To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

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

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

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

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

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

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

RENESAS

Application Note

78K/0

8-Bit Single-Chip Microcontrollers

Software

µPD78F0034A
µPD78F0034AY

Document No. U14425EE2V2AN00
Date Published February 2004

NOTES FOR CMOS DEVICES

1 PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

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

2 HANDLING OF UNUSED INPUT PINS FOR CMOS

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

3 STATUS BEFORE INITIALIZATION OF MOS DEVICES

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

Caution: **The following products are provided with an I²C bus interface circuit:**
µPD78002Y Subseries, µPD78014Y Subseries, µPD78018FY Subseries
µPD78054Y Subseries, µPD78058FY Subseries, µPD78064Y Subseries
µPD78075BY Subseries, µPD78078Y Subseries, µPD780018Y Subseries
µPD780024Y Subseries, µPD780034Y Subseries, µPD780058Y Subseries
µPD780308Y Subseries, µPD78070AY

Purchase of NEC I²C components conveys a license under the Philips I²C Patent Rights to use these components in an I²C system, provided that the system conforms to the I²C Standard Specification as defined by Philips.

The related documents in this publication may include preliminary versions. However, preliminary versions are not marked as such.

The export of this product from Japan is regulated by the Japanese government. To export this product may be prohibited without governmental license, the need for which must be judged by the customer. The export or re-export of this product from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

M4 94.11

Regional Information

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

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

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

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

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 01
Fax: 0211-65 03 327

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

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

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

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

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

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

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

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

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

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

NEC do Brasil S.A.
Electron Devices Division
Guarulhos, Brasil
Tel: 55-11-6465-6810
Fax: 55-11-6465-6829

Major Revisions in this Edition

Page	Description

The mark ★ shows major revised points.

Preface

Target reader

This manual is written for users who wish to understand the functions of the 78K/0 Family (µPD78F0034A, 780F034AY in order to design application systems in which it is used.

Purpose

This manual is intended to help the user understand by providing exam ples of programs in which the µPD78F0034A is used.

Organization

The contents of this application note are organized as follows.

- Basic settings for hardware-initialization
- Multiple interrupt servicing
- Standby function operation
- Pulse width measurement with free-running counter and one capture register
- Interfacing a serial EEPROM using the I^2C interface
- Using the 8-bit timer/event counter in cascade connection mode
- Using the asynchronous serial interface (UART0)
- Small temperature controller (using 10-bit A/D converter operations and 8-bit timer/event counter operation)

How to read this manual

It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, microcontrollers, and the C language.

For µPD78F0034A hardware functions
-> Refer to **µPD78F0034 Data Sheet.**

For 78K0 command functions
-> Refer to **78K/0 Series Instructions User's Manual**.

For 78K/0 electrical specifications
-> Refer to the **µPD78F0034 Data Sheet**.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active row:	xxx (overscore over pin or signal name) or representation/xxx ("/" before signal name)
Memory map address:	High order at high stage and low order at low stage
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information

Numerical representation: Binary ... xxxx or xxxxB
Decimal ... xxxx
Hexadecimal ... xxxxH or 0x xxxx

Prefixes representing powers of 2 (address space, memory capacity)
K (kilo): $2^{10} = 1024$
M (mega): $2^{20} = 1024^2$
G (giga): $2^{30} = 1024^3$

Related Documents The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Documents Related to µPD78K0034A

Document Title	Document Number
µPD780034A Data Sheet	U1394EE1V0DS00
78K/0 Instructions User's Manual	U12326EJ3V0UM00

Documents Related to Development Tools

Document Title	Document Number
IAR 78K/0 C Compiler	Part of Package CCIAR-CDR-78K0/K0S
IAR Assembler, Linker and Librarian	
SM78K/0 Simulator	U1018EJ3V0UM00

Table of Contents

Contents of Figures

Contents of Tables

Chapter 1 Basic Settings

1 Introduction

After Power-On-Reset some basic settings are done by the microcontroller itself. Some application specific hardware settings must be done from the user. For example clock generator, memory, ports, interrupts, etc.

2 Example Program

This example program shows a general hardware initialization, which could be done in many applications.

3 Basic Settings

The following basing settings should be done:

- Clock generator setting (PCC)
- Wait time after stop release (OSTS)
- Memory size (IMS)
- Port latches (Px, x = 0, 2...7)
- Port modes (PMx, x = 0, 2...7)
- Interrupt settings (IfxL, IfxH, MKxL, MKxH, PRxL, PRxH, x =0, 1)

3.1 Clock Generator/Memory Setting

3.1.1 Processor Clock Control Register (PCC)

The clock generator generates the clock to be supplied to the CPU and peripheral hardware and is controlled by the processor clock control register.

For the fastest execution speed the PCC register is set to PCC=0x00.

With PCC = 0x00 => $f_{cpu} = f_x$

For example: The number of clock cycles for the NOP-execution is given in the instruction set table with two clock cycles. Then the instruction execution time can be calculated with the following formula:

$$t_{instr} = \frac{2}{f_{cpu}}$$

With fx = 8.38MHz =>

$$t_{instr} = \frac{2}{f_{cpu}}$$

$$t_{instr} = \frac{2}{8.38\ MHz}$$

$$t_{instr} = 0.238\ \mu s$$

The NOP instruction is the fastest executable instruction, so the minimum instruction execution time is t_{instr} = 0,238µs@8.38MHz.

3.1.2 Oscillation Stabilization Time Select Register (OSTS)

After Reset the OSTS register is initialized to 0x04.

With fx = 8.38MHz =>

$$f_{stab} = \frac{2^{17}}{f_X}$$

$$f_{stab} = \frac{2^{17}}{8.38\ MHz}$$

$$f_{stab} = 15.6\ ms$$

If STOP mode is not used, no other selection is necessary.

If STOP mode is used past experiences has shown, that the following oscillation stabilization times – dependent from the oscillator type – should be selected:

Ceramic oscillator: ≥ 1.95 ms
Quarz oscillator: ≥ 7.81 ms

3.1.3 Memory Size Switching Register (IMS)

The 78F0034AY allows the user to select the internal memory capacity, so that the same memory map as that of 78002x and 78003x can be archieved.

For 78F0034AY: IMS = 0xC8 => RAM: 1024 Bytes
ROM: 32 Kbytes

3.2 Port Setting

The port latches of all ports should be set to 0 and the ports should be specified for input mode.

3.3 Interrupt Setting

All interrupts should be disabled and the interrupt request flags should be reset.

4 Pin Input/Output and Recommended Connection of Unused Pins

If no connection is provided to input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction.
Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to VDD or GND with a resistor, if it is considered to have a possibility of being an output pin.

For details, please refer to the user's manual.

5 Sample Program

```
/*=============================================================================
** PROJECT:        =  -
** MODULE          =  HW_Init.c
** SHORT DESC.     =  Hardware initialization
** DEVICE          =  uPD78F0034AY
** VERSION         =  1.0
** DATE            =  05/31/99
** LAST CHANGE     =  -
** ============================================================================
** Description:  -
**
** ============================================================================
** Environment:   Device:           uPD78F0034AY
**                Assembler:        A78000           Version 3.11D
**                C-Compiler:       ICC78000         Version 3.14A
**                Linker:           XLINK            Version 4.50C
**
** ============================================================================
** By:            NEC Electronics (Europe) GmbH
**                Oberrather Strasse 4
**                D-40472 Duesseldorf
**
**                DZ, NEC-EE, EAD-TPS
** ============================================================================
** Changes: -
**============================================================================
*/
#pragma language    =  extended
#pragma function    =  non_banked
#pragma memory      =  default

//————————————————————————————————
// Includes
//————————————————————————————————

#include <io78003x.h>
#include <in78000.h>

//————————————————————————————————
// Constants
//————————————————————————————————

// empty

//————————————————————————————————
// Variables
//————————————————————————————————

// External variables
//———————

// empty

// Internal variables
//———————

//empty

//————————————————————————————————
// Function prototypes
//————————————————————————————————

// External function prototypes
//———————————
// empty
```

```
// Internal function prototypes
//———————————————-

void vHardwareInit(void);              // Initialization hardware

/*===========================================================================
** MODULE          = vHardwareInit
** DESCRIPTION     = Initialization of some periheral hardware.
** PARAMETER       = -
** RETURN VALUE    = -
** VERSION         = 1.0
** DATE            = 05/31/99
** CHANGES         = -
** =========================================================================
** By:             NEC Electronics (Europe) GmbH
**                 Oberrather Strasse 4
**                 D-40472 Duesseldorf
**
**                 DZ, NEC-EE, EAD-TPS
**=========================================================================
*/

void vHardwareInit(void)
{
// clock generator setting
//———————————
    PCC  = 0x00;            // Use high speed mode (240ns @ 8.386MHz)
    OSTS = 0x01;            // 3.28ms wait after STOP release by interrupt
    IMS  = 0xC8;            // !!!! Select 1024 Byte RAM and 32k Byte ROM

// port setting
//———————
    P0=0x00;                                    // Set output latch to 0
    P2=0x00;                                    // Set output latch to 0
    P3=0x00;                                    // Set output latch to 0
    P4=0x00;                                    // Set output latch to 0
    P5=0x00;                                    // Set output latch to 0
    P6=0x00;                                    // Set output latch to 0
    P7=0x00;                                    // Set output latch to 0

    MEM = 0x01;                                 //  Important for keyReturn function

    PM0  = 0xFF;                                //  Port 0 = input (4-bits)
    PM2  = 0xFF;                                //  Port 2 = input (6-bits)
    PM3  = 0xFF;                                //  Port 3 = input (7-bits)
    PM4  = 0xFF;                                //  Port 4 = input
    PM5  = 0xFF;                                //  Port 5 = input
    PM6  = 0xFF;                                //  Port 6 = input (4-bits)
    PM7  = 0xFF;                                //  Port 7 = input (6 bits)

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

    PR0L = 0xFF;
    PR0H = 0xFF;
    PR1L = 0xFF;
}

/*===========================================================================
** main function
**=========================================================================
*/
void main(void)
{
    _DI();
   vHardwareInit();
    _EI();
```

```
    while(1)
        ;
}
```

Chapter 2 Multiple Interrupt Servicing

1 Introduction

Multiple interrupts occur when another interrupt request is acknowledged during execution of an interrupt.

2 Example Program

This example program shows the multiple interrupt servicing at the example of external interrupts INTP0 to INTP2. INTP2 has the lowest and INTP0 the highest default interrupt priority. INTP1 has a high programmable priority, the others a low programmable priority.

In this program an interrupt occurs in case of a falling edge on INTP0, INTP1, or INTP2. The falling edge is specified in EGN register.

3 Multiple Interrupt Setting

Multiple interrupts do not occur, if the interrupt request acknowledge enable state is selected (IE=1), except the non-maskable interrupts. When an interrupt request is received, interrupt requests acknowledge becomes disabled (IE=0). To enable multiple interrupts, it is necessary to set the IE flag with the EI instruction during interrupt servicing to enable interrupt acknowledge.

Two types of priority control are available, default priority control and programmable priority control. The last one is used for multiple interrupts.

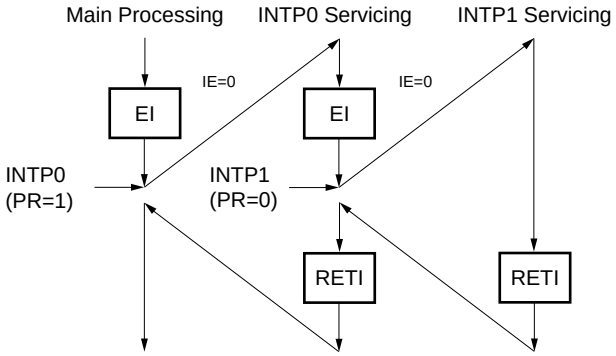
In the interrupt enable state, if an interrupt request with a priority less or higher than that of the currently serviced interrupt is generated, it is acknowledged for multiple interrupt servicing. If an interrupt with a priority less than that of the currently serviced interrupt is generated, it is not acknowledged for multiple interrupt servicing. This interrupt is held pending. When the current interrupt servicing ends, the pending interrupt is acknowledged after execution of one main process instruction.

4 Priority Specify Flag Registers PRx (x=0,1)

These registers are used to set the corresponding maskable interrupt priority orders (programmable priority control).

5 Multiple Interrupt Examples Referring to the Example Program

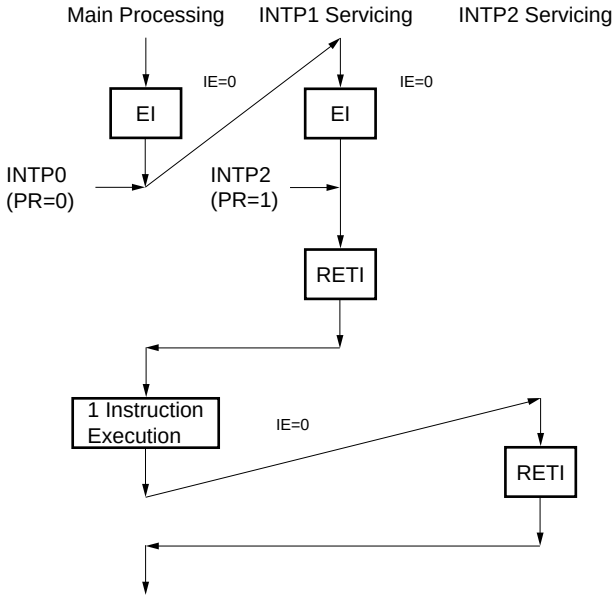
Figure 2-1: Multiple Interrupt



During servicing of interrupt INTP0, the interrupt request INTP1 is acknowledged, because of the higher default priority.

Multiple interrupt doesn´t occur due to priority control

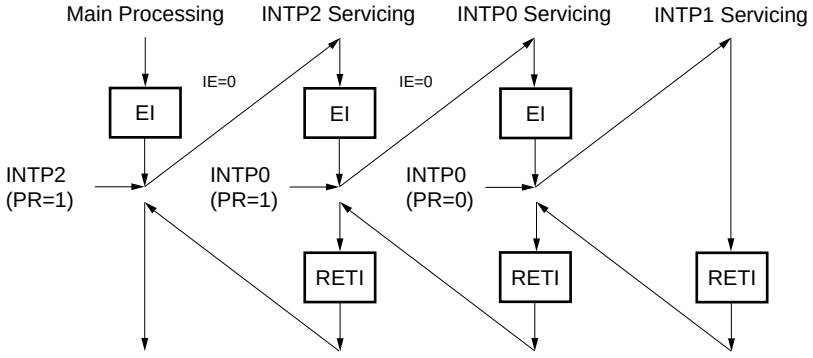
Figure 2-2: Multiple Interrupt Doesn't Occur



INTP2 is not acknowledged, because its priority is lower than that of INTP1.

Multiple interrupt occurs twice

Figure 2-3: Multiple Interrupt Occurs Twice

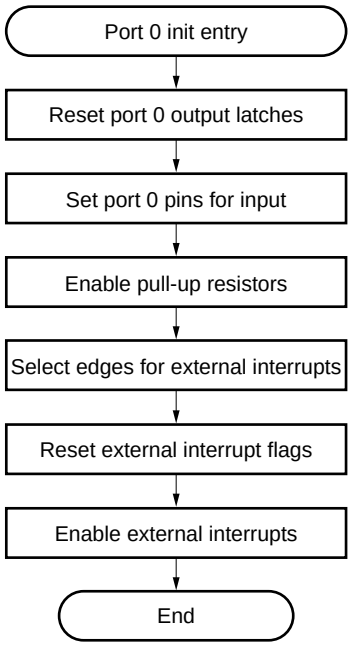


During servicing of interrupt INTP2, two interrupts requests INTP0 and INTP2, are acknowledged, and multiple interrupt servicing takes place.

6 Flow Charts

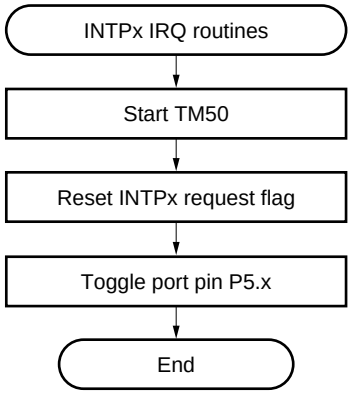
6.1 Configuration Port 0 for External Interrupts

Figure 2-4: Configuration Port 0



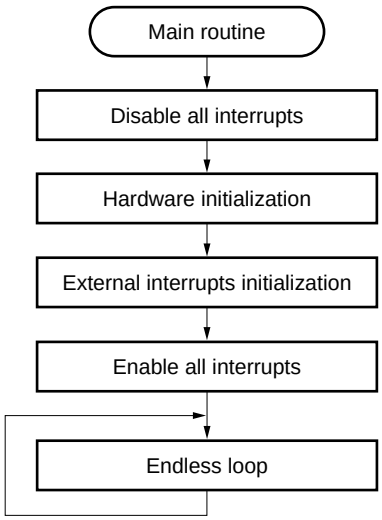
6.2 Interrupt Service Routine INTPx (x=0.2)

Figure 2-5: Interrupt Service Routine



6.3 Main Routine

Figure 2-6: Main Routine



7 Sample Program

```
/*=============================================================================
** PROJECT:        = -
** MODULE          = MultiInt.c
** SHORT DESC.     = -
** DEVICE          = uPD78F0034AY
** VERSION         = 1.0
** DATE            = 01.07.99
** LAST CHANGE     = -
** ============================================================================
** Description:  This program shows multiple interrupt servicing.
**
** ============================================================================
** Environment:    Device:             uPD78F0034AY
**                 Assembler:          A78000           Version 3.11D
**                 C-Compiler:         ICC78000         Version 3.14A
**                 Linker:             XLINK            Version 4.50C
**
** ============================================================================
** By:             NEC Electronics (Europe) GmbH
**                 Oberrather Strasse 4
**                 D-40472 Duesseldorf
**
**                 DZ, NEC-EE, EAD-TPS
** ============================================================================
** Changes:     -
**============================================================================
*/

#pragma language = extended

#include <io78003x.h>
#include <in78000.h>

void vHardwareInit(void);                          // Initialization hardware
void vExtIntInit(void);                            // Initialization external interrupts

                                                   // Interrupt service routine external
                                                   // interrupt INTP0
static interrupt [INTP0_vect] void vToggleP50(void);

                                                   // Interrupt service routine external
                                                   // interrupt INTP1
static interrupt [INTP1_vect] void vToggleP51(void);
                                                   // Interrupt service routine external
                                                   // interrupt INTP2
static interrupt [INTP2_vect] void vToggleP52(void);

//————————————————————————————
// External interrupts
//————————————————————————————

// Ports
//———

bit bShowINTP0 = P5.0;                             // Port to show that INTP0 has occured
bit bPM50  =  PM5.0;
bit bPU50  =  PU5.0;

bit bShowINTP1 = P5.1;                             // Port to show that INTP1 has occured
bit bPM51  =  PM5.1;
bit bPU51  =  PU5.1;

bit bShowINTP2 = P5.2;                             // Port to show that INTP2 has occured
bit bPM52  =  PM5.2;
bit bPU52  =  PU5.2;

bit bINTP0 =  P0.0;                                // Input port of INTP0
bit bPM00  =  PM0.0;
bit bPU00  =  PU0.0;

bit bINTP1 =  P0.1;                                // Input port of INTP1
bit bPM01  =  PM0.1;
bit bPU01  =  PU0.1;
```

```
bit bINTP2= P0.2;                               //  Input port of INTP2
bit bPM02 = PM0.2;
bit bPU02 = PU0.2;

// Interrupts
//------

bit bPMK0 = MK0L.1;                             //  External interrupt INTP0
bit bPIF0 = IF0L.1;
bit bPPR0 = PR0L.1;

bit bPMK1 = MK0L.2;                             //  External interrupt INTP1
bit bPIF1 = IF0L.2;
bit bPPR1 = PR0L.2;

bit bPMK2 = MK0L.3;                             //  External interrupt INTP2
bit bPIF2 = IF0L.3;
bit bPPR2 = PR0L.3;

/*=========================================================================
** Module name:      vHardwareInit
**
** Description:
**                   This module is to initialize some peripheral hardware.
**
** Operation:
**                   Sets the clock generator, the port modes and output latches,
**                   and initializes the interrupts.
**=========================================================================
*/

void vHardwareInit(void)
{
// clock generator setting
//-------------
    PCC  = 0x00;                                // Use high speed mode (240ns @ 8.386MHz)
    OSTS = 0x04;                                // 7.81ms wait after STOP release by interrupt
                                                // for quarz oscillator
    IMS  = 0xC8;                                // !!!! Select 1024 Byte RAM and 32k Byte ROM

// port setting
//-------
    P0=0x00;                                    //  Set output latch to 0
    P2=0x00;                                    //  Set output latch to 0
    P3=0x00;                                    //  Set output latch to 0
    P4=0x00;                                    //  Set output latch to 0
    P5=0x07;                                    //  Set output latch to 0
    P6=0x00;                                    //  Set output latch to 0
    P7=0x00;                                    //  Set output latch to 0

    MEM  = 0x01;                                //  Important for keyReturn function

    PM0  = 0xF0;                                //  Port 0  =  output (4-bits)
    PM2  = 0xC0;                                //  Port 2  =  output (6-bits)
    PM3  = 0x80;                                //  Port 3  =  output (7-bits)
    PM4  = 0x00;                                //  Port 4  =  output
    PM5  = 0x00;                                //  Port 5  =  output
    PM6  = 0x0F;                                //  Port 6  =  output (4-bits)
    PM7  = 0xC0;                                //  Port 7  =  output (6 bits)

    PU0=0x00;                                   // Disable pull-up resistors
    PU2=0x00;                                   // Disable pull-up resistors
    PU3=0x00;                                   // Disable pull-up resistors
    PU4=0x00;                                   // Disable pull-up resistors
    PU5=0x07;                                   // Disable pull-up resistors,
                                                // enable pull-up resistors 5.0..P5.2
    PU6=0x00;                                   // Disable pull-up resistors
    PU7=0x00;                                   // Disable pull-up resistors

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

