RENESAS

Application Note

IIC Communication with LCD Modules in V850ES Microcontrollers

Document no. U18520EU1V0UM00

Contents

1. INTRODUCTION

This document provides useful information and examples to help designers understand how to use the on-chip peripherals of NEC Electronics microcontrollers:

- Description of peripheral features
- Example program specifications
- Software flowcharts
- Applilet reference drivers
- Demonstration platforms used
- Hardware block diagram
- Software modules

Applilet is a software tool that generates driver code for the peripherals and a convenient means for generating code for fast evaluation. (Refer to the Applilet user's manual and other related documents for further details.)

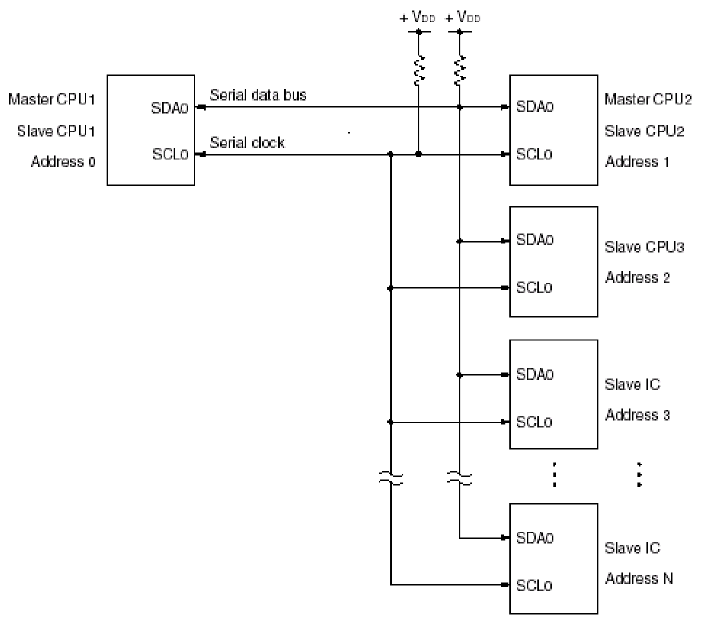
2. IIC COMMUNICATION

The inter-integrated circuit (also known as IIC or I^2C) is a synchronous communications protocol that allows a master device to initiate communication with one or more slave devices. The IIC bus uses two lines: serial clock (SCK) and serial data (SDA). These lines are generally implemented as open-collector outputs with pull-up resistors to allow any node on the bus to drive the lines and multiple master and slave devices to be connected. The last part of this document illustrates an example of IIC communication with an LCD module.

3. IIC COMMUNICATION WITH AN LCD MODULE

The IIC interface is used for 8-bit data transfers between two or more devices. The IIC bus uses two signaling lines, called serial data bus (SDA) and serial clock (SCL). These lines may be driven by any node or pulled up by external resistors. The protocol supports multiple devices attached to the bus. A device may be a master or slave or both; multiple masters and multiple slaves also can be attached to the bus.

Figure 1.



A master device can initiate transfers, and will specify 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 will detect the slave address specified at the beginning of the transfer; the least significant bit (LSB) of the slave address specifies whether the transfer is a read or a write cycle by the master. A slave device whose locally set slave address matches the address specified by the master responds by receiving or sending the data.

A master device, by responding to a slave address sent by another master, can also communicate as a slave, if the internal hardware supports this. NEC Electronics microcontrollers with IIC peripherals can operate as both a master and slave.

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

This section describes how to use the IIC peripheral of an NEC Electronics V850ES™ microcontroller for communication with an external liquid crystal display (LCD) module.

3.1 IIC0 Communication in V850ES Microcontrollers

The IIC0 peripherals in NEC Electronics V850ES microcontrollers support 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 current transfer ends
- Automatic generation and recognition of wait conditions in sending or receiving
- Selectable clock for IIC0 peripheral to support multiple data rates
- Standard IIC rate of 100,000 bits per second (bps) and high-speed rates up to 400,000 bps
- Digital filter for noise suppression in high-speed mode
- IIC extension codes
- Settable slave address register for response to different slave addresses

Figure 2. Simplified Block Diagram of IIC0 Serial Communication Circuit

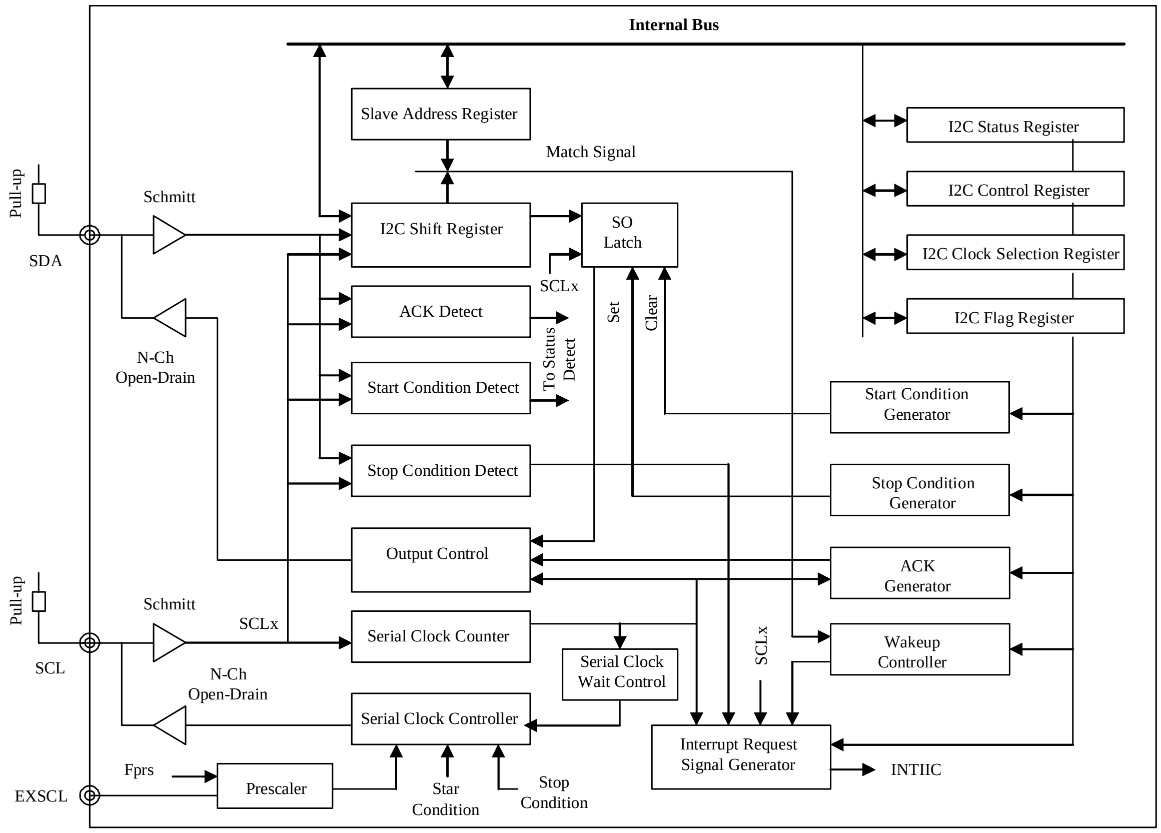


Table 1 provides a summary of the functional blocks and registers supporting IIC communication.

Table 1. Functional Blocks

Functional Block	Description
IIC shift register	Converts 8-bit serial data to 8-bit parallel data and vice versa
Slave address register	Stores local addresses when in slave mode
SO latch	Retains the SDA output level
Wakeup controller	Generates an interrupt when the address received matches the address value stored in the slave address register
Prescaler	Selects the sampling clock to be used
Serial clock counter	Counts the serial clocks that are output or input during transmit and receive operation; used to verify that 8-bit data has been transmitted or received

Functional Block	Description
Interrupt request signal generator	Generates an interrupt request at the falling edge of the eighth or ninth serial clock or upon detection of a stop condition
Serial clock controller	Generates SCL clock output from the sampling clock in master mode
Serial clock wait controller	Controls waiting time
ACK generator/detector Start/stop condition detector	Generates and detects each condition
Start condition generator	Generates a start condition
Stop condition generator	Generates a stop condition

Table 2. IIC Registers

Register	Description
IIC control register	Enables/disables IIC operation; sets wait timing
	Enables/disables an interrupt request upon detection of a stop condition
	Controls wait timing and interrupt request generation
	Controls ACK character
	Triggers a start or stop condition
IIC status register	Detects the master or slave status of a device
	Detects matching address, transmit, receive, ACK, START, STOP
IIC flag register	Indicates the operating mode and the status of the IIC bus
IIC clock selection register	Specifies the transfer clock for the IIC bus
IIC function expansion register	Specifies the function expansion of IIC communication
Port mode and port registers	Specifies port mode and the port functions

3.2 Configuring the IIC0 Peripheral for IIC Communication

This procedure explains how to initiate IIC communication.

1. Clear the IICE0 bit in the IICC0 control register to disable the peripheral while setting other registers.

2. Set the registers that control the port pins to enable drive of the SCL0 and SDA0 pins.

3. Set the communication parameters in the IICF0 flag register and IICC0 control register.

4. Select the clock source in the IICCL0 clock register and specify the speed in the IICX0 extension register.

5. Set the interrupt priority using the appropriate interrupt priority register bit.

6. Set the slave address for slave mode in the SVA0 slave address register.

7. Enable the INTIIC0 interrupt by clearing the appropriate interrupt mask bit.

8. Enable the IIC0 peripheral by setting the IICE0 bit in the IICC0 control register.

3.3 Communication in Master Mode

This procedure explains how to initiate communication in master mode.

1. 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.

2. Write the desired slave address using the LSB to set direction to the IIC0 register

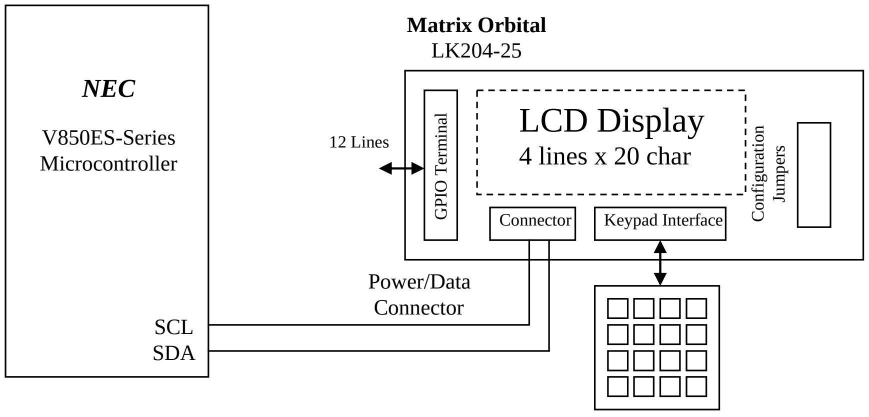
Additional communication in master mode is handled in response to the INTIIC0 interrupt. Communication in slave mode also is handled in response to an INTIIC0 interrupt received after a master sends a matching slave address.

3.4 Program Description and Specification

The program uses IIC write operations to send commands and strings of data to an LCD module through the IIC bus. The LCD module responds to the commands and data by displaying strings on the display.

The LCD module also contains a keypad interface. When you press a key on the keypad, the LCD module stores the keypress as ASCII code that can be read via an IIC read operation. The program polls the LCD module for key press codes and displays them on the LCD.

Figure 3. SCL and SDA Interface Connections



In this example, the SCL and SDA IIC interface lines, along with power and ground, are connected to the 4-pin power/data connector of the Matrix Orbital LCD module to enable the following:

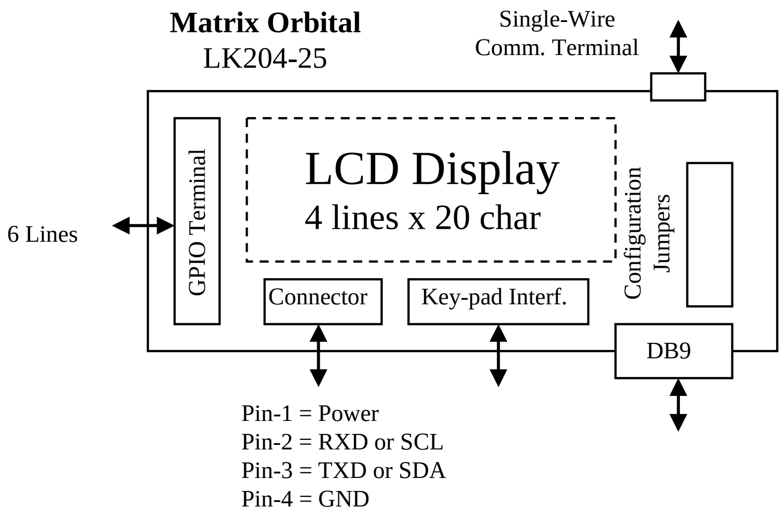
- Matrix Orbital LK204-25 LCD module as the IIC slave device
- SDA and SCL lines with external pull-up resistors that allow either side to drive or float
- IIC communication speed of 100,000 bps or lower (standard IIC data rate)
- IIC0 peripheral for the master, with the SCL0 and SDA0 pins controlled by the IIC0 peripheral
- Slave that responds to the IIC slave address 0x50 for slave communication with the master
- Master write cycles to the slave using multiple-byte transfers of commands or string data
- Master read cycles to the slave using one-byte transfers of key press values
- Slave response with key value 00H if no buffered key press is available

3.4.1 Matrix Orbital LCD Module

The Matrix Orbital LK204-25 is a standalone LCD module with the following features.

- 20-character by 4-line text display
- Built-in fonts
- Communication over IIC or RS-232 interface

Figure 4. Block Diagram of Matrix Orbital LCD Module



For IIC communication, the Matrix Orbital LK204-25 module can support:

- Data rates up to 400 kilobits per second (Kbps)
- Interface voltage level of 5 V_{DC}, CMOS level
- 7-bit address format, with the eighth bit indicating direction
- Default address of 0x50 for writing to the module
- Default address of 0x51 for reading keypad data

To display data, the LK204-25 uses

- Built-in dot-matrix fonts
- Up to eight user-defined characters

When text data is written, the LK204-25

- Receives a character
- Displays the character at its current position
- Displays a “space” in place of any undefined character

The LK204-25 uses a series of display commands that it sends with a special character (0xFE) prefix followed by a command byte and the command data (if required).

The LK204-25 supports the attachment of a keypad matrix with up to five columns by five rows, a maximum of 25 keys. When you press a key on the keypad, the key press is sent immediately via the RS-232 interface, buffered for later polling by the RS-232 interface, or read via the IIC interface.

Refer to the *LK204-25 User's Manual* for more information.

3.4.2 Program Flowcharts

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

- Code for program and port initialization, called before the main() program starts
- The main program loop, which polls the LCD module for key presses and transmits commands and strings through the IIC bus
- Subroutines for accessing the LCD module
- Subroutines for sending and receiving IIC data
- Interrupt service routines related to IIC communication

The programs also include sections not related directly to IIC operation, such as sections about timer operation.

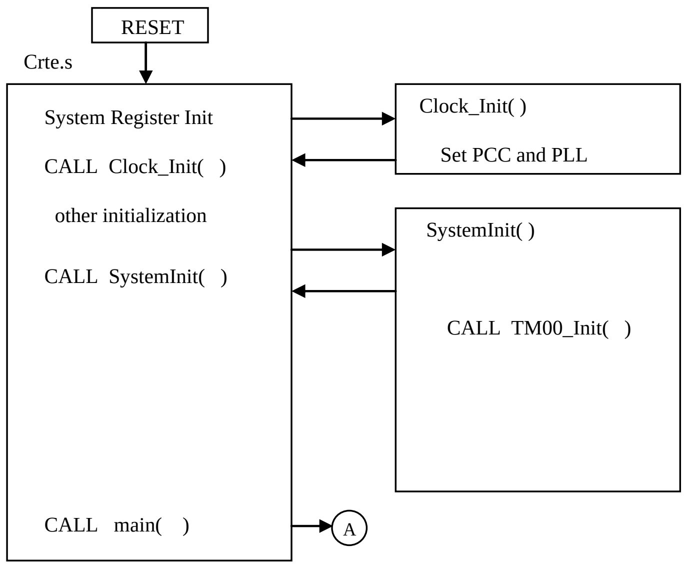
The flowcharts here illustrate the initialization sequence, the main program, and the subroutines and interrupt routines related to IIC operation. Flowcharts are not included for other initialization sequences, other peripheral operation subroutines, and other peripheral interrupt service routines. The software listings include this code.

3.4.3 Program Startup and Initialization

For V850ES programs written in C language, the assembly language startup file, generally named crte.s, supplies startup code for the C language program. This startup code specifies the reset vector that determines the starting point of the program after a hardware reset, and also provides the initial setup of system registers before calling the main() program.

When you use Applilet to generate a C language program for the microcontroller, the program automatically generates the crte.s startup file in assembly language. The crte.s file calls the Clock_Init() function to set the system clock and then the SystemInit() function to initialize subroutines for some (but not all) of the peripherals.

Figure 5. Startup Sequence



The SystemInit() program initializes the peripherals before the startup code calls the main() routine of the user program. Therefore, the main() routine does not need to call the individual peripheral initialization routines. (Note that the SystemInit() function does NOT automatically call the IIC0_Init() routine to initialize the IIC0 peripheral.)

