
		tmp = ASIS6;  /* clear and ignore any receive errors */
		tmp = RXB6;		/* get character received */
		if (tmp != LIN_S_Rx.checksum) {
			LIN_S_Error = MD_LIN_ERROR_DTNEQ;/* data not received as sent */
			break;
		} else {
```

```
                    /* checksum received as sent, transmission is done */
                    LIN_S_SendDone = TRUE;
                    LIN_S_Rx.state = LIN_STATE_IDLE;
              }
              break;

       default:
              /* unknown state, should not happen - data frame corrupted? */
              LIN_S_Error = MD_LIN_ERROR;             /* report generic error */
              break;
       } /* end of switch (LIN_S_Rx.state) */
/* end of LIN_S_Tx_Handler */
}

/*
**-------------------------------------------------------------------------
**     void LIN_S_Rx_Handler(void) is LIN slave receive character ISR
**      in this application, called from MD_INTSR6
**
** Parameters: None
** Returns: None
**            Changes LIN_S_Rx structure in receiving data
**            Sets LIN_S_Error on errors
**-------------------------------------------------------------------------
*/
void LIN_S_Rx_Handler(void)
{
UCHAR tmp,tmp2;
unsigned int i;

       /* handle receive interrupt based on state */
       switch (LIN_S_Rx.state) {
       case LIN_STATE_IDLE:
              /* idle state, ignore received character */
              tmp = ASIS6;  /* clear and ignore any receive errors */
              tmp = RXB6;          /* in case debugging */
              break;

       case LIN_STATE_SBF:
              /* waiting for SBF, got it */
              tmp = ASIS6;  /* clear and ignore any receive errors */
              tmp = RXB6;          /* not necessary ? */
              LIN_S_Rx.state = LIN_STATE_SYNC;  /* now waiting to measure sync */
#if (LIN_S_CALC_BAUDRATE == TRUE)
              /* set up timer to measure SYNC field, recalculate baud rate */
              LIN_S_TM_Setup();
#endif
              break;

       case LIN_STATE_SYNC:
              /* only will get here if not measuring SYNC with timer */
#if (LIN_S_CALC_BAUDRATE == FALSE)
       /* this code does not readjust baudrate, just expects SYNC */
       /* code could be left even if calculating baudrate; */
       /* actual transition from LIN_STATE SYNC to LIN_STATE_ID */
       /* will be done in timer ISR if timer is used to calc and readjust baudrate */
              /* have received next character, should be SYNC */
              /* ignore any errors, and recalculate baud rate */
              tmp = ASIS6;  /* clear and ignore any receive errors */
              tmp = RXB6;
              if (tmp != 0x55) {
                    LIN_S_Error = MD_LIN_ERROR_SYNCB;/* SYNC byte not correct */
                    break;
```

```
            }
            LIN_S_Rx.state = LIN_STATE_ID;    /* now waiting for ID field */
#else
            tmp2 = ASIS6;/* clear and ignore any receive errors */
            tmp = RXB6;         /* in case debugging */
            LIN_S_Error = MD_LIN_ERROR;             /* report generic error */
#endif
            break;

      case LIN_STATE_ID:
            /* received ID, check for error */
            tmp2 = ASIS6;/* read error register */
            tmp = RXB6;         /* read ID byte */
            if (tmp2 != 0x00) {
                  /* report error based on bits set in ASIS6 */
                  switch (tmp2 & 0x07) {
                   case 0x01:  LIN_S_Error = MD_LIN_ERROR_RCVOV;break; /* overflow */
                   case 0x02:  LIN_S_Error = MD_LIN_ERROR_RCVFE;break;       /* framing
*/
                   case 0x04:  LIN_S_Error = MD_LIN_ERROR_RCVPE;break; /* parity */
                   default:           LIN_S_Error = MD_LIN_ERROR_RCVMU;break; /* multiple
bits */
                  }
                  break;
            }
            tmp2 = LIN_S_CalcIdParity(tmp);  /* check if ID parity is correct */
            if (tmp2 != tmp) {
                  LIN_S_Error = MD_LIN_ERROR_IDPAR;/* ID parity error */
                  break;
            }
            /* set up receive data structure */
            LIN_S_Rx.id = tmp;         /* set ID */
            LIN_S_Rx.dir = tmp & LIN_ID_DIRMASK;    /* set direction */

            tmp = tmp & LIN_ID_LENMASK;       /* mask to length bits */
            if ((tmp == LIN_ID_2BYTE) || (tmp == LIN_ID_2BYTE_B)) {
                  LIN_S_Rx.len = 2;
            } else {
                  if (tmp == LIN_ID_4BYTE) {
                        LIN_S_Rx.len = 4;
                  } else {
                        LIN_S_Rx.len = 8;
                  }
            }
            LIN_S_Rx.count = 0;
            LIN_S_Rx.checksum = 0;
            if (LIN_S_Rx.dir == LIN_ID_DIR_OUT) {
                  /* direction is out (from Master), we are receiving data */
                  LIN_S_Rx.state = LIN_STATE_DATAW;
            } else {
                  /* direction is in (to Master), we need to send */
                  /* set flag ReceiveDone, handler in main to prep and send data */
                  LIN_S_Rx.state = LIN_STATE_DATAR;
                  LIN_S_ReceiveDone = TRUE;
            }
            break;

      case LIN_STATE_DATAW:
            /* receive data */
            tmp2 = ASIS6;/* get error status */
            tmp = RXB6;         /* get byte */
            if (tmp2 != 0x00) {
                  /* report error based on bits set in ASIS6 */
                  switch (tmp2 & 0x07) {
```

```
                    case 0x01:  LIN_S_Error = MD_LIN_ERROR_RCVOV; break; /* overflow */
                    case 0x02:  LIN_S_Error = MD_LIN_ERROR_RCVFE; break;       /* framing
*/
                    case 0x04:  LIN_S_Error = MD_LIN_ERROR_RCVPE; break; /* parity */
                    default:           LIN_S_Error = MD_LIN_ERROR_RCVMU; break; /* multiple
bits */
                  }
                  break;
            }
            /* store data */
            LIN_S_Rx.data[LIN_S_Rx.count] = tmp;
            /* update checksum */
            i = (unsigned int) LIN_S_Rx.checksum;
            i += (unsigned int) tmp;
            if (i > 0xFF) {
                  i = (i + 1) & (unsigned int) 0xFF;
            }
            LIN_S_Rx.checksum = (UCHAR) i;
            LIN_S_Rx.count = LIN_S_Rx.count + 1;    /* increment count */
            if (LIN_S_Rx.count == LIN_S_Rx.len) {
                  /* have received all data, wait for checksum */
                  LIN_S_Rx.state = LIN_STATE_SUMW;
            }
            break;

      case LIN_STATE_SUMW:
            /* receive data */
            tmp2 = ASIS6; /* get error status */
            tmp = RXB6;         /* get byte */
            if (tmp2 != 0x00) {
                  /* report error based on bits set in ASIS6 */
                  switch (tmp2 & 0x07) {
                    case 0x01:  LIN_S_Error = MD_LIN_ERROR_RCVOV; break; /* overflow */
                    case 0x02:  LIN_S_Error = MD_LIN_ERROR_RCVFE; break;       /* framing
*/
                    case 0x04:  LIN_S_Error = MD_LIN_ERROR_RCVPE; break; /* parity */
                    default:           LIN_S_Error = MD_LIN_ERROR_RCVMU; break; /* multiple
bits */
                  }
                  break;
            }
            LIN_S_Rx.rcvsum = tmp;                        /* store received checksum */
            i = LIN_S_Rx.checksum + tmp;            /* add received inverted sum to
checksum */
            if (i != 0xFF) {
                  LIN_S_Error = MD_LIN_ERROR_CKSUM; /* checksum error */
                  break;
            }
            LIN_S_Rx.state = LIN_STATE_IDLE;
            LIN_S_ReceiveDone = TRUE;
            break;

      case LIN_STATE_DATAR:
      case LIN_STATE_SUMR:
            /* interrupt should not happen in these states - ignore it */
            /* could report or handle error here */
            tmp2 = ASIS6; /* clear and ignore any receive errors */
            tmp = RXB6;         /* in case debugging */
            break;

      default:
            /* unknown state, should not happen - data frame corrupted? */
            tmp2 = ASIS6; /* clear and ignore any receive errors */
```

```
            tmp = RXB6;         /* in case debugging */
            LIN_S_Error = MD_LIN_ERROR;             /* report generic error */
            break;
      } /* end of switch (LIN_S_Rx.state) */
/* end of LIN_S_Rx_Handler */
}

UCHAR LIN_S_INTTM_count = 0;     /* count for number of interrupts on RXD rising edge */

unsigned int LIN_S_8Bit_count;   /* summing register for timer counts over 8 bit times */

/*
**-------------------------------------------------------------------------
** Abstract:
**     void LIN_S_TM_Setup(void)
** Sets up and starts timer to measure baudrate by timing SYNC bit widths
**
** Parameters: None
** Returns: None
**-------------------------------------------------------------------------
*/
void LIN_S_TM_Setup(void) {
      LIN_S_INTTM_count = 0;           /* clear count of interrupts */

      SRMK6 = 1;          /* mask serial receive interrupt */

      ISC |= 0x02;  /* set ISC bit 1 to send P14/RXD6 to TI000 */

      TM00_Start();/* start timer to measure bit cycle widths */
}

/*
**-------------------------------------------------------------------------
** Abstract:
**     void LIN_S_INTTM_isr(void) - Interrupt-service routine for INTTM010
**      Intercepted from MD_INTTM010()
**      In SYCN byte (0x55), will happen on each rising edge of RXD6;
**      TM00 will be cleared; CR010 will have number of counts in TM00
**      when edge occurred. Ignore first rising edge, use next four
**      to measure 4 * 2 = 8 bit times; adjust BRGC6 baudrate accordingly
**
**     Note: given expected baudrate, there is no expected overflow on CR010,
**     it is expected that 8 bit time counts will not overflow 16 bit sum,
**     and that new BRGC6 value will adjust to less than 0xFF; so no
**     range or value error handling done here. For actual use, overflows should
**     be handled, and result should be tested for between expected high and low values.
**
** Parameters: None
** Returns: None
**-------------------------------------------------------------------------
*/
void LIN_S_INTTM_isr(void) {
#if (LIN_S_CALC_BAUDRATE == TRUE)
      /* if calculating baudrate, manage timer interrupt */
UCHAR tmp;
unsigned int CR010_val = CR010;

      /* INTTM010 has occurred at rising edge of RXD */
      /* action depends on which edge */
      switch (LIN_S_INTTM_count) {
      case 0:
            /* first edge, end of start bit, start of bit 0; ignore, set count to 1 */
            LIN_S_INTTM_count = 1;
            break;
```

```
      case 1:
            /* second edge, start of bit 2, CR010 has time for bit 0 + bit 1 (2 bits) */
            LIN_S_8Bit_count = CR010_val;    /* just set value, has 2 bit times */
            LIN_S_INTTM_count = 2;
            break;
      case 2:
            /* third edge, start of bit 4, CR010 has time for bits 2 and 3, add to
previous */
            LIN_S_8Bit_count += CR010_val;   /* now has 4 bit times */
            LIN_S_INTTM_count = 3;
            break;
      case 3:
            /* fourth edge, start of bit 6, CR010 has time for bits 4 and 5, add to
previous */
            LIN_S_8Bit_count += CR010_val;   /* now has 6 bit times */
            LIN_S_INTTM_count = 4;
            break;
      case 4:
            /* fifth edge, start of stop bit, CR010 has time for bits 6 and 7, add to
previous */
            LIN_S_8Bit_count += CR010_val;   /* now has 8 bit times */
            /* we now have 8 bit times, can calculate baud rate */
            /* and we are in the stop bit, so ok to reenable RXD */