```
    MK1L   =  0xFF;

    PR0L   =  0xFF;
    PR0H   =  0xFF;
    PR1L   =  0xFF;
}

/*=============================================================================
** Module name: vExtIntInit
**
** Description:
**                  This module is to initialize the external interrupt control
**                  and mode registers.
**
** Operation:
**                -
**===========================================================================
*/

void vExtIntInit(void)
{
    bINTP0  = 0;                                // Set output latch to 0
    bPM00   = 1;                                // P0.0 = input
    bPU00   = 1;                                // Enable pull-up resistor for P0.0

    bINTP1  = 0;                                // Set output latch to 0
    bPM01   = 1;                                // P0.1 = input
    bPU01   = 1;                                // Enable pull-up resistor for P0.1

    bINTP2  = 0;                                // Set output latch to 2
    bPM02   = 1;                                // P0.2 = input
    bPU02   = 1;                                // Enable pull-up resistor for P0.2

    EGP     = 0x00;                             // Set external interrupts for
    EGN     = 0x07;                             // falling edge.

    bPIF0   = 0;                                // Reset INTP0 request flag
    bPMK0   = 0;                                // Enable INTP0
    bPPR0   = 0;                                // High priority level

    bPIF1   = 0;                                // Reset INTP1 request flag
    bPMK1   = 0;                                // Enable INTP1
    bPPR1   = 1;                                // Low priority level

    bPIF2   = 0;                                // Reset INTP2 request flag
    bPMK2   = 0;                                // Enable INTP2
    bPPR2   = 0;                                // High priority level
}

/*=============================================================================
** Module name: vToggleP50
**
** Description:
**              This module toggles port 5.0.
**
** Operation:
**               -
**===========================================================================
*/

static interrupt [INTP0_vect] void vToggleP50(void)
{
    _EI();
    bPIF0   = 0;                                // Reset INTP0 request flag

    bShowINTP0 = 0;                             // Set P5.0 to indicate that
                                                // INTP0 has acknowledged
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    bShowINTP0 = 1;                             // Reset P5.0 to indicate that
                                                // INTP0 has been serviced.
}
```

```
/*=============================================================================
** Module name: vToggleP51
**
** Description:
**             This module toggles port 5.1.
**
** Operation:
**                -
**===========================================================================
*/

static interrupt [INTP1_vect] void vToggleP51(void)
{
    _EI();
    bPIF1 = 0;                                  // Reset INTP1 request flag

    bShowINTP1 = 0;                             // Set P5.1 to indicate that
                                                // INTP1 has acknowledged
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();

    bShowINTP1 = 1;                             // Reset P5.1 to indicate that
                                                // INTP1 has been serviced.
}

/*=============================================================================
** Module name: vToggleP52
**
** Description:
**             This module toggles port 5.2.
**
** Operation:
**                -
**===========================================================================
*/

static interrupt [INTP2_vect] void vToggleP52(void)
{
    _EI();
    bPIF2 = 0;                                  // Reset INTP2 request flag

    bShowINTP2 = 0;                             // Set P5.2 to indicate that
                                                // INTP2 has acknowledged
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();

    bShowINTP2 = 1;                             // Reset P5.2 to indicate that
                                                // INTP2 has been serviced.
}

/*=============================================================================
** main function
**===========================================================================
*/

void main(void)
{
    _DI();
    vHardwareInit();                            // Hardware initialization
    vExtIntInit();                              // Initialize externeal interrupts
    _EI();                                      // Enable all interrupts

    while(1)
        ;
}
```

Chapter 3 Standby Function Operation

1 Introduction

The standby function is designed to decrease power consumption of the system. The following two modes are available, the HALT mode and the STOP mode.

2 Example Program

This example program shows the switching between main and subsystem clock and the switching into STOP and HALT mode.

The program starts with main clock. A falling edge on INTP0 switches the system clock to subsystem clock. A falling edge on INTP1 switches the system clock to main system clock. A falling edge on INTP2 sets the µC to STOP mode, if system clock is main clock. Otherwise, if system clock is subsystem clock, a falling edge on INTP2 sets the µC to HALT mode.

3 HALT Mode

The HALT instruction sets the HALT mode. The HALT mode is intended to stop the CPU operation clock. The system clock continues oscillating. In this mode current consumption is not decreased as much as in the STOP mode. However, the HALT mode is effective to restart operation immediately upon interrupt request and to carry out intermittent operations such as watch operations.

Table 3-1: HALT Mode Settings and Operation Status

Parameter	Main System Clock	Subsystem Clock
Clock Generator	Both main and subsystem clock can be oscillated	
CPU	Operation stops	
Port	Status before HALT mode setting is held	
16-bit Timer/Event Counter	Operable	Operable when TI00 is selected
8-bit Timer/Event Counter	Operable	Operable when TI50, TI 51 are selected as count clock
Watch Timer	Operable, if $f_{xx}/2^7$ is selected as clock count	Operable if f_{xt} is selected as count clock
Watchdog Timer	Operable	Operation stops
A/D Converter	Stop	
Serial Interface	Operable	Operable during external SCK
External Interrupt	Operable	

There are three ways to clear HALT mode:

- Clear upon unmasked interrupt request
- Clear upon non-maskable interrupt request
- Clear upon RESET input

4 STOP Mode

STOP instruction sets the STOP mode. In the STOP mode, the main system clock oscillator stops, stopping the whole system, thereby considerably reducing the CPU power consumption.
Data memory low-voltage hold (down to VDD = 1.6 V) is possible. Thus, the STOP mode is effective to hold data memory contents with ultra-low current consumption. Because this mode can be cleared upon interrupt request, it enables intermittent operations to be carried out.
However, because a wait time is necessary to secure oscillator stabilization after the stop mode is cleared, select the HALT mode if it is necessary to start processing immediately upon interrupt request.

Table 3-2: STOP Mode Settings and Operation Status

Parameter	Main System Clock	Subsystem Clock
Clock Generator	Only main system clock oscillation is stopped	
CPU	Operation stops	
Port	Status before STOP mode setting is held	
16-bit Timer/Event Counter	Operation stops	
8-bit Timer/Event Counter	Operable only when TI50, TI 51 are selected as count clock	
Watch Timer	Operable, if f_{xt} is selected as clock count	Operation stops
Watchdog Timer	Operation stops	
A/D Converter	Operation stops	
Serial Interface	Operable at external SCK	
External Interrupt	Operable	

There are three ways to clear STOP mode:

- Release upon unmasked interrupt request
- Release by RESET input

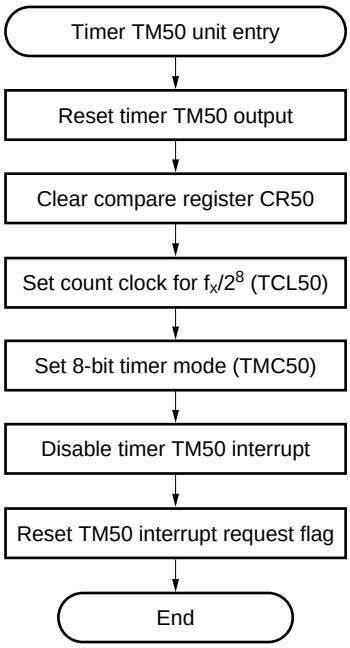
5 Processor Clock Control Register (PCC)

The clock generator is controlled by the processor clock control register (PCC). The PCC sets whether to use CPU clock selection, the ratio of division, main system clock oscillator operation/stop and subsystem clock oscillator internal feedback resistor.

6 Flow Charts

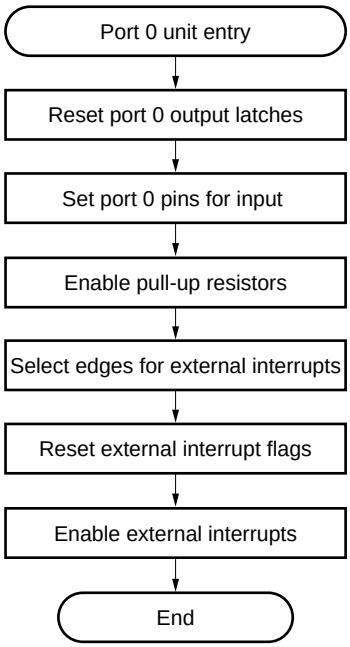
6.1 Configuration Timer 50

Figure 3-1: Configuration Timer 50



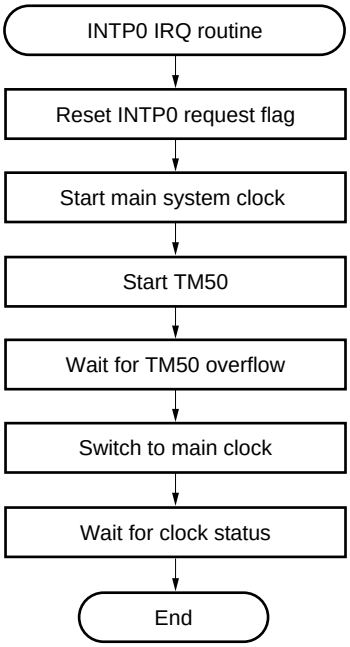
6.2 Configuration Port 0 for External Interrupts

Figure 3-2: Configuration Port 0



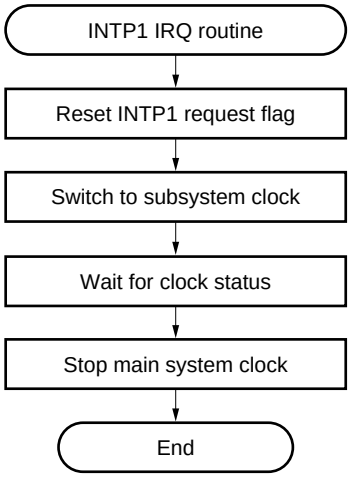
6.3 Interrupt Service Routine INTP0 (Subsystem to Main System Clock)

Figure 3-3: Interrupt Service Routine INTP0



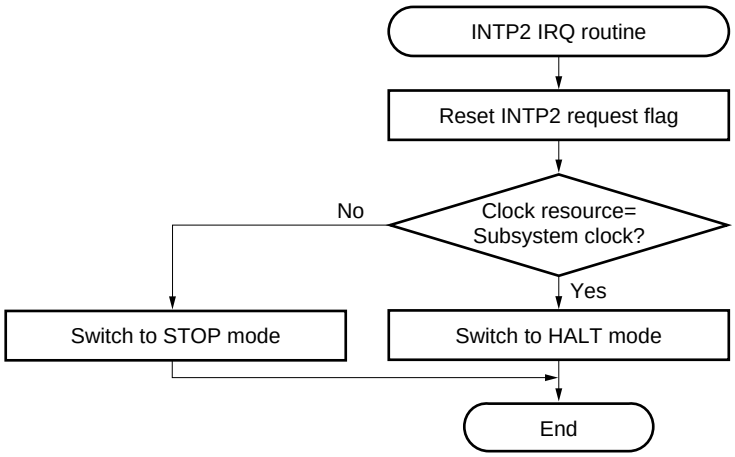
6.4 Interrupt Service Routine INTP1 (Main System to Subsystem Clock)

Figure 3-4: Interrupt Service Routine INTP1



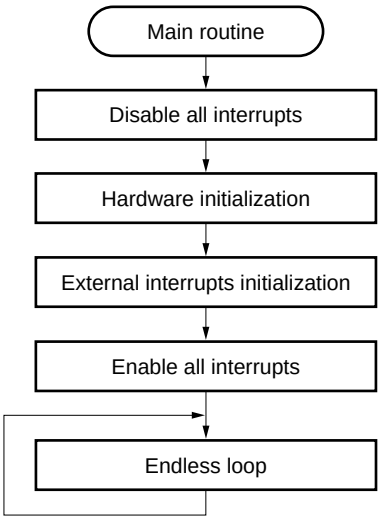
6.5 Interrupt Service Routine INTP2 (Switch to HALT/STOP Mode)

Figure 3-5: Interrupt Service Routine INTP2



6.6 Main Routine

Figure 3-6: Main Routine



7 Sample Program

```
/*=========================================================================
** PROJECT:       = -
** MODULE         = ClockGen.c
** SHORT DESC.    = -
** DEVICE         = uPD78F0034AY
** VERSION        = 1.0
** DATE           = 10.06.99
** LAST CHANGE    = -
** =========================================================================
** Description:   This program switches the system clock from to subsystem
**                clock.
**                The program starts with main clock. With a falling edge on
**                INTP0 the system clock will be switched to subsystem clock.
**                With a falling edge on INTP1 the system clock will be switched
**                back to main system clock. A falling edge on INTP2 sets the
**                µC to stop mode, if system clock is main clock. Otherwise,
**                if system clock is sub clock, a falling edge on INTP2 sets the
**                µC to halt mode.
** =========================================================================
** Environment:   Device:             uPD78F0034AY
**                Assembler:          A78000          Version 3.11D
**                C-Compiler:         ICC78000        Version 3.14A
**                Linker:             XLINK           Version 4.50C
**
** =========================================================================
** By:            NEC Electronics (Europe) GmbH
**                Oberrather Strasse 4
**                D-40472 Duesseldorf
**
**                DZ, NEC-EE, EAD-TPS
** =========================================================================
** Changes:     -
**=========================================================================
*/

#pragma language = extended

#include <io78003x.h>
#include <in78000.h>

void vHardwareInit(void);                           // Initialization hardware
void vTM50Init(void);                               // Initialization timer 50
void vExtIntInit(void);                             // Initialization external interrupts
void vWatchTimerInit(void);                         // Initialization watch timer
                                                    // Interrupt service routine to
                                                    // switch clock generator from sub to
                                                    // main system clock
static interrupt [INTP0_vect] void vSwitchSubToMain(void);
                                                    // Interrupt service routine to
                                                    // switch clock generator from main to
                                                    // sub system clock
static interrupt [INTP1_vect] void vSwitchMainToSub(void);
                                                    // Interrupt service routine to
                                                    // switch to standby-mode (STOP mode)
static interrupt [INTP2_vect] void vStandby(void);
                                                    // Interrupt service routine to
                                                    // toggle LED
static interrupt [INTWT_vect] void vToggleLED(void);


//-----------------------------------------
// General
//-----------------------------------------

bit bLEDPort = P5.0;                                // Port pin to toggle LED

//-----------------------------------------
// Timer 50
//-----------------------------------------

#define TM50_START 0x80
#define TM50_STOP  0x7F
```

```
// Interrupts
//————-

bit bTMMK50 = MK0H.6;
bit bTMIF50 = IF0H.6;
bit bTMPR50 = PR0H.6;

//————————————————————————————
// Clock generator
//————————————————————————————

bit bMCC = PCC.7;
bit bCLS = PCC.5;
bit bCSS = PCC.4;

//————————————————————————————
// Watch timer
//————————————————————————————

// Interrupts
//————-

bit bWTMK = MK1L.1;                              // Interrupt mask flag
bit bWTIF = IF1L.1;                              // Interrupt request flag

unsigned char uchHalfSec = 0,
              uchSec=0;

//————————————————————————————
// External interrupts
//————————————————————————————

// Ports
//———

bit bMainclock = P0.0;                           // Port for switching to subclock
bit bPM00 = PM0.0;
bit bPU00 = PU0.0;

bit bSubclock = P0.1;                            // Port for switching to subclock
bit bPM01 = PM0.1;
bit bPU01 = PU0.1;

bit bStandby = P0.2;                             // Port for switching to subclock
bit bPM02 = PM0.2;
bit bPU02 = PU0.2;

// Interrupts
//————-

bit bPMK0    = MK0L.1;                           // External interrupt INTP0
bit bPIF0    = IF0L.1;

bit bPMK1    = MK0L.2;                           // External interrupt INTP1
bit bPIF1    = IF0L.2;

bit bPMK2    = MK0L.3;                           // External interrupt INTP2
bit bPIF2    = IF0L.3;

/*=============================================================================
** Module name: vHardwareInit
**
** Description:
**                This module is to initialize some peripheral hardware.
**
** Operation:
**                Sets the clock generator, the port modes and output latches,
**                and initializes the interrupts.
**===========================================================================
*/

void vHardwareInit(void)
{
// clock generator setting
```

```
//————————————
    PCC   =  0x00;                          // Use high speed mode (240ns @ 8.386MHz)
    OSTS  =  0x04;                          // 7.81ms wait after STOP release by interrupt
    IMS   =  0xC8;                          // !!!! Select 1024 Byte RAM and 32k Byte ROM

// port setting
//———————
    P0=0x00;                                //  Set output latch to 0
    P2=0x00;                                //  Set output latch to 0
    P3=0x00;                                //  Set output latch to 0
    P4=0x00;                                //  Set output latch to 0
    P5=0x00;                                //  Set output latch to 0
    P6=0x00;                                //  Set output latch to 0
    P7=0x00;                                //  Set output latch to 0

    MEM = 0x01;                             //  Important for keyReturn function

    PM0  = 0xF0;                            //  Port 0 = output (4-bits)
    PM2  = 0xC0;                            //  Port 2 = output (6-bits)
    PM3  = 0x80;                            //  Port 3 = output (7-bits)
    PM4  = 0x00;                            //  Port 4 = output
    PM5  = 0x00;                            //  Port 5 = output
    PM6  = 0x0F;                            //  Port 6 = output (4-bits)
    PM7  = 0xC0;                            //  Port 7 = output (6 bits)

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

    PR0L  =  0xFF;
    PR0H  =  0xFF;
    PR1L  =  0xFF;
}

/*============================================================================
** Module name: vTM50Init
**
** Description:
**              This module is to initialize the timer 50 mode and control
**               registers.
**
** Operation:
**              Timer output is enabled, active high and at start reset.
**               The timer clock is fx.
**==========================================================================
*/

void vTM50Init(void)
{
    TMC50 &= TM50_STOP;                         // Timer 50 stop

    CR50 = 0xFF;

    TCL50 = 0x05;                               // 00000101 = 0x05
                                                // ——000 - TI50 falling edge
                                                // ——001 - TI50 rising edge
                                                // ——010 - fx
                                                // ——011 - fx/2^2
                                                // ——100 - fx/2^4
                                                // ——101 - fx/2^6
                                                // ——110 - fx/2^8
                                                // ——111 - fx/2^10

    TMC50 = 0x07;                               // 00000111 = 0x07
                                                // ||-||||0 - output disabled
                                                // ||-||||1 - output enabled
                                                // ||-|||0    — active level high
                                                // ||-|||1    — active level low
                                                // ||-|00     —— no change
```

```
                                                    // ||-|01—— timer output reset
                                                    // ||-|10—— timer output set
                                                    // ||-0——— single mode
                                                    // ||-1——— cascade mode
                                                    // |0———— clear and start mode by
                                                    // matching between CR50
                                                    // and TM50
                                                    // |1———— PWM (free-running mode)
                                                    // 0———— after cleaning to 0,
                                                    //            count op. disabled
                                                    // 1———— operation start

    bTMMK50 = 1;                                    // Disable timer 50 interrupt
    bTMIF50 = 0;                                    // Reset timer 50 interrupt flag
}

/*=============================================================================
** Module name: vExtIntInit
**
** Description:
**                This module is to initialize external interrupt control and
**               mode register.
**
** Operation:
**                -
**===========================================================================
*/

void vExtIntInit(void)
{
    bMainclock = 0;                                 // Set output latch to 0
    bPM00 = 1;                                      // P0.0 = input
    bPU00 = 1;                                      // Enable pull-up resistor for P0.0

    bSubclock = 0;                                  // Set output latch to 0
    bPM01 = 1;                                      // P0.1 = input
    bPU01 = 1;                                      // Enable pull-up resistor for P0.1

    bStandby = 0;                                   // Set output latch to 0
    bPM02 = 1;                                      // P0.2 = input
    bPU02 = 1;                                      // Enable pull-up resistor for P0.2