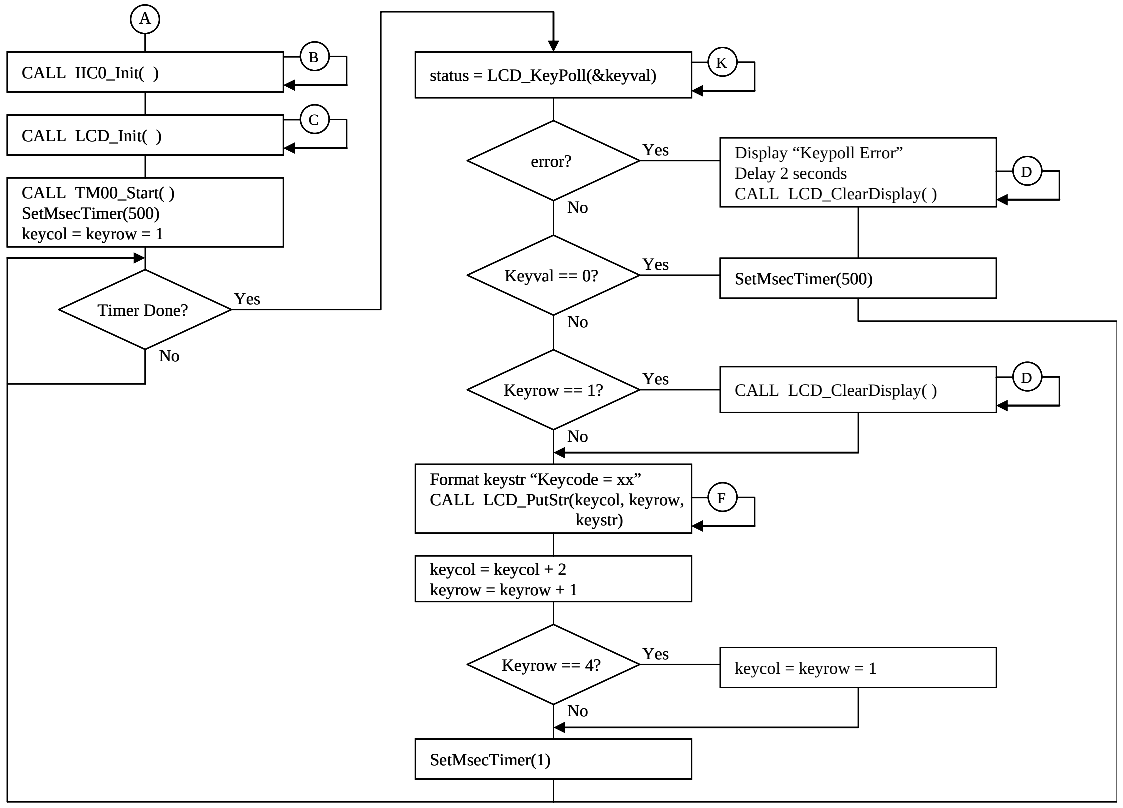
3.4.4 Main( ): Main Program for IIC Communication to the LCD Module

When the startup code calls the main( ) program after peripheral initialization, main() calls IIC0_Init() to initialize the IIC0 peripheral and LCD_Init() to initialize the LCD module by clearing the display, setting the keypad interface, and showing a startup message on the LCD.

Main() calls TM00_Start() to start the millisecond (ms) timer and SetMsecTimer(500) to set the timer for a 500 ms delay. Main() then sets the local **keycol** and **keyrow** variables to one, the display position for the initial string.

After completion of the main program loop, main() calls CheckMsecTimer() to check whether the ms timer has elapsed. If it has not elapsed, the program does nothing and checks the timer again.

Figure 6. Main() Program



If the timer has elapsed, the program calls LCD_KeyPoll(&keyval) to check whether there is key code to read. If the program encounters an IIC0 error, the LCD displays an error message for two seconds, after which the timer resets to 500 ms for the next poll and the routine returns to the top of the loop.

If there is no error, the LCD_KeyPoll() routine stores the value of the key code returned by the LCD module in **keyval**. If **keyval** is zero, there is no key press character available from the LCD module; the timer is set to poll again in 500 ms and the routine returns to the top of the loop.

If **keyval** is non-zero, the program has read a key press from the LCD module. If **keyrow** is set at 1, the program calls LCD_ClearDisplay() to clear the display. The program then formats a string in the **keystr** array to show the key values and then calls LCD_Putstr(keycol, keyrow, keystr); to write the string to the display at the current location.

The variables that set the string position are incremented: **keycol** is incremented by two and **keyrow** is incremented by one. If **keyrow** is now set to more than four (the number of rows on the display), the **keycol** and **keyrow** variables are set back to one.

Since the program has detected a key press, the timer delay before the next poll is set to 1 ms and the program returns to the top of the loop.

3.4.5 IIC0_Init(): Initialize IIC0 Peripheral

The IIC0_Init() routine initializes the IIC0 peripheral settings but does not start operation.

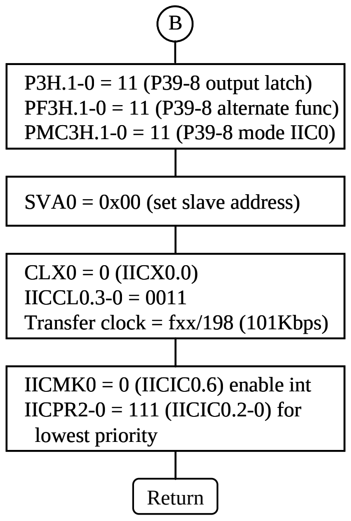
First the appropriate port pins are set for IIC operation. For IIC0, the SDA0 signal is on pin P38/SDA0 and the SCL0 signal is on pin P39/SCL0. The port output latches for these pins are set to 1. Note that port 3 is a 16-bit I/O port; when accessed in 8- or 1-bit mode, the port is referred to as *P3L* (for bits 0–7) or *P3H* (for bits 8–15). The P3H.1 output latch bit is the same as P3.9, the output latch for P39/SCL0. In a similar fashion, the appropriate bits in the PF3 Port Function register are set for the alternate function and the PMC3 Port Mode Control register bits are set for the IIC function.

The local slave address (if the processor will respond as a slave) is set to 00. This program does not support response as a slave.

The CLX0 bit in the IICX0 register and the SMC0 (IICCL0.3), CL01 (IICCL0.1) and CL00 (IICCL0.0) bits are set to control the clocking of the IIC0 peripheral. This setting causes the main system clock, fxx, to be divided by 198 to generate the SCL0 transfer clock. With this division, the clock rate will be 101,010 bps, just slightly over the standard rate of 100 Kbps. Since the Matrix Orbital LCD module supports up to 400 Kbps, this setting will work.

The IICIC0 interrupt control register is set to enable the INTIIC0 interrupt by setting the mask bit to 0, and the priority of this interrupt is set for the lowest-priority group.

Figure 7. IIC0_Init() Routine


B
P3H.1-0 = 11 (P39-8 output latch)
PF3H.1-0 = 11 (P39-8 alternate func)
PMC3H.1-0 = 11 (P39-8 mode IIC0)
SVA0 = 0x00 (set slave address)
CLX0 = 0 (IICX0.0)
IICCL0.3-0 = 0011
Transfer clock = fxx/198 (101Kbps)
IICMK0 = 0 (IICIC0.6) enable int
IICPR2-0 = 111 (IICIC0.2-0) for lowest priority
Return

3.4.6 LCD_Init(): Initialize LCD Module

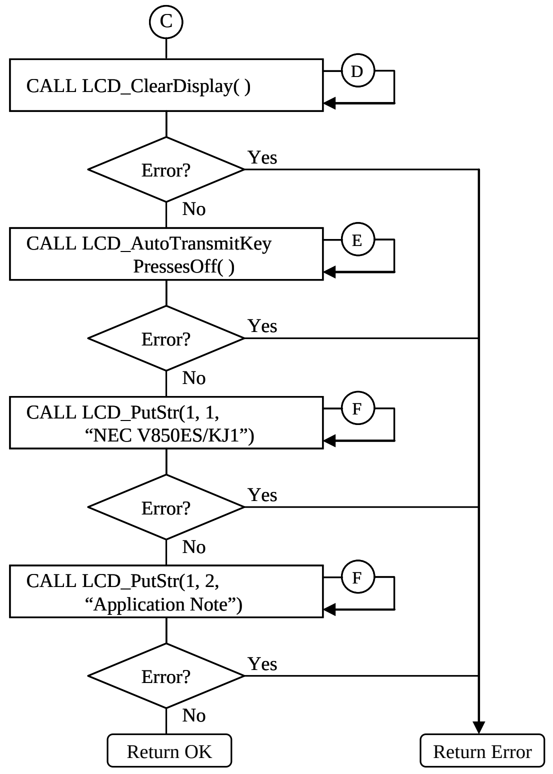
The LCD_Init() routine initializes the LCD module for display and key input. The routine first calls LCD_ClearDisplay() to clear any power-up display on the LCD panel.

The routine then calls LCD_AutoTransmitKeyPressesOff(), to set the LCD module to store key presses in a buffer. By default, the module is set to transmit key codes immediately as pressed using the RS-232 interface. To read key presses via the IIC interface, turn off the autotransmit mode to have the LCD module store key presses in a buffer (10 deep) and return one key press after each IIC read cycle.

The LCD_Init() routine displays “NEC V850ES/KJ1” on the first line of the screen and “Application Note” on the second by calling LCD_Putstr( ) with the appropriate values for column, row, and string.

If there are any errors, LCD_Init() stops initialization and returns an error; otherwise the program returns a value indicating that initialization was successful.

Figure 8. CD_Init() Routine


C
CALL LCD_ClearDisplay( )
D
Error?
Yes
No
CALL LCD_AutoTransmitKey PressesOff( )
E
Error?
Yes
No
CALL LCD_PutStr(1, 1, "NEC V850ES/KJ1")
F
Error?
Yes
No
CALL LCD_PutStr(1, 2, "Application Note")
F
Error?
Yes
No
Return OK
Return Error

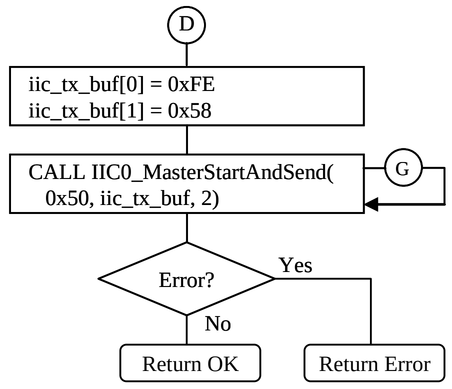
3.4.7 LCD_ClearDisplay(): Clear Screen on LCD Module

The LCD_ClearDisplay() routine clears the screen on the LCD module and sets up a transmit buffer containing two bytes. The first byte is set to FEH, the special character preceding commands for the Matrix Orbital LCD module, and the second to 58H, the command to clear the display.

The routine calls the IIC0_MasterStartAndSend(0x50, iic_tx_buf, 2) routine, which sends two bytes to the LCD module by executing an IIC write transfer to slave address 50H, the default slave address of the LCD module for write cycles.

If IIC0_MasterStartAndSend() enounters an error, the error is returned to the calling routine. Otherwise, IIC0_MasterStartAndSend() returns a value indicating success.

Figure 9.	LCD_ClearDisplay() Routine

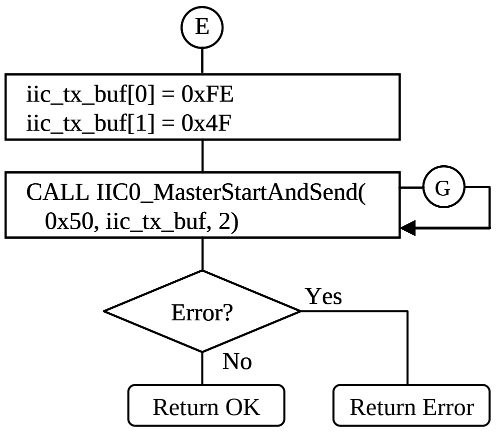


3.4.8 LCD_AutoTransmitKeyPressesOff(): Set LCD Module To Buffer Key Presses

The LCD_AutoTransmitKeyPressesOff() routine sets the LCD module to store key press codes until they are read. By default, the LCD module transmits key press codes via the RS-232 interface as soon as the codes are detected; this routine causes the LCD module to store key presses in a buffer (with a depth of ten key presses) until after completion of a read operation on the IIC bus.

The routine first creates a transmit buffer containing two bytes. The first is set to FEH, the special character preceding commands for the Matrix Orbital LCD module, and the second is set to 4FH, the **Auto Transmit Key Presses Off** command.

Figure 10. LCD_AutoTransmitKeyPressesOff() Routine



The routine calls the IIC0_MasterStartAndSend(0x50, iic_tx_buf, 2), which sends two bytes to the LCD module by executing an IIC write transfer to slave address 50H, the default slave address of the LCD module for write cycles.

If successful, IIC0_MasterStartAndSend() returns a value indicating success. In the event of error, the routine returns an error to the calling routine.

3.4.9 LCD_PutStr(col, row, *str): Display String on LCD at Specified Column, Row

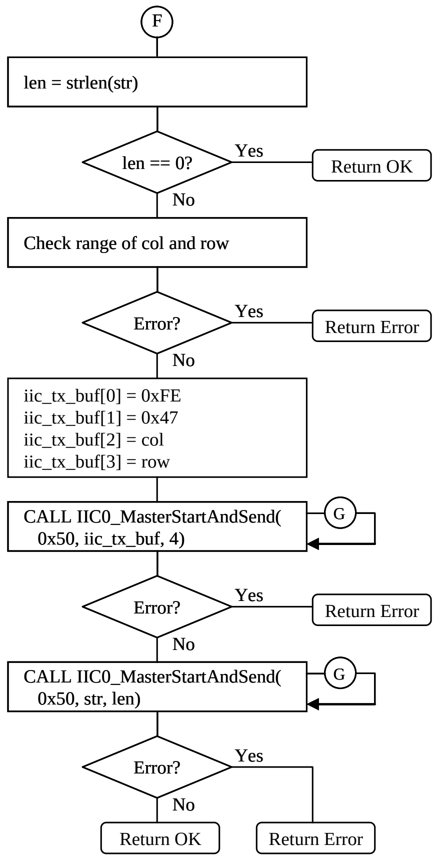
The LCD_PutStr() routine displays a string at a specified position on the LCD screen. The **col** parameter specifies the column position, with 1 being the first column. The **row** parameter specifies the row position, with 1 being the first row. The ***str** parameter points to a string, such as an array of bytes, with a zero byte terminating the string.

The routine first checks the length of the string. If the length is zero, nothing is written to the display and the return value indicates success. The routine then verifies that the **col** and **row** parameters are within the range allowed for the LCD and returns an error if they are not. For the LCD module used in this example, **col** has a range of 1 to 20 and **row** a range of 1 to 4.

The routine then creates a transmit buffer containing four bytes. The first is set to FEH, the special character preceding commands for the Matrix Orbital LCD module, and the second is set to 47H, the **Set Cursor Position** command. The third is the **col** parameter and the fourth is the **row** parameter. The routine executes an IIC write transfer to slave address 50H, the default slave write address, which calls IIC0_MasterStartAndSend(0x50, iic_tx_buf, 4) and sends the four bytes to the LCD.

In the event of error, IIC0_MasterStartAndSend() returns the error to the calling routine. Otherwise, IIC0_MasterStartAndSend() calls IIC0_MasterStartAndSend(0x50, str, len) to send the string of characters to the display by executing a second IIC write transfer, this time with the address of the passed string defining the data, and the length of the string defining the number of bytes to send. In the event of error, IIC0_MasterStartAndSend() returns the error to the calling routine; otherwise IIC0_MasterStartAndSend() returns a value indicating success.

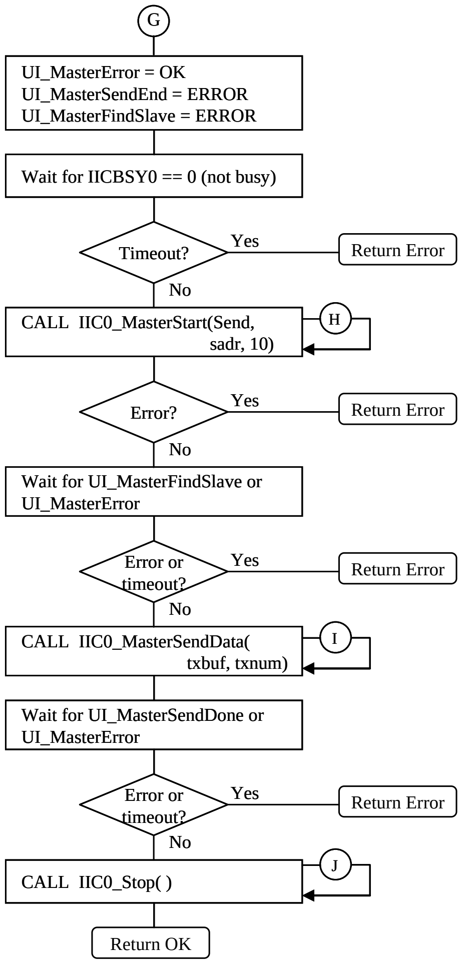
Figure 11. LCD_PutStr(col, row, *str) Routine



3.4.10 IIC0_MasterStartAndSend(sadr, *txbuf, txnum): Start Master, Send Data to IIC Slave

The IIC0_MasterStartAndSend(sadr, *txbuf, txnum) routine is a simple combination of IIC0_MasterStart() and IIC0_MasterSendData(), which is necessary for sending IIC data as a master. It was not created by Applilet. The **sadr** parameter is the slave address for send data, the ***txbuf** parameter points to an array of bytes to send, and the **txnum** parameter specifies the number of bytes to send.

Figure 12. IIC0_MasterStartAndSend(sadr, *txbuf, txnum) Routine


G
UI_MasterError = OK
UI_MasterSendEnd = ERROR
UI_MasterFindSlave = ERROR
Wait for IICBSY0 == 0 (not busy)
Timeout?
Yes
Return Error
No
CALL IIC0_MasterStart(Send, sadr, 10)
H
Error?
Yes
Return Error
No
Wait for UI_MasterFindSlave or UI_MasterError
Error or timeout?
Yes
Return Error
No
CALL IIC0_MasterSendData(txbuf, txnum)
I
Wait for UI_MasterSendDone or UI_MasterError
Error or timeout?
Yes
Return Error
No
CALL IIC0_Stop( )
J
Return OK

The routine first sets IIC communication flags to default settings; these flags will be affected by IIC0 callback functions during the IIC communication process.