            /* stop the timer */
            TM00_Stop();
            ISC &= 0xFD; /* clear ISC.1 to switch TI000 back to pin P00/TI000 */

            /* to convert our count to new BRGC6 value, divide by 16 by shifting four bits
*/
            /* to round, add 8 before shift (equivalent of 1/2 LSB) */
            CR010_val = (LIN_S_8Bit_count + 8) >> 4;

            /* stop RX and TX to change BRGC6 */
            ASIM6 &= 0x9F;            /* clear bits 6 (TXE6) and 5 (RXE6) */
            BRGC6 = (UCHAR) CR010_val;
            ASIM6 |= 0x60;            /* set bits 6 and 5 to restart TX/RX */

            LIN_S_Rx.state = LIN_STATE_ID;   /* now waiting for ID field */
            /* see if we get SRIF6 */
            while (SRIF6 == 0)
                  ;
            /* clear the SYNC from receiver */
            SRIF6 = 0;
            tmp = ASIS6;
            tmp = RXB6;
            /* re-enable receive interrupt */
            SRMK6 = 0;
            break;
      }
#else  /* LIN_S_CALC_BAUDRATE == TRUE is false */
      /* just return to timer ISR */
#endif /* LIN_S_CALC_BAUDRATE */
}
```

7.2.3.4 LIN Slave 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/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
```

7.2.3.5 LIN Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#ifndef      _MDSYSTEM_
#define      _MDSYSTEM_
/*
************************************************************************
** MacroDefine
************************************************************************
*/
#define      CG_X1STAB_SEL 0x5
#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
```

7.2.3.6 LIN Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
```

```
/*
************************************************************************************
** Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "port.h"
#include "timer.h"
#include "serial.h"
/*
************************************************************************************
** MacroDefine
************************************************************************************
*/

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     Init every Macro
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void SystemInit( void )
{
       /* Clock generator initiate */
       Clock_Init();
       /* Port initiate */
       PORT_Init();
       /* UART6 initiate */
       UART6_Init();
       /* TM00 initiate */
       TM00_Init();
       /* TM50 initiate */
       TM50_Init();
       /* TM51 initiate */
       TM51_Init();
}

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

```
      SystemInit( );
      EI( );
}
```

7.2.3.7 LIN Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/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;
}
```

7.2.3.8 LIN Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#ifndef _MDSERIAL_
#define _MDSERIAL_
/*
************************************************************************
** Global variables
************************************************************************
*/
#define      UART_BAUDRATE_M6    0x0
#define      UART_BAUDRATE_K6    0xd0

void UART6_Init( void );
MD_STATUS UART6_SendData( UCHAR* txbuf, UCHAR txnum );
MD_STATUS UART6_ReceiveData( UCHAR* rxbuf, UCHAR rxnum );
__interrupt void MD_INTSR6( void );
__interrupt void MD_INTST6( void );

enum TransferMode { Send, Receive };
```

```
#endif
```

7.2.3.9 LIN Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/

#pragma      interrupt    INTSR6       MD_INTSR6
#pragma      interrupt    INTST6       MD_INTST6

/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"

/* add includes for LIN */
#include "LIN.h"
#include "LIN_S.h"

/* uart6 transmission */
UCHAR* UART6_TX_ADDRESS;                /* uart transfer buffer address */
UCHAR UART6_TX_CNT;                     /* uart transfer number */

/* uart6 reception */
UCHAR* UART6_RX_ADDRESS;                /* uart receive buffer address */
UCHAR UART6_RX_CNT;                     /* uart receive data number */

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function initializes UART6 module.
```

```
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void UART6_Init( void )
{
       ASIM6 = 0;

       SetIORBit(P1, 0x08);                    /* port setting in transmit/receive mode */
       ClrIORBit(PM1, 0x08);

       SetIORBit(ASIM6, 0x04);                 /* data length 8 bits */
       ClrIORBit(ASIM6, 0x02);                 /* stop length 1 bits */
       SetIORBit(ASIM6, 0x01);
       ClrIORBit(ASIM6, 0x18);                 /* none parity mode */
       SetIORBit(ASICL6, 0x02);          /* LSB first transfer */
       ClrIORBit(ASICL6, 0x01);          /* normal output of TxD6 */
       CKSR6 = UART_BAUDRATE_M6;         /* baudrate selection */
       BRGC6 = UART_BAUDRATE_K6;
       ClrIORBit(IF0H, 0x02);
       ClrIORBit(MK0H, 0x02);                  /* transfer end interrupt enable */
       ClrIORBit(IF0H, 0x01);
       ClrIORBit(MK0H, 0x01);                  /* receive end interrupt enable */
       SetIORBit(PR0H, 0x01);                  /* interrupt low */
       SetIORBit(PR0H, 0x02);                  /* interrupt low */

       SetIORBit(ASIM6, 0x80);
       SetIORBit(ASIM6, 0x60);                 /* transmit/receive mode */
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for UART6 data receiving.
**
** Parameters:
**     rxbuf: receive buffer pointer
**     rxnum: buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS UART6_ReceiveData( UCHAR* rxbuf, UCHAR rxnum )
{
       if( rxnum < 1 ){
       return MD_ERROR;
 }

       UART6_RX_CNT = rxnum;
       UART6_RX_ADDRESS = rxbuf;

       return MD_OK;
}
```

```
/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for UART6 data transfering.
**
** Parameters:
**     txbuf: transfer buffer pointer
**     txnum: buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS UART6_SendData( UCHAR* txbuf, UCHAR txnum )
{
       if( txnum < 1 ){
              return MD_ERROR;    /* 1 frame data at least */
       }

       UART6_TX_ADDRESS = txbuf;
       UART6_TX_CNT = txnum;
       if( txnum == 1) {          /* only 1 frame data to transfer doesn't need contiously
transfer */
              if((ASIF6 & 0x02) == 0){
                     TXB6 = *UART6_TX_ADDRESS++;
              }
              else{
                     return MD_DATAEXISTS;
              }
       }
       else {
              if((ASIF6 & 0x02) == 0){
                     TXB6 = *UART6_TX_ADDRESS++;
                     while((ASIF6 & 0x02) == 1);             /* wait */
                     UART6_TX_CNT--;
                     TXB6 = *UART6_TX_ADDRESS++;
              }
              else{
                     return MD_DATAEXISTS;
              }
       }
       return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is the UART6 receive end interrupt handler.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTSR6( void )
{
       UCHAR err_type;
```

```
	UCHAR tmp;

	/* intercept receive interrupt for LIN if active */
	if (LIN_S_Active) {
		LIN_S_Rx_Handler();
		return;
	}

	err_type = ASIS6;
	tmp = RXB6;

	if( err_type != 0 ){
		return;
	}

	if( UART6_RX_CNT == 0 ){
		/* receive data end */
		return;
	}

	/* the interrut generated by receive end */
	*UART6_RX_ADDRESS ++= tmp;
	UART6_RX_CNT--;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is the UART6 transfer end interrupt handler.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTST6( void )
{
	/* intercept transmit interrupt for LIN if active */
	if (LIN_S_Active) {
		LIN_S_Tx_Handler();
		return;
	}

	if( UART6_TX_CNT == 0 ){
		/* send data complete */
		return;
	}

	if(((ASIF6 & 0x02) == 0) && ((ASIM6 & 0x80) == 0x80)){
		UART6_TX_CNT--;
		if( UART6_TX_CNT != 0 ){
			TXB6 = *UART6_TX_ADDRESS++;
		}
	}
}
```

7.2.3.10 LIN Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"
```

7.2.3.11 LIN Slave 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
*******************************************************************************
*/

#ifndef _MDTIMER_
#define _MDTIMER_
enum TMChannel { TMChannel0, TMChannel1 };
/*
*******************************************************************************
** 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 0x5
#define      TM_TM50_INTERVALVALUE      0x7c
#define      TM_TM50_SQUAREWIDTH 0x7c
#define      TM_TM50_PWMACTIVEVALUE     0x7c
#define      TM_TM51_CLOCK 0x2
#define      TM_TM51_INTERVALVALUE      0xcf
#define      TM_TM51_SQUAREWIDTH 0xcf
#define      TM_TM51_PWMACTIVEVALUE     0xcf
#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 TM00_Init(void);
void TM50_Init(void);
void TM51_Init(void);

/*timer start*/
void TM00_Start(void);
void TM50_Start(void);
void TM51_Start(void);
```

```
/*timer stop*/
void TM00_Stop(void);
void TM50_Stop(void);
void TM51_Stop(void);
MD_STATUS TM50_ChangeTimerCondition(UCHAR value);
MD_STATUS TM51_ChangeTimerCondition(UCHAR value);
MD_STATUS TM00_GetPulseWidth( ULONG *activewidth, ULONG *inactivewidth, enum TMChannel
channel );
__interrupt void MD_INTTM010(void);
__interrupt void MD_INTTM50(void);

#endif       /* _MDTIMER_*/
```

7.2.3.12 LIN Slave 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
************************************************************************
*/

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


/*
************************************************************************
** MacroDefine
************************************************************************
*/
/*TM00 pulse width measure*/
extern ULONG tm00_pos_width;
```

```
extern ULONG tm00_neg_width;

/*TM01 pulse width measure*/

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function initializes TM00_module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM00_Init( )
{
       TMC00=0x00;
                          /* internal count clock */
       PRM00 |= TM_TM00_CLOCK;
       /* TM00 pulse measurement funciton */
       SetIORBit(PM0,0x1);              /* TI000(P00) input */
#if 1
       /* this is correction to Applilet code; ISC bit for TI000 control is bit 1, not bit
zero */
       ClrIORBit(ISC,0x2);
#else
       /* original Applilet code, clears wrong bit */
       ClrIORBit(ISC,0x1);
#endif
       SetIORBit(PRM00, 0x10);                /* specifies raising edge of TI000 */
       CRC00 = 0x07;
       ClrIORBit(IF0H,0x80);
}
/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function starts the TM00 counter.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM00_Start()
{
       ClrIORBit(MK0H,0x80);
       TMC00 = 0x8;                     /* pulse width measure start */
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function stops the TM00 counter and clear the count register.
**
```

```
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM00_Stop()
{
       TMC00=0x0;
       SetIORBit(MK0H,0x80);
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is TM00 pulse width calculation.
**
** Parameters:
**     ULONG *activerwidth :            activewidth clock number
**     ULONG *inactivewidth :           inactivewidth clock nubmer
**     enum TMChannel channel :   channel to get pulse width
**
** Returns:
**     MD_ARGERROR
**     MD_OK
**
**-------------------------------------------------------------------------
*/
MD_STATUS TM00_GetPulseWidth(ULONG *activewidth, ULONG *inactivewidth,enum TMChannel
channel)
{
       if(channel == TMChannel0){
              *inactivewidth = tm00_neg_width;
              *activewidth = tm00_pos_width;
       }
       else{
              return MD_ARGERROR;
       }
       return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function can initialize TM50_module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM50_Init( void )
{
       ClrIORBit(TMC50, 0x80);
       TCL50 = TM_TM50_CLOCK;                  /* countclock=fx/64 */
       SetIORBit(PR0H, 0x20);                  /* low priority level */
       ClrIORBit(IF0H, 0x20);
```