    EGP = 0x00;                                     // Set external interrupts for
    EGN = 0x07;                                     // falling edge.

    bPIF0 = 0;                                      // Reset INTP0 request flag
    bPMK0 = 0;                                      // Enable INTP0

    bPIF1 = 0;                                      // Reset INTP1 request flag
    bPMK1 = 0;                                      // Enable INTP1

    bPIF2 = 0;                                      // Reset INTP2 request flag
    bPMK2 = 0;                                      // Enable INTP2
}

/*=============================================================================
** Module name: vWatchTimerInit
**
** Description:
**              This module is to initialize the watch timer.
**
** Operation:
**              Sets the watch timer for working with sub clock.
**              Each 0.5s an interrupt will be generated.
**===========================================================================
*/

void vWatchTimerInit(void)
{
    WTM = 0x53;                                     // 01010011 = 0x53
                                                    // ||||—|0 - Operation stop
                                                    // ||||—|1 - Operation enable
                                                    // ||||—0— Clear after operation stop
                                                    // ||||—1— Start
```

```
                                            // |000——— 2^4/fw
                                            // |001——— 2^5/fw
                                            // |010——— 2^6/fw
                                            // |011——— 2^7/fw
                                            // |100——— 2^8/fw
                                            // |101——— 2^9/fw
                                            // 0———— fx/2^7 (65.4kHz)
                                            // 1———— fxT    (32.768kHz)

    bWTMK = 0;                              // Enable watch timer interrupt
    bWTIF = 0;                              // Reset interrupt request flag
}

/*=============================================================================
** Module name: vSwitchSubToMain
**
** Description:
**              This module switches the clock generator from subsystem clock
**               to main clock.
**
** Operation:
**                -
**===========================================================================
*/

static interrupt [INTP0_vect] void vSwitchSubToMain(void)
{
    bPIF0 = 0;                              // Reset INTP0 request flag

    bMCC = 0;                               // Start main system clock

    TMC50 |= TM50_START;                    // Timer 50 start

    while(!bTMIF50);                        // Wait for timer 50 overflow
        _NOP();                             // CR50 = 0xFF
                                            // For sure, that main system clock is
                                            // running

    bTMIF50 = 0;                            // Reset timer 50 interrupt request flag

    bCSS = 0;                               // Switch to main system clock

    while(bCLS)                             // Wait for CPU clock status =
        _NOP();                             // main system clock

    WTM = 0x53;                             // Select watch timer count clock
                                            // fxT
    bWTMK = 0;                              // Enable watch timer interrupt
    bWTIF = 0;                              // Reset interrupt request flag
}

/*=============================================================================
** Module name: vSwitchMainToSub
**
** Description:
**              This module switches the clock generator from main clock
**               to subsystem clock.
**
** Operation:
**                -
**===========================================================================
*/

static interrupt [INTP1_vect] void vSwitchMainToSub(void)
{
    bPIF1 = 0;                              // Reset INTP1 request flag

    bCSS = 1;                               // Switch to subsystem clock

    while(!bCLS)                            // Wait for CPU clock status =
        _NOP();                             // subsystem clock

    WTM = 0xD3;                             // Select watch timer count clock to
                                            // fx/2^7
```

```
    bWTIF = 0;                                      // Reset interrupt request flag
    bWTMK = 0;                                      // Enable watch timer interrupt

    bMCC = 1;                                       // Stop main system clock
}

/*=============================================================================
** Module name: vStandby
**
** Description:
**              This module switchs µC to Stand-by mode (STOP mode).
**
** Operation:
**                -
**===========================================================================
*/

static interrupt [INTP2_vect] void vStandby(void)
{
    bPIF2 = 0;                                      // Reset INTP2 request flag
    bWTMK = 1;                                      // Disable watch timer interrupt
    if(bCLS)                                        // Check system clock source
    _HALT();                                        // => if subsystem clock, switch to HALT mode
    else
    {
    _STOP();                                        // => if main clock, switch to STOP mode
    bMCC = 1;
    }
}

/*=============================================================================
** Module name: vToggleLED
**
** Description:
**              This module is the interrupt service routine to toggle
**                the LED.
** Operation:
**              Each 0.5s an interrupt will be generated.
**              If CPU clock is main system clock, LED will toggle with f=1Hz.
**              If CPU clock is subsystem clock, LED will toggle with f=0.5Hz.
**===========================================================================
*/

static interrupt [INTWT_vect] void vToggleLED(void)
{

    IF1L.1=0;                                       // Reset interrupt request flag

    if(uchHalfSec == 0)                             // Time calculation
        uchHalfSec = 1;
    else
    {
        uchHalfSec=0;
       bLEDPort=~bLEDPort;
    }
}


/*=============================================================================
** main function
**===========================================================================
*/

void main(void)
{
    _DI();

   vHardwareInit();                                 // Hardware initialization
    vTM50Init();                                    // Timer 50 initialization
    vExtIntInit();                                  // Initialize externeal interrupts
   vWatchTimerInit();                               // Watch timer initialization

    _EI();                                          // Enable all interrupts
```

```
    PM5.0 = 0;                          // Port 5.0 = output
    bLEDPort = 0;                       // Set output latch to 0

    while(1)
         ;
}
```

Chapter 4
Pulse Width Measurement with Free-Running Counter and One Capture Register

1 Introduction

In many applications the 16-bit timer/event counter TM0 is used for multiple tasks.

2 Example Program

This example program initializes the 16-bit timer/event counter for free-running mode and matches between TM0 and CR01 and enables the external interrupt INTTM01.
The period and the pulse-width will be measured and the valid flag for ‘value available’ will be set.

Maximum frequency that can be measured: 40 kHz
Minimum frequency that can be measured: 130 Hz

3 TM0 Configuration

To use the 16-bit timer/event counter the timer operation must be stopped before setting the control registers. then the configuration of the control registers TMC0, CRC0, TOC0, PRM0 and PM7 is necessary.

3.1 16-Bit Timer Output Control Register (TOC0)

This register controls the operation of the 16-bit timer/event counter output control circuit.
For this application a timer output is not necessary.

=> TOC0 = 0x00

3.2 Prescaler Mode Register 0 (PRM0)

This register is used to set the 16-bit timer (TM0) count clock and TI00, TI01 input valid edges.

For this application the rising and the falling edge of TI00 is selected.

=> PRM0 = 0x30

3.3 Capture/Compare Control Register 0 (CRC0)

This register controls the operation of the capture/compare registers (CR00, CR01).

In this application the capture/compare register CR01 operates as capture register.

=> CRC0 = 0x04

3.4 16-Bit Timer Mode Control Register (TMC0)

This register sets the 16-bit timer operating mode, the 16-bit timer register clear mode, output timing, and detects an overflow.

The pulse width measurement should be done with a free-running counter.

=> TMC0 = 0x04

3.5 Port Mode Register 7 (PM7)

This register sets port 7 for input/output.

P70 is used for TI00 edge detection.

=> PM7 = 0xC1

4 Period and Pulse-Width Measurement Operation

The measurement method is measuring with TM0 used in free-running mode, that means the timer is never been reset or modified.

When the 16-bit timer register (TM0) is operated in free-running mode and the edge, specified in prescaler mode register (PRM0), is input to the TI00/TO0/P70 pin, the value of TM0 is taken into 16-bit capture/compare register 01 (CR01) and an external interrupt request signal (INTTM01) is generated.

4.1 Period Measurement Operation

For period measurement it is necessary to detect either each rising or each falling edge of the input signal.

When the first valid edge is detected on the TI00/TO0/P70 pin, the value of TM0 is taken into 16-bit capture/compare register CR01. In the interrupt service routine this value is taken into a buffer register. When the second valid edge is detected on the TI00/TO0/P70 pin, the value of TM0 is also taken in to 16-bit capture/compare register CR01 and the interrupt service routine takes this value into a buffer register.

The difference of these two values reflects the period of the input signal.
For calculation of the period the overflow of the timer must be considered.

4.2 Pulse-Width Measurement Operation

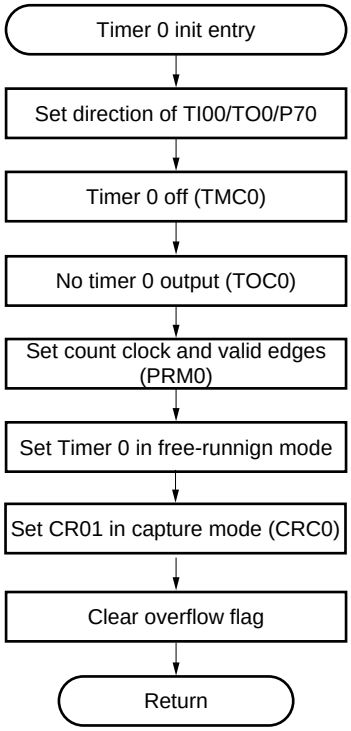
For pulse-width measurement it is necessary to detect each edge of the input signal.

When the rising edge is detected on the TI00/TO0/P70 pin, the value of TM0 is taken in to 16-bit capture/compare register CR01. In the interrupt service routine this value is taken into a buffer register. When the falling valid edge is detected on the TI00/TO0/P70 pin, the value of TM0 is also taken in to 16-bit capture/compare register CR01 and the interrupt service routine takes this value into a buffer register.
The difference of these two values reflects the pulse-width of the input signal. For calculation of the period the overflow of the timer must be considered.

5 Flow Charts

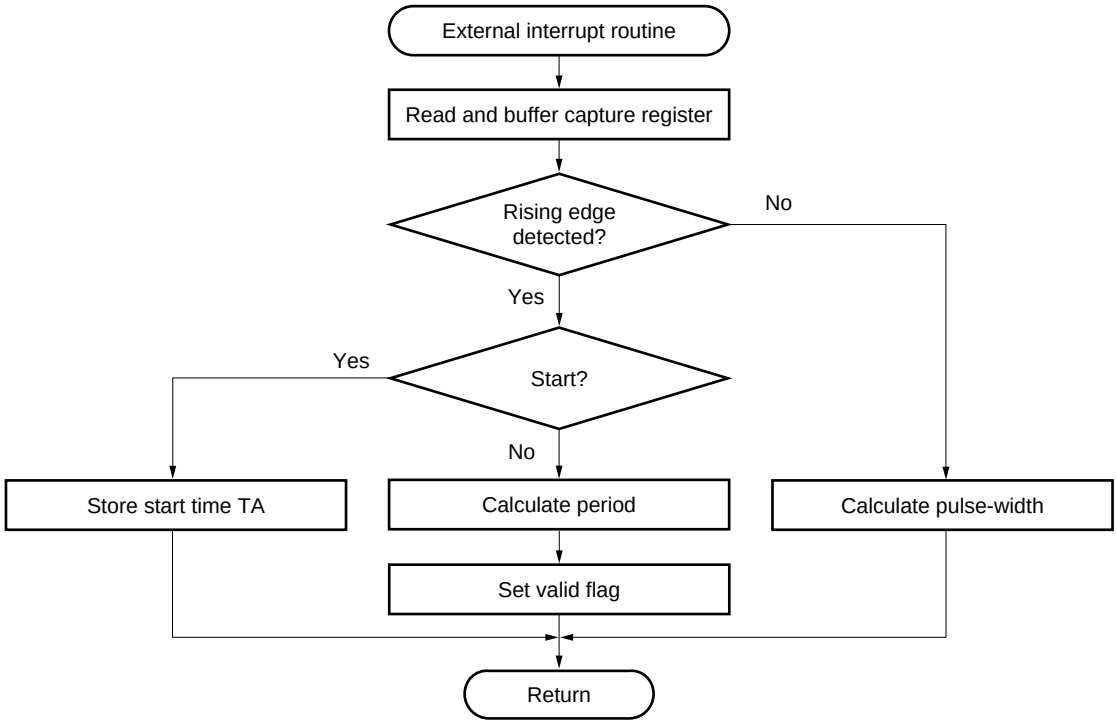
5.1 Configuration TM0

Figure 4-1: Configuration TM0



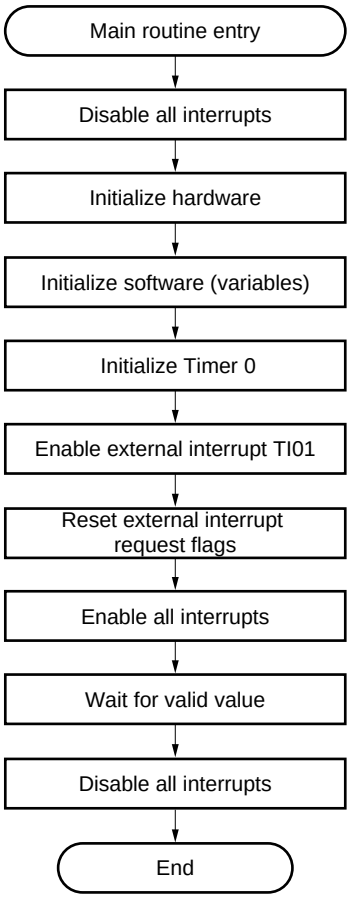
5.2 Period and Pulse-Width Calculation

Figure 4-2: Period and Pulse-Width Calculation



5.3 Main Routine

Figure 4-3: Main Routine



6 Sample Program

```
/*=============================================================================
** PROJECT          =  IE-78001-R-A Tests
** MODULE           =  TM0_PW.c
** SHORT DESC.      =  Example program demonstrates the 16-Bit timer/event counter
**                     TM0
** DEVICE           =  uPD78F0034AY
** VERSION          =  1.0
** DATE             =  26.03.1999
** LAST CHANGE      =  -
** ============================================================================
** Description:     Measurement of pulse-width and periof with free-running
**                  counter and one capture register.
**                  This module will do the needed hardware initialization:
**                  Configuration of 16-Bit timer, ports and interrupts.
**                  Pulse-width and period is measured and valid values are
**                  indicated by valid flag.
**
** ============================================================================
** Environment:     Device:             uPD78F0034AY
**                  Assembler:          A78000              Version 1.13A
**                  C-Compiler:         ICC78000            Version 3.13A
**                  Linker:             XLINK               Version 4.50
**
** ============================================================================
** By:              NEC Electronics (Europe) GmbH
**                  Oberrather Strasse 4
**                  D-40472 Duesseldorf
**
**                  dz, NEC-EE, EAD-TPS
** ============================================================================
Changes:
** ============================================================================
*/

#pragma language = extended

#include <io78003X.h>
#include <in78000.h>

void HardwareInit(void);                        // Initialization Hardware
void vTimer0Init(void);                         // Initialization Timer 0
void vSoftwareInit(void);;                      // Initialization measurement variables

static interrupt [INTTM01_vect] using[2] void vIntT01Exception(void); // Interrupt TI01

#pragma memory = saddr

typedef unsigned int Word;
typedef unsigned char Byte;

#pragma memory = default

//-------------------------------------------
// General
//-------------------------------------------

bit bCLOE   = CKS.4;                            // “PCL output enable” of clock output selection
                                                // register CKS

//-------------------------------------------
// TM0
//-------------------------------------------

// Ports
//----

bit bSignalDirPort = PM7.0;                     // Port direction for signal
bit bSignalInputPort = P7.0;                    // Signal input
```

```
// Interrupt bits
//————————-

bit bTMMK01 = MK0H.5;                          // External interrupt TI01 enable bit

bit bTMIF01 = IF0H.5;                          // External interrupt request flag TI01

bit bTMPR1  = PR0H.5;                          // External interrupt TI01 priority flag

// TM0
//——

bit bOVF0   = TMC0.0;                          // “Overflow”-Bit of 16-Bit timer mode
                                               // control register TMC0

//————————————————————————————————-
// Period and pulse-width
//————————————————————————————————-

#pragma memory = saddr

Word    wBufferCR00;                           // Buffer register for capture register CR01
Word    wBufferCR01;                           // Buffer register for capture register CR01
Word    wTA;                                   // Calculation registers for period and pulse
Word    wTB;                                   // calculation
Word    wPulse;                                // Result pulse-width
Word    wPeriod;                               // Result period

#pragma memory = default

bit bValid;                                    // Measurement done flag

/* ============================================================================
**  Module name: vHardwareInit
**
**  Description:
**               This module is to initialize some peripheral hardware.
**
**  Operation:
**               Sets the clock generator, the port modes and output latches
**               and initializes the interrupts.
**
** ============================================================================
*/

void vHardwareInit(void)                       // Hardware inizialization
{

// clock generator setting
//————————————
    CKS  = 0x08                                // setup PCL frequency = fxt = 32KHz
    PCC  = 0x00;                               // Use high speed mode (250ns @ 8MHz)
    OSTS = 0x01;                               // 3.28ms wait after STOP release by interrupt
    IMS  = 0xC8;                               // !!!! Select 1024 Byte RAM and 32k Byte ROM

// port setting
//————————-
    P0=0x00;                                   // Set output latch to 0
    P2=0x00;                                   // Set output latch to 0
    P3=0x00;                                   // Set output latch to 0
    P4=0x00;                                   // Set output latch to 0
    P5=0x00;                                   // Set output latch to 0
    P6=0x00;                                   // Set output latch to 0
    P7=0x00;                                   // Set output latch to 0

    PM0  = 0xF0;                               // Port 0 = output (4-bits)
    PM2  = 0xC0;                               // Port 2 = output (6-bits)
    PM3  = 0x80;                               // Port 3 = output (7-bits)
    PM4  = 0x00;                               // Port 4 = output
    PM5  = 0x00;                               // Port 5 = output
    PM6  = 0x0F;                               // Port 6 = output (4-bits)
```

```
    PM7  = 0xC0;                            // Port 7 = output (6 bits)

// interrupt setting
//————————

    IF0L = 0x00;
    IF0H = 0x00;
    IF1L = 0x00;
    MK0L = 0xFF;
    MK0H = 0xFF;
    MK1L = 0xFF;

    PR0L = 0xFF;
    PR0H = 0xFF;
    PR1L = 0xFF;
}

/* ============================================================================
**  Module name: vSoftwareInit
**
**  Description:
**              This module is to initialize the variables used by measurement.
**
**  Operation:
**
** ============================================================================
*/

void vSoftwareInit(void)
{
    wBufferCR01 = 0;                        // Reset buffer register for capture value
    wTA = 0;                                // Reset time tn-1
    wTB = 0;                                // Reset time tn
    wPulse = 0;                             // Reset result pulse-width
    wPeriod= 0;                             // Reset result period
    bValid=0;                               // Measurement not done
}

/* ============================================================================
**  Module name: vTimer0Init
**
**  Description:
**              This module is to initialize the timer 0 control and mode
**              registers.
**
**  Operation:
**
** ============================================================================
*/

void vTimer0Init(void)
{
    bSignalDirPort = 1;

    TMC0 = 0x00;                            // Timer 0 stop
    TOC0 = 0x00;                            // No timer 0 output

    PRM0 = 0x30;                            // 00110000 = Prescaler Mode Register 0
                                            // ||||—00 - count clock fx (8Mhz at 8MHz)
                                            // ||||—01 - count clock fx/4 (2Mhz at 8MHz)
                                            // ||||—10 - count clock fx/64 (125KMhz at
8MHz)
                                            // ||||—11 - count clock TIOo valid edge
                                            // ||00 —— TI00 falling edge
                                            // ||01 —— TI00 rising edge
                                            // ||11 —— TI00 both edges
                                            // 00 ——— TI01 falling edge
                                            // 01 ——— TI01 rising edge
                                            // 11 ——— TI01 both edges

    CRC0 = 0x04;                            // 00000100 = Capture/compare control register 0
```

```
                                            // ——||0 - CR00 operates as compare register
                                            // ——|0 — Capture CR00 on valid edge of TI01 (P71)
                                            // ——1 — CR01 operates as capture register

   TMC0 = 0x04                              //    00000100 = Timer Mode Control Register
                                            // ——|||0 - OVF0 bit cleared
                                            // 010 — free running mode, match between TM0 and
CR00/CR01

    bOVF0 = 0;                              // Clear overflow bit
}

/* ============================================================================
**  Module name: vIntT01Exception
**
**  Description:
**               This module is the interrupt service routine for the external
**                interrupt T01 operation.
**
**  Operation:
**               Interrupt source P70 = CR01, Signal = TA, TB, Pulse,
**                Period
**
** ============================================================================
*/

static interrupt [INTTM01_vect] using [2] void vIntT01Exception (void)
{
    wBufferCR01 = CR01;                     // Store capture value in buffer

   if(bSignalInputPort)                     // Check, if rising edge
    {                                       // Yes=>
        if(wTA==0)                          // Check, if New cycle start
        {                                   // Yes=> Period calculation and new cycle
                                            // start
            wTA=wBufferCR01;                // Store TA
            bOVF0=0;                        // Reset overflow flag
        }
        else                                // No => Check, if overflow and
        {                                   // calculate period
            if(!bOVF0)
                wPeriod = wBufferCR01 - wTA;
            else
               wPeriod = 65536 - wTA + wBufferCR01;
            wTA = 0;                        // Reset flags
            bOVF0= 0;
            bValid=1;                       // Set Valid flag for measurement end
        }
    }
    else
    {
        if(!bOVF0)                          // No =>
            wPulse = wBufferCR01 - wTA;     // Calculate pulse-width
        else
            wPulse = 65536 - wTA + wBufferCR01;
    }
}