UI_MasterError is set to MD_OK (to indicate no error), UI_MasterSendEnd is set to MD_ERROR to indicate that sending is not done, and UI_MasterFindSlave is set to MD_ERROR to indicate that a slave is not found at this point.

The routine then waits until the bus is not busy, as indicated by the IICBSY0 bit in the IICF0 flag register, to allow previous operations to complete or another master to finish using the bus.

The routine then calls the IIC0_MasterStart(Send, sadr, 10) function, to start IIC communication as a master for sending data, and specifies the slave address of **sadr**. A return indicates that either the IIC0 peripheral has been enabled and the slave address written to the IIC0 register for transmission, or the IIC communication channel is busy, which should not happen in this demonstration system.

The routine waits for UI_MasterFindSlave to be OK or for UI_MasterError to be set to an error condition. The routine times out if neither condition occurs within a set time.

The IIC0 peripheral receives an interrupt on the ninth clock bit; the interrupt is handled by the MD_INTIIC0 interrupt service routine. If the LCD module has been properly addressed with its slave address, the module responds with an ACK that sets the UI_MasterFindSlave flag. If there is no ACK, the interrupt service routine sets the UI_MasterError flag to MD_NACK.

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

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

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

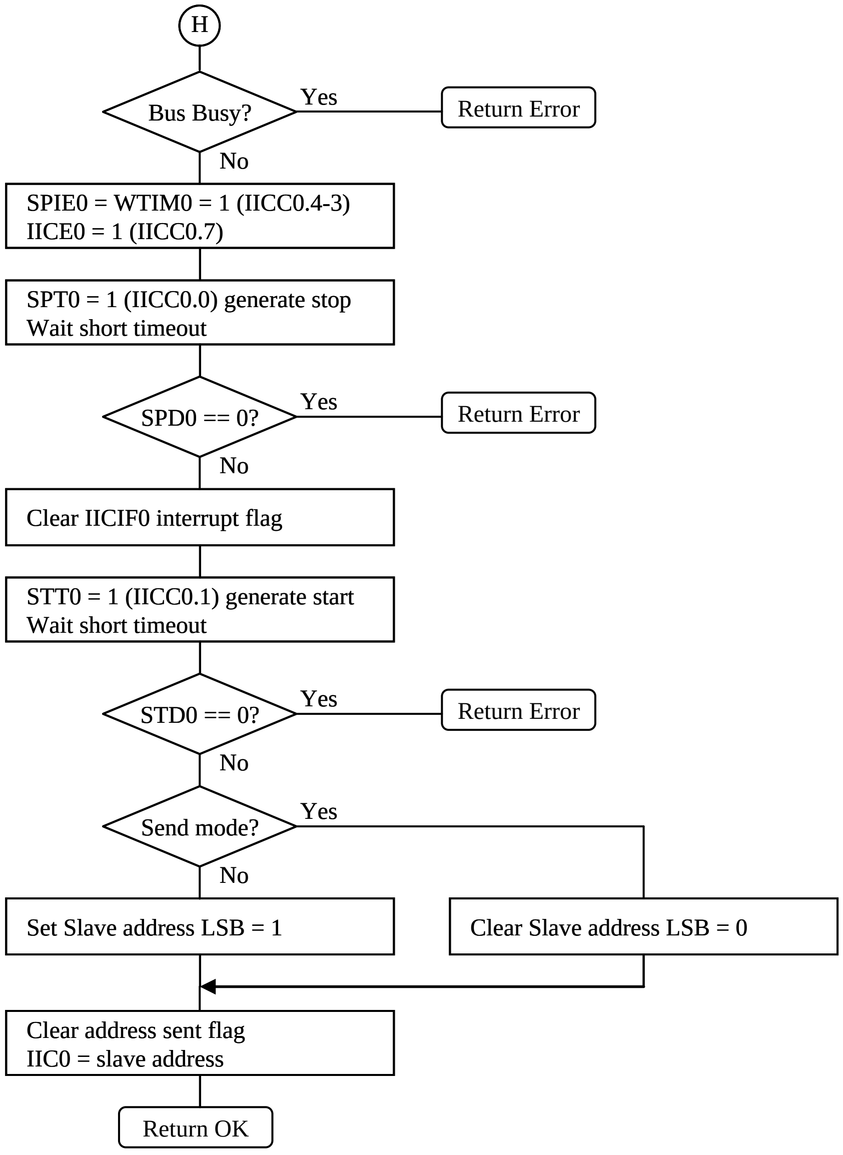
The IIC0_MasterStartAndSend() routine waits for the UI_MasterSendDone condition to be true or for UI_MasterError to be set to an error condition, and times out if neither condition occurs within a set time.

If an error occurs in transmission or if the routine times out, an error is returned. Otherwise, the UI_MasterSendDone flag was set and the transfer was successful. The routine calls IIC0_Stop() to end the transfer and returns a value indicating success.

3.4.11 IIC0_MasterStart(mode, adr, wait): Start Master Read or Write To IIC Slave

The IIC0_MasterStart(mode, adr, wait) function starts IIC communication in master mode. The **mode** parameter is either **Send** or **Receive**, the **adr** parameter specifies the slave address for communication, and the **wait** parameter controls the amount of time to wait for a start condition to be generated.

Figure 13. IIC0_MasterStart(mode, adr, wait) Routine



The routine checks to see of the IIC bus is busy, as indicated by the IICBSY0 bit in the IICF0 flag register, and returns an error if so. The IICC0 register bits (SPIE0 and WTIM0) are set to enable interrupts on stop conditions and to interrupt after nine clocks (master mode). The IICC0 bit 7 (IICE0) then is set to 1 to enable the IIC0 peripheral.

The SPT0 bit (IICC0.0) is set to 1 to generate a stop condition (recommended when IICE0 is enabled), and a short timeout occurs to wait for a stop condition. The SPD0 bit (IICS0.0) is set to 1 to indicate when a stop condition is detected. If no stop condition is detected, then an error is returned. The stop condition causes an interrupt so that the interrupt flag is cleared.

The STT0 bit (IICC0.1) is set to 1 to generate a start condition, and a wait occurs to allow the start condition to be registered. STD0 (IICS0.1) indicates that a start condition exists. If a start condition is not detected, then an error is returned.

If **mode** is set to Send, the least significant bit (LSB) of the slave address is set to 0, which indicates a master-to-slave transmission of data. If **mode** is set to receive, the LSB of the slave address is set to 1, which indicates a slave-to-master transmission of data. The “address sent” flag, which is used by the MD_INTIIC0() interrupt service routine to determine the action to take on interrupt, is cleared and the slave address is written to the IIC0 register to start the IIC transfer. The routine then returns, having started communication to the slave.

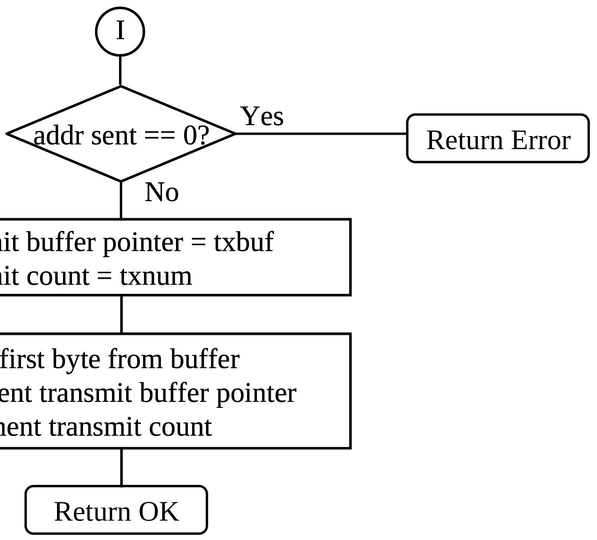
3.4.12 IIC0_MasterSendData(*txbuf, txnum): Master Send Data To IIC Slave

The IIC0_MasterSendData(*txbuf, txnum) function is called after IIC0_MasterStart() has been called, and after the UI_MasterFindSlave flag has been set to indicate that the slave is there and ready for data. The **txbuf** parameter points to an array of bytes to send, and the **txnum** parameter specifies a count of bytes to send.

The routine checks to see if the “address sent” flag has been set, and returns an error if it hasn’t. The latter would result if this routine were called before the INTIIC0 interrupt occurs at the end of slave address transmission.

The routine then sets the buffer address for data to be sent to the location specified by the **txbuf** pointer, sets the count of bytes to send to txnum, writes the first byte to the IIC0 register, increments the pointer, and decrements the count. The MD_INTIIC0 interrupt service routine sends the second and further bytes.

Figure 14. IIC0_MasterSendData(*txbuf, txnum) Routine


I
addr sent == 0?
Yes
Return Error
No
Transmit buffer pointer = txbuf
Transmit count = txnum
IIC0 = first byte from buffer
Increment transmit buffer pointer
Decrement transmit count
Return OK

3.4.13 MD_INTIIC0(): Interrupt Service Routine for INTIIC0 Interrupt

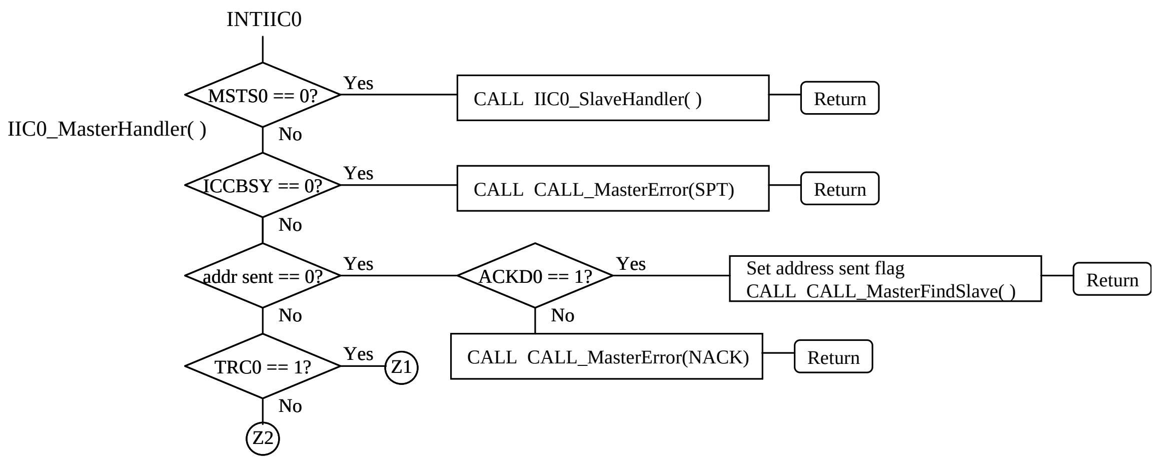
The MD_INTIIC0() interrupt service routine (ISR), and the functions it calls, do most of the work for IIC transfers. The ISR checks the MSTS0 bit in the IICS0 status register to determine whether a master or slave operation is in progress and then calls the appropriate handler routine, either IIC0_MasterHandler() or IIC0_SlaveHandler(). For communication with the LCD module, this example uses master communication, so the flowchart above shows the IIC0_MasterHandler() flow.

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

The handler first checks whether the bus is non-busy, in other words, whether a stop condition has been detected. This should not happen in normal master mode communication, so UI_MasterError() is called to indicate a stop error.

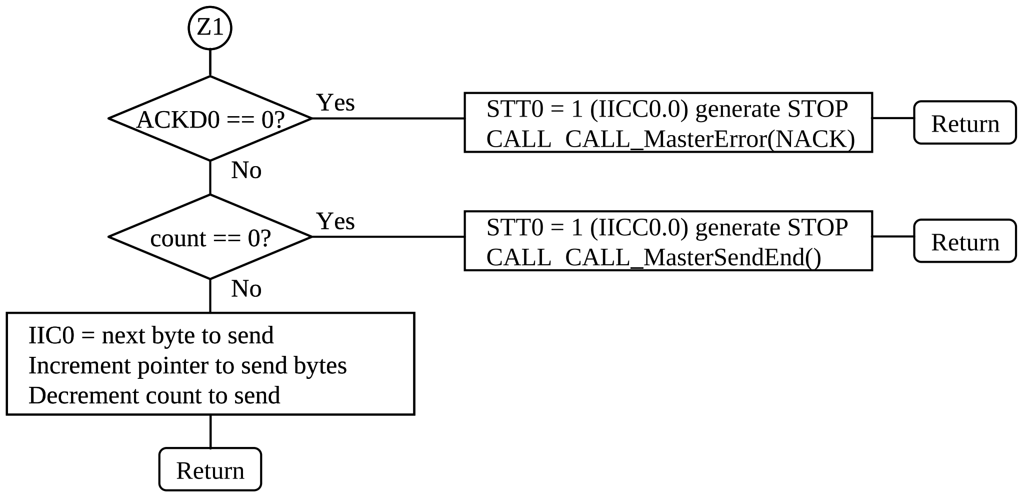
The handler then verifies whether the slave “address sent” flag has been set. If it has not been set, this interrupt occurs at the end of sending the slave address. The routine checks whether an ACK has been received from the slave; if not, the NACK error condition is set. If the ACK has been received, then the slave responded to the address properly; the UI_MasterFindSlave flag is set, the “address sent” flags is set true, and the handler returns.

Figure 15. MD_INTIIC0() Routine



After the first interrupt, further interrupts occur after data bytes have been sent or when a receive operation has been started. The handler checks to see if the transfer direction is sending (master to slave, TRC0 = 1) or receiving (slave to master, TRC0 = 0). The sending flow is shown in the figure below; the receiving flow is shown in the figure following.

Figure 16. Sending Flow

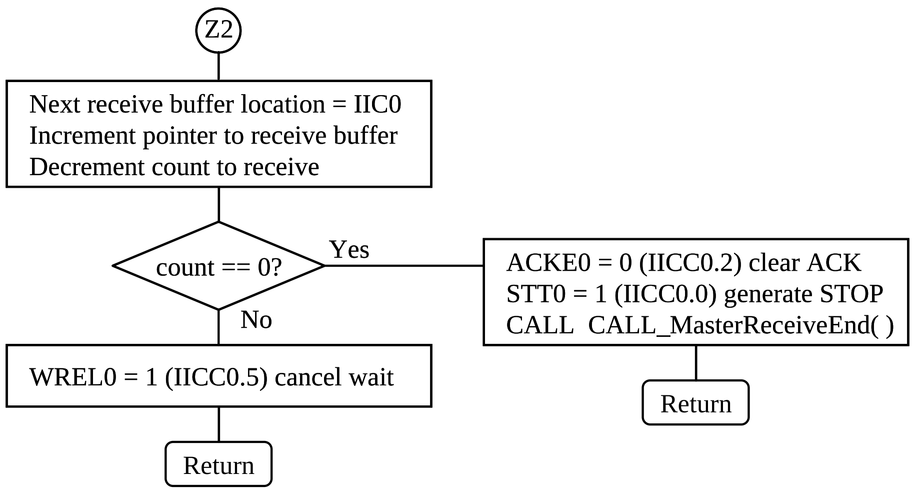

Z1
ACKD0 == 0?
Yes
STT0 = 1 (IICC0.0) generate STOP
CALL CALL_MasterError(NACK)
Return
No
count == 0?
Yes
STT0 = 1 (IICC0.0) generate STOP
CALL CALL_MasterSendEnd()
Return
No
IIC0 = next byte to send
Increment pointer to send bytes
Decrement count to send
Return

If sending, the handler expects an ACK from the slave after each data byte sent. If the ACK is not detected, a stop condition is set, the NACK error condition is set, and the handler returns.

If the ACK was received, the handler checks the count of bytes to see if there are any more left to send. If there are no more bytes to be sent, this interrupt occurred after the last byte was sent and the transfer is done. The handler sets a stop condition to inform the slave that the transfer is done, sets the UI_MasterSendDone flag, and returns.

If the count is not zero, there are more bytes to be sent. The handler writes the next byte to IIC0 to start transmitting, updates the count and buffer pointers, and returns to wait for the next INTIIC0 interrupt.

Figure 17. Receiving Flow


Z2
Next receive buffer location = IIC0
Increment pointer to receive buffer
Decrement count to receive
count == 0?
Yes
No
ACKE0 = 0 (IICC0.2) clear ACK
STT0 = 1 (IICC0.0) generate STOP
CALL CALL_MasterReceiveEnd( )
Return
WREL0 = 1 (IICC0.5) cancel wait
Return

If receiving, IIC0 peripheral has automatically generated a wait condition on the bus after the data byte was received, to have the slave wait until the data is dealt with. The handler reads the IIC0 register to get the byte received, stores it in the receive buffer, increments the pointer to the buffer, and decrements the count of bytes to receive.

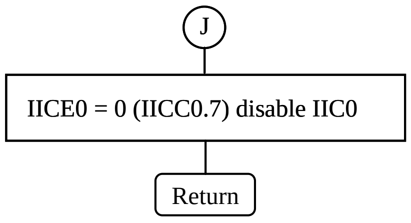
If there are no more bytes to be received, the transfer is done. The handler clears the ACKE0 bit, which will cause the slave to not see an ACK condition after the transfer (this tells the slave the transfer is done), sets a stop condition to finish the transfer, sets the UI_MasterReceiveDone flag, and returns.

If there are more bytes to receive, the handler sets the WREL0 bit to release the wait condition. At this point, the slave sees an ACK condition on the bus and begins sending the next byte. The handler returns to wait for the next INTIIC0 interrupt.

3.4.14 IIC0_Stop(): Stop IIC0 Operation

The IIC0_Stop() routine ends the IIC0 send or receive operation by clearing the IICE0 bit in the IICC0 control register to zero, disabling the IIC0 peripheral. The enable bit will be set again on the next call to IIC0_MasterStart().