```
	/* 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);
	ClrIORBit(MK0H, 0x20);                  /* INTTM50 enable */
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function can stop the TM50 counter and clear the count register.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void TM50_Stop( void )
{
	ClrIORBit(TMC50, 0x80);
	SetIORBit(MK0H, 0x20);                  /* INTTM50 disable */
}

/*
**-----------------------------------------------------------------------------
**
** 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 */
       /* 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;
}
```

7.2.3.13 LIN Slave 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
*******************************************************************************
*/

#pragma sfr
#pragma interrupt INTTM010 MD_INTTM010
#pragma interrupt INTTM50 MD_INTTM50
/*
*******************************************************************************
** Include files
*******************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"
```

```
#include "sw_0537.h"       /* for sw_isr() definition */
#include "LIN.h"           /* to interface with LIN timer management */
#include "LIN_S.h"


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

/* Timer00, Timer01 pulse width measure */
ULONG tm00_pos_width=0;
ULONG tm00_neg_width=0;
/*
**-------------------------------------------------------------------------
**
** Abstract:
**     TM00 INTTM010 interrupt-service routine.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTTM010( )
{
 if (LIN_S_Active) {
       LIN_S_INTTM_isr();  /* replace with LIN ISR handling */
 } else {
       /* original Applilet code */
       ULONG tm00_pul_cur_0 = 0;
       ULONG tm00_pul_cur_1 = 0;
       UCHAR stat = TMC00;
       ClrIORBit(TMC00, 0x1);                  /* OVF00 clear */
       tm00_pul_cur_0 = (ULONG)CR010;
       tm00_pul_cur_1 = (ULONG)CR000;
       if(stat & 0x1){
              if(tm00_pul_cur_0 > tm00_pul_cur_1){
                     tm00_pos_width = tm00_pul_cur_1;
                     tm00_neg_width = tm00_pul_cur_0 + 0x10000 - tm00_pul_cur_1;
              }
              else{
                     tm00_pos_width = tm00_pul_cur_1;
                     tm00_neg_width = tm00_pul_cur_0 + 0x10000 - tm00_pul_cur_1;
              }
       }
       else{
              tm00_pos_width = tm00_pul_cur_1;
              tm00_neg_width = tm00_pul_cur_0 - tm00_pul_cur_1;
       }
       /*TODO*/
 } /* end of if-else (LIN_S_Active) */
}
/*
**-------------------------------------------------------------------------
**
** Abstract:
**     TM50 INTTM50 interrupt-service routine
**
```

```
** Parameters:
**     None
**
** Returns:
**     None
**
**---------------------------------------------------------------------------
*/
__interrupt void MD_INTTM50( )
{
       sw_isr();    /* call function to check switch input */
}
```

7.3 Listings for CSI Demonstration Programs

7.3.1 Listings for CSI Master Program

This demonstration program runs on the M-78F0537 plus M-Station II platform. The files listed in this section are unique to this demonstration program. The non-unique files are listed in the M-78F0537 common files section below.

7.3.1.1 CSI Master Main.c

```
/* main.c for NEC Electronics Serial Application Note, CSI Master */
/*
************************************************************************
**
** 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "port.h"
#include "timer.h"
#include "serial.h"
```

```
/* add includes for non-Applilet functions */
#include "led_0537.h"
#include "sw_0537.h"
/*
************************************************************************
** MacroDefine
************************************************************************
*/
/*
**-----------------------------------------------------------------------
**
** Abstract:
**     main function
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void main( void )
{
UCHAR csi_tbuf[1];
UCHAR sw_val;
UCHAR count;

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;
       /* TODO. add user code */

       led_init();         /* initialize LEDs */
       sw_init();          /* initialize switch variables */
       sw_set_debounce(10);       /* set debounce counter to 10 for 10 msec stable time */


       TM00_Start();/* start timer for switch debouncing */

       count = 0;

       while(1){
              /* display count in LED display */
              led_dig(count);

              /* wait for a switch to be down */
              while (SW_LU_RU == (sw_val = sw_get()))
                     ;

              if (sw_val == SW_LU_RD) {
                     /* right (SW3) down, increment count, update display */
                     count = count + 1;
                     led_dig(count);
              }
              if (sw_val == SW_LD_RU) {
                     /* left (SW2) down, transmit cound via CSI if not busy */
                     if (CSI11_SendDone) {
                            CSI11_SendDone = FALSE;
                            csi_tbuf[0] = count;
                            CSI11_SendData(csi_tbuf, 1);
                     }
              }
```

```
            /* wait for switches to be up */
            while (SW_LU_RU != sw_get())
                  ;
      } /* end of while (1) loop */
}
```

7.3.1.2 CSI Master 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/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
```

7.3.1.3 CSI Master 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#ifndef      _MDSYSTEM_
#define      _MDSYSTEM_
/*
************************************************************************
** MacroDefine
************************************************************************
*/
#define      CG_X1STAB_SEL 0x5
#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
```

7.3.1.4 CSI Master 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
*****************************************************************************
*/
/*
*****************************************************************************
** Include files
*****************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "port.h"
#include "timer.h"
#include "serial.h"
/*
*****************************************************************************
** MacroDefine
*****************************************************************************
*/

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     Init every Macro
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void SystemInit( void )
{
      /* Clock generator initiate */
      Clock_Init();
      /* Port initiate */
      PORT_Init();
      /* CSI11 initiate */
      CSI11_Init();
      /* TM00 initiate */
      TM00_Init();
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     Init hardware setting
**
** Parameters:
```

```
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void hdwinit( void )
{
       DI( );
       SystemInit( );
       EI( );
}
```

7.3.1.5 CSI Master 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/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 */
       ClrIORBit(OSCCTL, 0x10);
       SetIORBit(MOC, 0x80);                    /* stop X1 clock */
       PCC = CG_CPU_CLOCKSEL;
}
```

7.3.1.6 CSI Master 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 :   NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
*******************************************************************************
*/
#ifndef _MDSERIAL_
#define _MDSERIAL_
/*
*******************************************************************************
** Global variables
*******************************************************************************
*/
void CSI11_Init( void );
MD_STATUS CSI11_SendData( UCHAR *txbuf, UCHAR txnum );
MD_STATUS CSI11_ReceiveData( UCHAR *rxbuf, UCHAR rxnum );
```

```
__interrupt void MD_INTCSI11( void );
void CALL_CSI11_Send( void );

enum TransferMode { Send, Receive };

/* added flag for CSI11_SendDone */
extern UCHAR CSI11_SendDone;

#endif
```

7.3.1.7 CSI Master 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/

#pragma      interrupt    INTCSI11     MD_INTCSI11

/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"
/* csi11 transmission */
UCHAR* CSI11_TX_ADDRESS;               /* csi11 transfer buffer address */
UCHAR CSI11_TX_LEN;                    /* csi11 transfer count number */
UCHAR CSI11_TX_CNT;

/* csi11 reception */
UCHAR* CSI11_RX_ADDRESS;               /* csi11 receive buffer head */
UCHAR  CSI11_RX_LEN;                   /* csi11 recive data count */
UCHAR CSI11_RX_CNT;

/*
```

```
**------------------------------------------------------------------------------
**
** Abstract:
**     This function initializes CSI11 module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**------------------------------------------------------------------------------
*/
void CSI11_Init( void )
{
       ClrIORBit(P0, 0x04);             /* port setting in transmit/receive mode */
       ClrIORBit(PM0, 0x04);
       ClrIORBit(CSIM11, 0x10);   /* data direction MSB */
       ClrIORBit(CSIM11, 0x20);

       SetIORBit(CSIC11, 0x10);   /* clock data phase3 */
       SetIORBit(P0, 0x10);
       ClrIORBit(PM0, 0x10);
       SetIORBit(CSIC11, 0x02);         /* fprs/8 */

       ClrIORBit(IF1H, 0x02);
       ClrIORBit(MK1H, 0x02);           /* transfer end interrupt enable */
       SetIORBit(PR1H, 0x02);           /* interrupt low */
       SetIORBit(CSIM11, 0x40);   /* transmit/receive mode */

       SetIORBit(CSIM11, 0x80);
}

/*
**------------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for CSI11 data transfering.
**
** Parameters:
**     txbuf:       transfer buffer pointer
**     txnum:       buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR
**
**------------------------------------------------------------------------------
*/
MD_STATUS CSI11_SendData( UCHAR* txbuf ,UCHAR txnum )
{
       if( txnum < 1 ){
              return MD_ERROR;
       }

       /* init CSI10 send parameter */
       CSI11_TX_LEN = txnum;                    /* send data length */
       CSI11_TX_CNT = 0;                /* send data count */
       CSI11_TX_ADDRESS = txbuf;        /* send buffer pointer */

       SOTB11 = *CSI11_TX_ADDRESS++;    /* started by writing txbuf[0] to SOTBn */

       return MD_OK;
```

```
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for CSI11 data receiving.
**
** Parameters:
**     rxbuf:        receive buffer pointer
**     rxnum:        buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS CSI11_ReceiveData( UCHAR* rxbuf, UCHAR rxnum )
{
       UCHAR tmp;

       if( rxnum < 1 ){
              return MD_ERROR;
       }

       /* init CSI11 receive parameter */
       CSI11_RX_LEN = rxnum;                    /* receive data length */
       CSI11_RX_CNT = 0;
       CSI11_RX_ADDRESS = rxbuf;         /* receive buffer pointer */

       tmp = SIO11;

       return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is the CSI11 transfer end interrupt handler.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTCSI11( void )
{
       /* send interrupt */
       if( CSI11_TX_CNT < CSI11_TX_LEN ){
              if( CSI11_TX_CNT == (CSI11_TX_LEN-1)){
                     CALL_CSI11_Send();         /* send finish */
              }
              else {
                     CSI11_TX_CNT++;
                     SOTB11 = *CSI11_TX_ADDRESS++;
              }
       }

       /* receive interrupt */
```

```
        if( CSI11_RX_CNT < CSI11_RX_LEN ){
                *(UCHAR*)CSI11_RX_ADDRESS++ = SIO11;
                CSI11_RX_CNT++;
                if( CSI11_RX_CNT == CSI11_RX_LEN ){
                        /*receive complete */
                        return;
                }
        }
}
```

7.3.1.8 CSI Master 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 :   NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"

/* added global flag for CSI11_SendDone */
UCHAR CSI11_SendDone = TRUE;

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     This function is a call back function to deal with data process after
**     some frame(s) data transfering of CSI11 interface.
**
** Parameters:
**     None
```

```
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void CALL_CSI11_Send( void )
{
       /* Set flag to indicate to main program that transmit is done */
       CSI11_SendDone = TRUE;
}
```

7.3.1.9 CSI Master 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
*******************************************************************************
*/

#ifndef _MDTIMER_
#define _MDTIMER_
/*
*******************************************************************************
** MacroDefine
*******************************************************************************
*/
#define      REGVALUE_MAX  0xff
#define      TM_TM00_CLOCK 0x0
#define      TM_TM00_INTERVALVALUE      0x1f3f
#define      TM_TM00_SQUAREWIDTH 0x1f3f
#define      TM_TM00_PPGCYCLE    0x1f3f
#define      TM_TM00_PPGWIDTH    0x00
#define      TM_TM00_ONESHOTCYCLE       0x1f3f
#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 0x2
#define       TM_TM50_INTERVALVALUE      0x00
#define       TM_TM50_SQUAREWIDTH 0x00
#define       TM_TM50_PWMACTIVEVALUE     0x00
#define       TM_TM51_CLOCK 0x2
#define       TM_TM51_INTERVALVALUE      0x00
#define       TM_TM51_SQUAREWIDTH 0x00
#define       TM_TM51_PWMACTIVEVALUE     0x00
#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 TM00_Init(void);

/*timer start*/
void TM00_Start(void);

/*timer stop*/
void TM00_Stop(void);
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
__interrupt void MD_INTTM000(void);

#endif       /* _MDTIMER_*/
```