/* ============================================================================
**  main function
** ============================================================================
*/

void main( void )
{
    _DI();                                  // Disable all interrupts
   vHardwareInit();                         // Peripheral settings
   vSoftwareInit();                         // Initialization measurement variables
   vTimer0Init();                           // Initialization timer 0

    bTMMK01 = 0;                            // Enable external interrupt TI01
```

```
    bTMIF01 = 0;                          // Reset external interrupt request flag TI01
    _EI();                                // Enable all interrupts

   while(bValid==0);                      // Wait until new value available
         ;
    _DI();                                // Disable interrupts
    bValid = 0;                           // No more valid values available
    while(1)                              // Endless loop
         ;
}
```

Chapter 5 Interfacing EEPROM Using the Serial Interface I²C

1 Introduction

The I²C™ bus uses a two-wire interface consisting of a serial data line (SDA) and a serial clock line (SCL). Each device on the bus has its own address and is able to transmit or receive data. Depending on the function each device operates as master or slave. The master is the device that initiates, controls, and terminates a transfer and generates all framing and clock signals. The slave device is any device addressed by the master.

2 Example Program

This example program interfaces a two wire 24Cxx EEPROM in the I²C 7-bit addressing mode. The device is written in “byte write” mode and read in “random address read” mode.

3 Configuration Serial Interface I²C

3.1 I²C Clock Select Register (IICCL0)

This register is used to set the transfer clock rate for the I²C bus.

In this application the standard mode is used with a clock speed of 97.5 kHz at 8.386 MHz. No digital filter is needed.

=> IICCL0 = 0x05

3.2 I²C Control Register (IICC0)

This register is used to enable/disable I²C operations, set wait timing, and set other I²C operations.

=> IICC0 = 0xE8

3.3 I²C Status Register (IICS0)

This register indicates the status of the I²C.

4 8-Bit Timer/Event Counter (TM50, TM51)

In this application only one 8-bit timer/event counter (TM51) in individual mode is used.

4.1 8-Bit Timer/Event Counter Configuration

4.1.1 Timer Register (TM51)

TM51 is an 8-bit read-only register, which counts the clock pulses.

4.1.2 8-Bit Compare Register (CR51)

When CR51 is used as a compare register, the value set in CR51 is constantly compared with the 8-bit counter (TM51) count value, and an interrupt request (INTTM51) is generated if they match. It is possible to rewrite the value of CR51 within 00H to FFH during count operation.

For this application the value of CR51 is initialized to 0xA3.

4.1.3 Timer Clock Select Register (TCL51)

The register sets count clocks of 8-bit timer/event counter (TM51) and the valid edge of TI51 input.

For this application the count clock is set to 4.096 kHz.

=> TMC = 0x07

Note: Stop the timer operation before writing to TCL51.

4.1.4 8-Bit Timer Mode Control Register (TMC51)

TMC51 is a register which sets up the timer mode

For this application the timer output F/F is not needed and reset.

=> the value written to TMC51 is 0x04

4.1.5 Port Mode Register 7 (PM7)

In this application no output is needed, so PM7 is unchanged.

4.2 Timer 51 Operation

The 8-bit timer/event counter operates as an interval timer which generates interrupt requests repeatedly at intervals of the count value preset to 8-bit compare register CR51.

In this application a timer interval of 40ms is needed. The value of the 8-bit compare register CR51 can be calculated by the following formula:

$$\text{interval time} = \frac{(CR51+1)}{f_{TM51}}$$

TCL51 = 0x07 $\Rightarrow f_{TM51} = \frac{f_x}{S^{11}}$

$\Rightarrow f_{TM51} = 4.096$ KHz @ 8.386MHz

CR51 = (interval time * f_{TM51})-1=163

CR51 = 163

5 EEPROM Device Operations

5.1 “Byte Write” Mode

In the “byte write” mode the master sends the EEPROM device address - including the write-bit -, the byte address in the EEPROM and one data byte. All bytes are acknowledged from the EEPROM. Then the master terminates the transfer by generating the stop condition.

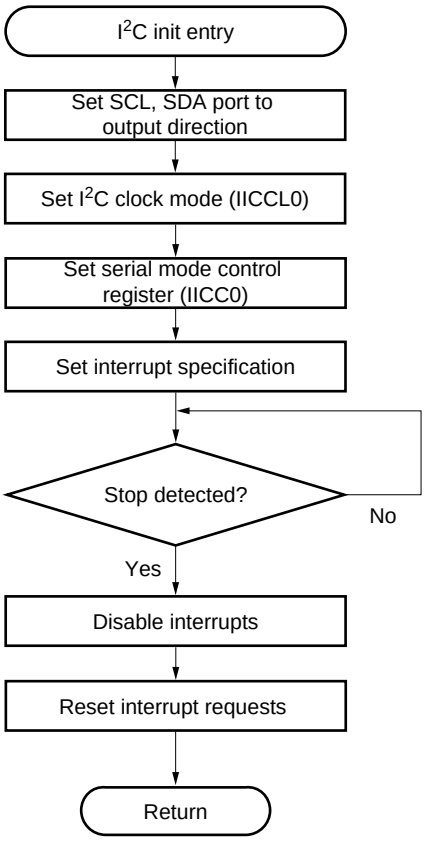
5.2 “Random Address Read” Mode

At first, a dummy write is performed to load the byte address into the address counter of the EEPROM . The master sends the EEPROM device address - including the write-bit - and the byte address in the EEProm. Each byte is acknowledged from the EEPROM. This is followed by a ReStart condition from the master, the EEPROM address is repeated with the read-bit. The EEPROM acknowledges this and outputs the addressed byte. The master has to not-acknowledge this byte and to terminate the transfer by a stop condition.

6 Flow Charts for I²C Interface

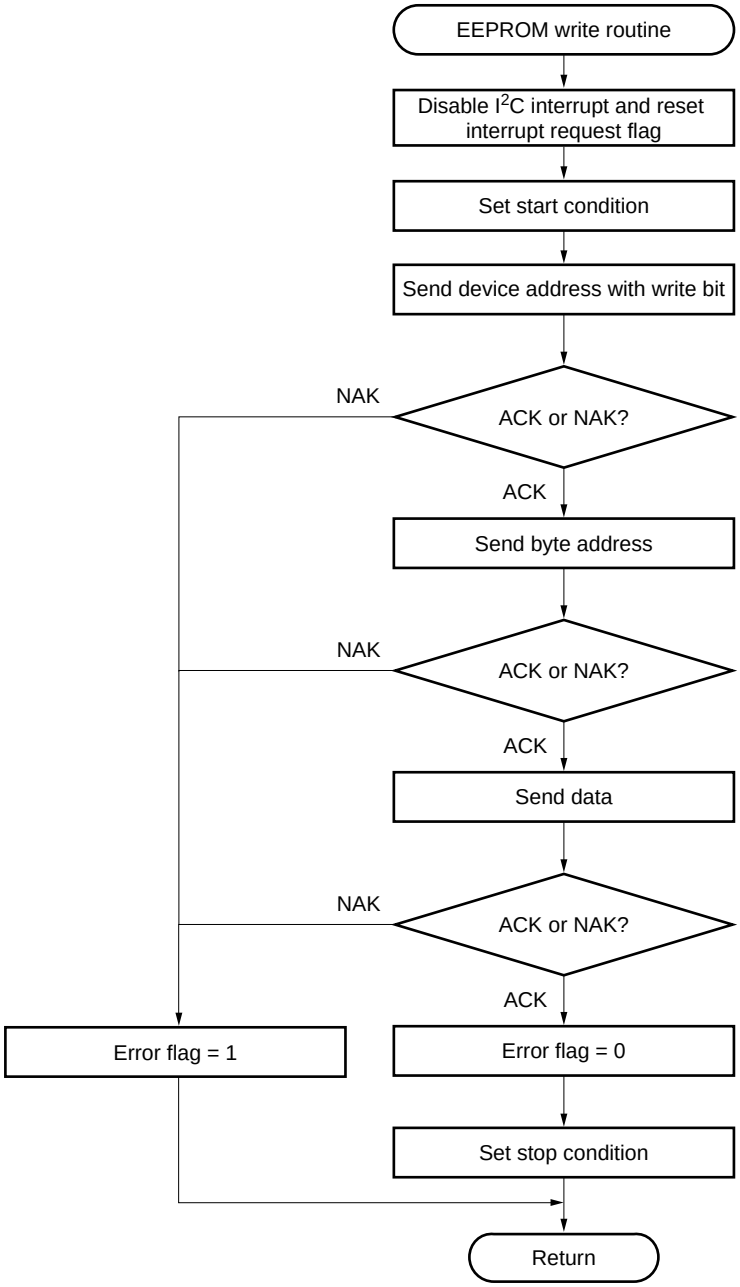
6.1 Configuration I²C

Figure 5-1: Configuration I²C



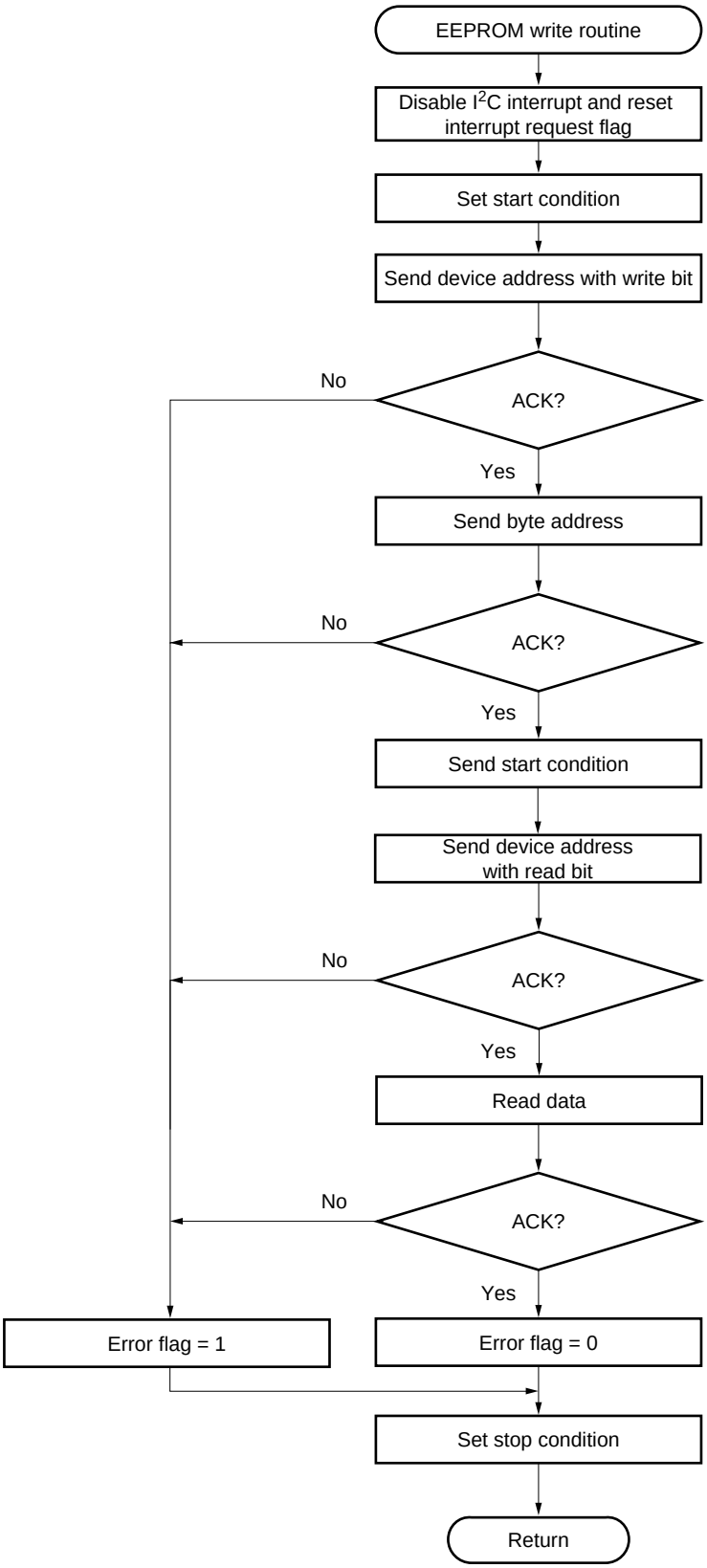
6.2 Write Data to EEPROM

Figure 5-2: Write Data to EEPROM



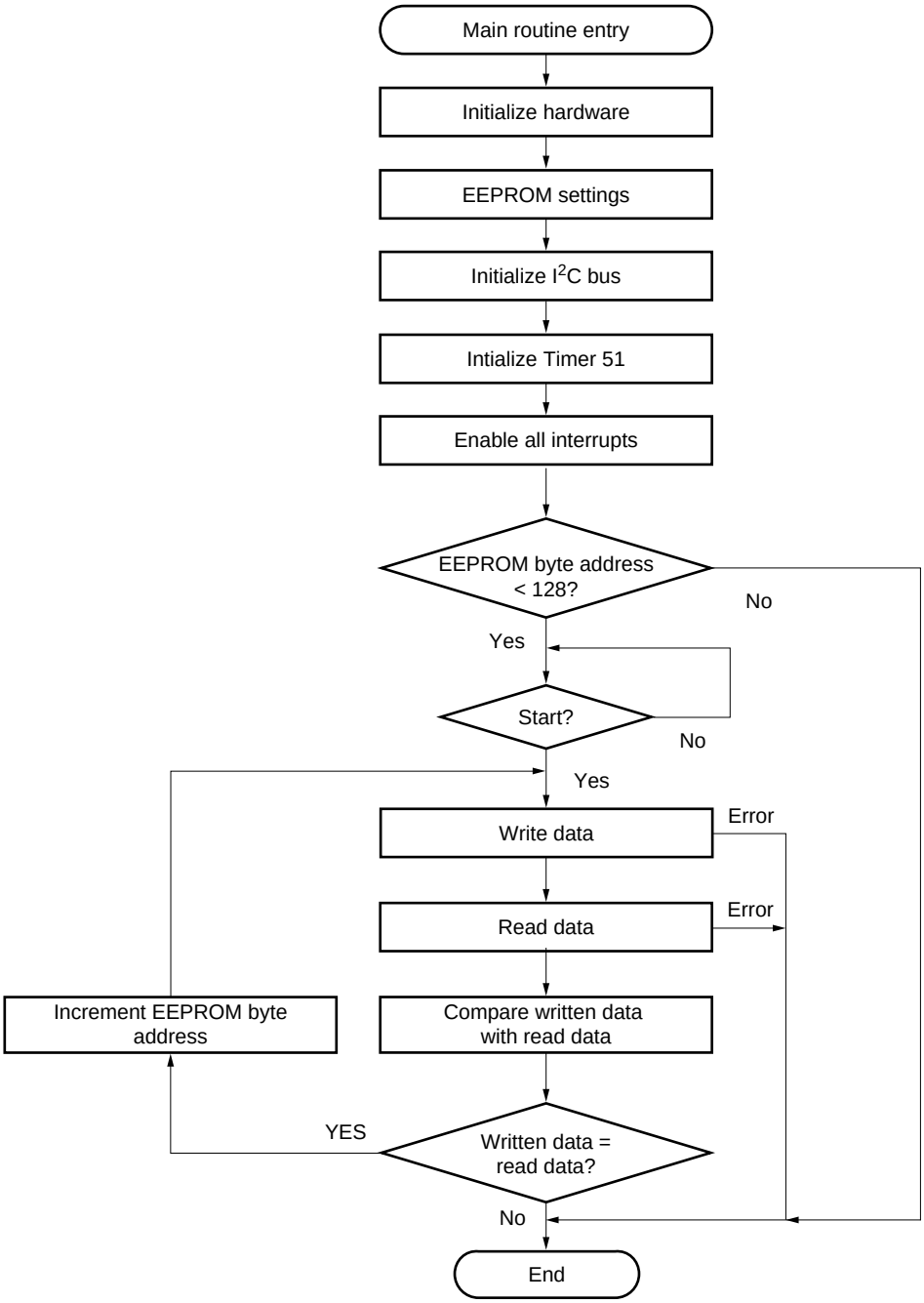
6.3 Read Data from EEPROM

Figure 5-3: Read Data from EEPROM



6.4 Main Routine

Figure 5-4: Main Routine



7 Sample Program

```
/*=============================================================================
** PROJECT         = IE-78001-R-A Tests
** MODULE          = I2CMast.c
** SHORT DESC.     = Read/Write of EEPROM24Cxx
** DEVICE          = uPD78F0034AY
** VERSION         = 1.0
** DATE            = 16.04.1999
** LAST CHANGE     = -
**=============================================================================
** Description:    The memory of a serial EEPROM is successively written with
**                 0x55 and 0xAA and then read. In case of an error, Port 5.0 is
**                 high. At the end of the test Port 5.1 is high. Low Level on
**                 Port 5.2 starts the test.
**                 The EEProm is written in “byte write” mode and read in
**                 “random address read” mode.
**
**=============================================================================
** Environment:    Device:              uPD78F0034AY
**                 Assembler:           A78000       Version 3.11D
**                 C-Compiler:          ICC78000           Version 3.14A
**                 Linker:              XLINK              Version 4.50C
**
**=============================================================================
** By:             NEC Electronics (Europe) GmbH
**                 Oberrather Strasse 4
**                 D-40472 Duesseldorf
**
**                 DZ, NEC-EE, EAD-TPS
**=============================================================================
Changes:
**=============================================================================
*/

#pragma language = extended

#include <io78003X.h>
#include <in78000.h>

void vHardwareInit(void);                       // Initialization Hardware
void vEEPromInit(void);                         // Initialization EEProm
void vI2CInit(void);                            // Initialization I2C bus
void vTM51Init(void);                           // Initialization Timer 51

                                                // EEProm write
void vEEPromWrite(unsigned char uchByteAddress, unsigned char uchDataIn);
                                                // EEProm read
void vEEPromRead(unsigned char uchByteAddress);
void vEEPromWait(void);                         // Wait time for EEProm write time


//————————————————————————————————-
// EEPROM
//————————————————————————————————-

// Ports
//———

bit bEEPromErrorDirPort     = PM5.0;            // Port direction bit for EEProm error
bit bEEPromErrorDataPort    = P5.0;             // Output latch bit for EEProm error
bit bEEPromReadyDirPort     = PM5.1;            // Port direction bit for EEProm ready
bit bEEPromReadyDataPort    = P5.1;             // Output latch bit for EEProm ready
bit bEEPromStartDirPort     = PM5.2;            // Port direction bit for EEProm start
bit bEEPromStartDataPort    = P5.2;             // Output latch bit for EEProm start
bit bEEPromStartPullUpPort  = PU5.2;            // Pull-up resistor for EEProm start

// EEProm
//———-

#define DEVICE_ADDRESS 0xA0                     // device address excl. R/W bit

#pragma memory = saddr
unsigned char uchEEPromAddress;                 // EEProm device address, incl. R/W bit
```

```
unsigned char uchEEPromByteAddress;                    // EEProm byte address
unsigned char uchEEPromData;                           // EEProm data

bit bEEPromError;
bit bEEPromBreak;
#pragma memory = default

//————————————————————————————————-
// Timer51
//————————————————————————————————-

#define TM51_START 0x80                                // Start timer 51
#define TM51_STOP 0x7F                                 // Stop timer51

// Interrupts
//——————

bit bTMMK51 = MK0H.7;                                  // Timer 51 interrupt enable bit
bit bTMIF51 = IF0H.7;                                  // Timer 51 interrupt request flag
bit bTMPR51 = PR0H.7;                                  // Timer 51 interrupt priority flag

//————————————————————————————————-
// I2C bus
//————————————————————————————————-

// Ports
//———

bit bSDA0DirPort   = PM3.2;                            // Port direction bit for SDA0
bit bSDA0DataPort  = P3.2;                             // Output latch bit for SDA0
bit bSCL0DirPort   = PM3.3;                            // Port direction bit for SCL0
bit bSCL0DataPort  = P3.3;                             // Output latch bit for SCL0