Figure 18. IIC0_Stop() Routine



3.4.15 LCD_KeyPoll(*pkey): Poll LCD Module For Keypad Input

The LCD_KeyPoll(*pkey) routine checks whether the LCD module has a key press available in its buffer. The **pkey** parameter is a pointer to a byte location to store the key code read. If no key is available, or if an error occurs, zero is stored for the key code.

First the routine checks to see if the **pkey** parameter is zero. If so, the pointer has not been set, and an error is returned.

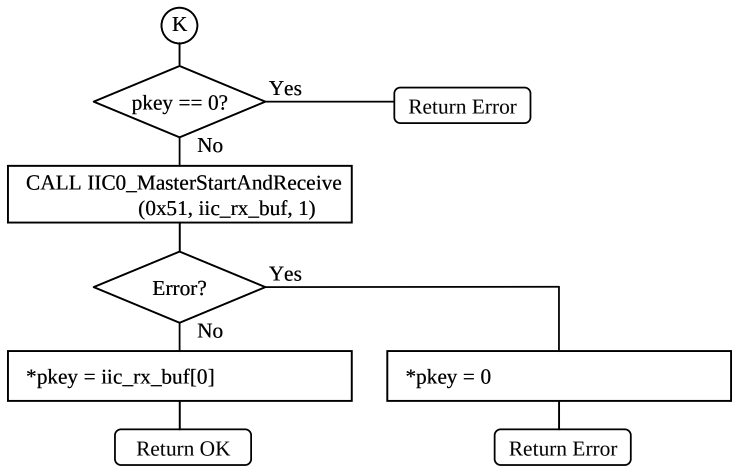
The routine then calls IIC0_MasterStartAndReceive(0x51, iic_rx_buf, 1) to start an IIC transfer to read a byte of data from the LCD module. The **0x51** parameter is the IIC slave address of the LCD module for reading; **iic_rx_buf** is a pointer to an array of bytes to receive the data, and the **1** specifies one byte to receive.

If an error is returned, the key code stored is zero, and an error condition is returned.

If there is no error, the key code returned from the LCD module is in the first location of the buffer. The location pointed to by **pkey** is set to **iic_rx_buf[0]** to return the key code read, and the routine returns a value indicating success.

If no key is available in the LCD module key buffer, the LCD module returns a value of 0 for the key code.

Figure 19. LCD_KeyPoll(*pkey) Routine



3.4.16 IIC0_MasterStartAndReceive(sadr, *rxbuf, rxnum): Start Master, Receive Data From IIC Slave

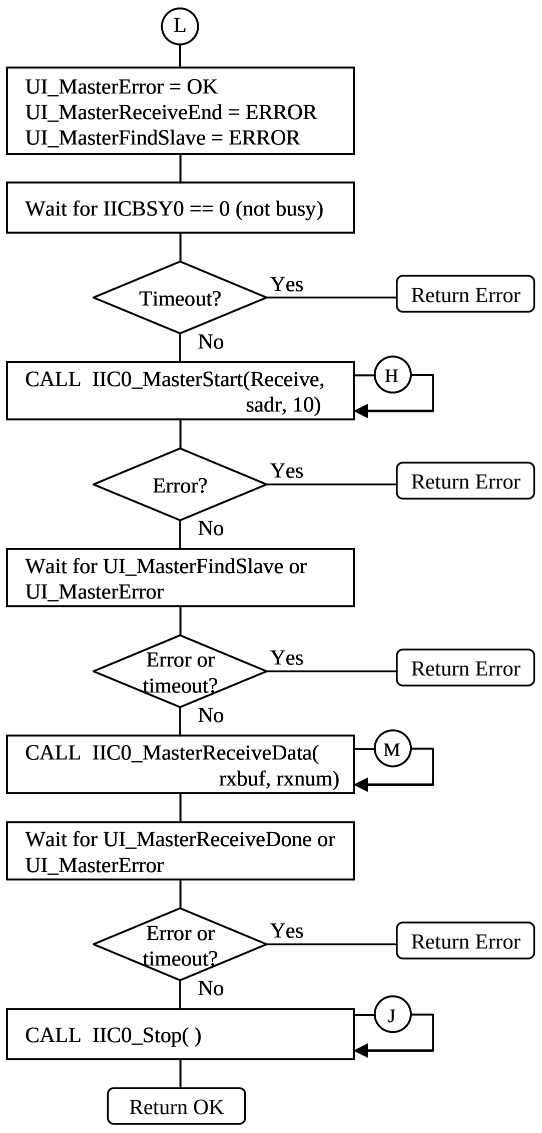
The IIC0_MasterStartAndReceive(sadr, *rxbuf, rxnum) routine is a combination function of IIC0_MasterStart() and IIC0_MasterReceiveData(). This function was not created by Applilet, but is a simple combination of the two routines necessary for receiving IIC data as a master. The **sadr** parameter is the slave address to receive data from, the ***rxbuf** parameter is a pointer to an array of bytes to receive the data read, and the **rxnum** parameter is the number of bytes to receive.

The routine first sets IIC communication flags to default settings; these flags will be affected by IIC0 callback functions during the IIC communication process.

UI_MasterError is set to MD_OK (to indicate no error), UI_MasterReceiveEnd is set to MD_ERROR to indicate that receiving is not done; UI_MasterFindSlave is set to MD_ERROR to indicate that a slave is not found at this point.

The routine then waits for the bus not to be busy as indicated by the IICBSY0 bit in the IICF0 flag register to allow previous operations to complete or in case another master is using the bus.

Figure 20. IIC0_MasterStartAndReceive(sadr, *rxbuf, rxnum) Routine


L
UI_MasterError = OK
UI_MasterReceiveEnd = ERROR
UI_MasterFindSlave = ERROR
Wait for IICBSY0 == 0 (not busy)
Timeout?
Yes
Return Error
No
CALL IIC0_MasterStart(Receive, sadr, 10)
H
Error?
Yes
Return Error
No
Wait for UI_MasterFindSlave or UI_MasterError
Error or timeout?
Yes
Return Error
No
CALL IIC0_MasterReceiveData(rxbuf, rxnum)
M
Wait for UI_MasterReceiveDone or UI_MasterError
Error or timeout?
Yes
Return Error
No
CALL IIC0_Stop( )
J
Return OK

The routine then calls IIC0_MasterStart(Send, sadr, 10) function to start IIC communication as a master for sending data, and specifies the slave address of **sadr**. Upon return, either the IIC0 peripheral has been enabled and the slave address has been written to the IIC0 register for transmission, or the IIC communication channel is busy, which should not happen in this demonstration system.

The routine then waits for either UI_MasterFindSlave to be OK, or for UI_MasterError to be set to an error condition, and will timeout if neither condition occurs within a set time.

The IIC0 peripheral gets an interrupt on the ninth clock bit, which will be handled by the MD_INTIIC0 ISR. If the LCD module has been properly addressed with its slave address, it will respond with an ACK that will set the UI_MasterFindSlave flag. If there is no ACK, the ISR sets the UI_MasterError flag to MD_NACK.

If the slave is not found, IIC0_MasterStartAndReceive() returns an error code. If the slave has been found, then the routine calls IIC0_MasterReceiveData(rxbuf, rxnum) to specify the location and amount of data to receive.

IIC0_MasterReceiveData() stores the buffer address and count in preparation for the interrupt at the end of the first byte received. It then signals the slave to send the first byte.

At this point, after every byte of data has been received, an INTIIC0 interrupt occurs. The MD_INTIIC0 ISR handles the interrupts by storing the byte of data received, by setting UI_MasterReceiveDone if all bytes have been received, or by setting an error if there is no ACK from the slave.

The IIC0_MasterStartAndReceive() routine waits for the UI_MasterReceiveDone condition to be true, or for UI_MasterError to be set to an error condition, and times out if neither condition occurs within a set time.

If an error occurs in transmission, or if the routine times out, an error is returned. Otherwise, the UI_MasterReceiveDone flag was set and the transfer was successful. The routine calls IIC0_Stop() to end the transfer and returns a value indicating success.

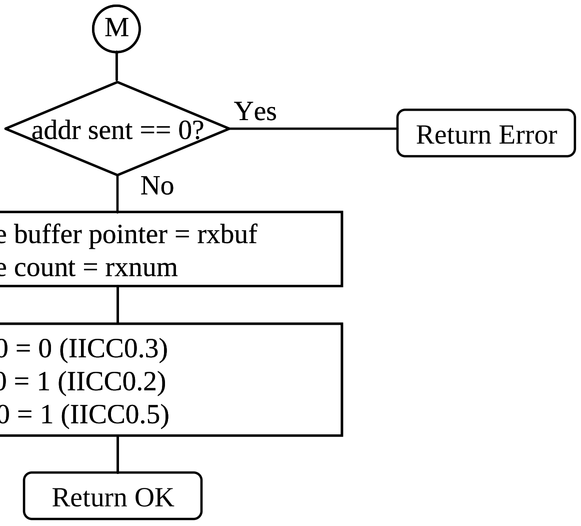
3.4.17 IIC0_MasterReceiveData(*rxbuf, rxnum): Master Receive Data From IIC Slave

The IIC0_MasterReceiveData(*rxbuf, rxnum) function is called after IIC0_MasterStart() has been called and after the UI_MasterFindSlave flag has been set to indicate that the slave is there and ready to send data. The **rxbuf** parameter points to an array of bytes to send and the **rxnum** parameter specifies a count of bytes to send.

The routine checks to see of the “address sent” flag has been set and returns an error if it hasn’t. This condition would result if the routine were called before the INTIIC0 interrupt occurs at the end of slave address transmission.

The routine then sets the buffer address for data to receive to the location pointed to by **rxbuf** and sets the count of bytes to receive to **rxnum**.

Figure 21. IIC0_MasterReceiveData(*rxbuf, rxnum) Routine



At this point, the IIC0 peripheral has automatically asserted a wait condition to pause slave operations until the master is ready. The WTIM0 bit is set to 0 to provide an ACK after eight bits of data sent by the slave. The ACKE0 bit is set to 1 to set the ACK state after a byte is received, and the WREL0 bit is set to 1 to release the wait condition. The slave then begins transmitting the first byte of data.

When this byte has been received by the IIC0 peripheral, the peripheral sets the wait condition again and the INTIIC0 interrupt occurs. The MD_INTIIC0() ISR handles the actual reading of data received, and prepares for the next byte or for ending the transfer.

3.5 Applilet Reference Driver

The Applilet program generator can automatically generate C or assembly language source code to manage peripherals for NEC Electronics microcontroller devices. Refer to Section 5 to find out what version of Applilet is used.

Applilet produces the basic initialization code and main function for the program, the clock initialization code, and the initialization and driver code for the IIC0 and TM00 timer peripherals. After Applilet produces the basic code, users must add additional code to customize the program’s functionality.

This section describes how to configure Applilet to produce code for these peripherals and also lists the files and routines produced. Additional files not generated by Applilet, such as those written for LCD module access, are also listed.

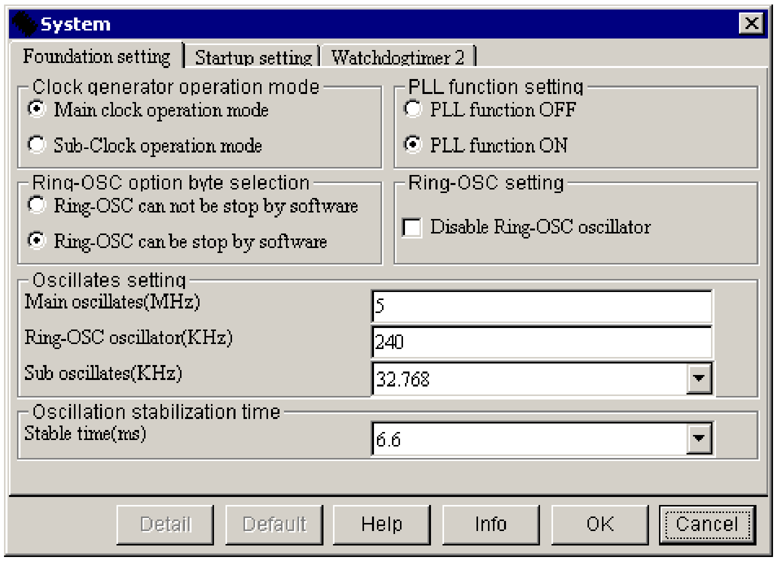
When Applilet is started, a new project file is created and saved as a .prx file. Applilet shows a screen allowing different peripheral blocks to be selected for setup.

3.5.1 Configuring Applilet for Clock Initialization

The **System** box allows you to control clock initialization in the Clock_Init() routine.

1. On the **Foundation Setting** tab, select **Main clock operation mode** to enable operation of the external crystal.
2. Select **PLL function On** to set the clock to use the PLL multiplier.
3. Set the external clock frequency to 5 MHz, which results in a system clock of 20 MHz with PLL on.
4. Verify that **Ring-OSC can be stop by software** is selected; if not, the watchdog timer 2 (WDTM2) cannot be stopped. Watchdog timer 2 would reset the program periodically if not disabled or cleared within a certain interval of time. To make the program code clearer, the watchdog timer is not used.

Figure 22. System Dialog Box



Clock generator operation mode
[✓ Main clock operation mode]
Sub-Clock operation mode

PLL function setting
PLL function OFF
[✓ PLL function ON]

Ring-OSC option byte selection
Ring-OSC can not be stop by software
[✓ Ring-OSC can be stop by software]

Ring-OSC setting
☐ Disable Ring-OSC oscillator

Oscillates setting

Main oscillates(MHz)	5
Ring-OSC oscillator(KHz)	240
Sub oscillates(KHz)	32.768

Oscillation stabilization time

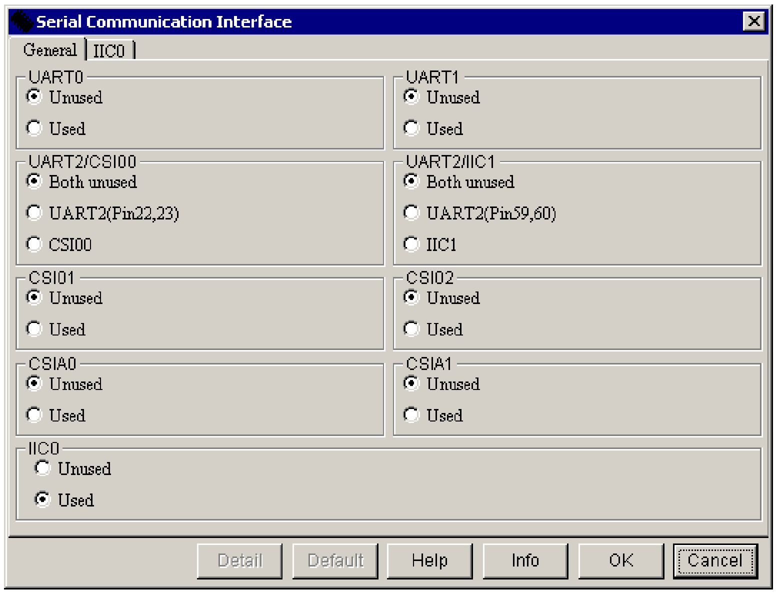
Stable time(ms)	6.6

3.5.2 Configuring Applilet for Operation with the IIC0 Peripheral

The **Serial Communication Interface** dialog box allows you to enable the IIC0 peripheral

1. In the **IIC0** box on the **General** tab, select **Used** to activate the **IIC0** tab, which will allow you to specify settings that will control the code generated for the IIC0_Init() routine and for the routines used to read and write data.

Figure 23. General Tab (Serial Communication Interface Dialog Box)



Serial Communication Interface

General | IIC0

UART0: [✓ Unused] Used

UART1: [✓ Unused] Used

UART2/CSI00: [✓ Both unused] UART2(Pin22,23) CSI00

UART2/IIC1: [✓ Both unused] UART2(Pin59,60) IIC1

CSI01: [✓ Unused] Used

CSI02: [✓ Unused] Used

CSIA0: [✓ Unused] Used

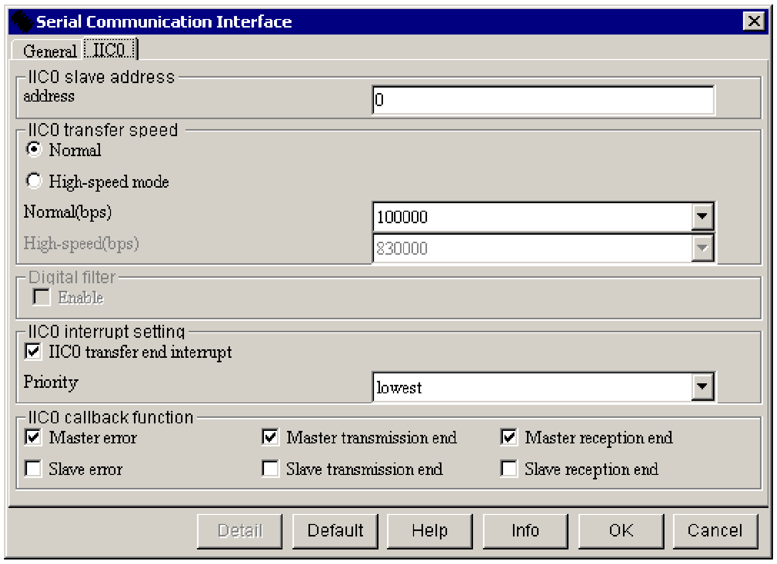
CSIA1: [✓ Unused] Used

IIC0: Unused [✓ Used]

2. The IIC0 peripheral can operate in both slave and master mode or switch between the two. Unless the device is in the middle of a transfer initiated as a master, it is in slave mode and will respond to IIC transmissions from an external master that matches its slave address. This demonstration does not operate in slave mode, so the slave address remains the default 00H.

   In the **IIC0 transfer speed box** on the **IIC0** tab, select **Normal** to activate the list of communication speeds available. Speeds are based on divisions of the peripheral clock, so the available choices will depend on that. For this demonstration, select 100000 (100 Kbps), to be compatible with standard IIC bus rates. Since the digital filter is only available in high-speed mode, that option is disabled.