7.3.1.10 CSI Master 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
*******************************************************************************
*/

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


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

/*TM01 pulse width measure*/

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function initializes TM00_module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void TM00_Init( )
{
       TMC00=0x00;
                          /* internal count clock */
       PRM00 |= TM_TM00_CLOCK;
       SetIORBit(PR0H, 0x40);                  /* low priority level */
       ClrIORBit(IF0H, 0x40);
       /* TM00 interval */
       ClrIORBit(CRC00,0x01);
       CR000 = TM_TM00_INTERVALVALUE;
}
/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function starts the TM00 counter.
**
** Parameters:
**     None
**
** Returns:
**     None
```

```
**
**-------------------------------------------------------------------------
*/
void TM00_Start()
{
      TMC00 = 0x0c;                    /* interval timer start */
      ClrIORBit(MK0H, 0x40);                 /* INTTM000 enable */
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function stops the TM00 counter and clear the count register.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM00_Stop()
{
      TMC00=0x0;
      SetIORBit(MK0H, 0x40);                  /* INTTM000 stop */
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function changes TM00 condition.
**
** Parameters:
**     USHORT* :     array_reg
**     USHORT :      array_num
** Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num)
{
 switch (array_num){
 case 2:
 CR010=*(array_reg + 1);
 case 1:
 CR000=*(array_reg + 0);
 break;
 default:
 return MD_ERROR;
 }
       return MD_OK;
}
```

7.3.1.11 CSI Master 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
************************************************************************
*/

#pragma sfr
#pragma interrupt INTTM000 MD_INTTM000
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"

/* add include for switch routines */
#include "sw_0537.h"

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

/* Timer00, Timer01 pulse width measure */
/*
**-----------------------------------------------------------------------
**
** Abstract:
**     TM00 INTTM000 interrupt-service routine
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
__interrupt void MD_INTTM000( )
{
```

```
        /* call switch ISR routine for debouncing switches */
        sw_isr();
}
```

7.3.2 Listings for CSI Slave Program

This demonstration program runs on the DemoKit-LG2 platform. The files listed in this section are unique to this demonstration program. The non-unique files are listed in the DemoKit-LG2 common files section below.

7.3.2.1 CSI Slave Main.c

```
/* main.c for NEC Electronics Serial Application Note, CSI Slave */
/*
************************************************************************
**
** 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "int.h"
#include "timer.h"
#include "serial.h"
/*
************************************************************************
** MacroDefine
************************************************************************
*/
```

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

//-----------------------------------------------------------------------------
// Global string constants
//-----------------------------------------------------------------------------
const unsigned char *s_WAITING = "WAITING ";

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     main function
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/

void main( void )
{
UCHAR csi_buf[3];
UCHAR csi_len = 1;
UCHAR msg_buf[10];

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;
       /* TODO. add user code */

       /* initialize IIC */
       IIC0_Init();

       /* initialize LCD */
       LCD_Init();

       sw3_in = 0;
       LCD_string(s_WAITING, 0);

       /* set up first receive operation */
       CSI10_ReceiveData(csi_buf, csi_len);

       while(1){
              if (CSI10_ReceiveDone) {
                     sprintf((char *)msg_buf, "RX %02X", csi_buf[0]);
                     LCD_string(msg_buf, 0);
                     CSI10_ReceiveDone = FALSE;
                     CSI10_ReceiveData(csi_buf, csi_len);
              }
              if (sw3_in != 0) {
                     LCD_string(s_WAITING, 0);
                     sw3_in = 0;
              }
       }
}
```

7.3.2.2 CSI Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/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();
	/* CSI10 initiate */
	CSI10_Init();
	/* TM50 initiate */
	TM50_Init();
	/* TM51 initiate */
	TM51_Init();
}

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

7.3.2.3 CSI Slave 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 :   NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC Electronics/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
```

7.3.2.4 CSI Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/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 */
}
```

7.3.2.5 CSI Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/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;
}
```

7.3.2.6 CSI Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#ifndef _MDSERIAL_
#define _MDSERIAL_
/*
************************************************************************
** Global variables
************************************************************************
*/
#define      IIC0_SLAVEADDRESS   0x0

void CSI10_Init( void );
MD_STATUS CSI10_ReceiveData( UCHAR* rxbuf, UCHAR rxnum );
__interrupt void MD_INTCSI10( void );
void CALL_CSI10_Receive( void );

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);

/* added status flag for CSI10 Receive Done */
extern UCHAR CSI10_ReceiveDone;

#endif
```

7.3.2.7 CSI Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/

#pragma      interrupt    INTCSI10     MD_INTCSI10

#pragma interrupt   INTIIC0            MD_INTIIC0

/*
```

```
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"
/* csi10 transmission */
UCHAR* CSI10_TX_ADDRESS;                /* csi10 transfer buffer address */
UCHAR CSI10_TX_LEN;                     /* csi10 transfer count number */
UCHAR CSI10_TX_CNT;

/* csi10 reception */
UCHAR* CSI10_RX_ADDRESS;                /* csi10 receive buffer address */
UCHAR  CSI10_RX_LEN;                    /* csi10 recive data count */
UCHAR CSI10_RX_CNT;
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 CSI10 module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void CSI10_Init( void )
{
       SetIORBit(PM1, 0x02);            /* port setting in receive mode */
       ClrIORBit(CSIM10, 0x10);         /* data direction MSB */

       SetIORBit(CSIC10, 0x10);         /* clock data phase3 */
       SetIORBit(CSIC10, 0x07);         /* external clock selected */
       SetIORBit(PM1, 0x01);

       ClrIORBit(IF0H, 0x04);
       ClrIORBit(MK0H, 0x04);                 /* transfer end interrupt enable */
       SetIORBit(PR0H, 0x04);                 /* interrupt low */
       ClrIORBit(CSIM10, 0x40);         /* receive mode */

       SetIORBit(CSIM10, 0x80);
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for UART6 data receiving.
**
```

```
** Parameters:
**     rxbuf: receive buffer pointer
**     rxnum: buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR
**
**---------------------------------------------------------------------------
*/
MD_STATUS CSI10_ReceiveData( UCHAR* rxbuf, UCHAR rxnum )
{
       UCHAR tmp;

       if( rxnum < 1 ){
              return MD_ERROR;
       }

       /* init CSI10 receive parameter */
       CSI10_RX_LEN = rxnum;             /* receive data count */
       CSI10_RX_CNT = 0;          /* receive data count */
       CSI10_RX_ADDRESS = rxbuf;  /* receive buffer pointer and head */

       tmp = SIO10;

       return MD_OK;
}

/*
**---------------------------------------------------------------------------
**
** Abstract:
**     This function is the CSI10 interrupt handler.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**---------------------------------------------------------------------------
*/
__interrupt void MD_INTCSI10( void )
{
       /* send interrupt */
       if( CSI10_TX_CNT < CSI10_TX_LEN ){
              if( CSI10_TX_CNT == (CSI10_TX_LEN-1) ){
              }
              else {
                     SOTB10 = *CSI10_TX_ADDRESS++;
                     CSI10_TX_CNT++;
              }
       }

       /* receive interrupt */
       if( CSI10_RX_CNT < CSI10_RX_LEN ){
              *(UCHAR*)CSI10_RX_ADDRESS ++= SIO10;
              CSI10_RX_CNT++;

#if 0  /* this was generated by Applilet, incorrect */
              if( CSI10_RX_CNT == CSI10_TX_LEN ){
#else  /* this is correct */
              if( CSI10_RX_CNT == CSI10_RX_LEN ){
#endif
```

```
                    /* receive complete */
                    CALL_CSI10_Receive();
                }
        }
}

/*
**-------------------------------------------------------------------------
**
** 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;
}
```

7.3.2.8 CSI Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/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;

UCHAR CSI10_ReceiveDone = FALSE;

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is a call back function to deal with data process after
**     some frame(s) data receiving of CSI10 interface.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void CALL_CSI10_Receive( void )
{
      /* set flag so main program knows data received */
      CSI10_ReceiveDone = TRUE;
}

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

7.4 Listings for IIC Demonstration Programs

7.4.1 Listings for IIC Master Program

This demonstration program runs on the DemoKit-KA1+ platform. The files listed in this section are the complete set of files for this demonstration program.

7.4.1.1 IIC Master Main.c

```
/* main.c for NEC Electronics Serial Application Note, IIC Master */
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0S/KB1+, 78K0S/KA1+,
** 78K0S/KY1+ 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002-2004
** 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 :  NEC Electronics78K0SKX1H.lib V1.46 [2 Mar 2005]
**
** Device :  uPD78F9222
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "int.h"
#include "port.h"

/* add include for psuedo-IIC functions */
#include "iic_dkka1.h"
```

```
/*
*****************************************************************************
** MacroDefine
*****************************************************************************
*/

/*#define LED1 P2.3 */      /* LED D1 */
#define LED2 P13.0   /* LED D2 */
#define LED3 P4.5 /* LED D3 */
#define LED4 P12.3   /* LED D4 /*

/*
*****************************************************************************
** Function Prototypes
*****************************************************************************
*/
void LED1(unsigned char b0);
void init_LED (void);
void drive_LED(unsigned char value);

/*
*****************************************************************************
** Global variables
*****************************************************************************
*/

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function implements main function.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void main( void )
{
unsigned char i=0;
unsigned char count = 0xFF;
unsigned char nack;
unsigned char rdval;
unsigned char swval;

      init_LED();               /* LED port initialization */
      iic_init();               /* IIC port pin initialization */

      while(1){
 if(IntP0Flag)      /* Key1 pressed? */
 {
 IntP0Flag=0; /* Reset status flag Key1 */
 switch (i) {
 case 0:
      /* show low nibble of count */
      drive_LED(count & 0xF);
      break;
 case 1:
      /* show result of writing */
```

```
      nack = iic_dkwr(count);    /* write count to DemoKit-LG2 */
      if (nack == 0) {
            drive_LED(0x00);     /* ok, blank LED */
      } else {
            drive_LED(0x08);     /* error, LED1 on */
      }
      count = count - 0x11;      /* move from FF to EE, DD, etc. */
      break;
 case 2:
      /* read last key from DemoKit-LG2, show value */
      rdval = iic_dkrd(&nack);   /* read byte from DemoKit-LG2 */
      if (nack == 0) {
            switch (rdval) {
             case 0:      swval = 0;    break; /* none down */
             case 1:      swval = 1;    break; /* UP */
             case 2:      swval = 2;    break; /* DOWN */
             case 4:      swval = 3;    break; /* RIGHT */
             case 8:      swval = 4;    break; /* LEFT */
             case 16:     swval = 5;    break; /* SELECT */
             default:     swval = 7;    break; /* multiple */
            }
            drive_LED(swval);    /* show switch value read */
      } else {
            drive_LED(0x08);     /* LED1 indicates error */
      }
      break;
 case 3:
                           drive_LED(0x00);          /* blank to restart cycle */
                           break;
 default:
       break;
 }
 if(i<3)
 i++;
 else
 i=0;
 } /* endo of if (IntP0Flag) */
      } /* end while (1) */
} /* end of main() */

/* use subroutine to set LED1 on/off, because rd/modify/wr on P2.3 */
/* will change IIC bits of P2 */
void LED1(unsigned char b0)
{
unsigned char tmp;
      tmp = P2 & 0xF9;           /* read port, clear P2.2 and P2.1 */
      if (b0 & 0x01) {
            tmp = tmp | 0x08;    /* set P2.3 bit */
      } else {
            tmp = tmp & 0xF7;    /* clear P2.3 bit */
      }
      P2 = tmp;                        /* write updated P2 value, changed P2.3, cleared
P2.2,P2.1 */
}

/*
**-----------------------------------------------------------------------------
** Module: init_LED
** Function: Initialization of LED ports
**-----------------------------------------------------------------------------
*/
void init_LED (void)
{
      /* note port mode registers initialized in Port_Init() */
```

```
 LED1(0);           /* drive LED1 */
 LED2=0;       /* drive LED2 */
 LED3=0; /* drive LED3 */
 LED4=0;       /* drive LED4 */

 LED1(1);           /* switch LED1 off */
 LED2=1;       /* switch LED2 off */
 LED3=1;       /* switch LED3 off */
 LED4=1;       /* switch LED4 off */
}

/*
**-----------------------------------------------------------------------------
** Module: drive_LED
** Function: Output of 4-bit value to LED port
**-----------------------------------------------------------------------------
*/
void drive_LED(unsigned char value)
{
 LED1(~(value>>3)); /* use function to drive LED1 */
 LED2 = ~(value>>2);
 LED3 = ~(value>>1);
 LED4 = ~(value);
}
```