// Interrupt bits
//————————-

bit bIICMK0 = MK0H.2;                                  // I2C interrupt enable bit
bit bIICIF0 = IF0H.2;                                  // I2C interrupt request flag
bit bIICPR0 = PR0H.2;                                  //  I2C interrupt priority flag

// I2C bus
//————————

bit bCL00   =  IICCL0.0;                               // Selection of data transfer rate
bit bDFC0   =  IICCL0.2;                               // Control of digital filter operation
bit bSMC0   =  IICCL0.3;                               // Operation mode switching

bit bSPT0   =  IICC0.0;                                // Stop condition trigger
bit bSTT0   =  IICC0.1;                                // Start condition trigger
bit bACKE0  =  IICC0.2;                                // Acknowledge control
bit bWTIM0  =  IICC0.3;                                // Control of wait and interrupt request
                                                       // generation
bit bSPIE0  =  IICC0.4;                                // Enable/Disable generation of interrupt
                                                       // request when stop condition is detected
bit bWREL0  =  IICC0.5;                                // Cancel wait
bit bLREL0  =  IICC0.6;                                // Exit from the current communication
operation
                                                       // SCL0 and SDA0 lines go into high imped-
ance
                                                       // state
                                                       // MUST BE DONE BEFORE SPT0 WILL BE SET,
                                                       // OTHERWISE SPD0 WILL BE CLEARED
bit bIICE0  =  IICC0.7;                                // Enable I2C operation

bit bSPD0   =  IICS0.0;                                // Detection of stop condition
bit bSTD0   =  IICS0.1;                                // Detection of start condition
bit bACKD0  =  IICS0.2;                                // Detection of ACK

#define WRITE  0xFE
#define READ                                           0x01

#define SET_START_CONDITION        {bSTT0 = 1; while(!bSTD0);}

#define SET_STOP_CONDITION         {bSPT0 = 1; while(!bSPD0);}
```

```
#define SEND_DEVICE_ADDRESS        {ICC0 = DEVICE_ADDRESS;}

#define WAIT_READ_INT_AND_ACK   {while(!bIICIF0){;}; \
                                        bIICIF0=0;}

#define WAIT_WRITE_INT_AND_ACK  {while(!bIICIF0){;}; \
                                        bIICIF0=0; \
                                       if(!bACKD0){SET_STOP_CONDITION; \
                                                   bEEPromError=1; \
                                                   return;}}

/* ============================================================================
**  Module name: vHardwareInit
**
**  Description:
**               This module is to initialize some peripheral hardware.
**
**  Operation:
**               Sets the clock generator, the port modes and output latches
**              and initializes the interrupts.
** ============================================================================
*/

void vHardwareInit(void)                        // Hardware inizialization
{

// clock generator setting
//————————————
    CKS  = 0x08;                                // setup PCL frequency = fxt = 32KHz
    PCC  = 0x00;                                // Use high speed mode (250ns @ 8MHz)
    OSTS = 0x01;                                // 3.28ms wait after STOP release by interrupt
    IMS  = 0xC8;                                // !!!! Select 1024 Byte RAM and 32k Byte ROM

// port setting
//———————-
    P0=0x00;                                    // Set output latch to 0
    P2=0x00;                                    // Set output latch to 0
    P3=0x00;                                    // Set output latch to 0
    P4=0x00;                                    // Set output latch to 0
    P5=0x00;                                    // Set output latch to 0
    P6=0x00;                                    // Set output latch to 0
    P7=0x00;                                    // Set output latch to 0

    PM0  = 0xF0;                                // Port 0 = output (4-bits)
    PM2  = 0xC0;                                // Port 2 = output (6-bits)
    PM3  = 0x80;                                // Port 3 = output (7-bits)
    PM4  = 0x00;                                // Port 4 = output
    PM5  = 0x00;                                // Port 5 = output
    PM6  = 0x0F;                                // Port 6 = output (4-bits)
    PM7  = 0xC0;                                // Port 7 = output (6 bits)

// interrupt setting
//—————————

    IF0L = 0x00;
    IF0H = 0x00;
    IF1L = 0x00;

    MK0L = 0xFF;
    MK0H = 0xFF;
    MK1L = 0xFF;

    PR0L = 0xFF;
    PR0H = 0xFF;
    PR1L = 0xFF;
}

/* ============================================================================
**  Module name: vEEPromInit
**
**  Description:
**               This module is to initialize the eeprom related registers
**                and flags.
**
```

```
**  Operation:
**                -
** ========================================================================
*/

void vEEPromInit(void)
{
    bEEPromErrorDataPort = 0;                // Set output latch low
    bEEPromErrorDirPort = 0;                 // Set EEProm error for output direction
    bEEPromReadyDataPort = 0;                // Set Output latch low
    bEEPromReadyDirPort = 0;                 // Set EEProm ready for output direction
    bEEPromStartDataPort = 0;                // Set output latch low
    bEEPromStartDirPort = 1;                 // Port direction bit for EEProm start
    bEEPromStartPullUpPort = 1;              // Pull-up resistor for EEProm start

   uchEEPromAddress = DEVICE_ADDRESS;        // Set device address
    uchEEPromData = 0;                       // EEProm data

    bEEPromError = 0;                        // No EEProm error
    bEEPromBreak = 0;                        // EEProm break
}

/* ========================================================================
**  Module name: vI2CInit
**
**  Description:
**               This module is to initialize the i2c module control and mode
**                registers.
**
**  Operation:
**                -
** ========================================================================
*/

void vI2CInit(void)
{
    bSDA0DataPort = 0;                       // Set output latch low
    bSCL0DataPort = 0;                       // Set output latch low
    bSDA0DirPort  = 0;                       // Set SDA0 to output direction
    bSCL0DirPort  = 0;                       // Set SCL0 to output direction

// IICCL0
//———-

    bSMC0 = 0;                               // Operation in standard mode
    bCL00 = 1;                               // Clock speed fx/86 (97.5kHz@8.386MHz)
    bDFC0 = 1;                               // Digital filter off

// IICC0
//———

    bIICE0 = 1;                              // Enable I2C operation
    bLREL0 = 1;                              // Exit from the current communication operation
                                             // SCL0 and SDA0 lines go into high impedance
                                             // state
                                             // MUST BE DONE BEFORE SPT0 WILL BE SET,
                                             // OTHERWISE SPD0 WILL BE CLEARED
    bSPT0  = 1;                              // Generate stop condition
    bSPIE0 = 0;                              // Disable generation of interrupt request when
                                             // stop condition is detected
    bWTIM0 = 1;                              // Interrupt request is generated at the ninth
                                             // clock's falling edge
    bACKE0 = 0;                              // No auto acknowledge
    bWREL0 = 1;                              // Release in any case the wait state
    bSTT0  = 0;                              // Generate no start condition

    while(!bSPD0)                            // Wait for stop detection
         _NOP();

    bIICMK0 = 1;                             // Disable I2C interrupt
    bIICIF0 = 0;                             // Reset I2C interrupt request flag
}

/* ========================================================================
```

```
**  Module name: vTM51Init
**
**  Description:
**               This module is to initialize the timer 51 control and mode
**                registers.
**
**  Operation:
**                -
** ============================================================================
*/

void vTM51Init(void)
{
    TMC51 &= TM51_STOP;                       // Timer 51 stop

    CR51 = 163;

    TCL51 = 0x07;                             // 00000111 = 0x07
                                              // ——000 - TI51 falling edge
                                              // ——001 - TI51 rising edge
                                              // ——010 - fx/2
                                              // ——011 - fx/2^3
                                              // ——011 - fx/2^5
                                              // ——011 - fx/2^7
                                              // ——011 - fx/2^9
                                              // ——011 - fx/2^11

    TMC51 = 0x04;                             // 00000100 = 0x04
                                              // ||-||||0 - output disabled
                                              // ||-||||1 - output enabled
                                              // ||-|||0— active level high
                                              // ||-|||1— active level low
                                              // ||-|00—— timer output no change
                                              // ||-|01—— timer output reset
                                              // ||-|10—— timer output set
                                              // ||-0——— single mode
                                              // ||-1——— cascade mode
                                              // |0———— clear and start mode by matching
                                              // CR51
                                              // |1———— PWM (free-running) mode
                                              // 0———— after cleaning to 0, count up
                                              // disabled
                                              // 1———— count operation start

    bTMMK51 = 1;                              // Disable timer 51 interrupt
    bTMIF51 = 0;                              // Reset timer 51 interrupt request flag
}

/* ============================================================================
**  Module name: uchEEPromWrite
**
**  Description:
**               This module writes data to the EEProm through I2C bus.
**
**  Operation:
**                -
** ============================================================================
*/

void vEEPromWrite(unsigned char uchByteAddress, unsigned char uchDataIn)
{

    bIICMK0 = 1;                              // Disable I2C interrupt
    bACKE0  = 0;                              // No autoacknowledge

   SET_START_CONDITION;
                                              // Send device address and
    IIC0 = uchEEPromAddress & WRITE;          // set write mode
   WAIT_WRITE_INT_AND_ACK;

    IIC0 = uchByteAddress;                    // Send EEProm byte address
   WAIT_WRITE_INT_AND_ACK;

    IIC0 = uchDataIn;                         // Send data
```

```
   WAIT_WRITE_INT_AND_ACK;

    SET_STOP_CONDITION;

    vEEPromWait();                          // Wait EEProm write time = 40ms

    bEEPromError = 0;                       // No EEprom error

    return;
}

/* ============================================================================
**  Module name: vEEPromRead
**
**  Description:
**               This module reads data from the EEProm through I2C bus.
**
**  Operation:
**                -
** ============================================================================
*/

void vEEPromRead(unsigned char uchByteAddress)
{
    bIICMK0 = 1;                            // Disable I2C interrupt
    bACKE0  = 0;                            // No autoacknowledge

   SET_START_CONDITION;
                                            // Send device address and
    IIC0 = uchEEPromAddress & WRITE;        // set write mode
   WAIT_WRITE_INT_AND_ACK;

    IIC0 = uchByteAddress;                  // Send EEProm byte address
   WAIT_WRITE_INT_AND_ACK;

   SET_START_CONDITION;                     // Repeated start condition
                                            // Send device address and
    IIC0 = uchEEPromAddress | READ;         // set read mode
   WAIT_WRITE_INT_AND_ACK;

    bWTIM0 = 0;                             // Interrupt request at the eighth clock's
                                            // falling edge
    bACKE0 = 1;                             // Autoacknowledge
    bWREL0 = 1;                             // Cancel wait

   WAIT_READ_INT_AND_ACK;                   // Read data
    uchEEPromData = IIC0;

   SET_STOP_CONDITION;

    bEEPromError = 0;                       // No EEProm error

    bWTIM0 = 1;                             // Interrupt request at the ninth clock's
                                            // falling edge
    bACKE0 = 0;                             // No autoacknowledge

    return;
}

/* ============================================================================
**  Module name: vEEPromWait
**
**  Description:
**               This module is the wait routine for EEProm write cycles.
**
**  Operation:
**                The EEProm write time is 40ms.
**               TCL51 = 0x07 => fx/2^11 = 4.096MHz@8.386MHz (->vTM51Init)
**                              => timer clock time = 244µs.
**               EEProm wait time = (CR51+1) * 244µs
**                With CR51 = 163:
**               EEProm wait time = (163+1) * 244µs = 40ms.
**
** ============================================================================
```

```
*/

void vEEPromWait(void)
{
    CR51 = 163;                                 // Set compare value
    bTMIF51 = 0;                                // Reset timer 51 interrupt request flag
    TMC51 |= TM51_START;                        // Start timer 51
    while(!bTMIF51)                             // Wait for timer 51 interrupt request
         ;                                      // = end of EEProm wait time 40ms
    TMC51 &= TM51_STOP;                         // stop timer 51
    bTMIF51 = 0;                                // Reset timer 51 interrupt request flag
}

/* ============================================================================
**  main function
** ============================================================================
*/

void main(void)
{
    _DI();                                      // Disable all interrupts
    vHardwareInit();                            // Peripheral settings
    vEEPromInit();                              // Initialization measurement variables
    vI2CInit();                                 // Initialization I2C
    vTM51Init();
    _EI();                                      // Enable all interrupts

    while(1)
    {
        bEEPromReadyDataPort = 1;               // EEProm ready
        _NOP();

       while(bEEPromStartDataPort)              // Wait for start
             ;

        bEEPromErrorDataPort = 0;               // No EEPRom error
        bEEPromReadyDataPort = 0;               // EEProm not ready

                                                // Write all 128 bytes EEProm memory with
                                                // 0x55 and 0xAA
    for(uchEEPromByteAddress=0;uchEEPromByteAddress<128;uchEEPromByteAddress++)
        {                                       // Write data 0x55
          vEEPromWrite(uchEEPromByteAddress,0x55);

           if(!bEEPromError)                    // Check, if EEProm write error
            {                                   // No => Read written data
               uchEEPromData = 0;
              vEEPromRead(uchEEPromByteAddress);
                                                // Check, if data ok
              if(bEEPromError|| (uchEEPromData!=0x55))
                    bEEPromBreak = 1;           // No => EEProm break
            }
            else
                bEEPromBreak = 1;               // Yes=> EEProm break

           if(!bEEPromBreak)                    // If no EEprom error
            {                                   // Select next EEProm byte address
               uchEEPromByteAddress+=1;
                                                // Write data 0xAA
             vEEPromWrite(uchEEPromByteAddress,0xAA);

              if(!bEEPromError)                 // Check, if EEprom write error
               {                                // No => Read written data
                   uchEEPromData = 0;
                  vEEPromRead(uchEEPromByteAddress);
                                                // Check, if data ok
                  if(bEEPromError|| (uchEEPromData!=0xAA))
                        bEEPromBreak = 1;       // No => EEProm break
               }
               else
                   bEEPromBreak = 1;                // Yes=> EEProm break
           }
        }
       if(bEEPromBreak)                         // If EEProm error
```

```
            bEEPromErrorDataPort = 1;  // Set port latch to display
                                       // EEProm error

        while(!bEEPromStartDataPort)   // Wait for start
              ;
    }
}
```

Chapter 6 Using the 8-Bit Timer/Event Counter in Cascade Connection Mode

1 Introduction

In many applications the timer is used for multiple tasks.

2 Example Program

This example program provides an example of a stop watch using the 8-bit timer/event counter in cascade connection mode with 16-bit resolution.

3 TM50 and TM51 Configuration for Cascade Connection Mode

To use the 8-bit timer/event counter the timer operation must be stopped before setting the control registers and then the configuration of these registers is necessary.

3.1 Timer Clock Select Register TCL50

This register sets the count clocks of 8-bit timer/event counter and the valid edge of TI50, TI51 input.
The count clock in cascade mode is selected in TCL50, so the TCL51 need not to be selected.

In this application the count clock is set to fx/28.

=> TCL50 = 0x06

3.2 8-Bit Timer Mode Control Registers TMC50 and TMC51

The cascade mode of the timers is adjusted, when TMC50 and TMC51 are initialized to:

TMC50 = 0000xxx0B, x = don’t care
TMC51 = 0001xxx0B, x = don’t care

In this application the value written to TMC50 is 0x00 and the value written to TMC51 is 0x10.

3.3 Other Settings

INTTM51 of TM51 is generated when TM51 count value matches CR51, even if cascade-connection is used. Ensure to mask TM51 to prohibit interrupt.

4 Timer Operation

The timer operates as 16-bit interval timer in cascade-connection mode, which generates interrupt requests repeatedly at intervals of the count value preset to 8-bit compare registers CR51and CR50.

In this application a timer interval of 1s is needed as time base. The value of combined compare registers CR51 and CR50 can be calculated by the following formula:

$$\text{interval time} = \frac{((CR51,CR50)+1)}{f_{TM50}}$$

TCL50 = 0x06 $\Rightarrow f_{TM50} = \frac{f_x}{s^8}$

$\Rightarrow f_{TM50}$ = 32.758 KHz @ 8.386 MHz

(CR51, CR50) = (interval time * f_{TM50})-1

(CR51, CR50) = 32757

=> CR51 = 7F
=> CR50 = F5

5 External Interrupts

In this application the stop watch is started and stopped via external interrupts.
The port 0 can be used as an external interrupt request input.

5.1 External interrupt Configuration

To use the port 0 for external interrupt request, it must be specified with the port mode register PM0. If a falling edge for interrupt request is necessary, an on-chip pull-up resistor can be used with the pull-up resistor option register PU0.
The valid edge for external interrupts can be specified with the external interrupt rising/falling edge enable register EGP/EGN.

5.1.1 Port Mode Register PM0

For external interrupt the used port pins must be specified for input.
In this application for starting the stop watch INTP0 is used, and for stopping the stop watch INTP1 is used.

=> The port mode bits PM0.0 and PM0.1 must be set to 1.

5.1.2 Pull-up Resistor Option Register PU0

In this application a falling edge for the external interrupts is needed. With open port pins and used pull-up registers the input is on high level. So, if the input signal is switched to low level, a falling edge is generated.

=> For using the needed pull-up resistors the pull-up resistor option bits PU0.0 and PU0.1 must be set to 1.

5.1.3 External Interrupt Rising/Falling Edge Enable Registers EGP/EGN

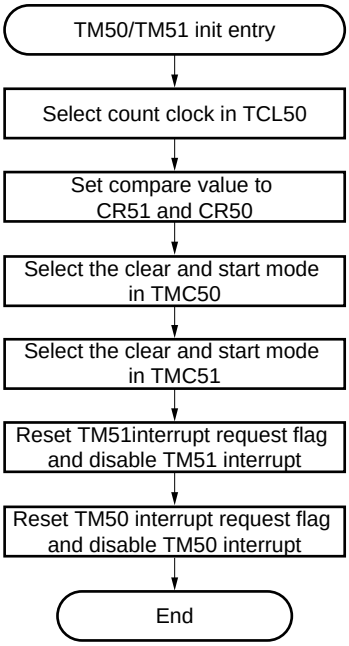
To enable the falling edge for external interrupts INTP0 and INTP1 the registers EGP and EGN has to be setup with the following values:

=> EGP = 0x00, EGN = 0x03

6 Flow Charts

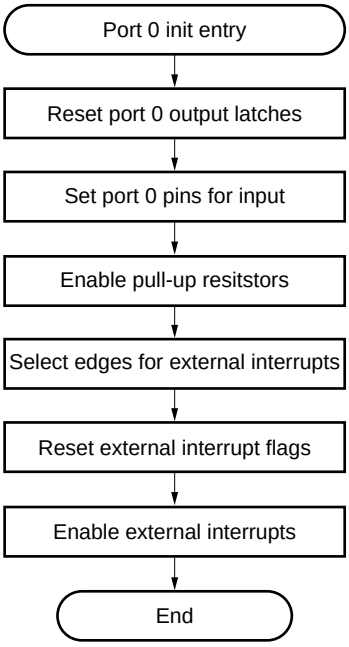
6.1 Configuration Timer 50 and Timer 51 for Cascade Mode

Figure 6-1: Configuration Timer 50 and Timer 51 for Cascade Mode



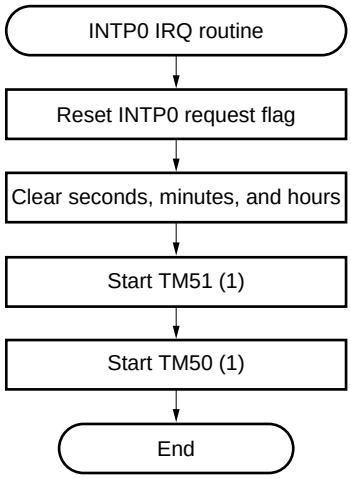
6.2 Configuration Port 0 for External Interrupts

Figure 6-2: Configuration Port 0



6.3 Interrupt Service Routine INTP0 (Start)

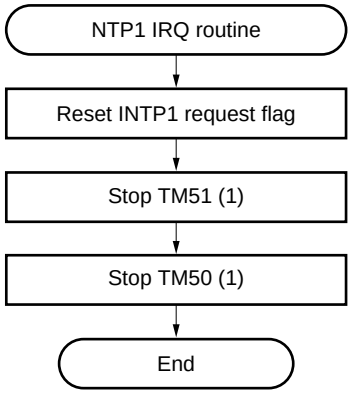
Figure 6-3: Interrupt Service Routine



(1) Start TM51 at first

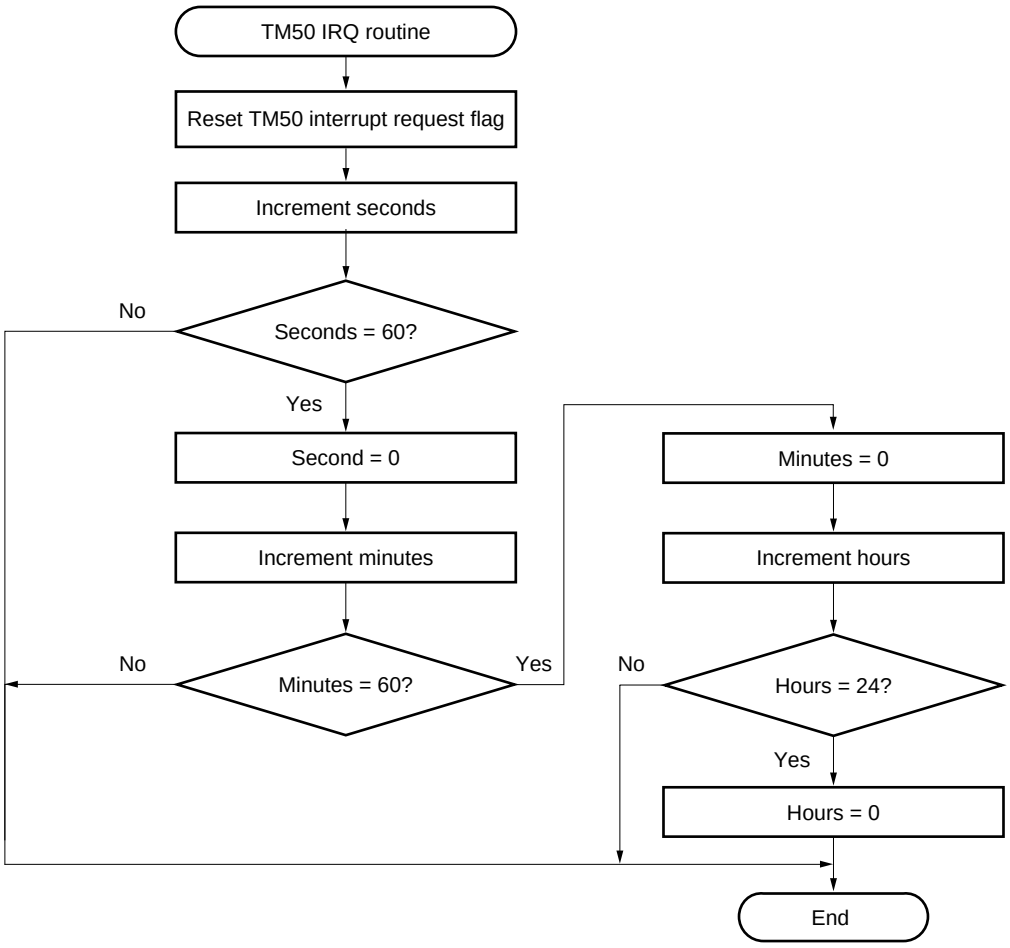
6.4 Interrupt Service Routine INTP1 (Stop)

Figure 6-4: INTP1



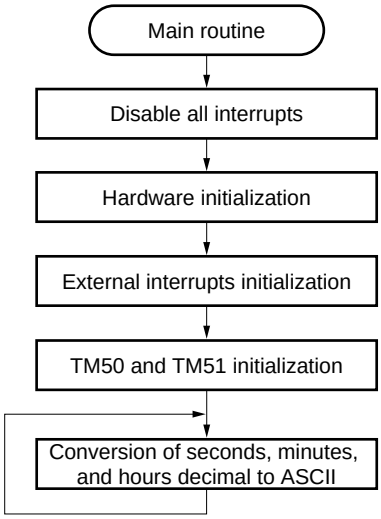
6.5 Interrupt Service Routine Timer 50

Figure 6-5: Timer 50



6.6 Main Routine

Figure 6-6: Main Routine



7 Sample Program