Figure 24. IIC0 Tab (Serial Communication Interface Dialog Box)


Serial Communication Interface

General | IIC0

IIC0 slave address
address: 0

IIC0 transfer speed
(●) Normal
( ) High-speed mode
Normal(bps): 100000
High-speed(bps): 830000

Digital filter
[ ] Enable

IIC0 interrupt setting
[✓] IIC0 transfer end interrupt
Priority: lowest

IIC0 callback function
[✓] Master error [✓] Master transmission end [✓] Master reception end
[ ] Slave error [ ] Slave transmission end [ ] Slave reception end

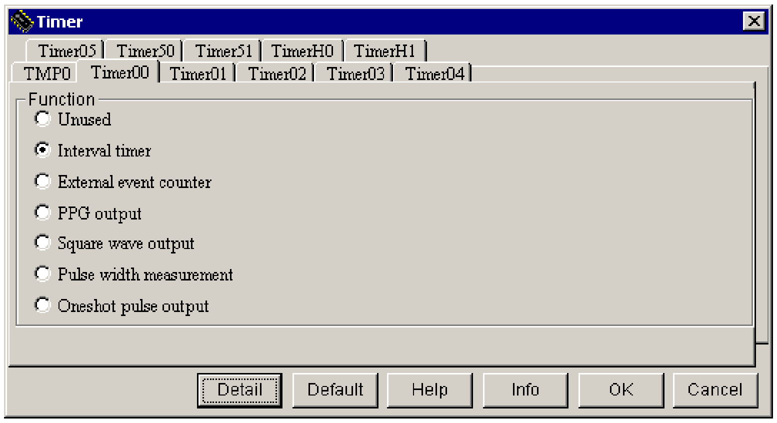
3. In the **IIC0 interrupt setting** box, select **IIC0 transfer end interrupt** and **lowest** priority.

4. Since Applilet generates several standard routines for operation, and has the option to generate blank callback routines in which you may insert code to deal with particular events, this example uses the callback routines to support master communication. Therefore, in the **IIC0 callback function** box, select **Master error**, **Master transmission end**, and **Master reception end**.

3.5.3 Configuring Applilet for Timer 00 (TM00)

The **Timer** dialog box displays a number of tabs showing the available timers. On the **Timer00** tab, select the **Interval timer** to provide a periodic interrupt.

Figure 25. Timer Dialog Box

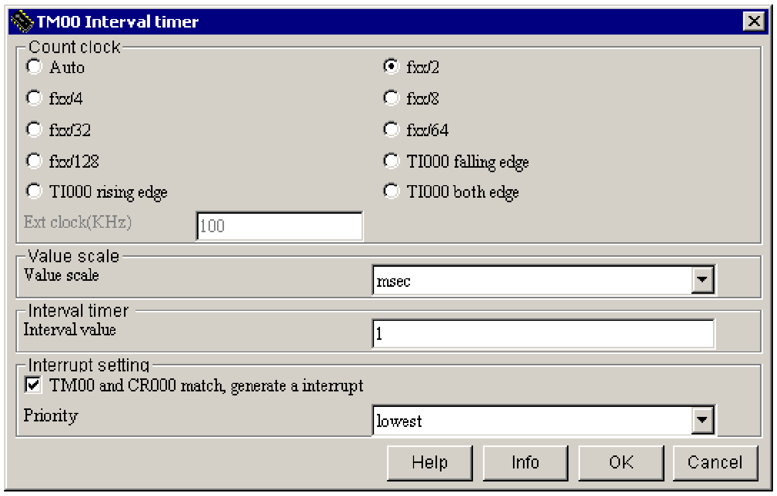

Timer

Timer05 | Timer50 | Timer51 | TimerH0 | TimerH1 |
TMP0 | Timer00 | Timer01 | Timer02 | Timer03 | Timer04 |

Function
- Unused
- [✓ Interval timer]
- External event counter
- PPG output
- Square wave output
- Pulse width measurement
- Oneshot pulse output

1. Click **Detail** to display the **TM00 Interval timer** dialog box.

Figure 26. TM00 Interval Timer Dialog Box


TM00 Interval timer

Count clock
- Auto
- [✓ fxx/2]
- fxx/4
- fxx/8
- fxx/32
- fxx/64
- fxx/128
- TI000 falling edge
- TI000 rising edge
- TI000 both edge

Ext clock(KHz) 100

Value scale
Value scale: msec

Interval timer
Interval value: 1

Interrupt setting
[✓] TM00 and CR000 match, generate a interrupt
Priority: lowest

2. In the **Count clock** box, select **fxx/2** to use a 10 MHz clock for the timer.

3. In the **Value scale** box, select **msec** for milliseconds.

4. In the **Interval time** box, enter **1**.

5. In the **Interrupt setting** box, select **TM00 and CR000 match, generate an interrupt** to have Timer 00 generate an interrupt every millisecond. The interrupt will be used to count down a millisecond timer for timing delays.

3.5.4 Generating Code With Applilet

At this point, select the **Generate code** option. Applilet will show the peripherals and functions to be generated, and allow you to select a directory to store the source code.

When you press the **Generate** button, Applilet creates the code in several C language source files (extension .c), C header files (extension .h), and assembly language source and header files (extensions .s and .inc), and shows the list of files created.

1. To support the initial startup code, Applilet generates the crte.s assembly source file. For clock initialization, Applilet generates system.inc and system.s. The SystemInit() function is generated in systeminit.c.

2. To support the IIC0 peripheral, Applilet generates serial.h, serial.c, and serial_user.c (described in the next section).

3. To support the Timer 00 peripheral, Applilet generates timer.h, timer.c and timer_user.c.

4. Several other files are generated, including a main.c file with a blank main function, and a link directive file, 850.dir, which controls the linking process.

3.5.5 Applilet-Generated Files

For the demonstration program, Applilet generates the source files described in the following table.

Table 3. Description of Applilet-Generated Source Files

File	Function
Macrodriver.h	General header file for Applilet-generated programs
Crte.s	Reset vector, program startup code
System.inc	Assembly-language header for system.s
System.s	Assembly source for Clock_Init() routine
System_user.s	Empty file (would contain code for System interrupt if used)
Systeminit.c	SystemInit() routine for peripheral initialization
Main.c	The main program routine
Inttab.s	Interrupt vectors with RETI for unused interrupts
Serial.h	Header file for serial.c
Serial.c	CSI00 functions generated by Applilet
Serial_user.c	Callback functions for UART1 and CSI00, for user code
Timer.h	Header file for timer.c

File	Function
Timer.c	Timer 00 functions
Timer_user.c	User code for INTTM000 interrupt
850.dir	Link directive file

3.5.6 Applilet-Generated Files for IIC0 Operation

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

3.5.6.1 Serial.h

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

2. We have added external declarations for the variables used to signal IIC0 states, such as UI_MasterSendEnd, and for the IIC0_MasterStartAndSend() and IIC0_MasterStartAndReceive() functions.

3.5.6.2 Serial.c

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

1. The **void IIC0_Init(void);** routine initializes the IIC0 peripheral as specified in the Applilet IIC0 detail dialog.

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

3. The **MD_STATUS IIC0_SlaveStart(UCHAR adr);** routine prepares the IIC0 to respond to slave mode operation at the specified slave address. This routine is not used in the demonstration program.

4. The **void IIC0_Stop(void);** routine stops the IIC0 peripheral.

5. The **MD_STATUS IIC0_MasterSendData(UCHAR* txbuf , UINT txnum);** routine is called after IIC0_MasterStart( ) has started a "Send" transfer. The **txbuf** parameter points to an array of bytes to send, and the **txnum** parameter specifies the number of bytes to send. This routine sets the buffer pointer and count, and sends the

first byte in the buffer. Further bytes are sent in the MD_INTIIC0( ) interrupt service routine.

6. The **MD_STATUS IIC0_MasterReceiveData(UCHAR* rxbuf, UINT rxnum);** routine is called after IIC0_MasterStart( ) has started a "Receive" transfer. The **rxbuf** parameter specifies the address of a buffer to store received characters, and **rxnum** specifies the number of characters to receive.

7. The **MD_STATUS IIC0_SlaveSendData(UCHAR* txbuf, UINT txnum);** routine is called to send data when operating as a slave. This routine is not used in the demonstration program.

8. The **MD_STATUS IIC0_SlaveReceiveData(UCHAR* rxbuf, UINT rxnum);** routine is called to receive data when operating as a slave. This routine is not used in the demonstration program.

9. The **__interrupt void MD_INTIIC0(void);** IIC0 interrupt service routine, invoked by the INTIIC0 interrupt, calls either IIC0_MasterHandler( ) or IIC0_SlaveHandler(), depending on the IIC0 master or slave state.

10. The **MD_STATUS IIC0_SlaveHandler(void);** routine, called by MD_INTIIC0( ), handles IIC0 interrupts in slave mode.

11. The **MD_STATUS IIC0_MasterHandler(void);** routine, called by MD_INTIIC0(), handles IIC0 interrupts in master mode.

3.5.6.3 Serial_user.c Source File

The **serial_user.c** source file contains stub functions for user code. The functions are empty on code generation to allow you to add application-specific code. We have added the following variables to enable subroutines to check the state of IIC0 communication.

Table 4. **IIC0 Communication Subroutines**

Name	Description
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

1. The **void CALL_IIC0_MasterError(MD_STATUS flag);** routine is called by the IIC0_MasterHandler( ) routine when an error is detected. Code was added to store the **flag** parameter in UI_MasterError.

2. The **void CALL_IIC0_MasterReceiveEnd(void);** routine is called by IIC0_MasterHandler( ) when the receive count reaches zero. Code was added to set the UI_MasterReceiveEnd flag to MD_OK.

3. The **void CALL_IIC0_MasterSendEnd(void);** routine is called by IIC0_MasterHandler( ) when the last transmit byte has been sent. Code was added to set the UI_MasterSendEnd flag to MD_OK.

4. The **void CALL_IIC0_SlaveAddressMatch(void);** routine is called by IIC0_SlaveHandler( ) when the slave address received in slave mode matches the set slave address for the IIC0 peripheral. This routine is not used in the demonstration program, and is left with no executable code.

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

The following routines were not generated by Applilet, but were written for this demonstration program.

1. The **MD_STATUS IIC0_MasterStartAndSend(UCHAR sadr, UCHAR* txbuf, UINT txnum);** routine combines the IIC0_MasterStart() and IIC0_MasterSendData() functions, since these are done together to accomplish sending data to a particular slave. First IIC0_MasterStart(Send, sadr, 10) is called to set up a sending transfer to the specified slave address; the routine then waits for UI_MasterFindSlave to be set, to indicate that the slave has responded.

   Then IIC0_MasterSendData(txbuf, txnum) is called, to send the specified data. The routine then waits for the UI_MasterSendDone flag to be set, to indicate that the transfer has been completed. The UI_MasterError flag is monitored while waiting, to check for error conditions in the sending of slave address or transfer of data.

2. The **MD_STATUS IIC0_MasterStartAndReceive(UCHAR sadr, UCHAR* rxbuf, UINT rxnum);** routine combines the IIC0_MasterStart() and IIC0_MasterReceiveData() functions, since these are done together to accomplish receiving data from a particular slave. First IIC0_MasterStart(Receive, sadr, 10) is called to set up a receiving transfer from the specified slave address; the routine then waits for UI_MasterFindSlave to be set, to indicate that the slave has responded.

Then IIC0_MasterReceiveData(rxbuf, rxnum) is called, to receive the specified number of bytes to the specified buffer. The routine then waits for the UI_MasterReceiveDone flag to be set, to indicate that the transfer has been completed. The UI_MasterError flag is monitored while waiting, to check for error conditions in the sending of slave address or transfer of data.

3.5.7 Files for LCD Module Routines

The following files were written for Matrix Orbital LCD module handling.

3.5.7.1 Lcd_mo.h Header File

The **lcd_mo.h** header file contains declarations of functions for LCD module access and definitions of maximum column and row values for the selected display.

3.5.7.2 Lcd_mo.c Source File

The **lcd_mo.c** source file contains the following functions for LCD module access:

1. The **MD_STATUS LCD_Init(void);** routine initializes the LCD module by clearing the display, setting key buffering, and displaying initial messages.
2. The **MD_STATUS LCD_ClearDisplay(void);** routine clears the LCD screen.
3. The **MD_STATUS LCD_AutoTransmitKeyPressesOff(void);** routine sets the LCD module to buffer key presses rather than sending immediately on key press.
4. The **MD_STATUS LCD_PutStr(int col, int row, char *str);** routine writes a string of characters to the LCD at the specified column and row address.
5. The **MD_STATUS LCD_KeyPoll(unsigned char *pkey);** routine reads the LCD module to see if a key press has occurred.

3.6 Demonstration Platform

A demonstration platform was chosen from the NEC Electronics America development tools available at the time this document was prepared. In some cases, you may be able to duplicate the same hardware using standard off-the-shelf components along with an NEC Electronics microcontroller of your choice.

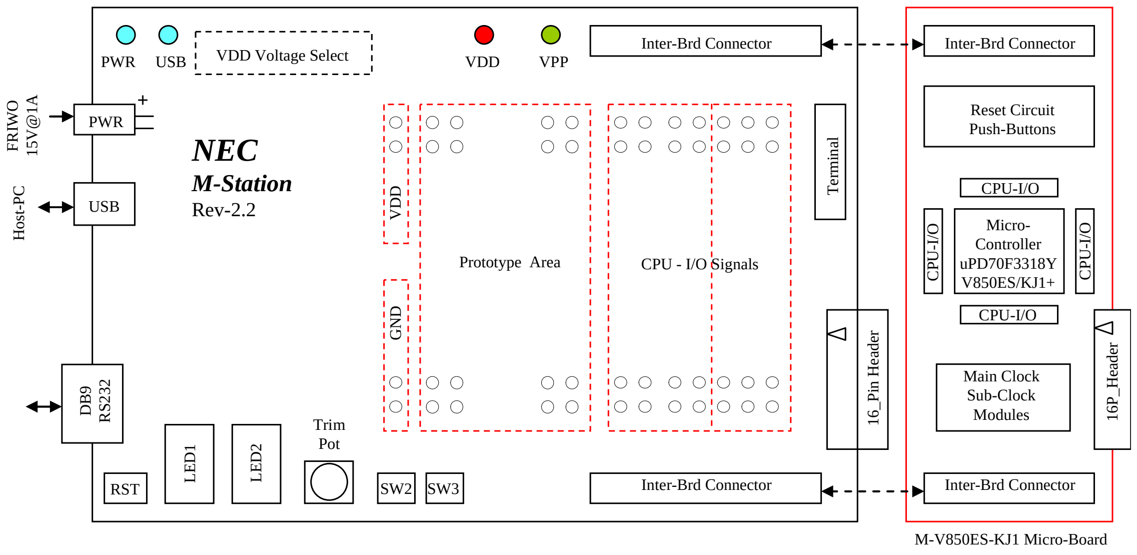
3.6.1 Resources

To demonstrate the program, we used the following resources:

- M-V850ES-KJ1 micro-board with 32-bit µPD70F3318Y (V850ES/KJ1+) MCU
- M-Station II evaluation system with
  - TB1 terminal block with SCL, SDA, and GND signals
  - M-Station II +15V power supply jumpered to the TB1 terminal block
- Matrix Orbital LK204-25-GW-V LCD module
  - 4-line, 20-character/line LCD
  - Keypad interface supporting a 5 × 5 matrix of keys
  - 9 to 15V voltage range
  - Breadboard cable for connection to M-Station
  - 15-button keypad for key input

For detailed information about this hardware, refer to the appropriate user's manual, available from NEC Electronics America upon request. For complete information about the Matrix Orbital LK204-25, refer to the manufacturer's user's manual.

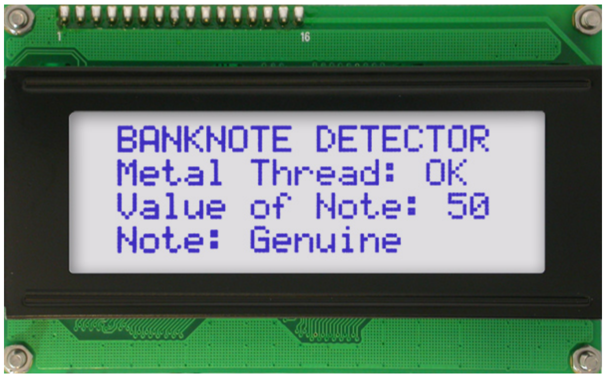
Figure 27.



The M-V850ES-KJ1 micro-board attaches to the M-Station II through two 100-pin connectors. When the micro-board is attached, the µPD70F3318Y microcontroller's IIC pins automatically connect to the terminal block on the M-Station II. The SCL and SDA lines have 2 kΩ pull-up resistors mounted on the M-Station II side of the communication channel.

The LK202-25-GW-V Matrix Orbital LCD module connects to the M-Station II terminal block through a cable. The M-Station terminal block has connections for SCL, SDA, and GND. The M-Station II can be set to connect +15V to the terminal block, and so can supply power and ground, as well as communication signals to the LCD module.

Figure 28. Matrix Orbital Display



The Matrix Orbital LK204-25-GW-V display has been modified from the factory default settings for use with IIC communication:

- ♦ Standard communication channel jumpers for RS-232 removed
- ♦ Communication channel jumpers for IIC communication inserted
- ♦ 2.2K pull-up resistors added to SCL and SDA lines
- ♦ 15-button keypad added to keypad connector, to demonstrate key press input

3.6.2 Demonstration of Program

If the hardware has been configured correctly, and the µPD70F3318Y microcontroller has been programmed with the demonstration program code, you can initiate the demonstration as follows.

1. Connect the Matrix Orbital LK204-25-GW-V to the M-Station II terminal block with the breadboarding cable (see hardware block diagram section).