7.4.1.2 IIC Master Iic_dkka1.h

```
/* iic_dkka1.h      */
/*     header for DemoKit-KA1+ board for IIC communication */

#ifndef _IIC_DKKA1_H
#define _IIC_DKKA1_H

/*****************************************************************/
/* Define definitions */
/*****************************************************************/


/*****************************************************************/
/* Export functions */
/*****************************************************************/
extern void iic_init(void);             /* init IIC operation */

/* DemoKit-LG2 DKLG2_I2 access functions */
unsigned char iic_dkrd(unsigned char *pnack);  /* read byte from DemoKit-LG2 */
unsigned char iic_dkwr(unsigned char idata);   /* write byte to DemoKit-LG2 */

#endif /* _IIC_DKKA1_H */
```

7.4.1.3 IIC Master Iic_dkka1.c

```
/*     iic_dkka1.c - routines for IIC I/O                                    */
/*     for DemoKit-KA1+ board                                                    */

/* 78F9222 does not have built-in IIC support, so          */
/* IIC access done by operations on a parallel port        */
/* P21 is connected to SCL                                                   */
/* P22 is connected to SDA                                                   */
/* SCL and SDA have external pull-up resistors             */
/* P21 and P22 are either inputs (PM2.1/2 = 1)             */
/* or outputs to drive low (PM2.1/2 = 0              */

/* Note: this code is adapted from EV81 monitor source code,             */
/* developed by Tom Spivey, NEC Electronics Electronics America Inc. Sr. FAE    */

/* need pragma declaration to access SFR's in C      */
#pragma sfr
/* need pragma to use NOP() function to insert NOP instruction */
#pragma nop

#include "iic_dkka1.h"

#define      SCL    PM2.1
#define SDA  PM2.2
#define SDA_IN      P2.2
#define SCL_IN P2.1

/* local variables */

/*     void iic_init(void)       */
/*           set up IIC                */
void iic_init(void)
{
unsigned char tmp;

       tmp = PMC2;         /* get current state of PMC2 */
       tmp &= 0xF9;  /* set bits 1 and 2 low to set port mode */
       PMC2 = tmp;         /* and update */

       tmp = PM2;          /* get current state of port mode register */
       tmp |= 0x06;  /* set bits 1 and 2 high to set input mode */
       PM2 = tmp;          /* update PM2 */

       tmp = P2;           /* read current state of P2 */
       tmp &= 0xF9;  /* set bits 1 and 2 low */
       P2 = tmp;           /* set P21 and P22 low */

       /* To output 0, set PM2.1/2 low to enable low output */
       /* To output 1, set PM2.1/2 high to configure as input; external pull-up provides 1
*/
}

/* local routines to set SCL clock high or low, allowing delays to be inserted */
/* for higher CPU clock rates */
/* SCL clock rate should be no more than 400 KHz maximum; no more than 100 KHz if the */
/* VDD voltage is lower than 4.5V. */
/* To insure at most 100 KHz, or 10 us cycle time, we insert delays of 5 us before and */
/* 5 us after clock changes. There is some additional overhead in calling these */
/* subroutines, so we are assured of no greater than 100 KHz clock rate. */
/* For 78F9222 operating at 8 MHz, minimum instruction cycle time is 0.25 us; so */
/* to get a 5 us delay, we use 4 * 5 = 25 NOP instructions. */
```

```
void iic_SCL_0(void)
{
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
       SCL = 0;
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
}

void iic_SCL_1(void)
{
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
       SCL = 1;
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
       NOP(); NOP(); NOP(); NOP(); NOP();
}

/* iic_iack sends an ACKnowlege bit (low true) over the software generated IIC bus */
void iic_iack(void)
{
       iic_SCL_0(); /* sclk = L */
       SDA = 0;            /* data = L */
       iic_SCL_1(); /* sclk = H */
}

/* iic_inack sends a not_ACKnowlege bit (high) over the software generated IIC bus */
void iic_inack(void)
{
       iic_SCL_0(); /* sclk = L */
       SDA = 1;            /* data = H */
       iic_SCL_1(); /* sclk = H */
}

/* iic_istart sends a start condition over the software generated IIC bus */
/*     H -> L transition on SDA while SCL high creates start condition */
void iic_istart(void)
{
       SDA = 1;            /* data = H */
       iic_SCL_1(); /* sclk = H */
       SDA = 0;            /* data = L (start condition) */
}

/* iic_irecv gets the 8-bit data read from the IIC bus. */
/* Data is received MSB first. */
unsigned char iic_irecv(void)
{
unsigned char rcv_val;     /* value received shifted in here */
unsigned char i;           /* counter for loop */

       rcv_val = 0;
       iic_SCL_0(); /* take sclk low now to avoid a stop condition */
       SDA = 1;            /* set data high so EEPROM can pull it low */
       for (i = 0; i < 8; i++) {
```

```
		rcv_val = rcv_val << 1;		/* shift up */
		iic_SCL_0();			/* sclk = L */
		iic_SCL_1();			/* sclk = H */
		if (SDA_IN == 1)		/* check data input */
			rcv_val |= 0x01;	/* set bit if input is high */
	}

	return rcv_val;
}

/* iic_isend sends an 8-bit value over the IIC bus. */
/*     Data is send MSB first */
/* returns zero if no error, 1 if error (NACK) */
unsigned char iic_isend(unsigned char val)
{
unsigned char i;		/* counter for loop */

	for (i = 0; i < 8; i++) {
		iic_SCL_0();  /* sclk = L */
		if ((val & 0x80) == 0)
			SDA = 0;	/* bit is zero, data = L */
		else
			SDA = 1;	/* bit is one, data = H */
		iic_SCL_1();  /* sclk = H */
		val = val << 1;	/* check next bit */
	}

	/* check for slave ACKnowlege */
	iic_SCL_0();		/* sclk = L */
	SDA = 1;			/* data = H (allow slave to pull low) */
	iic_SCL_1();		/* sclk = H */
#if 1
	/* allow wait - wait for SCL to go high, for slave cancel wait */
	while (SCL_IN == 0)
		;
#endif
	if (SDA_IN == 1)	/* error if ACK high (false) */
		return 1;		/* indicate error */
	iic_SCL_0();		/* sclk = L (reset clock to clear slave ACK) */
	return 0;			/* indicate no error */
}

/* iic_istop sends a stop condition over the software generated IIC bus. */
/* l -> H transition on SDA while SCL high creates stop condition. */
void iic_istop(void)
{
	iic_SCL_0();  /* sclk = L */
	SDA = 0;		/* data = L */
	iic_SCL_1();  /* sclk = H */
	SDA = 1;		/* data = H (stop condition) */
}


/* iic_dkrd performs byte read of DemoKit-LG2 running DKLG2_I2 program */
/*     parameter pnack contains pointer to address to store nack (error indication) */
/*     data returned is byte read */
unsigned char iic_dkrd(unsigned char *pnack)
{
unsigned char nack;
unsigned char read_val;

	/* send slave address */

	iic_istart();			/* send start bit */
```

```
	nack = iic_isend(0x21);		/* slave address for READ (b0=1) */

	if (nack == 0) {		/* proceed if no error on read */
		read_val = iic_irecv();		/* read data */
		iic_istop();			/* terminate cycle */
	}

	if (nack != 0) {		/* if we had an error */
		iic_istop();		/* terminate the cycle */
		read_val = 0;		/* return a zero */
	}

	if (pnack != 0) {		/* don't store nack if pointer is NULL */
		*pnack = nack;			/* otherwise store the error status */
	}

	return read_val;		/* return the data */
}

/* iic_dkwr performs byte write to a slave DemoKit-LG2 running DKLG2_I2 program */
/*     parameter idata is the data to write */
/* return value is nack (0 if no error, 1 if error) */
unsigned char iic_dkwr(unsigned char idata)
{
unsigned char nack;	/* error indication */

	/* send slave address */
	iic_istart();		/* send start bit */
	nack = iic_isend(0x20);	/* send slave address for write */

	if (nack == 0) {	/* proceed if no error */
		nack = iic_isend(idata);	/* send data */
	}

	iic_istop();		/* stop, whether error or not */

	return nack;		/* report error or not */
}
```

7.4.1.4 IIC Master Macrodriver.h

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0S/KB1+, 78K0S/KA1+,
** 78K0S/KY1+ 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002-2004
** 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 file is general header file.
** APIlib :  NEC Electronics78K0SKX1H.lib V1.46 [2 Mar 2005]
**
** Device :  uPD78F9222
**
** Compiler : NEC Electronics/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      MD_TRUE             1
#define      MD_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 */

/* main clock and subclock as clock source */
enum ClockMode { MainClock, RingClock };
```

```
/* clear IO register bit and set IO register bit */
#define      ClrIORBit(Reg, ClrBitMap)  Reg &= ~ClrBitMap
#define      SetIORBit(Reg, SetBitMap)  Reg |= SetBitMap

#define      SYSTEMCLOCK  8000000
#define      SUBCLOCK     0
#define      MAINCLOCK    8000000
#define      FXPCLOCK     8000000
#define      FRCLOCK      240000


#endif
```

7.4.1.5 IIC Master System.h

```
/*
*******************************************************************************
**
** This device driver was created by Applilet for the 78K0S/KB1+, 78K0S/KA1+,
** 78K0S/KY1+ 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002-2004
** 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 :  NEC Electronics78K0SKX1H.lib V1.46 [2 Mar 2005]
**
** Device :  uPD78F9222
**
** Compiler : NEC Electronics/CC78K0
**
*******************************************************************************
*/
#ifndef      _MDSYSTEM_
#define      _MDSYSTEM_
/*
*******************************************************************************
** MacroDefine
*******************************************************************************
*/
#define      OPTION_BYTE  0x98
#define      CG_Mainosc   0x8

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

void Clock_Init( void );

#endif
```

7.4.1.6 IIC Master Systeminit.c

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0S/KB1+, 78K0S/KA1+,
** 78K0S/KY1+ 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002-2004
** 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 initial function.
** APIlib :  NEC Electronics78K0SKX1H.lib V1.46 [2 Mar 2005]
**
** Device :  uPD78F9222
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "int.h"
#include "port.h"
/*
************************************************************************
** MacroDefine
************************************************************************
*/

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     This function initializes every Macro.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void SystemInit( void )
{
```

```
	/* Clock generator initiate */
	Clock_Init();
	/* Port initiate */
	PORT_Init();
	/* INT initiate */
	INT_Init();
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function initializes hardware settings.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void hdwinit( void )
{
	DI( );
	SystemInit( );
	EI( );
}
```