```
/*=============================================================================
**  PROJECT:      = IE-78000-R-A  Tests
**  MODULE        = tm50casc.c
**  SHORT  DESC.  = -
**  DEVICE        = uPD78F0034AY
**  VERSION       = 1.0
**  DATE          = 21.04.99
**  LAST  CHANGE  = -
**  ===========================================================================
**  Description:  The  time  variables  of  a  stop  watch  (1-Sec  resolution)  is
**                incremented  by  the  timer  50  interrupt.
**                The  timer  50  and  timer  51  are  cascade-connected  (16-Bit
**                resolution).  The  main  routine  converts  converts  the  time
**                variables  into  an  ascii  string.  This  string  could  be
**                displayed,  e.g.  on  a  LCD  (not  implemented).
**                The  external  interrupt  INTP0  starts  the  stop  watch.
**                The  external  interrupt  INTP1  stops  the  stop  watch.
**
**  ===========================================================================
**  Environment:  Device:             uPD78F0034AY
**                Assembler:          A78000           Version  3.11D
**                C-Compiler:         ICC78000         Version  3.14A
**                Linker:             XLINK            Version  4.50C
**
**  ===========================================================================
**  By:           NEC  Electronics  (Europe)  GmbH
**                Oberrather  Strasse  4
**                D-40472  Duesseldorf
**
**                dz,  NEC-EE,  EAD-TPS
**  ===========================================================================
**  Changes:      name/date/description
**===========================================================================
*/

#pragma  language  =  extended

#include  <io78003x.h>
#include  <in78000.h>

void  vHardwareInit(void);                // Initialization  hardware
void  vExtIntInit(void);                  // Initialization  external  interrupts
void  vInitTM5051(void);                  // Initialization  timer  50  and  timer  51

                                          // Interrupt  service  routine  to  start  clock
static  interrupt  [INTP0_vect]  void  vClockStart(void);
                                          // Interrupt  service  routine  to  stop  clock
static  interrupt  [INTP1_vect]  void  vClockStop(void);
                                          // Interrupt  service  routine  for  timer  50
static  interrupt[INTTM50_vect]  void  vIntTM50(void);

//————————————————————————————
//  General
//————————————————————————————

#pragma  memory  =  saddr

unsigned  char  uchHour;
unsigned  char  uchMinute;
unsigned  char  uchSecond;

unsigned  char  uchTime[9]  =  {0x30,0x30,':',0x30,0x30,':',0x30,0x30,'\0'};

#pragma  memory  =  default

//————————————————————————————
//  External  interrupts
//————————————————————————————

//  Ports
//———
```

```
bit  bStart = P0.0;                          // Port  for  Clock  start
bit  bStop  = P0.1;                          // Port  for  clock  stop

bit  bPM00  = PM0.0;
bit  bPM01  = PM0.1;

bit  bPU00  = PU0.0;
bit  bPU01  = PU0.1;

//  Interrupts
//------

bit  bPMK0  = MK0L.1;                        // External  interrupt  INTP0
bit  bPIF0  = IF0L.1;

bit  bPMK1  = MK0L.2;                        // External  interrupt  INTP1
bit  bPIF1  = IF0L.2;

//---------------------------------------
//  Timer50
//---------------------------------------

#define  TM50_START  0x80
#define  TM50_STOP    0x7F


//  Interrupts
//------

bit  bTMMK50  =  MK0H.6;

bit  bTMIF50  =  IF0H.6;

bit  bTMPR50  =  PR0H.6;

//---------------------------------------
//  Timer51
//---------------------------------------

#define  TM51_START  0x80
#define  TM51_STOP    0x7F


//  Interrupts
//------

bit  bTMMK51  =  MK0H.7;

bit  bTMIF51  =  IF0H.7;

bit  bTMPR51  =  PR0H.7;

/*============================================================================
**    Module  name:  vHardwareInit
**
**  Description:
**                     This  module  is  to  initialize  some  peripherals  hardware.
**
**  Operation:
**                     Sets  the  clock  generator,  the  port  modes  and  output  latches
**                    and  initializes  the  interrupts.
**============================================================================
*/

void  vHardwareInit(void)                    // Hardware  initialization
{
//  clock  generator  setting
//-------------
     PCC   = 0x00;                           // Use  high  speed  mode  (238ns@8.386MHz)
     OSTS  = 0x01;                           // 3.28ms  wait  after  stop  release  by  inter-
rupt
     IMS   = 0xC8;                           // !!!!  Select  1024  Byte  RAM  and  32  kByte
ROM
```

```
//  port  setting
//-------
     P0=0x00;                                  // Set output latch to 0
     P2=0x00;                                  // Set output latch to 0
     P3=0x00;                                  // Set output latch to 0
     P4=0x00;                                  // Set output latch to 0
     P5=0x00;                                  // Set output latch to 0
     P6=0x00;                                  // Set output latch to 0
     P7=0x00;                                  // Set output latch to 0

     PM0 = 0xF0;                               // Port  0 = output  (4-bits)
     PM0 = 0xC0;                               // Port  2 = output  (6-bits)
     PM0 = 0x80;                               // Port  3 = output  (7-bits)
     PM0 = 0x00;                               // Port  4 = output
     PM0 = 0x00;                               // Port  5 = output
     PM0 = 0x0F;                               // Port  6 = output  (4-bits)
     PM0 = 0xC0;                               // Port  7 = output  (6-bits)

//  interrupt  setting
//---------

      IF0L  =  0x00;
      IF0H  =  0x00;
      IF1L  =  0x00;

      MK0L  =  0xFF;
      MK0H  =  0xFF;
      MK1L  =  0xFF;

      PR0L  =  0xFF;
      PR0H  =  0xFF;
      PR1L  =  0xFF;
}

/*=============================================================================
**    Module  name:  vTM5051Init
**
**  Description:
**                    This module is to initialize the timer 50 and timer 51 control
**                   mode registers.
**
**  Operation:
**                  -
**===========================================================================
*/

void  vTM5051Init(void)                    // Hardware  initialization
{
      TMC50  &=  TM50_STOP;                // Timer  50  stop
      TMC51  &=  TM51_STOP;                // Timer  51  stop

//  Select  count  clock  in  TM50
//  Cascade-connected  TM51  need  not  be  selected  =>  No  TCL51  setting

      TCL50  =  0x06;                      // 00000110  =  0x06
                                           // ---000  -  TI51  falling  edge
                                           // ---001  -  TI51  rising  edge
                                           // ---010  -  fx/2
                                           // ---011  -  fx/2^2
                                           // ---100  -  fx/2^4
                                           // ---101  -  fx/2^6
                                           // ---110  -  fx/2^8
                                           // ---111  -  fx/2^10

//  Compared  value  =  time  base  for  1  second  @  8.386MHz

      CR50  =  0xF5;
      CR51  =  0x7F;

//  Select  the  clear  &  start  mode  by  match  of  TM50  and  CR50/TM51  and  CR51
//  TM50  ->  TMC50  =  0000xxx0B,  x  =  don't  care
//  TM51  ->  TMC51  =  0001xxx0B,  x  =  don't  care

      TMC50  =  0x00;                      // 00010000  =  0x01
```

```
                                        // ||-||||0  -  output  disabled
                                        // ||-||||1  -  output  enabled
                                        // ||-|||0—  active  level  high
                                        // ||-|||1—  active  level  low
                                        // ||-|00——  no  change
                                        // ||-|01——  timer  output  reset
                                        // ||-|10——  timer  output  set
                                        // ||-0———  single  mode
                                        // ||-1———  cascade  mode
                                        // |0————  clear  and  start  by
                                        // |        matching  between  CR50  and  TM50
                                        // |1————  cascade  mode
                                        // 0————  after  cleaning  to  0,  count  up
                                        //                disabled
                                        // 1————  count  operation  start

     TMC51  =  0x10;                    // 00000000  =  0x00
                                        // ||-||||0  -  output  disabled
                                        // ||-||||1  -  output  enabled
                                        // ||-|||0—  active  level  high
                                        // ||-|||1—  active  level  low
                                        // ||-|00——  no  change
                                        // ||-|01——  timer  output  reset
                                        // ||-|10——  timer  output  set
                                        // ||-0———  single  mode
                                        // ||-1———  cascade  mode
                                        // |0————  clear  and  start  by
                                        // |          matching  between  CR51  and  TM51
                                        // |1————  cascade  mode
                                        // 0————  after  cleaning  to  0,  count  up
                                        //               disabled
                                        // 1————  count  operation  start

//  INTTM51  of  TM51  is  generated  when  TM51  count  value  matches  CR51,  even  if
//  cascade  connection  is  used.  Ensure  to  mask  TM51  to  ptohibit  interrupt.

     bTMIF51  =  0;                     // Reset  timer  51  interrupt  request  flag
     bTMMK51  =  1;                     // Disable  timer51  interrupt  in  cascade
                                           mode

     bTMIF50  =  0;                     // Reset  timer  51  interrupt  request  flag
     bTMMK50  =  0;                     // Enable  timer  50  interrupt
}

/*=============================================================================
**    Module  name:  vExtIntInit
**
**  Description:
**                    This  module  is  to  initialize  external  interrupt  control  and
**                   mode  registers.
**
**  Operation:
**                  -
**===========================================================================
*/

void  vExtIntInit(void)
{
     bStart  =  0;                      // Set  output  latches  to  0
     bStop   =  0;

     bPM00   =  1;                      // P0.0  =  input
     bPM01   =  1;                      // P0.1  =  input

     bPU00   =  1;                      // Enable  pull-up  resistor  for  P0.0
     bPU01   =  1;                      // Enable  pull-up  resistor  for  P0.1

     EGP  =  0x00;                      // Set  external  interrupts  for
     EGN  =  0x03;                      // for  falling  edges

     bPIF0  =  0;                       // Reset  INTP0  request  flag
     bPMK0  =  0;                       // Enable  INTP0

     bPIF1  =  0;                       // Reset  INTP1  request  flag
```

```
     bPMK1 = 0;                          // Enable  INTP1
}

/*=============================================================================
**    Module  name:  vClockStart
**
**  Description:
**                   This module is to initialize the timer 50 and timer 51 control
**                  mode registers.
**
**  Operation:
**                 -
**===========================================================================
*/

static  interrupt  [INTP0_vect]  void  vClockStart(void)
{
     bPIF0 = 0;                          // Reset  INTP0  request  flag
     uchHour = 0;
     uchMinute = 0;
     uchSecond = 0;

//  When  TMC51.TCE51  is  set  to  1,  and  then  TMC50.TCE50  is  set  to  1,  count
//  operation  starts.

      TMC51 |= TM51_START;               // Timer  51  start
      TMC50 |= TM50_START;               // Timer  50  start
}

/*=============================================================================
**    Module  name:  vClockStart
**
**  Description:
**                   This module is to initialize the timer 50 and timer 51 control
**                  mode registers.
**
**  Operation:
**                 -
**===========================================================================
*/

static  interrupt  [INTP1_vect]  void  vClockStop(void)
{
     bPIF1 = 0;                          // Reset  INTP1  request  flag

      TMC50 &= TM50_STOP;                // Timer  50  stop
      TMC51 &= TM51_STOP;                // Timer  51  stop
}

/*=============================================================================
**    Module  name:  vIntTM50
**
**  Description:
**                   This module is to initialize the timer 50 and timer 51 control
**                  mode registers.
**
**  Operation:
**                 -
**===========================================================================
*/

static  interrupt  [INTTM50_vect]  void  vIntTM50(void)
{
     bTMIF50 = 0;                        // Reset  timer  50  interrupt  request  flag

      uchSecond++;                       // Calculation  of  second,  minute,
       if(uchSecond==60)                 // and  hour  up  to  23:59:59
     {
          uchSecond = 0;
          uchMinute++;
           if(uchMinute==60)
         {
              uchMinute = 0;
              uchHour++;
```

```
                if(uchHour==24)
                    uchHour = 0;
            }
        }
}

/*================================================================================
**  main  function
**================================================================================
*/
void  main(void)
{
     _DI();                                 // Disable  all  interrupts
      vHardwareInit();                      // Peripheral  settings
      vExtIntInit();                        // Initialization  external  interrupts
      vTM5051Init();                        // Initialization  timer  50  and  timer  51
     _EI();

     do                                     // Conversion  of  decimals  to  ASCII
     {
            uchTime[0]  =  uchHour/10+0x30;
            uchTime[1]  =  uchHour%10+0x30;

            uchTime[3]  =  uchMinute/10+0x30;
            uchTime[4]  =  uchMinute%10+0x30;