2. Power on the M-Station II.

3. Power on V_{DD} at 5V to the M-V850E-KJ1 micro-board. At this point, the program will run and try to access the LCD module. Since the LCD module is not powered, it will receive errors. The program will be in the state of polling for keys, getting an error, displaying the key poll error (which will not be seen), clearing the display, and polling again.

4. Connect +15V power to the LK204-25-GW-V by connecting JP2 pins 1, 2, and 3 together on the M-Station II. You will see the default power-on display of the module, and then the demonstration program will clear the display.

5. Press **Reset** on the M-Station II to start the application program. Observe the LCD, which should be showing:

   ```
   NEC Electronics V850ES/KJ1
   SPI/IIC Application Note
   ```

6. Press four keys on the keypad. The LCD will display the key code values on successive lines, as in this example:

   ```
   Keycode = 48H
       Keycode = 57H
           Keycode = 51H
               Keycode = 55H
   ```

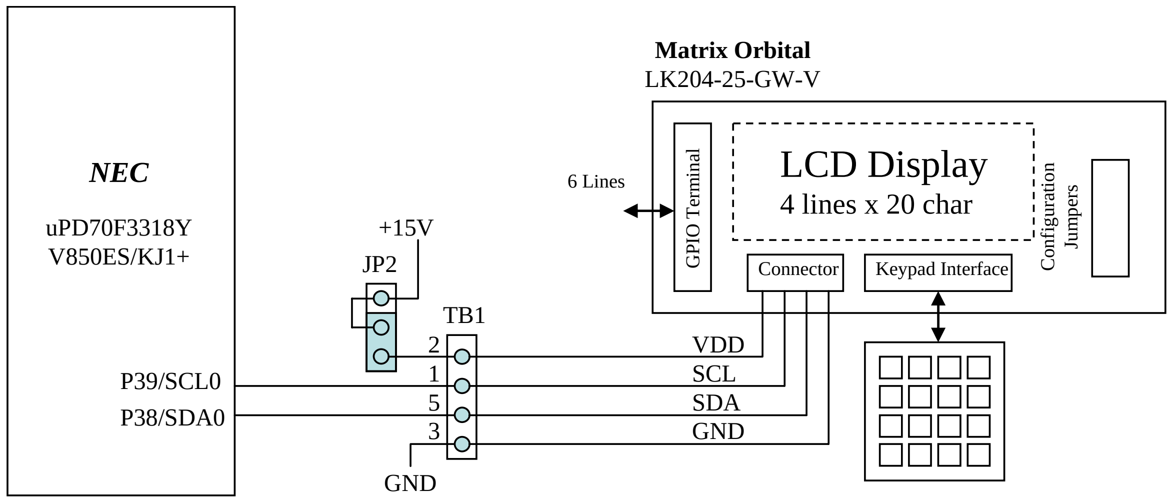
   If you press the keypad a fifth time, the display will clear and display the new value:

   ```
   Keycode = 58H
   ```

3.7 Hardware Block Diagram

The connections of SCL and SDA from the µPD70F3318Y MCU to the Matrix Orbital LCD module are shown in the figure below.

Figure 29. Connections from the M-Station-II Terminal Block to the LCD Module



To have P39/SCL0 of the µPD70F3318Y brought to the 100-pin connectors, it is necessary to close shorting block SB7 on the M-V850ES-KJ1, and have SB6 open. The factory default is to have SB7 open and SB6 closed.

The P39/SCL0 and P38/SDA0 lines of the µPD70F3318Y are brought to the 100-pin connector between the M-V850ES-KJ1 micro-board and the M-Station II. They are connected to the SCL and SDA lines on the M-Station II by shorting blocks, which are connected by default. SB11 on the M-Station II connects P39/SCL0 to SCL, and SB12 on the M-Station II connects P38/SDA0 to SDA.

SCL and SDA on the M-Station II have 2 kΩ pull-up resistors attached, to provide termination for the signals on the M-Station side, and are brought to terminal block TB1, pins 1 and 5, respectively.

To have the M-Station II supply power and ground for the LCD module, connect pins JP2.1 and JP2.2 with wire wrap or solder. This will connect the M-Station +15V supply (JP2.1) to the V_{IN} supply for the LIN transceiver (JP2.2), which is the normal connection of JP2. Then, placing a jumper between JP2.2 and JP2.3 (VBAT signal, pin 2 on TB1) will provide +15V to the V_{DD} line of the LK204-25-GW-V. The –V designation on the LCD module part number indicates a wide voltage range of 9 to 15V. This jumper should only be connected after power has been supplied to the µPD70F3318Y microcontroller.

GND is provided on TB1 pin 3.

The connection of signals from the µPD70F3318Y to the Matrix Orbital LK204-GW-V is shown in Table 5.

Table 5.

MCU Pin	SB	M-V850ES-KJ1 Connector	M-Station II Connector	SB	TB1 Pin	Signal	LK204-25-GW-V Connector Pin
					2	+15V	1
P39/SCL0	SB7	P1.11	J1.11	SB11	1	SCL	2
P38/SDA0	--	P1.12	J1.12	SB12	5	SDA	3
					3	GND	4

3.8 Software Modules

The following files make up the software modules for the demonstration program. The table below shows which files were generated by Applilet, and which of those needed modification to create the demonstration program.

The listings for these files are located in Section 5.

Table 6.

File	Purpose	Generated By Applilet	Modified By User
Main.c	Main program	Yes	Yes
Macrodriver.h	General definitions used by Applilet	Yes	No
Crte.s	Reset vector, program startup code	Yes	No
Inttab.s	Interrupt vectors for non-used interrupts (RETI)	Yes	No
System.inc	Clock-related definitions	Yes	No
System.s	Clock_Init() function	Yes	Yes[Note 1]
System_user.c	File for System interrupt	Yes	No
Systeminit.c	SystemInit() and hdwinit() functions	Yes	No
Serial.h	IIC0-related definitions	Yes	Yes[Note 2]

File	Purpose	Generated By Applilet	Modified By User
Serial.c	IIC0-related functions	Yes	Yes[Note 2]
Serial_user.c	User code in IIC0 callback routines	Yes	Yes[Note 2]
Timer.h	Timer-related definitions	Yes	Yes[Note 3]
Timer.c	Timer-related functions	Yes	No
Timer_user.c	User code for Timer interrupt service	Yes	Yes[Note 3]
850.dir	Link directive file	Yes	Yes[Note 4]
Lcd_mo.h	LCD module definitions	No	--
Lcd_mo.c	LCD module functions	No	--

Notes:

1. System.s was modified to correct an error in the Clock_Init() routine which resulted in excessive time spent waiting for the PLL to stabilize.

2. Serial.h was modified to add the declarations of new functions IIC0_MasterStartAndSend() and IIC0_MasterStartAndReceive(). Serial.c was modified to correct the order of setting of P3 in IIC0_Init(), and to force a stop condition after IICE0 setting in IIC0_MasterStart(). Serial_user.c had variables added, code inserted in callback functions to set these variables, and the routines IIC0_MasterStartAndSend() and IIC0_MasterStartAndReceive() added.

3. Timer.h was modified to add the declaration of new functions defined in timer_user.c. Timer_user.c was modified to add the code to handle the periodic INTTM000 interrupt in the MD_INTTM000() routine, which counts down the millisecond timer, and to add the SetMsecTimer() and CheckMsecTimer() routines.

4. Note 4: As produced by Applilet, the 850.dir link directive file did not specify an SCONST segment for initialized string constants. Since the demonstration program uses string constants, an SCONST segment directive was added to 850.dir.

4. DEVELOPMENT TOOLS

The following software and hardware tools were used in the development of this document.

Table 7. Software Tools

Tool	Version	Comments
Applilet	E1.46c	Source code generation tool for NEC Electronics microcontrollers
V850ESKX1H.mcu	V1.33	Applilet MCU configuration for µPD70F3318Y (V850ES/KJ1+)
PM Plus	V6.10	Project manager for program compilation and linking
CA850	V3.00	C compiler, assembler, linker for NEC Electronics V850ES microcontrollers
DF3318Y.800	V1.01	Device file for µPD70F3318Y (V850ES/KJ1+) microcontrollers

Table 8. Hardware Tools

Tool	Version	Comments
M-Station 2	V2.1E	Base platform for NEC Electronics America micro-board demonstration
M-V850ES-KJ1	V1.0	Micro-board for V850ES/KJ1+ with µPD70F3318YGJ CPU

5. SOFTWARE LISTINGS

This application program is based on specific files and a number that are used in other application notes such as the one titled “NEC Electronics CSI-to-SPI Peripheral Communication”. For this reason the files are listed in two separate sections here.

Since the Applilet program generation tool was used for both programs, some filenames, such as serial.h, serial.c, or serial_user.c, are found in each program. At first glance, these files may seem to be identical, because Applilet may place a large amount of similar code in each version of the file.

However, there are differences in initialization values for registers, or differences in generated code, depending on the options selected in Applilet. The files listed in the sections for each demonstration program may be different from the same-named files in other sections.

5.1 Files for IIC To LCD Module Demonstration Program

5.1.1 Main.c

```
/*
*****************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+,
V850ES/KG1+,
**  V850ES/KF1+, V850ES/KE1+ 32-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:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*****************************************************************************
*/
/*
** ***************************************************************************
** Include files
** ***************************************************************************
*/
```

```
#include "macrodriver.h"
#include "timer.h"
#include "serial.h"

/* added include for LCD panel/keypad */
#include "lcd_mo.h"

/*
** ***************************************************************************
** MacroDefine
** ***************************************************************************
*/
char keystr[64];  /* buffer to write key code string */

/*
**---------------------------------------------------------------------------
**
**  Abstract:
**          main function
**
**  Parameters:
**          None
**
**  Returns:
**          None
**
**---------------------------------------------------------------------------
*/

void  main( void )
{
UCHAR keyval;           /* key code returned by panel */
UCHAR keycol;     /* column to display string */
UCHAR keyrow;     /* row to display string */
MD_STATUS status; /* status from operations */

      /* initialize IIC0 interface */
      IIC0_Init();

      /* initialize the LCD display and keypad */
      LCD_Init();

      TM00_Start();           /* start timer for millisecond counting */
      SetMsecTimer(500);      /* set for 1/2 second polling */
      keycol = 1;             /* set initial string display location */
      keyrow = 1;             /* for row 1, column 1 */

      while (1) {
            if (CheckMsecTimer()) {
                  /* timer expired, poll keypad */
                  status = LCD_KeyPoll(&keyval);
                  if (status != MD_OK) {
                        /* error on polling */
                        LCD_PutStr(1, 4, "KeyPoll Error");  /* display error
*/
```

```
                        SetMsecTimer(2000);
                        while (!CheckMsecTimer());                /* for 2
sec */
                        LCD_ClearDisplay();                             /*
then clear display */
                        SetMsecTimer(500);                              /*
poll again in 500 msec */
                  } else {
                        /* no error on polling, see if we have a key */
                        if (keyval == 0x00) {
                              /* no key, set up to poll again in 500 msec */
                              SetMsecTimer(500);
                        } else {
                              /* we have a key, handle it */
                              if (keyrow == 1) {
                                    LCD_ClearDisplay();     /* clear the
display when we write to row 1 */
                              }
                              /* format string for display */
                              sprintf(keystr, "Keycode = %02XH", keyval);
                              /* send string to display at keycol/keyrow
position */
                              LCD_PutStr(keycol, keyrow, keystr);
                              /* change position for next line */
                              keycol += 2;
                              keyrow += 1;
                              if (keyrow > LCD_ROW_NUM) {
                                    /* if we pass the last row, start at the
top again */
                                    keycol = 1;
                                    keyrow = 1;
                              }
                              SetMsecTimer(1);  /* if key pressed, poll again
quickly */
                        }
                  } /* end check status of polling */

            } /* end of CheckMsecTimer */

      } /* end of while(1) loop */

} /* end of main() */
```

5.1.2 Lcd_mo.h

```
/*
*****************************************************************************
**
**
**  This file was created for the NEC V850ES SPI/IIC Application Note
**
**  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_mo.h
**  Abstract :    This file implements header for lcd_mo.c
**
**  Device:    uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*****************************************************************************
**
*/
#ifndef     _LCD_MO_H_
#define     _LCD_MO_H_
/*
*****************************************************************************
**
**  MacroDefine
*****************************************************************************
**
*/

/* LCD display defines */
#define LCD_ROW_NUM           4           /* number of rows (1 - 4) */
#define LCD_COL_NUM           20          /* number of columns (1 - 20) */

/* LCD functions */
MD_STATUS LCD_Init(void);                                             /*
init LCD display/keypad */
MD_STATUS LCD_ClearDisplay(void);                               /* clear
LCD display screen */
MD_STATUS LCD_AutoTransmitKeyPressesOff(void);            /* set for key
buffering */
MD_STATUS LCD_PutStr(int col, int row, char *str);        /* write string
to display */
MD_STATUS LCD_KeyPoll(unsigned char *pkey);                     /* check if
key press available */

#endif /* _LCD_MO_H_ */
```

5.1.3 Lcd_mo.c

```
/*
*****************************************************************************
**
**
**  This file was created for the NEC V850ES SPI/IIC Application Note
**
**  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_mo.c
**  Abstract :    This file implements functions for IIC LCD display/keyboard
**
**  Device:    uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*****************************************************************************
**
*/
/*
*****************************************************************************
**
**  Include files
*****************************************************************************
**
*/
#include "macrodriver.h"
#include "serial.h"           /* for IIC routines */
#include "lcd_mo.h"           /* includes for this file */

/* local definitions for LCD panel IIC access */
#define LCD_IIC_WADR    0x50  /* LCD display IIC slave address for write */
#define LCD_IIC_RADR    (LCD_IIC_WADR + 1)      /* slave address for read */

/* buffers for IIC0 operation */
unsigned char iic_tx_buf[16]; /* for this app note, only need four locations
*/
unsigned char iic_rx_buf[16]; /* really only need one character here */

/*
**---------------------------------------------------------------------------
--
**  Abstract:
**    Function to do LCD display/keypad Initialization
**
**  Parameters: None
**  Returns:
**    MD_OK if initialize is successful, MD_ERROR if fails
**---------------------------------------------------------------------------
--
*/
MD_STATUS LCD_Init(void)
{
MD_STATUS status;

      /* clear the display */
      status = LCD_ClearDisplay();

      /* set Auto Transmit Key Presses Off */
      if (status == MD_OK) {
            status = LCD_AutoTransmitKeyPressesOff();
      }

      /* display text messages */
      if (status == MD_OK) {
            status = LCD_PutStr(1, 1, "NEC V850ES/KJ1");
      }

      if (status == MD_OK) {
            status = LCD_PutStr(1, 2, "SPI/IIC App Note");
      }
      return status;
}

/*
**-------------------------------------------------------------------------
--
**  Abstract:
**    Function to clear the LCD display
**
**  Parameters: None
**  Returns:
**    MD_OK if operation is successful, MD_ERROR if fails
**-------------------------------------------------------------------------
--
*/
MD_STATUS LCD_ClearDisplay(void)
{
MD_STATUS status;

      iic_tx_buf[0] = 0xFE;   /* special character */
      iic_tx_buf[1] = 0x58;   /* command = Clear Display */

      /* transmit two bytes ghrough IIC0 */
      status = IIC0_MasterStartAndSend(LCD_IIC_WADR, iic_tx_buf, 2);
      return status;
}

/*
**-------------------------------------------------------------------------
--
**  Abstract:
**    Function to turn automatic transmit of key presses off, so can poll via
IIC0
**
**  Parameters: None
**  Returns:
**    MD_OK if operation is successful, MD_ERROR if fails
**-------------------------------------------------------------------------
--
*/
MD_STATUS LCD_AutoTransmitKeyPressesOff(void)
{
MD_STATUS status;

      iic_tx_buf[0] = 0xFE;   /* special character */
      iic_tx_buf[1] = 0x4F;   /* command = Auto Transmit Key Presses Off */

      /* transmit two bytes ghrough IIC0 */
      status = IIC0_MasterStartAndSend(LCD_IIC_WADR, iic_tx_buf, 2);
      return status;
}

/*
**----------------------------------------------------------------------------
--
**  Abstract:
**    Function to write a string to LCD panel display
**
**  Parameters:
**    int col      character position (1 - LCD_COL_NUM) to start string
**    int row      line (1 - LCD_ROW_NUM) to display string
**      char *str    pointer to string to display, null terminated
**  Returns:
**      MD_OK        if string displayed properly
**    MD_ERROR     if error in IIC communication
**    MD_ARGERROR  if col/row are out of display range
**
**----------------------------------------------------------------------------
--
*/
MD_STATUS LCD_PutStr(int col, int row, char *str)
{
MD_STATUS status = MD_OK;
unsigned int len = strlen(str);

      if (len == 0) {
            return MD_OK;                /* skip writing empty strings */
      }
      if ( (col == 0) || (col > LCD_COL_NUM) || (row == 0) || (row >
LCD_ROW_NUM) ) {
            return MD_ARGERROR;          /* starting position out of range
*/
      }
      /* issue command to set cursor position */
      iic_tx_buf[0] = 0xFE;                    /* special character */
      iic_tx_buf[1] = 0x47;                    /* Command - Set Cursor */
      iic_tx_buf[2] = (unsigned char)col; /* column */
      iic_tx_buf[3] = (unsigned char)row; /* row */
      status = IIC0_MasterStartAndSend(LCD_IIC_WADR, iic_tx_buf, 4);    /*
send four bytes */

      /* if cursor set ok, send the string */
      if (status == MD_OK) {
            /* send the string */
            status = IIC0_MasterStartAndSend(LCD_IIC_WADR, (unsigned char
*)str, len);
      }
      return status;
}
```

```
/*
**----------------------------------------------------------------------------
--
**  Abstract:
**    Function to poll LCD display keypad for key press
**
**  Parameters:
**    pkey = pointer to unsigned character
**  Returns:
**    MD_OK if operation is successful, MD_ERROR if fails,
**    MD_ARGERROR if pointer is null
**    *pkey set to 00 if error, keycode read otherwise
**      if no key press is in the buffer, display returns 00 for key code
**----------------------------------------------------------------------------
--
*/
/* poll LCD display keypad for key pressed */
MD_STATUS LCD_KeyPoll(unsigned char *pkey)
{
MD_STATUS status = MD_OK;

      if (pkey == 0) {
            return MD_ARGERROR;           /* pointer is invalid */
      }

      /* read one character from keypad to receive buffer */
      status = IIC0_MasterStartAndReceive(LCD_IIC_RADR, iic_rx_buf, 1);

      if (status == MD_OK) {
            *pkey = iic_rx_buf[0];  /* read successful, return key code if
any */
      } else {
            *pkey = 0;                    /* return no key if error */
      }
      return status;
}
```