7.4.1.7 IIC Master System.c

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0S/KB1+, 78K0S/KA1+,
** 78K0S/KY1+ 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002-2004
** 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 :   NEC Electronics78K0SKX1H.lib V1.46 [2 Mar 2005]
**
** Device :   uPD78F9222
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
/*
```

```
************************************************************************
** Include files
************************************************************************
*/
#pragma section @@CNST OPT AT 80H
#include "macrodriver.h"
#include "system.h"
/*
************************************************************************
** MacroDefine
************************************************************************
*/
const char OPTION = OPTION_BYTE;

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     This function initializes the clock generator.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void Clock_Init( void )
{

       SetIORBit(LSRCM, 0x01);                 /* stop low speed Internal-OSC */
       ClrIORBit(PPCC, 0x03);
       ClrIORBit(PCC, 0x02);
       ClrIORBit(OSTS, 0x03);
}
```

7.4.1.8 IIC Master Int.h

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0S/KB1+, 78K0S/KA1+,
** 78K0S/KY1+ 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002-2004
** 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 :  NEC Electronics78K0SKX1H.lib V1.46 [2 Mar 2005]
**
```

```
** Device :   uPD78F9222
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#ifndef      _MDINT_
#define      _MDINT_
/*
************************************************************************
** MacroDefine
************************************************************************
*/
enum ExternalINT { EX_INTP0, EX_INTP1, EX_INTP2, EX_INTP3 };

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

enum MaskableSource { INT_LVI=1, INT_INTP0, INT_INTP1, INT_TMH1, INT_TM000,
              INT_TM010, INT_AD, INT_INTP2=9, INT_INTP3, INT_TM80,
              INT_SRE6, INT_SR6, INT_ST6 };
void INT_Init( void );
__interrupt void MD_INTP0( void );

/* add definition of flag for INTP0 occurred */
extern unsigned char IntP0Flag ;  /* flag for KEY 1 pressed */

#endif
```

7.4.1.9 IIC Master Int.c

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0S/KB1+, 78K0S/KA1+,
** 78K0S/KY1+ 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002-2004
** 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 :   NEC Electronics78K0SKX1H.lib V1.46 [2 Mar 2005]
**
** Device :   uPD78F9222
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
```

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

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     This function initializes the external INT, including enable or disable,
**     priority setting
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void INT_Init( void )
{
       ClrIORBit(INTM0, 0x0c);
       ClrIORBit(ISC,0x01);
       SetIORBit(PM3, 0x01);                   /* set P30 input mode */
       ClrIORBit(IF0, 0x04);
       ClrIORBit(MK0, 0x04);                   /* enable INTP0 */
}
```

7.4.1.10 IIC Master Int_user.c

```
/*
*************************************************************************
**
** This device driver was created by Applilet for the 78K0S/KB1+, 78K0S/KA1+,
** 78K0S/KY1+ 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002-2004
** 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 :  NEC Electronics78K0SKX1H.lib V1.46 [2 Mar 2005]
**
** Device :  uPD78F9222
**
```

```
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#pragma      interrupt    INTP0  MD_INTP0

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

unsigned char IntP0Flag = 0;     /* flag for KEY 1 pressed */

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     INTP0 Interrupt-service routine.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
__interrupt void MD_INTP0( void )
{
      IntP0Flag=1;         /* Set status flag INTP0 (Key1) */
}
```

7.4.1.11 IIC Master Port.h

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0S/KB1+, 78K0S/KA1+,
** 78K0S/KY1+ 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002-2004
** 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 : port.h
** Abstract : This file implements device driver for port module.
** APIlib :  NEC Electronics78K0SKX1H.lib V1.46 [2 Mar 2005]
**
** Device :  uPD78F9222
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/

#ifndef      _MDPORT_
#define      _MDPORT_
/*
************************************************************************
** MacroDefine
************************************************************************
*/
#define      PORT_PM2     0xf7
#define      PORT_PU2     0x0
#define      PORT_P2      0x0
#define      PORT_PM3     0xff
#define      PORT_PU3     0x0
#define      PORT_P3      0x0
#define      PORT_PM4     0xdf
#define      PORT_PU4     0x0
#define      PORT_P4      0x0
#define      PORT_PM12    0xf7
#define      PORT_PU12    0x0
#define      PORT_P12     0x0
#define      PORT_P13     0x0

void PORT_Init( void );

#endif
```

7.4.1.12 IIC Master Port.c

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0S/KB1+, 78K0S/KA1+,
** 78K0S/KY1+ 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002-2004
** 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 : port.c
** Abstract : This file implements device driver for port module.
** APIlib :  NEC Electronics78K0SKX1H.lib V1.46 [2 Mar 2005]
**
** Device :  uPD78F9222
**
** Compiler : NEC Electronics/CC78K0
**
```

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

/*
************************************************************************************
** Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "port.h"
/*
************************************************************************************
** Constants
************************************************************************************
*/


/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function initializes the I/O port module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void PORT_Init( void )
{
       /* Initialize the port registers */
       P2 = PORT_P2;
       P3 = PORT_P3;
       P4 = PORT_P4;
       P12 = PORT_P12;
       P13 = PORT_P13;

       /* Initialize the pull-up resistor option registers */
       PU2 = PORT_PU2;
       PU3 = PORT_PU3;
       PU4 = PORT_PU4;
       PU12 = PORT_PU12;

       /* Initialize the mode registers */
       PM2 = PORT_PM2;
       PM3 = PORT_PM3;
       PM4 = PORT_PM4;
       PM12 = PORT_PM12;

}
```

7.4.2 Listings for IIC Slave Program

This demonstration program runs on the DemoKit-LG2 platform. The files listed in this section are unique to this demonstration program. The non-unique files are listed in the DemoKit-LG2 common files section below.

7.4.2.1 IIC Slave Main.c

```
/* main.c for NEC Electronics Serial Application Note, IIC Slave */
/*
************************************************************************
**
** 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "int.h"
#include "timer.h"
#include "serial.h"

/* include for sprintf */
#include "stdio.h"

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

/* include files for DemoKit-LG2 LCD */
#include "lcd.h"
#include "defines.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 )
{
int i;
unsigned char buf[10];

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;

       /* initialize IIC */
       IIC0_Init();
       /* set slave address */
       SVA0 = IIC0_SLAVEADDRESS;

       /* initialize LCD */
       LCD_Init();

       LCD_string((UCHAR *)"DKLG2 I2", 0);           /* display initial string */

       IIC0_SlaveStart(IIC0_SLAVEADDRESS);           /* set up slave operations */
       UI_SlaveReceiveEnd = MD_ERROR;                /* clear flag */
       UI_SlaveError = MD_OK;                              /* clear error */

       while(1){
              /* check for data received from Master */
              if (UI_SlaveReceiveEnd == MD_OK) {
                     /* show data in LCD */
                     sprintf((char *)buf, "RCV %02X", iic0_rxbuf[0]);
                     LCD_string(buf, 0);
                     /* set up slave operation again */
                     UI_SlaveReceiveEnd = MD_ERROR;
                     UI_SlaveError = MD_OK;
                     IIC0_SlaveStart(IIC0_SLAVEADDRESS);
              }
              /* check for IIC error */
              if (UI_SlaveError != MD_OK) {
                     /* show slave error code in LCD */
                     sprintf((char *)buf, "SE %04X", UI_SlaveError);
                     LCD_string(buf,0);
                     /* set up slave operation again */
                     UI_SlaveReceiveEnd = MD_ERROR;
```

```
                UI_SlaveError = MD_OK;
                IIC0_SlaveStart(IIC0_SLAVEADDRESS);
            }
            /* check for switch pressed */
            if (sw3_in != 0) {
                /* depending on switch, set string */
                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;    /* clear switch */
            }
        }
}
```

7.4.2.2 IIC Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/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( );
}
```

7.4.2.3 IIC Slave 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 : NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device : uPD78F0537
**
** Compiler :     NEC Electronics/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;
/* global value used to save last sw3 press */
extern __sreg volatile unsigned char sw3_save;

#endif
```

7.4.2.4 IIC Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/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 */
}
```

7.4.2.5 IIC Slave 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 :   NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC Electronics/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;
/* global value used to save last sw3 press */
__sreg volatile unsigned char sw3_save;

/*
**-----------------------------------------------------------------------
**
** 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
 sw3_save = sw3_first;     // save for reading by IIC
       } else {
 sw3_in=0;
 }
}
```

7.4.2.6 IIC Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#ifndef _MDSERIAL_
#define _MDSERIAL_
/*
************************************************************************
** Global variables
************************************************************************
*/
#define      IIC0_SLAVEADDRESS   0x20


enum TransferMode { Send, Receive };
MD_STATUS IIC0_MasterStart( enum TransferMode , UCHAR , UCHAR );
MD_STATUS IIC0_MasterSendData( UCHAR* , UINT );
MD_STATUS IIC0_MasterReceiveData( UCHAR* , UINT );
MD_STATUS IIC0_SlaveStart( UCHAR );
MD_STATUS IIC0_SlaveSendData( UCHAR* , UINT );
MD_STATUS IIC0_SlaveReceiveData( 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_SlaveSendEnd( void );
void CALL_IIC0_MasterSendEnd( void );
void CALL_IIC0_MasterReceiveEnd( void );
void CALL_IIC0_SlaveReceiveEnd( void );
void CALL_IIC0_MasterError( MD_STATUS flag );
void CALL_IIC0_SlaveError( 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;

extern MD_STATUS UI_SlaveError;
extern MD_STATUS UI_SlaveReceiveEnd;

/* added definition for initialize routine */
```

```
void IIC0_Init( void );

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

extern UCHAR iic0_rxbuf[10];     /* buffer to receive data from master */
extern UCHAR iic0_txbuf[10];     /* buffer to transmit data to master */

#endif
```

7.4.2.7 IIC Slave 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
#define FIX_PM6_SET 1      /* move setting of PM6 to after IICE0 set on */

#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 */

#if (FIX_PM6_SET == 1)
#else
       ClrIORBit(PM6, 0x03);                  /* port setting */
#endif
// remove setting of P6 for 78F0397 (78K0/LG2)
// keep for 78F0537
//     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 */
#if (FIX_PM6_SET == 1)
      ClrIORBit(PM6, 0x03);                  /* port setting */
#endif

      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 start IIC0 by slave mode.
**
** Parameters:
```

```
**     UCHAR adr : set address for select slave
**
** Returns:
**     MD_OK
**     MD_ARGERROR : bit0 must be set 0
**
**-------------------------------------------------------------------------
*/
MD_STATUS IIC0_SlaveStart( UCHAR adr )
{
       if( adr & 1 ){
              return MD_ARGERROR; /* bit0 isn't 0 */
       }
       SetIORBit(IICC0, 0x98);          /* start transfer */
#if (FIX_PM6_SET == 1)
       ClrIORBit(PM6, 0x03);                  /* port setting */
#endif
       SVA0 = adr;               /* set slave address */
       iic0_s_sta_flag = 0;

       return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for stop IIC0.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void IIC0_Stop( void )
{
#if (FIX_PM6_SET == 1)
       SetIORBit(PM6, 0x03);                  /* port setting */
#endif
       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:
**    This function is responsible for IIC0 data transfering by slave mode.
**
** Parameters:
**    UCHAR* txbuf :      send buffer pointer
**    USHORT txnum :      buffer size
**
** Returns:
**    MD_OK
**    MD_ERROR :   address incomplete
**
**-------------------------------------------------------------------------
```