            uchTime[6]  =  uchSecond/10+0x30;
            uchTime[7]  =  uchSecond%10+0x30;
     }
      while(1);                             // Endless  loop
}
```

Chapter 7 Asynchronous Serial Interface (UART0)

1 Introduction

Asynchronous serial I/O is widely used in the industry, most commonly for RS232 communication.
The data transmission/reception is provided at programmable baud rates, generated either by a dedicated on-chip baud rate generator or by scaling the input clock. The on-chip baud rate generator dedicated to the UART enables a wide range of selectable baud rates. The data frame of transmitted/received data consists of a start bit, character bits, a parity bit and one or two stop bits.

2 Example Program

This example program initializes the UART0 control registers for asynchronous serial interface mode. The serial input data, the serial output data and errors on input data routines are interrupt driven.
The received data will be written to a ring buffer (read buffer). After that the main routine reads the received data from the ring buffer (read buffer) and writes it to a second ring buffer (write buffer). The first data written to the ring buffer (write buffer) will also be written to the transmit shift register TXS0, for starting the transmission process.

3 Serial Interface Configuration

To use the asynchronous serial interface the data transfer and the baud rate must be selected and the configuration of the control registers ASIM0 and BRGC0 is necessary.

3.1 Baud Rate

The baud rate generator is configured in writing the appropriate value to BRGC0 register.

The transmit/receive clock that is used to generate the baud rate is obtained by dividing the main system clock. The baud rate is determined according to the following formula:

$$[\text{Baud rate}] = \frac{f_x}{2^{n+1}*(k+16)} \ [\text{Hz}]$$

fx = Oscillation frequency of main system clock
n = Value set by TPS00 to TPS02 in the baud rate control register BRGC0 ($0 \le n \le 7$)
k = Value set by MDL00 to MDL02 in the baud rate control register

Example for 9600 Baud:

Oscillation frequency of main system clock: f_X = 8.386 MHz
5-bit counter as clock source $f_{SCK} = f_x /24$: n = 4
Input clock selection for baud rate generator: k = 11

With BRGC0 = 0x4B, the calculated value for the baud rate, according to the above equation is:

$$[\text{Baud rate}] = \frac{8.386\ \text{MHz}}{2^{4+1}*(11+16)}\ [\text{Hz}]$$

[Baud rate] = 9706 [Hz]

In this case the baud rate error to 9600 Baud is 1.10%. The error tolerance range for baud rates depends on the number of bits per frame and the counter's division rate [1/(k+16)].

3.2 Asynchronous Serial Interface Mode

After the setup of the baud rate generator, it is necessary to configure the ASIM0 control register. The operating mode, the data frame and reception error handling are dependent of the ASIM0 register setting.

Example UART specification:

– UART transmit/receive mode
– Receive completion interrupt request is not issued when an error occurs
– 1 start bit, 8 data bits, no parity, 1 stop bit

=> ASIM0 = 0xCA

4 Transmitting and Receiving Interrupt Driven

4.1 Transmission

Transmit operation is started when transmit data is written to transmit shift register TXS0. A start bit, parity bit and stop bit(s) are automatically added to the data. Starting transmit operation shifts out the data in TXS0, after which a transmission complete interrupt request INTST0 is generated.

4.2 Reception

Once reception of one data frame is completed, the received data in the shift register is transferred to receive buffer register (RXB0) and a reception complete interrupt (INTSR0) occurs.
Even if an error occurred, the received data in which the error occurred is still transferred to RXB0. When ASIM0 bit 1 (ISRM0) is cleared upon occurrence of an error, INTSR0 occurs. When ISRM0 bit is set, INTSR0 does not occur.
If the RXE0 bit is reset during a receive operation, receive operation is stopped immediately. At this time, the contents of RXB0 and ASIS0 do not change, nor does INTSR0 or INTSER0 occur.

4.3 Receive Errors

Three types of errors can occur during a receive operation: parity error, framing error or overrun error. If, as result of a data reception, an error flag is set in the asynchronous serial interface status register (ASIS0), a reception error interrupt request (INTSER0) will occur.

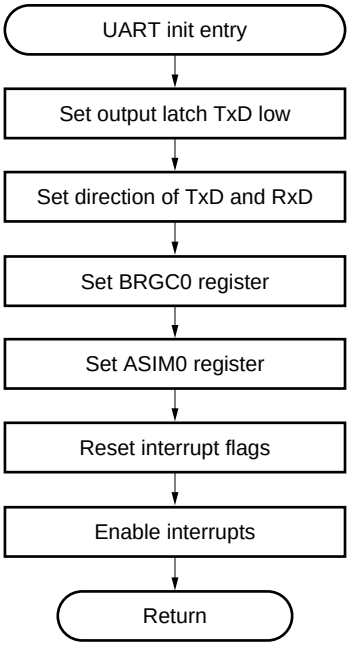
5 Ring Buffer

The ring buffers - used by this example program – are two 16 byte arrays each, organized by a write index and a read index.
The read index marks the first byte, which could be read out of the buffer and the write index marks the next free position to write a byte.
The buffer is a ring buffer, so it could be possible that the value of the read index is higher than the value of the write index. In this case the bytes between the read index and the physical end of the buffer as well as the bytes between the physical start and the write index are valid. In contrast to this the bytes between the write index and the read index are empty and could take more bytes.
Due to the ring organization, the buffer is empty, if the read index and the write index are equal. The buffer is full, if the write index marks the byte before the read index.

6 Flow Charts

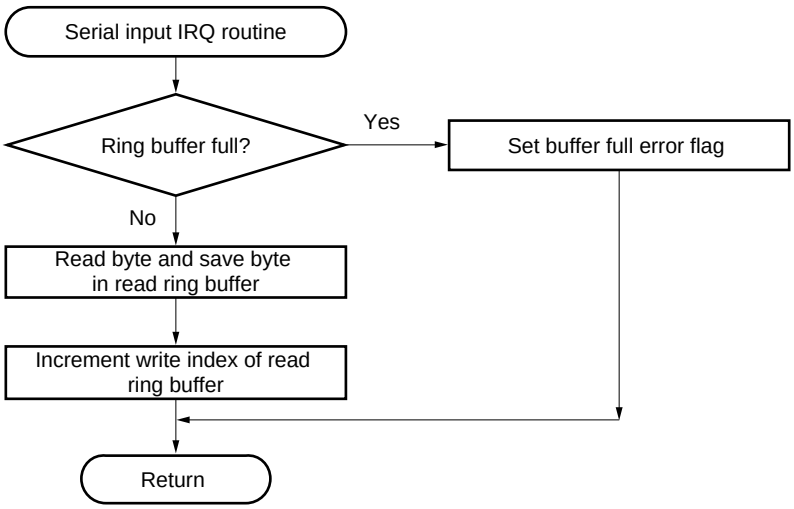
6.1 Configuration UART0

Figure 7-1: Configuration UART0



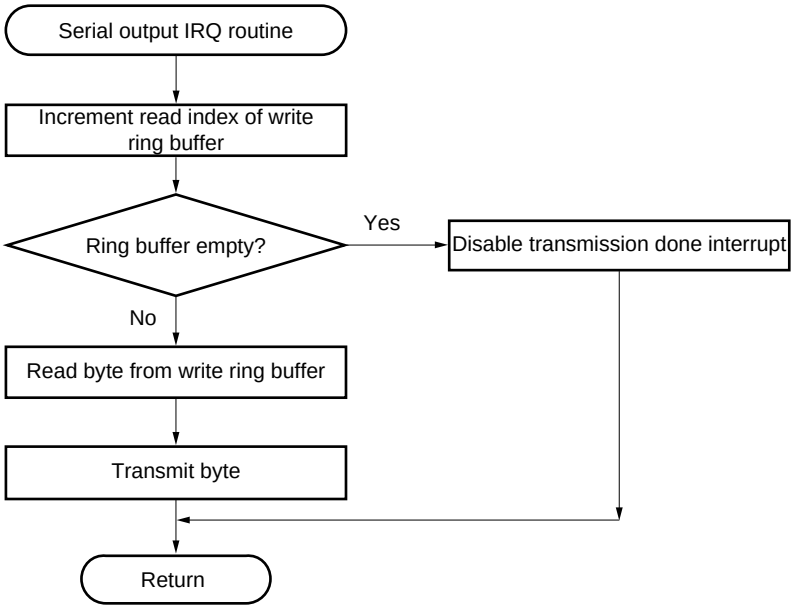
6.2 Receiving Data

Figure 7-2: Receiving Data



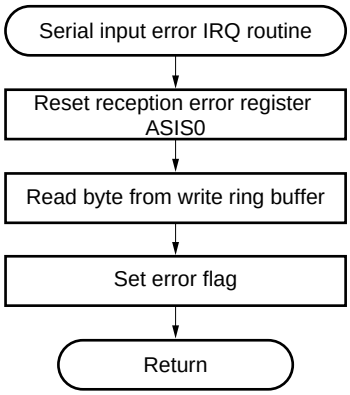
6.3 Transmitting Data

Figure 7-3: Transmitting Data



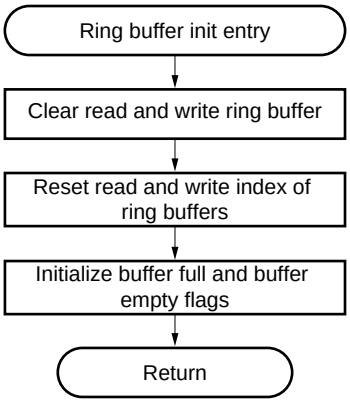
6.4 Reception Error Interrupt

Figure 7-4: Reception Error Interrupt



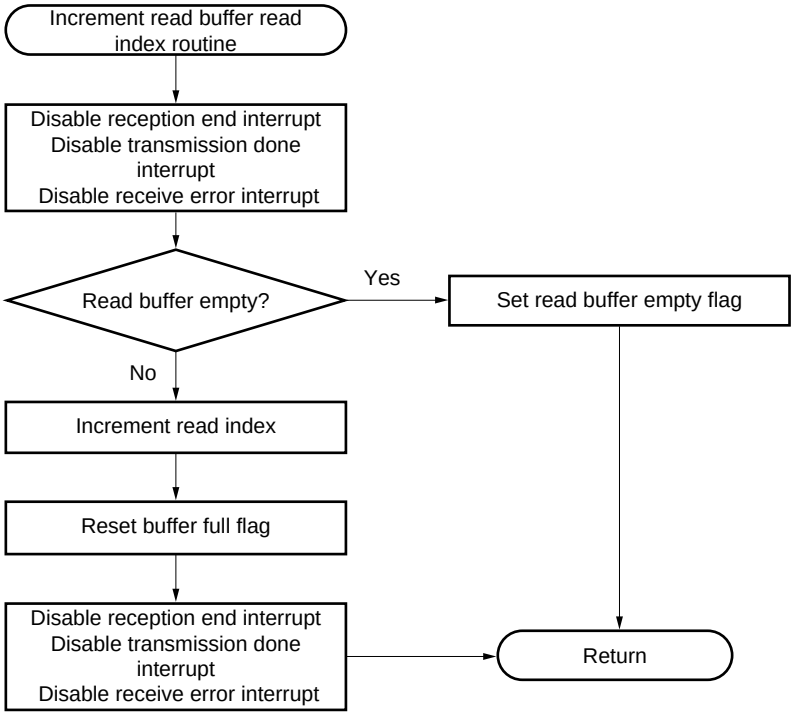
6.5 Configuration Ring Buffer

Figure 7-5: Configuration Ring Buffer



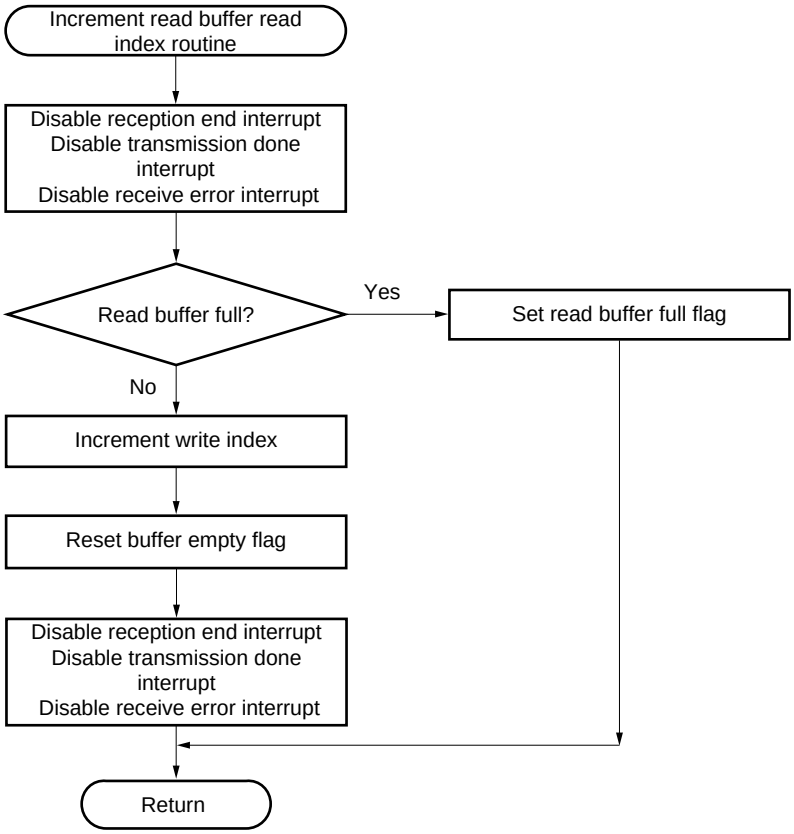
6.6 Increment Read Buffer Read Index

Figure 7-6: Increment Read Buffer Read Index



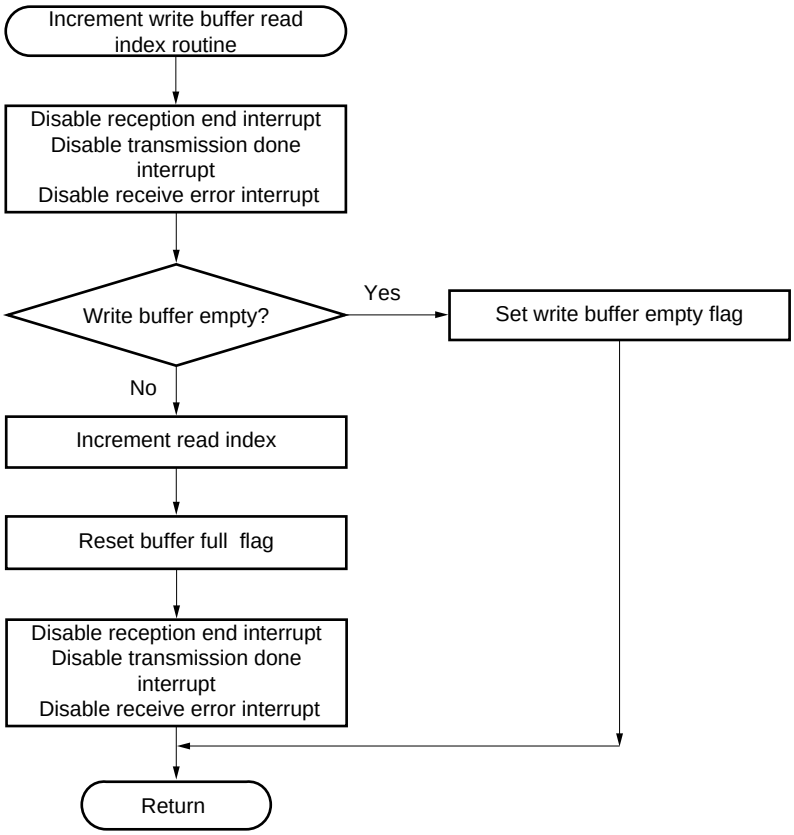
6.7 Increment Read Buffer Write Index

Figure 7-7: Increment Read Buffer Write Index



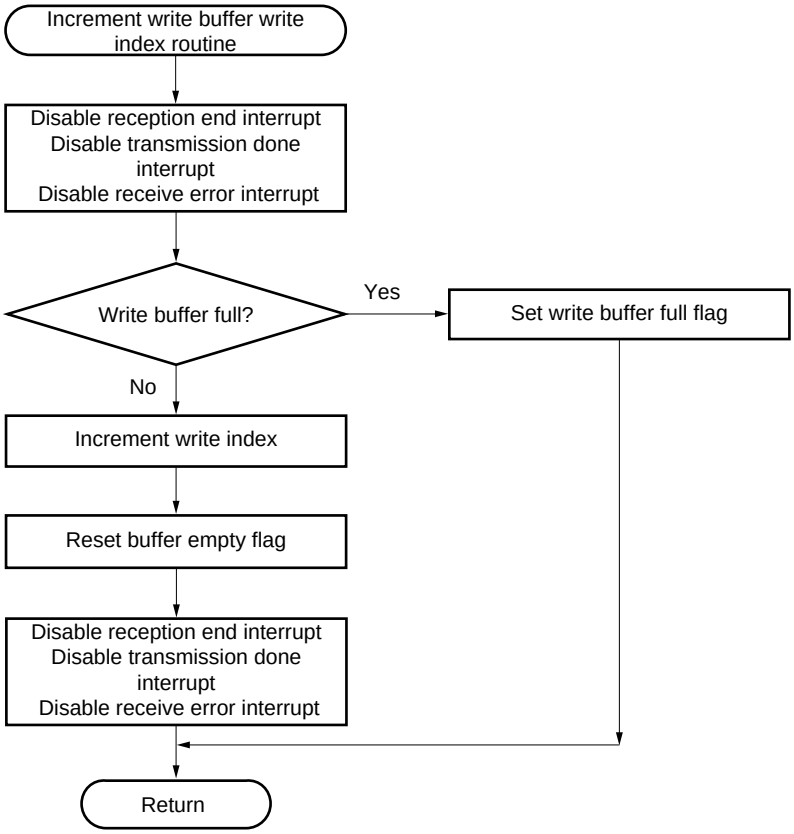
6.8 Increment Write Buffer Read Index

Figure 7-8: Increment Write Buffer Read Index



6.9 Increment Write Buffer Write Index

Figure 7-9: Increment Write Buffer Write Index



7 Sample Program

```
/*=============================================================================
** PROJECT         = IE-78001-R-A Tests
** MODULE          = UART0_TR.C
** SHORT DESC.     = This example program demonstrates the asynchronous serial
**                   interface UART.
** DEVICE          = uPD78F0034AY
** VERSION         = 1.0
** DATE            = 22.03.1999
** LAST CHANGE     = -
** ============================================================================
** Description:    Serial interface channel 0 is set by using the asynchronous
**                 serial interface mode register (ASIM0), the asynchronous
**                 serial interface status register (ASIS0), baud-rate generator
**                 control register (BRGC0) and the oscillation mode register
**                 (OSMS).
**                 The example program will receive a message and stores the
**                 message in the read buffer. The main routine reads the data
**                 from the read buffer and stores it in the write buffer.
**                 The data will be send by using the UART0.
**
**                 Baudrate: 9600Baud
**                 Data bits: 8
**                 Parity:    none
**                 Stop bits: 1
**                 Flow control: none
**
** ============================================================================
** Enviroment:     Device:             uPD78F0034AY
**                 Assembler:          A78000         Version 1.13A
**                 C-Compiler:         ICC78000       Version 3.13A
**                 Linker:             XLINK          Version 4.50
**
** ============================================================================
**  By:            NEC Electronics (Europe) GmbH
**                 Oberrather Strasse 4
**                 D-40472 Duesseldorf
**
**                 dz, NEC-EE, EAD-TPS
** ============================================================================
Changes:
** ============================================================================
*/

#pragma language = extended

#include <io78003x.h>
#include <in78000.h>
#include <string.h>

//#define Bd1200 1
#define Bd9600 1
//#define Bd115200 1

#define TRUE 1
#define FALSE 0

void vHardwareInit(void);                          // Initialization hardware
void vUART0Init(void);                             // Initialization UART0
void vRbInit(void);                                // Initialization ring buffers
void vIncRdIdxRdBuffer(void);                      // Increments the read buffer read index
void vIncWrIdxRdBuffer(void);                      // Increments the read buffer write index
void vIncRdIdxWrBuffer(void);                      // Increments the write buffer read index
void vIncWrIdxWrBuffer(void);                      // Increments the write buffer write index
bit bGetData(unsigned char *);                     // Reads data from read buffer
bit bSendData(unsigned char *);                    // Writes data to write buffer

                                                   // Interrupt service routine for byte reception
static interrupt [INTSR0_vect] void vIntUART0Receive(void);
                                                   // Interrupt service routine for byte
                                                   // transmission
static interrupt [INTST0_vect] void vIntUART0Transmit(void);
                                                   // Interrupt service routine for a byte
```

```
                                                    // reception error
static interrupt [INTSER0_vect] void vIntUART0ReceptionError(void);

//————————————————————————————————————-
// UART0
//————————————————————————————————————-

// Port bits
//——————
bit bRxDDirPort  = PM2.3;                           // Port direction bit for RxD
bit bTxDDirPort  = PM2.4;                           // Port direction bit for TxD
bit bTxDDataPort = P2.4;                            // Output latch bit for TxD

// Interrupt bits
//————————-
bit bSERMK0 = MK0L.5;                               // Receive error interrupt
bit bSRMK0  = MK0L.6;                               // Reception end interrupt
bit bSTMK0  = MK0L.7;                               // Transmission end interrupt

bit bSERIF0 = IF0L.5;                               // Receive error interrupt request flag
bit bSRIF0  = IF0L.6;                               // Reception interrupt request flag
bit bSTIF0  = IF0L.7;                               // Transmission done interrupt flag

bit bSERPR0 = PR0L.5;
bit bSRPR0  = PR0L.6;
bit bSTPR0  = PR0L.7;

//————————————————————————————————————-
// Read and write buffer
//————————————————————————————————————-

#define BUFFERSIZE 16                               // Buffer size

#define NO_BUFFER_ERROR 0                           // 'Ring buffer no error' error code
#define RD_BUFFER_FULL_ERROR 1                      // 'Read buffer full' error code
#define RD_BUFFER_EMPTY_ERROR 2                     // 'Read buffer empty' error code
#define WR_BUFFER_FULL_ERROR 3                      // 'Write buffer full' error code
#define WR_BUFFER_EMPTY_ERROR 4                     // 'Write buffer empty' error code
#define RCV_OVERRUN_ERROR 5                         // Overrun error code
#define RCV_FRAMING_ERROR 6                         // Framing error code
#define RCV_PARITY_ERROR 7                          // Parity error code
#define RCV_FATAL_ERROR 8                           // Multiple errors code

#pragma memory=saddr

// Read buffer
//———————
unsigned char uchRdBuffer[BUFFERSIZE];              // Read buffer
unsigned char uchRdIdxRdBuffer;                     // Read index read buffer
unsigned char uchWrIdxRdBuffer;                     // Write index read buffer

// Write buffer
//————————-
unsigned char uchWrBuffer[BUFFERSIZE];              // Write buffer
unsigned char uchRdIdxWrBuffer;                     // Read index write buffer
unsigned char uchWrIdxWrBuffer;                     // Write index write buffer

// Flags Read/Write buffer
//———————————
bit bRdBufferFull;                                  // Flag read buffer full
bit bRdBufferEmpty;                                 // Flag read buffer empty
bit bWrBufferFull;                                  // Flag write buffer full
bit bWrBufferEmpty;                                 // Flag write buffer empty

#pragma memory=default

unsigned char uchCommErrorCode;                     // Communication error code

bit b;

/* ======================================================================
**  Module name: vHardwareInit
**
**  Description:
```

```
**              This modul is to initialize some peripheral hardware.
**
**  Operation:
**              Sets the clock generator, the port modes and output latches
**               and initializes the interrupts.
** ========================================================================
*/
void vHardwareInit(void)
{

// clock generator setting
//————————————
    PCC  = 0x00;                                // Use high speed mode (250ns @ 8MHz)
    OSTS = 0x01;                                // 3.28ms wait after STOP release by interrupt
    IMS  = 0xC8;                                // !!!! Select 1024 Byte RAM and 32k Byte ROM

// port setting
//———————
    P0=0x00;                                    // Set output latch to 0
    P2=0x00;                                    // Set output latch to 0
    P3=0x00;                                    // Set output latch to 0
    P4=0x00;                                    // Set output latch to 0
    P5=0x00;                                    // Set output latch to 0
    P6=0x00;                                    // Set output latch to 0
    P7=0x00;                                    // Set output latch to 0

    MEM = 0x01;                                 // Important for keyReturn function

    PM0  = 0xF0;                                // Port 0  =  output (4-bits)
    PM2  = 0xC0;                                // Port 2  =  output (6-bits)
    PM3  = 0x80;                                // Port 3  =  output (7-bits)
    PM4  = 0x00;                                // Port 4  =  output
    PM5  = 0x00;                                // Port 5  =  output
    PM6  = 0x0F;                                // Port 6  =  output (4-bits)
    PM7  = 0xC0;                                // Port 7  =  output (6 bits)

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

    PR0L = 0xFF;
    PR0H = 0xFF;
    PR1L = 0xFF;
}

/* ========================================================================
**  Module name: vUART0Init
**
**  Description:
**           This modul is to initialize UART control and mode registers.
**
**  Operation:
**           Sets the dual ports to input for reception and output for
**           transmission.
**           Data format 9600Bd, 8 data bits, 1 stop bit, no parity
**           Enables transmit and receive mode.
** ========================================================================
*/
void vUART0Init(void)
{
    bRxDDirPort = 1;                            // Set RxD to input direction
    bTxDDirPort = 0;                            // Set TxD to output direction

    bTxDDataPort = 0;                           // Set output latch low

#ifdef Bd115200
    BRGC0 = 0x12;                               // 115200Bd, ERR% = 1.10% @ 8.386MHz
#else
#ifdef Bd9600
```