5.1.4 Inttab.s

```
--/*
--
****************************************************************************
--**
--**  This device driver was created by Applilet for the V850ES/KJ1+,
V850ES/KG1+,
--**  V850ES/KF1+, V850ES/KE1+ 32-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 : inttab.s
--**  Abstract : This file implements interrupt vector table
--**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
--**
--
****************************************************************************
--*/

--INT vector

--------------------------------------------------------------------
--    variable initiate
--------------------------------------------------------------------

      --.section "RESET", text
      --jr      __start

      .section "NMI", text                --nmi pin input
      reti

      .section "INTWDT1", text            --WDT1 OVF nonmaskable
      reti

      .section "INTWDT2", text            --WDT2 OVF nonmaskable
      reti

      .section "TRAP00", text             --TRAP instruction
      .globl      __trap00
__trap00:
      reti

      .section "TRAP10", text             --TRAP instruction
      .globl      __trap01
__trap01:
      reti

      .section "ILGOP", text              --illegal op code
      .globl      __ilgop
__ilgop:
      reti


      .section "INTWDTM1", text           --WDT1OVF maskable
      reti

      .section "INTP0", text              --INTP0 pin
      reti

      .section "INTP1", text              --INTP1 pin
      reti

      .section "INTP2", text              --INTP2 pin
      reti
```

```
.section "INTP3", text              --INTP3 pin
reti

.section "INTP4", text              --INTP4 pin
reti

.section "INTP5", text              --INTP5 pin
reti

.section "INTP6", text              --INTP6 pin
reti

.section "INTTM001", text           --TM00 and CR001 match
reti

.section "INTTM010", text           --TM01 and CR010 match
reti

.section "INTTM011", text           --TM01 and CR011 match
reti

.section "INTTM50", text            --TM50 and CR50 match
reti

.section "INTTM51", text            --TM51 and CR51 match
reti

.section "INTCSI00", text           --CSI00 transfer complete
reti

.section "INTCSI01", text           --CSI01 transfer complete
reti

.section "INTSRE0", text            --UART0 reception error occurence
reti

.section "INTSR0", text             --UART0 reception completion
reti

.section "INTST0", text             --UART0 translation completion
reti

.section "INTSRE1", text            --UART1 reception error occurence
reti

.section "INTSR1", text             --UART1 reception completion
reti

.section "INTST1", text             --UART1 translation completion
reti

.section "INTTMH0", text            --TMH0 and CMP00/CMP01 match
reti
```

```
      .section "INTTMH1", text            --TMH1 and CMP10/CMP11 match
      reti

      .section "INTCSIA0", text           --CSIA0 transfer completion
      reti

      .section "INTAD", text              --AD conversion end
      reti

      .section "INTKR", text              --key return interrupt
      reti

      .section "INTWTI", text             --watchtimer interval
      reti

      .section "INTWT", text              --watchtimer referemce time
      reti

      .section "INTBRG", text             --watchtimer counter BRG and PRSCM
match
      reti

      .section "INTTM020", text           --TM02 and CR020 match
      reti

      .section "INTTM021", text           --TM02 and CR021 match
      reti

      .section "INTTM030", text           --TM03 and CR030 match
      reti

      .section "INTTM031", text           --TM03 and CR031 match
      reti

      .section "INTCSIA1", text           --CSIA1 transfer completion
      reti

      .section "INTTM040", text           --TM04 and CR040 match
      reti

      .section "INTTM041", text           --TM04 and CR041 match
      reti

      .section "INTTM050", text           --TM05 and CR050 match
      reti

      .section "INTTM051", text           --TM05 and CR051 match
      reti

      .section "INTCSI02", text           --CSI02 transfer completion
      reti

      .section "INTSRE2", text            --UART2 reception error occurence
      reti
      .section "INTSR2", text             --UART2 reception completion
      reti

      .section "INTST2", text             --UART2 translation completion
      reti

      .section "INTIIC1", text            --IIC1 transfer completion
      reti

-- end of file
```

5.1.5 Systeminit.c

```
/*
*****************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+,
V850ES/KG1+,
**  V850ES/KF1+, V850ES/KE1+ 32-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 initiate
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*****************************************************************************
*/
/*
** **************************************************************************
** Include files
** **************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"
#include "serial.h"
/*
** **************************************************************************
** MacroDefine
** **************************************************************************
*/
extern unsigned long _S_romp;

/*
```

```
**---------------------------------------------------------------------------
**
**  Abstract:
**    Init every Macro
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**---------------------------------------------------------------------------
*/
void  SystemInit( void )
{

      _rcopy(&_S_romp, -1);

      __asm("di");                        /* disable interrupt */

      TM00_Init( );                       /* TM00 initiate */
      __asm("ei");                        /* enable interrupt */
}
```

5.1.6 Serial.h

```
/*
*****************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KX1+
**  32-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 : serial.h
**  Abstract : This file implements a device driver for the SERIAL module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*****************************************************************************
*/
#ifndef _MDSERIAL_
#define _MDSERIAL_
```

```
#define ADR_CSIA0B0           0xfffffe00  /* CSIA0 automatic transfer RAM
address */
#define ADR_CSIA1B0           0xfffffe20  /* CSIA1 automatic transfer RAM
address */

#define CSIA_AUTORAMSIZE            32    /* CSIA automatic transfer RAM size
*/
#define IIC_RECEIVEBUFSIZE          32

#define     IIC0_SLAVEADDRESS 0x0


enum TransferMode { Send, Receive };
void IIC0_Init( void );
MD_STATUS IIC0_MasterStart( enum TransferMode , UCHAR , UCHAR );
MD_STATUS IIC0_MasterSendData( UCHAR* txbuf, UINT txnum );
MD_STATUS IIC0_MasterReceiveData( UCHAR* rxbuf, UINT rxnum );
MD_STATUS IIC0_SlaveStart( UCHAR );
MD_STATUS IIC0_SlaveSendData( UCHAR* txbuf, UINT txnum );
MD_STATUS IIC0_SlaveReceiveData( UCHAR* rxbuf, UINT rxnum );
void IIC0_Stop( 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 CALL_IIC0_MasterReceiveEnd( );
void CALL_IIC0_MasterError( MD_STATUS flag );

/* added new function combining MasterStart and MasterSendData */
MD_STATUS IIC0_MasterStartAndSend(UCHAR sadr, UCHAR* txbuf, UINT txnum);
/* added new function combining MasterStart and MasterReceiveData */
MD_STATUS IIC0_MasterStartAndReceive(UCHAR sadr, UCHAR* rxbuf, UINT rxnum);

#endif      /* _MDSERIAL_ */
```

5.1.7 Serial.c

```
/*
*****************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KX1+
**  32-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 : serial.c
**  Abstract : This file implements a device driver for the SERIAL module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
```

```
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
****************************************************************************
*/

#include "macrodriver.h"
#include "serial.h"

#pragma interrupt INTIIC0 MD_INTIIC0
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 */

#define FIX_APPLILET_IIC0_INIT   1  /* fix Applilet-generated code for init
*/
#define FIX_APPLILET_IIC0_START      1    /* fix Applilet-generated code for
start */

/*
**----------------------------------------------------------------------------
--------------------------------
**
**  Abstract:
**    Initialize the IIC0 interface:acknowledge setting,start signal
generation
**    setting,stop signal generation setting,wait setting and INTIIC0
priority
**    setting.
**
**  Parameters:
**    None.
**
**  Returns:
**    None.
**
**----------------------------------------------------------------------------
--------------------------------
*/
void IIC0_Init( void )
```

```
{
#if (FIX_APPLILET_IIC0_INIT == 1)
      /* reorder setting of P3 registers per documentation */
      SetIORBit(P3H, 0x03);   /* set P38 and P39 output latch to 1 */
      SetIORBit(PF3H, 0x03);  /* set P38 and P39 for alternate function */
      SetIORBit(PMC3H, 0x03); /* set P38 and P39 port mode control for IIC0
*/
#else
      /* original code generated by Applilet */
      SetIORBit(PMC3H, 0x03);
      SetIORBit(PF3H, 0x03);
      SetIORBit(P3H, 0x03);
#endif
      SVA0 = IIC0_SLAVEADDRESS;
      ClrIORBit(IICX0, 0x01);       /* Normal mode, fxx/198 */
      ClrIORBit(IICCL0, 0x0b);
      SetIORBit(IICCL0, 0x03);
      ClrIORBit(IICIC0, 0x40);            /* IIC0 interrupt enable */
      SetIORBit(IICIC0, Lowest);          /* Set IIC0 interrupt Lowest */

      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 :  BUS busy
**    MD_ARGERROR cannot get start condition
**
**-----------------------------------------------------------------------------
-------------------------------
*/
MD_STATUS IIC0_MasterStart( enum TransferMode mode, UCHAR adr, UCHAR wait )
{
#if (FIX_APPLILET_IIC0_START == 1)
int wait2 = 1000; /* value used in waiting for stop condition */
#endif
      /* BUS check */
      __asm("di");
      if( IICF0 & 0x40 ){
            __asm("ei");            /* BUS busy */
            return MD_ERROR;
      }

      /* START IIC0 */
      SetIORBit(IICC0, 0x18);       /* SPIE0 = WTIM0 = 1 */
#if (FIX_APPLILET_IIC0_START == 1)
      if ( !(IICC0 & 0x80) ) {
            /* not enabled, enable and generate stop condition */
            SetIORBit(IICC0, 0x80); /* IICE0 = 1 */
            SetIORBit(IICC0, 0x01); /* SPT0 = 1 to generate stop condition */
            while ( wait2-- ) ;           /* delay to wait for stop condition
to be recognized */
            if ( !(IICS0 & 0x01)) { /* check stop condition */
                  __asm("ei");
                  return MD_ERROR;
            }
            /* stop condition will generate an interrupt if SPIE0 = 1, clear
if set */
            if (IICIF0 == 1) {
                  IICIF0 = 0;
            }
      }
#else
      SetIORBit(IICC0, 0x80);       /* IICE0 = 1 */
#endif

      SetIORBit(IICC0, 0x02);       /* generate start condition */
      __asm("ei");

      /* 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 */
      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 )
{
      ClrIORBit(IICC0, 0x80);       /* stop transfer */
      return;
}

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

```
**    This function is responsible for IIC0 data transfering by master mode.
**    and call back function is provided as interface to high level user.
**
** Parameters:
**    UCHAR* txbuf : Address of transfer buffer.
**    UINT txnum The number of data to transmit(frame number).
**
** 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.
**    and call back function is provided as interface to high level user.
**
** Parameters:
**    UCHAR* rxbuf : Address of receive buffer.
**    USHORT rxnum : The number of data shuld be received.
**
** 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.
**    and call back function is provided as interface to high level user.
**
** Parameters:
**    UCHAR* txbuf : Address of send buffer.
**    USHORT txnum : The number of data shuld be send.
**
** 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.
**    and call back function is provided as interface to high level user.
**
** Parameters:
**    UCHAR* rxbuf : Address of receive buffer.
**    USHORT rxnum : The number of data shuld be received.
**
** 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 Application 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 Application 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 ADDRESS */
      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;
}
```

5.1.8 Serial_user.c

```
/*
*****************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KX1+
**  32-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 : serial_user.c
**  Abstract : This file gives callback functions for serial module.
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*****************************************************************************
*/
/*
** ***************************************************************************
** Include files
** ***************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"
UCHAR       *IIC0_M_TX_ADDRESS;     /* write data buffer */
UCHAR       *IIC0_M_RX_ADDRESS, *IIC0_M_RX_ORGADDRESS;

UCHAR       *IIC0_S_TX_ADDRESS;     /* write data buffer */
UCHAR       *IIC0_S_RX_ADDRESS, *IIC0_S_RX_ORGADDRESS;      /* read data
buffer */

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

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**    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;

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

      /* wait for bus not busy */
      i = 10000;
      do { i--; } while ( (i > 0) && ( IICF0 & 0x40 ) );
      if (i == 0) {
            return MD_ERROR;
      }

      /* Start, data write */
      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;
      }

      IIC0_Stop();      /* done with communication, stop */

      return MD_OK;     /* no error */
}

/*
**--------------------------------------------------------------------------
--
**
**  Abstract:
**    Combines IIC0_MasterStart(Send, (sadr), 10)
**       and IIC0_MasterReceiveData(UCHAR* rxbuf, UINT rxnum)
**
**  Parameters:
**    UCHAR sadr :      set address for select slave
**    UCHAR* rxbuf :    receive buffer pointer
**    UINT rxnum :      number of bytes to receive
**
**  Returns:
**    MD_OK
**    MD_ERROR
**    MD_ARGERROR
**  MD_NACK - timeout on slave address
**
**--------------------------------------------------------------------------
--
*/

MD_STATUS IIC0_MasterStartAndReceive(UCHAR sadr, UCHAR* rxbuf, UINT rxnum)
{
MD_STATUS status;
int i,j;

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

      /* wait for bus not busy */
      i = 10000;
      do { i--; } while ( (i > 0) && ( IICF0 & 0x40 ) );
      if (i == 0) {
            return MD_ERROR;
      }

      /* Start, data read */
      status = IIC0_MasterStart(Receive, 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_MasterReceiveData(rxbuf, rxnum);  /* send data bytes */
      if (status != MD_OK) {
            return status;
      }
      i = 10000;
      do {
            i--;
      } while ( (UI_MasterError == MD_OK) && (UI_MasterReceiveEnd ==
MD_ERROR) && (i > 0) );

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

      IIC0_Stop();      /* done with communication, stop */

      return MD_OK;     /* no error */
}
```

5.1.9 Timer_user.c

```
/*
*****************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+,
V850ES/KG1+,
**  V850ES/KF1+ and V850ES/KE1+ 32-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 : timer_user.c
**  Abstract : This file implements a device driver for the timer interrupt
service routine
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*****************************************************************************
*/
/*
****************************************************************************
**Include files
****************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"
#pragma interrupt INTTM000 MD_INTTM000

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

/* counter for millisecond timer */
volatile unsigned int milliseconds;

/*
**-----------------------------------------------------------------------
----------
**
**  Abstract:
**    TM00 INTTM000 Interrupt service routine
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-----------------------------------------------------------------------
----------
*/
__interrupt void MD_INTTM000( void )
{
      /* count down millisecond timer */
      if (milliseconds > 0)
            milliseconds--;
}

/* set the millisecond timer */
void SetMsecTimer(int time)
{
      milliseconds = time;
}

/* check the millisecond timer */
BOOL CheckMsecTimer(void)
{
      if (milliseconds > 0)
            return MD_FALSE;
      return MD_TRUE;
}
```

5.1.10 850.dir

```
#*
#*****************************************************************************
*
#**
#**  This device driver was created by Applilet for the V850ES/KX1+
#**  32-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 : 850.dir
#**  Abstract : This is the link file for CA850
#**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
#**
#*****************************************************************************
*
#*

#CONST     : !LOAD ?R V0x400{
#     .const        = $PROGBITS ?A .const;
#      };
# changed above to add SCONST at 0x400, followed by CONST, then TEXT, with
OPT first at 0x7a
OPT     : !LOAD ?R V0x7a{
         .opt        = $PROGBITS ?A .opt;
    };

SCONST     : !LOAD ?R V0x400{
      .sconst        = $PROGBITS ?A .sconst;
      };
```

```
CONST     : !LOAD ?R {
      .const        = $PROGBITS ?A .const;
      };

TEXT  : !LOAD ?RX {
      .pro_epi_runtime = $PROGBITS  ?AX;
      .text        = $PROGBITS      ?AX;
};
  DATA      : !LOAD ?RW V0x3ffe000 {

      .data       = $PROGBITS ?AW;
      .sdata            = $PROGBITS ?AWG;
      .sbss       = $NOBITS   ?AWG;
      .bss        = $NOBITS   ?AW;
};

STACK : !LOAD ?RW V0x3ffee00{
      .stack            = $PROGBITS     ?AW     .stack;
};

__tp_TEXT @ %TP_SYMBOL{TEXT};
__gp_DATA @ %GP_SYMBOL{DATA} &__tp_TEXT{DATA};
__ep_DATA @ %EP_SYMBOL;
```

5.2 Files Common to Serial Communication Demonstration Programs

```
Macrodriver.h
/*
*****************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+,
V850ES/KG1+,
**  V850ES/KF1+ and V850ES/KE1+ 32-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
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*****************************************************************************
*/

#ifndef     _MDSTATUS_
#define     _MDSTATUS_
```

```
#pragma ioreg              /*enable use the register directly in ca850
compiler*/

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

#define     MD_ON       1
#define     MD_OFF             0

#define     MD_TRUE     1
#define     MD_FALSE    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_ADDRESSMATCH            MD_STATUSBASE+0x3 /*IIC slave address match*/
#define MD_OVF                           MD_STATUSBASE+0x4 /*timer count
overflow*/
#define MD_DMA_END                       MD_STATUSBASE+0x5 /*DMA transfer end*/
#define MD_DMA_CONTINUE                  MD_STATUSBASE+0x6 /*DMA transfer
continue*/
#define MD_SPT                           MD_STATUSBASE+0x7 /*IIC stop*/
#define MD_NACK                          MD_STATUSBASE+0x8       /*IIC no ACK*/
#define MD_SLAVE_SEND_END                MD_STATUSBASE+0x9       /*IIC slave send
end*/
#define MD_SLAVE_RCV_END                 MD_STATUSBASE+0x0 /*IIC slave receive
end*/
#define MD_MASTER_SEND_END               MD_STATUSBASE+0x11      /*IIC master send
end*/
#define MD_MASTER_RCV_END                MD_STATUSBASE+0x12      /*IIC master
receive end*/