```
*/
MD_STATUS IIC0_SlaveSendData(UCHAR* txbuf, UINT txnum)
{
       if( iic0_s_sta_flag == 0 ){
              return MD_ERROR;          /* address incomplete */
       }
       /* set parameter */
       iic0_s_send_size = txnum;
       iic0_s_send_pbuf = txbuf;

       SetIORBit(IICC0, 0x08);                /* start transfer */

       IIC0 = *iic0_s_send_pbuf ++ ;          /* set first data */
       iic0_s_send_size--;

       return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for IIC0 data receiving by slave mode.
**
** Parameters:
**     UCHAR* rxbuf :      receive buffer pointer
**     USHORT rxnum :      buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR :    address incomplete
**
**-------------------------------------------------------------------------
*/
MD_STATUS IIC0_SlaveReceiveData(UCHAR* rxbuf, UINT rxnum)
{
       if( iic0_s_sta_flag == 0 ){
              return MD_ERROR;          /* address incomplete */
       }
       /* set parameter */
       iic0_s_rev_size = rxnum;
       iic0_s_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 ) ){
			CALL_IIC0_SlaveSendEnd();
			return MD_SLAVE_SEND_END;
		} else {
			CALL_IIC0_SlaveError(MD_SPT);
			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{
				CALL_IIC0_SlaveError(MD_ERROR);
				return MD_ERROR;
			}
		}
		else{
			CALL_IIC0_SlaveError( MD_ERROR );
			return MD_ERROR;
		}
	}
```

```
	/* slave send control */
	else if( IICS0 & 0x08 ){
		if(!( IICS0 & 0x04 )){			/* check ACKD0 -> acknowledge */
			CALL_IIC0_SlaveError(MD_NACK);
			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 */
			CALL_IIC0_SlaveReceiveEnd( );
			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;
}
```

7.4.2.8 IIC Slave 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 :   NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
/*
```

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

/* added include to allow user code to get sw3_save */
#include "int.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;

MD_STATUS UI_SlaveError;
MD_STATUS UI_SlaveReceiveEnd;
UCHAR iic0_rxbuf[10];      /* buffer to receive data from master */
UCHAR iic0_txbuf[10];      /* buffer to transmit data to master */
/*
**-----------------------------------------------------------------------------
**
** 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:
**     Slave Error,
**     callback function open for users operation
**
** Parameters:
**     MD_STATUS flag
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_SlaveError( MD_STATUS flag )
{
       /* user operation */
       switch (flag) {
        case (MD_SPT):
              break;        /* don't report Stop status errors */
        case (MD_ERROR):
        case (MD_NACK):
        default:
              UI_SlaveError = flag;
              break;
       }
       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:
**     Slave recevice finish
**     callback function open for users operation
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/

void CALL_IIC0_SlaveReceiveEnd( void )
{
      /* user operation */
      UI_SlaveReceiveEnd = MD_OK;             /* set flag so main program can act on data
*/
      return;
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     Slave send finish,
**     callback function open for users operation
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void CALL_IIC0_SlaveSendEnd( void )
{
      /* user operation */
      /* send is done, or stop condition from master (or local), set up as slave */
      IIC0_SlaveStart(IIC0_SLAVEADDRESS);
      return;
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     IIC0 slave address match
**     callback function for users operation
**
** Parameters:
```

```
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_SlaveAddressMatch( void )
{
       /* user operation */
       if (IICS0 & 0x08) { /* check TRC0 state */
              /* TRC0 == 1, master reads from slave */
              iic0_txbuf[0] = sw3_save;
              IIC0_SlaveSendData(iic0_txbuf, 1);            /* send one byte back */
       } else {
              /* TRC0 == 0, master writes to slave */
              IIC0_SlaveReceiveData(iic0_rxbuf, 1);   /* expect one byte */
       }
}

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

7.5 Common File Listing for Programs on M-78F0537 Platform

The files listed in this section are common to the demonstration programs run on the M-78F0537 micro-board plus M-Station II platform. This section thus includes the following demonstration programs:

- UART demonstration program
- LIN master demonstration program
- LIN slave demonstration program
- CSI master demonstration program

7.5.1 M-78F0537 Common Led_0537.h

```
/* led_0537.h                                                                  */
/*     header for M-78F0537 CPU board for LED digit display */
/* Version 1.1      01-13-2006                                                 */

#ifndef _LED_0537_H
#define _LED_0537_H

/*********************************************************************/
/* Define definitions */
/*********************************************************************/

/* LED Patterns for decimal and hex digits, characters */
/* for individual bits,     ---A---      */
/* 0=on 1=off                    | |    */
/* bit 0 = segment A             F B    */
/* bit 1 = segment B             | |    */
/* bit 2 = segment C              ---G---      */
/* bit 3 = segment D             | |    */
/* bit 4 = segment E             E C    */
/* bit 5 = segment F             | |    */
/* bit 6 = segment G              ---D--- DP   */
/* bit 7 = decimal point                       */

#define LED_PAT_0 0xC0
#define LED_PAT_1 0xF9
#define LED_PAT_2 0xA4
#define LED_PAT_3 0xB0
#define LED_PAT_4 0x99
#define LED_PAT_5 0x92
#define LED_PAT_6 0x82
#define LED_PAT_7 0xF8
#define LED_PAT_8 0x80
#define LED_PAT_9 0x98
#define LED_PAT_A 0x88
#define LED_PAT_B 0x83
#define LED_PAT_C 0xC6
#define LED_PAT_D 0xA1
#define LED_PAT_E 0x86
#define LED_PAT_F 0x8E
#define LED_PAT_BLANK     0xFF
```

```
#define LED_PAT_DP         0x7F
#define LED_PAT_DASH       0xBF
#define LED_PAT_ULINE      0xF7
#define LED_PAT_OLINE      0xFE
#define LED_PAT_EQUAL      0xB7

/*****************************************************************/
/* Export functions */
/*****************************************************************/
extern void led_init(void);                                    /* init ports for LED
output */
extern void led_out_right(unsigned char val);  /* output value to right LED     */
extern void led_out_left(unsigned char val);   /* output value to left LED            */
extern void led_dig_right(unsigned char num);  /* display number in right LED   */
extern void led_dig_left(unsigned char num);   /* display number in left LED    */
extern void led_dig(unsigned char num);              /* display number as hex         */
extern void led_dig_bcd(unsigned char bcdnum); /* display number as BCD         */

#endif /* _LED_0537_H */
```

7.5.2 M-78F0537 Common Led_0537.c

```
/*     led_0537.c - routines for LED display                    */
/*           for M-78F0537 CPU board on M-Station base board    */
/* Version:  1.1    01-13-2006                                  */

/*           P70-P77 = output to right digit (LED2)            */
/*           P40-P43 = output to left digit (LED1) bits 0-3 */
/*           P50-P53 = output to left digit (LED1) bits 4-7 */

/*     To connect ports to LEDs on M-Station 1.1, make the
       following jumper connections between ROW1 and ROW2.
       To connect ports to LEDs on M-Station 2, make sure
       the default SBxx connections are inserted.
       Port   LED          M-Station 1.1 M-Station 2.2
       ----   ---          ------------- -------------
       P70          2-A          R1.25 - R2.25 SB27
       P71          2-B          R1.26 - R2.26 SB28
       P72          2-C          R1.27 - R2.27 SB29
       P73          2-D          R1.28 - R2.28 SB30
       P74          2-E          R1.29 - R2.29 SB31
       P75          2-F          R1.30 - R2.30 SB32
       P76          2-G          R1.31 - R2.31 SB33
       P77          2-DP   R1.32 - R2.32 SB34

       P40          1-A          R1.17 - R2.17 SB35
       P41          1-B          R1.18 - R2.18 SB36
       P42          1-C          R1.19 - R2.19 SB37
       P43          1-D          R1.20 - R2.20 SB38

       P50          1-E          R1.21 - R2.21 SB39
       P51          1-F          R1.22 - R2.22 SB40
       P52          1-G          R1.23 - R2.23 SB41
       P53          1-DP   R1.24 - R2.24 SB42
*/
```

```
/* NOTE: on M-Station Base V1.0 prototype, P40-P53 are      */
/* located at ROW4.1-8, and need to be wirewrapped to*/
/* connect to ROW2.17-24 to drive LED1.                     */

/* need pragma declaration to access SFR's in C      */
#pragma sfr

#include "led_0537.h"

/* table of bit patterns for seven-segment digits */
static unsigned char dig_tab[] = {
 LED_PAT_0,   /* 0 */
 LED_PAT_1,   /* 1 */
 LED_PAT_2,   /* 2 */
 LED_PAT_3,   /* 3 */
 LED_PAT_4,   /* 4 */
 LED_PAT_5,   /* 5 */
 LED_PAT_6,   /* 6 */
 LED_PAT_7,   /* 7 */
 LED_PAT_8,   /* 8 */
 LED_PAT_9,   /* 9 */
 LED_PAT_A,   /* A */
 LED_PAT_B,   /* B */
 LED_PAT_C,   /* C */
 LED_PAT_D,   /* D */
 LED_PAT_E,   /* E */
 LED_PAT_F    /* F */
};

/*     void led_init(void) */
/*           set up ports for display of LED digits */
void led_init(void)
{
#if 0  /* ports initialized in Port_Init() by Applilet */
       PM7 = 0x00;         /* set all port 7 to output */
       PM4 = 0x00;         /* set all port 4 to output */
       PM5 = 0x00;         /* set all port 5 to output */
#endif
}

/* void led_out_right(unsigned char val) */
/*           output raw data to right LED */
void led_out_right(unsigned char val)
{
       P7 = val;
}

/* void led_out_left(unsigned char val) */
/*           output raw data to left LED */
void led_out_left(unsigned char val)
{
       P4 = val & 0x0F;
       P5 = (val >> 4) & 0x0F;
}

/* void led_dig_right(unsigned char num) */
/*     display number in right LED */
void led_dig_right(unsigned char num)
{
       if (num > 0x0F) {
              led_out_right(LED_PAT_BLANK);
              return;
```

```
        }
        led_out_right(dig_tab[num]);
}

/* void led_dig_left(unsigned char num) */
/*     display number in left LED */
void led_dig_left(unsigned char num)
{
        if (num > 0x0F) {
                led_out_left(LED_PAT_BLANK);
                return;
        }
        led_out_left(dig_tab[num]);
}

/* void led_dig(unsigned char num) */
/*            display number as hex digits */
/*            num - number to display */
/*                  bits 0-3 in right digit */
/*                  bits 4-7 in left digit */
void led_dig(unsigned char num)
{
        led_out_right(dig_tab[num & 0x0F]);
        led_out_left(dig_tab[(num >> 4) & 0x0F]);
}

/* void led_dig_bcd(unsigned char bcdnum) */
/*            display two digits of BCD coded bcdnum */
/*            bcdnum - number to display in BCD */
/*                  0 - 9 displayed as right decimal digit, left blank */
/*                  10 - 99 displayed as two decimal digits */
/*                  100 - 255 displayed as blank */
void led_dig_bcd(unsigned char bcdnum)
{
unsigned char tens_dig;

        if (bcdnum > 99) {
                led_out_right(LED_PAT_BLANK);    /* display both digits blank */
                led_out_left(LED_PAT_BLANK);
                return;
        }

        if (bcdnum < 10) {
                led_out_right(dig_tab[bcdnum]);  /* just display right LED */
                led_out_left(LED_PAT_BLANK);            /* blank left LED */
                return;
        }

        /* 10 <= bcdnum <= 99 */
        tens_dig = 0;
        do {                        /* calculate ten's place and remainder */
                bcdnum -= 10;/* by multiple subtractions of 10 */
                tens_dig++;         /* while counting up the tens digit */
        } while (bcdnum >= 10);
        /* now tens_dig has ten's place */
        /* and bcdnum has remainder */
        led_out_right(dig_tab[bcdnum]);
        led_out_left(dig_tab[tens_dig]);
}
```

7.5.3 M-78F0537 Common 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;** Device :  uPD78F0537
;**
;** Compiler :      NEC Electronics/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
```

7.5.4 M-78F0537 Common 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;** Device :  uPD78F0537
;**
;** Compiler :      NEC Electronics/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
```

7.5.5 M-78F0537 Common Port.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 : port.h
** Abstract : This file implements device driver for PORT module.
** APIlib :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
*****************************************************************************
*/
#ifndef      _MDPORT_
#define      _MDPORT_
/*
*****************************************************************************
** MacroDefine
*****************************************************************************
*/
#define      PORT_PM0     0xff
#define      PORT_PU0     0x0
#define      PORT_P0      0x0
#define      PORT_PM1     0xff
#define      PORT_PU1     0x0
#define      PORT_P1      0x0
#define      PORT_PM2     0xff
#define      PORT_P2      0x0
#define      PORT_PM3     0xff
#define      PORT_PU3     0x6
#define      PORT_P3      0x0
#define      PORT_PM4     0xf0
#define      PORT_PU4     0x0
#define      PORT_P4      0x0
#define      PORT_PM5     0xf0
#define      PORT_PU5     0x0
#define      PORT_P5      0x0
#define      PORT_PM6     0xff
#define      PORT_P6      0x0
#define      PORT_PM7     0x0
#define      PORT_PU7     0x0
#define      PORT_P7      0x0
#define      PORT_PM12    0xff
#define      PORT_PU12    0x0
#define      PORT_P12     0x0
#define      PORT_P13     0x0
#define      PORT_PM14    0xff
#define      PORT_PU14    0x0
#define      PORT_P14     0x0
#define      PORT_ADPC    0x0

void PORT_Init( void );

#endif
```

7.5.6 M-78F0537 Common Port.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 : port.c
** Abstract : This file implements device driver for PORT module.
** APIlib :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "port.h"
/*
************************************************************************
** Constants
************************************************************************
*/

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     This function initializes the I/O module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void PORT_Init( void )
{
      /* initialize the port registers */
      P0 = PORT_P0;
      P1 = PORT_P1;
      P2 = PORT_P2;
      P3 = PORT_P3;
      P4 = PORT_P4;
      P5 = PORT_P5;
      P6 = PORT_P6;
      P7 = PORT_P7;
      P12 = PORT_P12;
```

```
        P13 = PORT_P13;
        P14 = PORT_P14;

        /* initialize the Pull-up resistor option registers */
        PU0 = PORT_PU0;
        PU1 = PORT_PU1;
        PU3 = PORT_PU3;
        PU4 = PORT_PU4;
        PU5 = PORT_PU5;
        PU7 = PORT_PU7;
        PU12 = PORT_PU12;
        PU14 = PORT_PU14;

        /* initialize the mode registers */
        PM0 = PORT_PM0;
        PM1 = PORT_PM1;
        PM2 = PORT_PM2;
        ADPC = PORT_ADPC;
        PM3 = PORT_PM3;
        PM4 = PORT_PM4;
        PM5 = PORT_PM5;
        PM6 = PORT_PM6;
        PM7 = PORT_PM7;
        PM12 = PORT_PM12;
        PM14 = PORT_PM14;
}
```

7.5.7 M-78F0537 Common Sw_0537.h

```
/* sw_0537.h */
/*     header for M-78F0537 CPU board for base board switch reading */
/* Version:  1.1    01-13-2006    */

#ifndef _SW_0537_H
#define _SW_0537_H

/*****************************************************************/
/* Define definitions */
/*****************************************************************/

/* symbolic definitions for switch inputs */
/* SW2 = left switch = P31 */
/* SW3 = right switch = P32 */
/*                                                                P32
             P31 */
#define      SW_LU_RU     0x06   /* left up, right up        1            1     */
#define SW_LD_RU    0x04   /* left down, right up      1            0     */
#define SW_LU_RD    0x02   /* left up, right down      0            1     */
#define SW_LD_RD    0x00   /* left down, right down 0        0     */

#define      SW_DEF_DEB_COUNT    8      /* default debounce counter      */

/*****************************************************************/
/* Export functions */
```

```
/*****************************************************************/
extern void sw_init(void);              /* init ports and variables for switch input */
extern unsigned char sw_chk(void);      /* get undebounced switch input */
extern unsigned char sw_get(void);      /* get debounced switch input */
extern void sw_set_debounce(unsigned char count);    /* set deboune cound */
extern void sw_isr(void);               /* debounce routine, called by timer ISR */

#endif /* _SW_0537_H */
```

7.5.8 M-78F0537 Common Sw_0537.c

```
/*     sw_0537.c - routines for switch input                      */
/*           for M-78F0537 CPU board on M-Station base board      */
/* Version:  1.1    01-13-2006                                    */

/*           P31 = input for left switch (SW2)                   */
/*           P32 = input for right switch (SW3)                        */

/*     To connect ports to switches on M-Station 1.1, make the
       following jumper connections between ROW1 and ROW2.
       To connect ports to swtiches on M-Station 2, make sure
       the default SBxx connections are inserted.

       Port   Switch M-Station 1.1 M-Station 2.2
       ----   ---          ------------ -------------
       P31           SW2           R1.5 - R2.5         SB7
       P32           SW3           R1.6 - R2.6         SB8
*/

/* need pragma declaration to access SFR's in C      */
#pragma sfr

#include "sw_0537.h"

/* local variables for switch handling */
static unsigned char sw_last;           /* last debounced switch value */
static unsigned char sw_new;            /* new value being debounced */
static unsigned char sw_deb_value;      /* value of debounce counter */
static unsigned char sw_deb_count;      /* debounce counter */

/*     void sw_init(void) */
/*           set up ports for switch input */
void sw_init(void)
{
#if 0  /* initialization done in Port_Init() by Applilet */
       /* set P31 and P32 to inputs */
       PM3.1 = 1;
       PM3.2 = 1;
       /* set pullups on P31 and P32 */
       PU3.1 = 1;
       PU3.2 = 1;
#endif
       /* set static variables */
       sw_last = SW_LU_RU; /* default is right up, left up (no switch pressed) */
       sw_deb_value = SW_DEF_DEB_COUNT;  /* set default debounce counter value */
```

```
	sw_deb_count = SW_DEF_DEB_COUNT;  /* set counter to max */
}

/* unsigned char sw_chk(void) */
/*     return input from switches, undebounced */
unsigned char sw_chk(void)
{
	return P3 & 0x06;
}

/* void sw_set_debounce(unsigned char count) */
/*     set the debounce counter value */
void sw_set_debounce(unsigned char count)
{
	sw_deb_value = count;      /* set new debounce counter value */
	sw_deb_count = count;      /* set counter to max */
}

/* unsigned char sw_get(void) */
/*     return debounced switch input */

unsigned char sw_get(void)
{
	return sw_last;
}


/* void sw_isr( void ) */
/* this routine called by periodic timer interrupt to poll and debounce switches */
/* after a new value has been seen steadily for sw_deb_value times, sw_last is updated */
void sw_isr( void )
{
unsigned char val;

	val = sw_chk();            /* get current value */
	/* if value is the same as before, no change; reset debounce and return */
	if (val == sw_last) {
		sw_deb_count = sw_deb_value;     /* reset debounce counter to max */
		return;
	}

	/* val != sw_last, there is a new input */
	/* if it's not the same as the previous new one, */
	/* set the NEW new one, reset the debounce counter and return */
	if (val != sw_new) {
		sw_new = val;
		sw_deb_count = sw_deb_value;
		return;
	}

	/* val != sw_last, val == sw_new */
	/* count down the debounce counter */
	sw_deb_count--;

	/* if we have counted down to zero, we have seen the same sw_new */
	/* for debounce count times, it is now the debounced switch value */
	if (sw_deb_count == 0) {
		sw_last = val;
		sw_deb_count = sw_deb_value;
		return;
	}

	/* if still debouncing, just return */
```

```
        return;
}
```

7.6 Common Listing Files for Programs on DemoKit-LG2 Platform

The files listed in this section are common to the demonstration programs run on the DemoKit-LG2 demonstration kit platform. This section thus includes the following demonstration programs:

- CSI slave demonstration program
- IIC slave demonstration program

7.6.1 DemoKit-LG2 Common 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/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
```

7.6.2 DemoKit-LG2 Common 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/CC78K0
**
*********************************************************************************
*/
#ifndef      _MDSYSTEM_
#define      _MDSYSTEM_
/*
*********************************************************************************
** MacroDefine
*********************************************************************************
*/
#define      CG_X1STAB_SEL 0x5
#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
```

7.6.3 DemoKit-LG2 Common 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 :  NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC Electronics/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;
}
```

7.6.4 DemoKit-LG2 Common 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/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_*/
```

7.6.5 DemoKit-LG2 Common 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/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;
}
```

7.6.6 DemoKit-LG2 Common 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC Electronics/CC78K0
**
*********************************************************************************
*/

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


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

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

7.6.7 DemoKit-LG2 Common 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;** Device :  uPD78F0537
;**
;** Compiler :      NEC Electronics/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
```

7.6.8 DemoKit-LG2 Common 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: NEC Electronics78K0KX2.lib V1.01 [09 Aug. 2005]
;**
```

```
;** Device :  uPD78F0537
;**
;** Compiler :      NEC Electronics/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
```

7.6.9 DemoKit-LG2 Common define.h

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

#define BAUDRATE 115200
```

7.6.10 DemoKit-LG2 Common Lcd.h

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

```
**
** This file was created for the NEC Electronics 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 Electronics/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_ */
```

7.6.11 DemoKit-LG2 Common Lcd.c

```
/*
*****************************************************************************
**
** This file was created for the NEC Electronics 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 Electronics/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);
 }
}
```

7.6.12 DemoKit-LG2 Common LcdDrvApp.h

```
/*********************************************************************
;
; NNNNNN NN EEEEEEEEEEEEEEEE CCCCCCCCCCCCCC
; NNNNNNNN NN EEEEEE CCCCCC
; NNNNNNNNNN NN EEEEEE CCCCCC
; NN NNNNNNNN NN EEEEEEEEEEEEEEEE CCCCCC
; NN NNNNNNNN NN EEEEEE CCCCCC
; NN NNNNNNNNNN EEEEEE CCCCCC
; NN NNNNNN EEEEEEEEEEEEEEE CCCCCCCCCCCCCC
;
; 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 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
;
; +-------+-------+-------+-------+-------+-------+-------+-------+
; 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 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 */

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

7.6.13 DemoKit-LG2 Common LcdDrvApp.c

```
/*****************************************************************************
;
; NNNNNN NN EEEEEEEEEEEEEEEE CCCCCCCCCCCCCC
; NNNNNNNN NN EEEEEE CCCCCC
; NNNNNNNNNN NN EEEEEE CCCCCC
; NN NNNNNNNN NN EEEEEEEEEEEEEEEE CCCCCC
; NN NNNNNNNN NN EEEEEE CCCCCC
; NN NNNNNNNNN EEEEEE CCCCCC
; NN NNNNNN EEEEEEEEEEEEEEE CCCCCCCCCCCCCC
;
; 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 >---*/
```