/*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
n=0,1,2*/
#define MD_OVERRUNERROR            MD_ERRORBASE+0x3  /*UARTn overrun error
n=0,1,2*/
#define MD_FRAMEERROR                    MD_ERRORBASE+0x4  /*UARTn frame error
n=0,1,2*/
#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_ODDBUF            MD_ERRORBASE+0x8  /*in 16bit transfer
mode,buffer size should be even*/
#define MD_DATAEXISTS              MD_ERRORBASE+0x9  /*Data to be
transferred next exists in TXBn register*/

/* macro fucntion definiton */
#define LockInt( ) { __asm("stsr 5,r10"); __asm("push r10"); __asm("di"); }
#define UnlockInt( ) {  __asm("pop r10"); __asm("ldsr r10,5"); }

/*main clock and subclock as clock source*/
enum ClockMode { MainClock, SubClock };
void  Clock_Init( void );
/*clear IO register bit and set IO register bit */
#define ClrIORBit(Reg, ClrBitMap)   Reg &= ~ClrBitMap
#define SetIORBit(Reg, SetBitMap)   Reg |= SetBitMap

enum INTLevel{Highest,Level1,Level2,Level3,Level4,Level5,Level6,Lowest};
enum TrigEdge { None, RisingEdge,FallingEdge, BothEdge };


#define     SYSTEMCLOCK 20000000
#define     SUBCLOCK    32768
#define     MAINCLOCK   5000000

#endif
```

5.2.1 Crte.s

```
#  This device driver was created by Applilet for the V850ES/KX1+
#  32-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 : crte.s
#  Abstract : start file for CA850
#  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
#
#                  |        :        |
#                  |        :        |
#         tp -> -+-----------------+ __start     __tp_TEXT
#                  |  start up       |
#                  |---------------- |
#   text section   |                 |
#                  | user program    |
#                  |                 |
#                  |-----------------|
#                  | library         |
#               -+-----------------+
#                  |        :        |
#                  |        :        |
#               -+-----------------+ __argc
#                  |      0          |
#                  |---------------- | __argv
#   data section   |    #.L16        |
#                  |---------------- | .L16
#                  | 0x0,0x0,0x0,0x0 |
#                 -+-----------------+
#                  |                 |
#   sdata section  |                 |
#                  |                 |
#            gp-> -+-----------------+             __ssbss
#                  |                 |
#   sbss section   |                 |
#                  |                 |
#                  +-----------------+ __stack     __esbss     __sbss
#                  | stack area      |
#   bss section    |                 |
#                  |  0x200 bytes    |
#            sp-> -+-----------------+ __stack + STACKSIZE     __ebss
#
#===========================================================================
=

#---------------------------------------------------------------------------
-
#     special symbols
#---------------------------------------------------------------------------
-
      .extern __tp_TEXT, 4
      .extern __gp_DATA, 4
      .extern __ep_DATA, 4
      .extern     __ssbss, 4
      .extern __esbss, 4
      .extern __sbss, 4
      .extern __ebss, 4


#---------------------------------------------------------------------------
-
#      C program main function
#---------------------------------------------------------------------------
-
      .extern     _SystemInit
      .extern     _main
      .extern     _Clock_Init

#---------------------------------------------------------------------------
-
#     for argv
#---------------------------------------------------------------------------
-
```

```
      .data
      .size __argc, 4
      .align      4
__argc:
      .word 0
      .size __argv, 4
__argv:
      .word #.L16
.L16:
      .byte 0
      .byte 0
      .byte 0
      .byte 0


#-----------------------------------------------------------------------------
-
#     dummy data declaration for creating sbss section
#-----------------------------------------------------------------------------
-
      .sbss
      .lcomm      __sbss_dummy, 0, 0


#-----------------------------------------------------------------------------
-
#     system stack
#-----------------------------------------------------------------------------
-
      .set  STACKSIZE, 0x200
      .bss
      .lcomm      __stack, STACKSIZE, 4


#-----------------------------------------------------------------------------
-
#     RESET handler
#-----------------------------------------------------------------------------
-

      .section   "RESET", text
      jr      __start

#-----------------------------------------------------------------------------
-
#     start up
#           pointers:  tp - text pointer
#                    gp - global pointer
#                    sp - stack pointer
#                    ep - element pointer
#     exit status is set to r10
#-----------------------------------------------------------------------------
-
      .text
      .align      4
      .globl      __start
      .globl  __exit
      .globl      __startend
      .extern  ___PROLOG_TABLE
__start:
      mov   #__tp_TEXT, tp          -- set tp register
      mov   #__gp_DATA, gp          -- set gp register offset
      add   tp, gp                  -- set gp register
      mov   #__stack+STACKSIZE, sp  -- set sp register
      mov   #__ep_DATA, ep          -- set ep register

      .option warning

      mov    1, r11           -- on-chip debug mode
      set1  5, PMC0[r0]
      set1  5, P0[r0]
      st.b   r11, PRCMD[r0]
      st.b   r11, OCDM[r0]

      nop
      nop
      nop
      nop
      nop
      mov   0x1, r11
      st.b r11, VSWC[r0]                  --mainclock over 16.6MHz

      jarl  _Clock_Init, lp         -- call Clock_Init function

      mov   #__ssbss, r13           -- clear sbss section
      mov   #__esbss, r12
      cmp   r12, r13
      jnl   .L11
.L12:
      st.w  r0, [r13]
      add   4, r13
      cmp   r12, r13
      jl    .L12
.L11:

      mov   #__sbss, r13            -- clear bss section
      mov   #__ebss, r12
      cmp   r12, r13
      jnl   .L14
.L15:
      st.w  r0, [r13]
      add   4, r13
      cmp   r12, r13
      jl    .L15
.L14:

      mov     #___PROLOG_TABLE, r12   -- for prologue/epilogue runtime
      ldsr    r12, 20               -- set CTBP (CALLT base pointer)
      -- IRAM clean up --
      mov 0x3ffd800, r10      -- IRAM start address
      mov 0x3fff000, r11      -- IRAM end address
_clear_loop:                  -- IRAM clean up
      st.w r0, 0x0[r10]
      add 4, r10
      cmp r11,r10
      jnz _clear_loop

      ld.w  $__argc, r6       -- set argc
      movea $__argv, gp, r7         -- set argv
      jarl  _SystemInit, lp         -- call SystemInit function
      jarl  _main,lp          -- call main function
__exit:
      halt                    -- end of program
```

5.2.2 System.inc

```
--/*
--
*****************************************************************************
--**
--**  This device driver was created by Applilet for the V850ES/FE2,
V850ES/FF2,V850ES/FG2
--**  and V850ES/FJ2 32-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.inc
--**  Abstract : This file implements a device driver for the SYSTEM module
--**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
--**
--  Device:  uPD70F3318Y
--
--  Compiler:  NEC/CA850
--
--
*****************************************************************************
--*/
.set  CG_Mainosc, 0x5
.set  CG_SECURITY0,     0xff
.set  CG_SECURITY1,     0xff
.set  CG_SECURITY2,     0xff
.set  CG_SECURITY3,     0xff
.set  CG_SECURITY4,     0xff
.set  CG_SECURITY5,     0xff
.set  CG_SECURITY6,     0xff
```

```
.set  CG_SECURITY7,     0xff
.set  CG_SECURITY8,     0xff
.set  CG_SECURITY9,     0xff
```

5.2.3 System.s

```
--/*
--
****************************************************************************
--
--  This device driver was created by Applilet for the V850ES/KF1+,
V850ES/KG1+,
--  V850ES/KJ1+ 32-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.s
--  Abstract : This file implements a device driver for the SYSTEM module
--  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
--
--
--  Compiler:  NEC/CA850
--
--
****************************************************************************
--*/
      .include "system.inc"
      .section "OPTION_BYTES", text
      .byte  0                      --Set to option byte (Ring-OSC cannot be
stopped)
      .byte  0
      .byte  0
      .byte  0
      .byte  0
      .byte  0

      .section "SECURITY_ID", text
      .byte CG_SECURITY0            -- Security ID head
      .byte CG_SECURITY1
      .byte CG_SECURITY2
      .byte CG_SECURITY3
      .byte CG_SECURITY4
      .byte CG_SECURITY5
      .byte CG_SECURITY6
      .byte CG_SECURITY7
      .byte CG_SECURITY8
      .byte CG_SECURITY9            -- Security ID tail

      .text
      .globl      _Clock_Init
      .align      4
--/*
--**-------------------------------------------------------------------------
--
--**
--**  Abstract:
--**  Init the Clock Generator and Watchdog timer
--**
--**  Parameters:
--**  None
--**
--**  Returns:
--**  None
--**
--**-------------------------------------------------------------------------
--
--*/
_Clock_Init:
      add    -8, sp
      st.w  r11, 0[sp]
      st.w  r12, 4[sp]

      -- disable interrupt
      stsr  5, r11
      ori         0x80, r11, r11
      ldsr  r11, 5

      clr1  0, SYS[r0]              -- reset SYS register

      mov   r0, r11
      ld.b  PCC[r0], r12
      andi  0xf8, r12, r12
      or          r12, r11
      st.b  r11, PRCMD[r0]
      st.b  r11, PCC[r0]

      nop
      nop
      nop
      nop
      nop
      -- PLL start
      set1  0, PLLCTL[r0]
      -- PLL work
.if 1 -- fix bad code generated by Applilet
-- need to set r11 to some value before starting this loop!
      -- Lock 200 us
      movea 0x800, r0, r11
.endif
__CG_LOOP4:
      nop
      nop
      nop
      addi  -1, r11, r11
      cmp   r0, r11
      bnz   __CG_LOOP4
      set1  1, PLLCTL[r0]
      -- enable interrupt
      stsr  5, r11
      andi  0x7f, r11, r11
      ldsr  r11, 5
      -- pop
      ld.w  0[sp], r11
      ld.w  4[sp], r12
      add   8, sp
      --disable watchdog timer 2
      mov   0x1f, r11
      st.b  r11, WDTM2[r0]

      jmp [lp]
```

5.2.4 System_user.c

```
/*
*****************************************************************************
**
**  This device driver was created by Applilet for the V850ES/FE2,
V850ES/FF2,V850ES/FG2
**  and V850ES/FJ2 32-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_user.c
**  Abstract : This file implements a device driver for the SYSTEM interrupt
service routine
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**
**  Compiler:  NEC/CA850
**
*****************************************************************************
*/
/*
** ************************************************************************
** Include files
** ************************************************************************
*/

#include "macrodriver.h"
/*
** ************************************************************************
** MacroDefine
** ***************************************************************************
*/
```

5.2.5 Timer.h

```
/*
******************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+,
V850ES/KG1+,
**  V850ES/KF1+ and V850ES/KE1+ 32-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 : timer.h
**  Abstract : This file implements a device driver for the timer module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
******************************************************************************
*/

#ifndef _MDTIMER_
#define _MDTIMER_

/*
** ****************************************************************************
**MacroDefine
** ****************************************************************************
*/
#define     TM_TMP0_CLOCK     0x0
#define     TM_TMP0_INTERVALVALUE   0x00
#define     TM_TMP0_INTERVALVALUE2  0x00
#define     TM_TMP0_ONESHOTOUTPUTCYCLE    0x00
#define     TM_TMP0_ONESHOTOUTPUTDELAY    0x00
#define     TM_TMP0_EXTTRIGGERCYCLE 0x00
#define     TM_TMP0_EXTTRIGGERDELAY 0x00
#define     TM_TMP0_PWMCYCLE  0x00
#define     TM_TMP0_PWMWIDTH  0x00
#define     TM_TMP0_CCR0COMPARE     0x00
#define     TM_TMP0_CCR1COMPARE     0x00
#define     TM00_Clock  0x0
#define     TM00_INTERVALVALUE      0x270f
#define     TM00_SQUAREWIDTH  0x270f
#define     TM00_PPGCYCLE     0x270f
```

```
#define     TM00_PPGWIDTH     0x00
#define     TM00_ONESHOTCYCLE 0x270f
#define     TM00_ONEPULSEDELAY      0x00
#define     TM01_Clock  0x0
#define     TM01_INTERVALVALUE      0x00
#define     TM01_SQUAREWIDTH  0x00
#define     TM01_PPGCYCLE     0x00
#define     TM01_PPGWIDTH     0x00
#define     TM01_ONESHOTCYCLE 0x00
#define     TM01_ONEPULSEDELAY      0x00
#define     TM02_Clock  0x0
#define     TM02_INTERVALVALUE      0x00
#define     TM02_SQUAREWIDTH  0x00
#define     TM02_PPGCYCLE     0x00
#define     TM02_PPGWIDTH     0x00
#define     TM02_ONESHOTCYCLE 0x00
#define     TM02_ONEPULSEDELAY      0x00
#define     TM03_Clock  0x0
#define     TM03_INTERVALVALUE      0x00
#define     TM03_SQUAREWIDTH  0x00
#define     TM03_PPGCYCLE     0x00
#define     TM03_PPGWIDTH     0x00
#define     TM03_ONESHOTCYCLE 0x00
#define     TM03_ONEPULSEDELAY      0x00
#define     TM04_Clock  0x0
#define     TM04_INTERVALVALUE      0x00
#define     TM04_SQUAREWIDTH  0x00
#define     TM04_PPGCYCLE     0x00
#define     TM04_PPGWIDTH     0x00
#define     TM04_ONESHOTCYCLE 0x00
#define     TM04_ONEPULSEDELAY      0x00
#define     TM05_Clock  0x0
#define     TM05_INTERVALVALUE      0x00
#define     TM05_SQUAREWIDTH  0x00
#define     TM05_PPGCYCLE     0x00
#define     TM05_PPGWIDTH     0x00
#define     TM05_ONESHOTCYCLE 0x00
#define     TM05_ONEPULSEDELAY      0x00
#define     TM50_Clock  0x5
#define     TM50_INTERVALVALUE      0x1e
#define     TM50_SQUAREWIDTH  0x1e
#define     TM50_PWMACTIVEVALUE     0x1e
#define     TM51_Clock  0x5
#define     TM51_INTERVALVALUE      0x1e
#define     TM51_SQUAREWIDTH  0x1e
#define     TM51_PWMACTIVEVALUE     0x1e
#define     TMH0_Clock  0x3
#define     TMH0_INTERVALVALUE      0x7c
#define     TMH0_SQUAREWIDTH  0x7c
#define     TMH0_PWMCYCLE     0x7c
#define     TMH0_PWMDELAY     0x3d
#define     TMH0_CARRIERDELAY 0x7c
#define     TMH0_CARRIERWIDTH 0x3d
#define     TMH1_Clock  0x3
#define     TMH1_INTERVALVALUE      0x7c
```

```
#define     TMH1_SQUAREWIDTH  0x7c
#define     TMH1_PWMCYCLE     0x7c
#define     TMH1_PWMDELAY     0x3d
#define     TMH1_CARRIERDELAY 0x7c
#define     TMH1_CARRIERWIDTH 0x3d


/*timer00 to 05,50,51,H0,H1 configurator initiation*/
void TM00_Init( void );

/*timer00 to 05 free running start,50,51,H0,H1 timer start*/
void TM00_Start( void );

/*timer00 to 05,50,51,H0,H1 timer stop*/
void TM00_Stop( void );
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
__interrupt void MD_INTTM000( void );

/* added functions in timer_user.c for millisecond timer */
void SetMsecTimer(int time);  /* set the timer */
BOOL CheckMsecTimer(void);          /* check the timer */

#endif
```

5.2.6 Timer.c

```
/*
****************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+,
V850ES/KG1+,
**  V850ES/KF1+ and V850ES/KE1+ 32-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 : timer.c
**  Abstract : This file implements a device driver for the timer module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
****************************************************************************
*/

/*
** ************************************************************************
** Include files
** *************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"
/*
** *************************************************************************
**MacroDefine
** *************************************************************************
*/
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    Initiate TM00, select founction and input parameter
**    count clock selection, INT init
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-------------------------------------------------------------------------
*/
void TM00_Init( void )
{
      TMC00 = 0x0;                                  /* stop TM00 */
      ClrIORBit(PRM00, 0x3);
      ClrIORBit(SELCNT1, 0x1);

      SELCNT1 |= ( TM00_Clock&0x4)>>2;    /* internal count clock */
      PRM00 |=( TM00_Clock&0x3);

      /* INTTM000 setting */
      TM0IC00 = Lowest;
      SetIORBit(TM0IC00, 0x40);
      /* TM00 interval */
      ClrIORBit(CRC00, 0x01);
      CR000 = TM00_INTERVALVALUE;
      CR001 = 0xffff;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    start the TM00 counter
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**
```

```
**---------------------------------------------------------------------------
*/
void TM00_Start( void )
{
      TMC00 = 0x0c;                          /* interval timer start */
      ClrIORBit(TM0IC00, 0x40);              /* enable INTTM000 */
}

/*
**---------------------------------------------------------------------------
**
**  Abstract:
**    stop the TM00 counter and clear the count register
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**---------------------------------------------------------------------------
*/
void TM00_Stop( void )
{
      TMC00 = 0x0;                                 /* stop TM00 */
      SetIORBit(TM0IC00, 0x40);              /* disable INTTM000 */
}

/*
**---------------------------------------------------------------------------
**
**  Abstract:
**    Change 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:
                  CR001=*(array_reg + 1);
         case 1:
                  CR000=*(array_reg + 0);
                  break;
         default:
                  return MD_ERROR;
         }
```

```
        return MD_OK;
}
```