```
    BRGC0 = 0x4B;                                   // 9600Bd, ERR% = 1.10% @ 8.386MHz
#else
    BRGC0 = 0x7B;                                   // 1200Bd, ERR% = 1.10" @ 8.386MHz
#endif // Bd9600
#endif // Bd115200

    ASIM0 = 0xCA;                                   // 11001010
                                                    // |||||||0__ UART mode, no infrared
                                                    // ||||||1___ Reception error interrupt
                                                    // |||||0____ 1 Stop bit
                                                    // ||||1_____ 8 Data bits
                                                    // ||00______ no parity
                                                    // 11________ UART mode, transmit and receive

    bSTIF0  = 0;                                    // Reset Transmit done interrupt request flag
    bSERIF0 = 0;                                    // Reset receive error interrupt request
    bSRIF0  = 0;                                    // Reset reception interrupt request flag

    bSERMK0 = 0;                                    // Enable receive error interrupt
    bSRMK0  = 0;                                    // Enable reception end interrupt
    bSTMK0  = 0;                                    // Enable transmission end interrupt

   uchCommErrorCode = NO_BUFFER_ERROR;              // Reset data reception/buffer error
}

/* ============================================================================
**  Module name: vRbInit
**
**  Description:
**            This modul initializes the ring buffers.
**
**  Operation:
**            Contents of ring buffers are set to 0,
**            indizes are set to 0, and flags are set/reset.
** ============================================================================
*/

void vRbInit(void)
{
    unsigned char i;

    for(i = 0; i < BUFFERSIZE; i++)                 // Initialization read/write buffer
     {
         uchRdBuffer[i] = 0;
         uchWrBuffer[i] = 0;
     }

    uchRdIdxRdBuffer = 0;                           // Reset read index read buffer
    uchWrIdxRdBuffer = 0;                           // Reset write index read buffer
    uchRdIdxWrBuffer = 0;                           // Reset read index write buffer
    uchWrIdxWrBuffer = 0;                           // Reset write index write buffer

    bRdBufferFull  = 0;                             // Read buffer is not full
    bRdBufferEmpty = 1;                             // Read buffer is empty
    bWrBufferFull  = 0;                             // Write buffer is not full
    bWrBufferEmpty = 1;                             // Write buffer is empty

   uchCommErrorCode = NO_BUFFER_ERROR;              // No ring buffer error
}

/* ============================================================================
**  Module name: vIncRdIdxRdBuffer
**
**  Description:
**               This modul is to increment the read buffer read index.
**
**  Operation:
**
** ============================================================================
*/

void vIncRdIdxRdBuffer(void)
{
    bSRMK0 = 1;
```

```
    bSERMK0= 1;
    bSTMK0 = 1;

   if(!bRdBufferEmpty)                          // Check if read buffer is empty
    {
      if(uchRdIdxRdBuffer==BUFFERSIZE-1)
           uchRdIdxRdBuffer=0;                  // No => Increment read buffer
       else                                     //       read index
           uchRdIdxRdBuffer++;

        bRdBufferFull=0;                        // Read buffer is no longer full
    }

  if(uchRdIdxRdBuffer==uchWrIdxRdBuffer)
       bRdBufferEmpty=1;                        // Yes=> Set read buffer empty flag

    bSRMK0  = 0;
    bSERMK0 = 0;
    bSTMK0  = 0;
}

/* ============================================================================
**  Module name: vIncWrIdxRdBuffer
**
**  Description:
**               This modul is to increment the read buffer write index.
**
**  Operation:
**
** ============================================================================
*/

void vIncWrIdxRdBuffer(void)
{
    bSRMK0  = 1;
    bSERMK0 = 1;
    bSTMK0  = 1;

   if(!bRdBufferFull)                           // Check if read buffer full
    {
      if(uchWrIdxRdBuffer==BUFFERSIZE-1)
           uchWrIdxRdBuffer=0;                  // No => Increment read buffer
       else                                     //        write index
           uchWrIdxRdBuffer++;

        bRdBufferEmpty=0;                       // Read buffer is no longer empty
    }

  if(((uchRdIdxRdBuffer==0)&&(uchWrIdxRdBuffer==BUFFERSIZE-1))
    ||(uchWrIdxRdBuffer==uchRdIdxRdBuffer-1))
       bRdBufferFull=1;                         // Yes=> Set read buffer full flag

    bSRMK0  = 0;
    bSERMK0 = 0;
    bSTMK0  = 0;
}

/* ============================================================================
**  Module name: vIncRdIdxWrBuffer
**
**  Description:
**               This modul is to increment the write buffer read index.
**
**  Operation:
**
** ============================================================================
*/

void vIncRdIdxWrBuffer(void)
{
    bSRMK0  = 1;
    bSERMK0 = 1;
    bSTMK0  = 1;
```

```
   if(!bWrBufferEmpty)                          // Check if write buffer is empty
    {
      if(uchRdIdxWrBuffer==BUFFERSIZE-1)
           uchRdIdxWrBuffer=0;                  // No => Increment write buffer
       else                                     //       read index
           uchRdIdxWrBuffer++;

       bWrBufferFull=0;                         // Write buffer is no longer full
    }

  if(uchRdIdxWrBuffer==uchWrIdxWrBuffer)
       bWrBufferEmpty=1;                        // Yes=> Set write buffer empty flag

    bSRMK0  = 0;
    bSERMK0 = 0;
    bSTMK0  = 0;
}

/* ============================================================================
**  Module name: vIncWrIdxWrBuffer
**
**  Description:
**               This modul is to increment the write buffer write index.
**
**  Operation:
**
** ============================================================================
*/
void vIncWrIdxWrBuffer(void)
{
    bSRMK0  = 1;
    bSERMK0 = 1;
    bSTMK0  = 1;

   if(!bWrBufferFull)                           // Check if write buffer full
    {
      if(uchWrIdxWrBuffer==BUFFERSIZE-1)
           uchWrIdxWrBuffer=0;                  // No => Increment write buffer
       else                                     // write index
           uchWrIdxWrBuffer++;

       bWrBufferEmpty=0;                        // Write buffer is no longer empty
    }

  if(((uchRdIdxWrBuffer==0)&&(uchWrIdxWrBuffer==BUFFERSIZE-1))
    ||(uchWrIdxWrBuffer==uchRdIdxWrBuffer-1))
       bWrBufferFull=1;                         // Yes=> Set write buffer full flag

    bSRMK0  = 0;
    bSERMK0 = 0;
    bSTMK0  = 0;
}

/* ============================================================================
**  Module name: bGetData
**
**  Description:
**             This modul reads data from the read buffer.
**
**  Operation: If ring buffer is not empty, read byte.
**             Save byte.
**             Increment read buffer read index.
**             Return code for read byte ok or
**             return error code if necessary.
**
** ============================================================================
*/

bit bGetData(unsigned char *puchByte)
{
   if(!bRdBufferEmpty)                          // Check if read buffer is empty
    {       //No => Read byte from read buffer
      *puchByte = uchRdBuffer[uchRdIdxRdBuffer];
       vIncRdIdxRdBuffer();                     // Increment read buffer read index
```

```
        return (TRUE);
    }
    else
    {       //Yes=> Return error code
       uchCommErrorCode = RD_BUFFER_EMPTY_ERROR;
        return (FALSE);
    }

}

/* ============================================================================
**  Module name: bSendData
**
**  Description:
**              This modul writes data to the write buffer and/or transmitts
**              the first byte.
**
**  Operation:  If ring buffer is not full, write byte.
**              Transmit first byte.
**              Save byte to write buffer for logic order of buffer events.
**              Increment buffer pointer.
**              Return code for send byte ok or
**              return error code if necessary.
**
** ============================================================================
*/

bit bSendData(unsigned char *puchByte)
{
    if (!bWrBufferFull)                          // Check if write buffer is full
    {
        if(bWrBufferEmpty)                       // No => Check if write buffer is empty
        {
            bSTIF0=0;                            // Yes=> Reset Transmit done interrupt request
                                                 //       flag (for sure)
            bSTMK0=0;                            //       Enable transmission done interrupt
            TXS0 = *puchByte;                    //       Transmit first byte
        }  //                                      No => Write first byte to write buffer
       uchWrBuffer[uchWrIdxWrBuffer] = *puchByte; // for logic order of buffer events
        vIncWrIdxWrBuffer();                     //       Increment write buffer write index
        return (TRUE);
    }
    else
    {                                            // Yes=> Return error code
       uchCommErrorCode = WR_BUFFER_FULL_ERROR;
        return (FALSE);
    }
}

/* ============================================================================
**  Module name: vIntUART0Receive
**
**  Description:
**              This modul is the interrupt service routine for a receive
**              operation.
**
**  Operation:
**              If read buffer is not full, read byte and save it to
**              read buffer.
**              Increment read buffer write index.
**              Set error code if necessary.
**
** ============================================================================
*/

static interrupt [INTSR0_vect] void vIntUART0Receive(void)
{
   if(!bRdBufferFull)                            // Check if read buffer is full
    {                                            // No => Save received byte
       uchRdBuffer[uchWrIdxRdBuffer]=RXB0;
        vIncWrIdxRdBuffer();                     //       Increment read buffer write index
    }
    else                                         // Yes=> Set error code
      uchCommErrorCode=RD_BUFFER_FULL_ERROR;
```

```
}

/* ============================================================================
** Module name: vIntUART0Transmit
**
** Description:
**              This modul is the interrupt service routine for a transmit
**               operation.
**
** Operation:
**               Increment write buffer read index.
**              If write buffer is empty, disable transmission done interrupt,
**               else transmit byte from write buffer.
**
** ============================================================================
*/

static interrupt [INTST0_vect] void vIntUART0Transmit(void)
{
   vIncRdIdxWrBuffer();                          // Increment write buffer read index
   if(bWrBufferEmpty)                            // Check if buffer is empty
        bSTMK0 = 1;                              // Yes=>  Disable transmission done
    else                                         //        interrupt
    {                                            // No =>  Send byte
      TXS0=uchWrBuffer[uchRdIdxWrBuffer];
    }
}

/* ============================================================================
** Module name: vUART0ReceptionError
**
** Description:
**             This modul is the interrupt service routine for a reception
**             error.
**
** Operation:
**             Read the reception error and set the data reception error
**             flag.
**
** ============================================================================
*/

static interrupt [INTSER0_vect] void vIntUART0ReceptionError(void)
{
    unsigned char uchDummy;

   uchCommErrorCode=ASIS0;                       // Read reception error
    uchDummy=RXB0;                               // Save received byte
                                                 // Set error code
  if(uchCommErrorCode==1) {uchCommErrorCode = RCV_OVERRUN_ERROR;}
  if(uchCommErrorCode==2) {uchCommErrorCode = RCV_FRAMING_ERROR;}
  if(uchCommErrorCode==3) {uchCommErrorCode = RCV_PARITY_ERROR;}
   else {uchCommErrorCode = RCV_FATAL_ERROR;}
}

/* ============================================================================
** main function
** ============================================================================
*/
void main (void)
{
    unsigned char byte;
    unsigned int count;

    _DI();                                       // Disable all Interrupts
    vHardwareInit();                             // Peripheral settings
    vUART0Init();                                // Initialization UART0
    vRbInit();                                   // Initialization ring buffer
    _EI();                                       // Enable interrupts

   while(bRdBufferEmpty)
        ;

    count=0;
```

```
    while(1)
     {
             while(bGetData(&byte))
              {
                  b=bSendData(&byte);
              }
     }

    while(1)
         ;
}
```

Chapter 8 Small Temperature Controller

1 Introduction

The intention of this application note is to demonstrate the use of the 10-bit A/D converter and the 8-bit timer/ event counter in a temperature controller.

2 Example Program

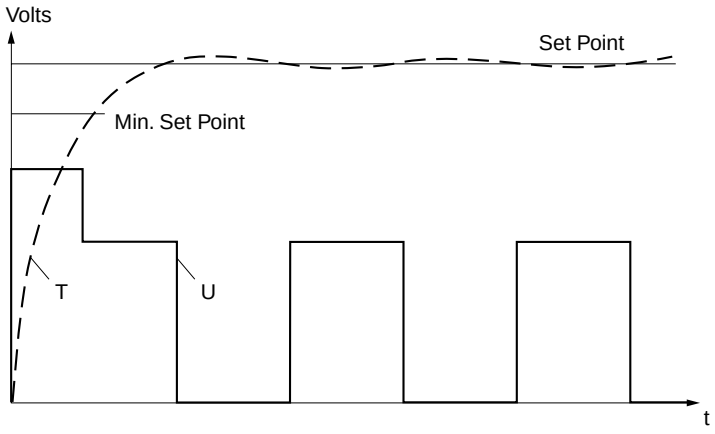
The actual temperature value is measured by a thermistor in a voltage divider. The output voltage of the divider is measured with the 10-bit A/D converter. The software of the temperature controller compares the actual value with the set point and calculates an output voltage to control a heating element. The control voltage is output as PWM signal, by using the 8-bit timer/event counter.

The method of controlling the temperature is with a three-step controller.
This method uses a constant voltage until the temperature reaches a fixed temperature lower than the temperature set point. The voltage is then changed to a lower value until temperature reaches the set point. Above the set point the voltage is turned off.

3 Controlling Strategy

First, output the control voltage at a high rate until just under the temperature set point. The control voltage is then shut down and a lower voltage is applied until the temperature reaches the set point. The voltage is then shut off, allowing the temperature to decay just under the set point. A low voltage is then applied again to bring the temperature up to the set point.

Figure 8-1: Controlling Strategy



4 Algorithm

The controller starts by measuring the actual voltage on the thermistor to determine the initial control voltage rate (high, low or off). After one PWM cycle the actual voltage is measured again.

4.1 Algorithm Parameters

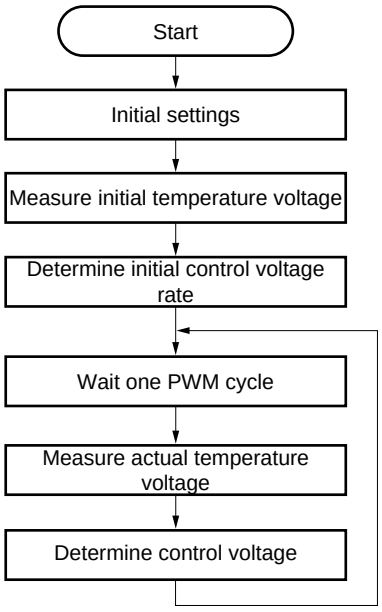
These parameters are #defines in the code, meaning the code must be recompiled to change the parameters.

Table 8-1 Controller Parameters

Parameter	Unit	Range	Resolution	Format	Description
SETPOINT	Volt	0...5	0.0049	Fixed Point	Set Point temperature
MINSETPOINT	Volt	0...5	0.0049	Fixed Point	Set Point lower control voltage
OUTPUT_VALUE_1	Volt	0...5	0.0196	Fixed Point	Maximum control voltage
OUTPUT_VALUE_2	Volt	0...5	0.0196	Fixed Point	Lower control voltage
OUTPUT_VALUE_3	Volt	0...5	0.0196	Fixed Point	Minimum control voltage

Controller flow chart:

Figure 8-2: Flow Chart



5 A/D Converter

The A/D converter consists of eight channels (ANI0...ANI07) with 10-Bit resolution. The conversion is based on the successive approximation method and the conversion result is held in the 16-Bit A/D conversion result register (ADCR0).

5.1 A/D Converter Configuration

5.1.1 Analog Input Channel Specification Register (ADS0)

This register specifies the analog voltage input port for A/D conversion.

For this application the analog input channel ANI0 is selected.

=> ADS0 = 0x00

5.1.2 A/D Converter Mode Register (ADM0)

This register sets the conversion time, for analog input to be A/D converted, conversion start/stop, and external trigger.

For this application the software start, 17.1 µs conversion time, and no external trigger are selected.

=> ADM0 = 0x00

5.2 A/D Conversion Operation

When bit 6 (TRG0) and bit 7 (ADCS0) of the A/D converter mode register (ADM0) are set to 0 and 1, respectively, A/D conversion of the voltage applied to the analog input pin specified by the analog input channel specification register (ADS0) starts.

Upon the end of the A/D conversion, the conversion result is stored in the A/D conversion result register ADCR0), and the interrupt request signal (INTAD0) is generated.

The first A/D conversion value just after A/D conversion operation start may not fall within the rating.

In this application the A/D conversion is started, the first A/D conversion result is thrown away (in accordance to the cautions in the user’s manual), and the second A/D conversion result is stored in a buffer register.

6 8-Bit Timer/Event Counter (TM50, TM51)

The 8-bit timer/event counter (TM50, TM51) provides the following two modes:

– Using 8-bit timer/event counter alone (individual mode)
– Using the cascade connection (16-bit resolution: cascade connection mode, combined with timerTM51)

In this application only one 8-bit timer/event counter (TM50) in individual mode is used.

6.1 Mode Using 8-Bit Timer/Event Counter Alone (Individual Mode)

In this application only the PWM output function is used, so only the PWM output function is described.

6.2 8-Bit Timer/Event Counter Configuration

6.2.1 Timer Register (TM50)

TM50 is an 8-bit read-only register counting the clock pulses.

6.2.2 8-Bit Compare Register (CR50)

When CR50 is used as a compare register, the value set in CR50 is constantly compared with the 8-bit counter (TM50) count value, and an interrupt request (INTTM50) is generated if they match. It is possible to rewrite the value of CR50 within 00H to FFH during count operation.

For this application the value of CR50 is initialized to 0x00.

6.2.3 Timer Clock Select Register (TCL50)

This register sets count clocks of 8-bit timer/event counter (TM50) and the valid edge of TI50 input.

For this application the count clock is set to 2.097 MHz.

=> TMC = 0x03

Note: Stop the timer operation before writing to TCL50.

6.2.4 8-Bit Timer Mode Control Register (TMC50)

TMC50 is a register, which sets up the timer mode.

For this application the PWM (free-running) mode, and reset of timer output F/F is selected. The timer output must be enabled.

=> TMC50 = 0x45

6.2.5 Port Mode Register 7 (PM7)

This register sets port 7 for input/output.

P72 is used for TM50 output.

=> PM7 = 0xC0

6.3 8-Bit PWM Output Operation

8-bit timer event counter operates as PWM output when bit 6 (TMC506) of 8-bit timer mode control register (TMC50) is set to 1. The duty rate pulse determined by the value set to 8-bit compare register (CR50). Set the active level width of PWM pulse to CR50, and the active level can be selected with bit 1 of TMC50 (TMC501). Count clock can be selected with bit 0 to 2 (TCL500 to TCL502) of timer clock select register (TCL50). Enable/ disable for PWM output can be selected with bit 0 of TMC50 (TOE50)

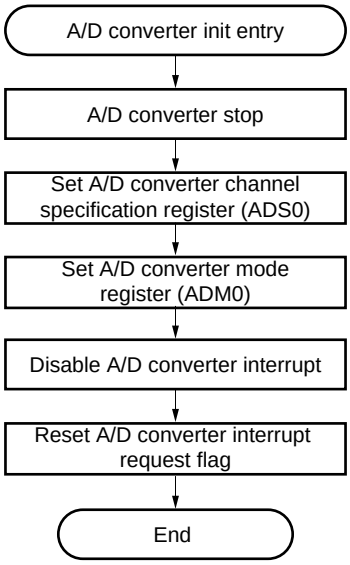
The PWM output outputs inactive level after count operation starts until overflow is generated. When overflow is generated, the active level is output until TM50 matches the count value of CR50. After the CR50 matches the count value, PWM outputs the inactive level again until overflow is generated.
When the count operation is stopped with TCE50 = 0, PWM output comes to inactive level.

In this application each timer overflow an interrupt (INTTM50) is generated, in the interrupt service routine a new value is written to CR50 and a flag for this new value is set.

7 Flow Charts

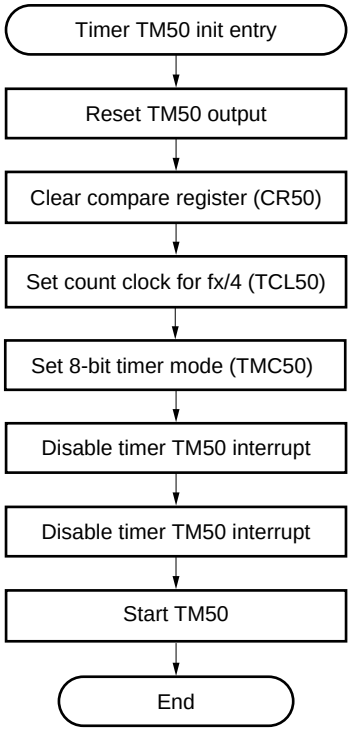
7.1 Configuration A/D Converter

Figure 8-3: A/D Converter



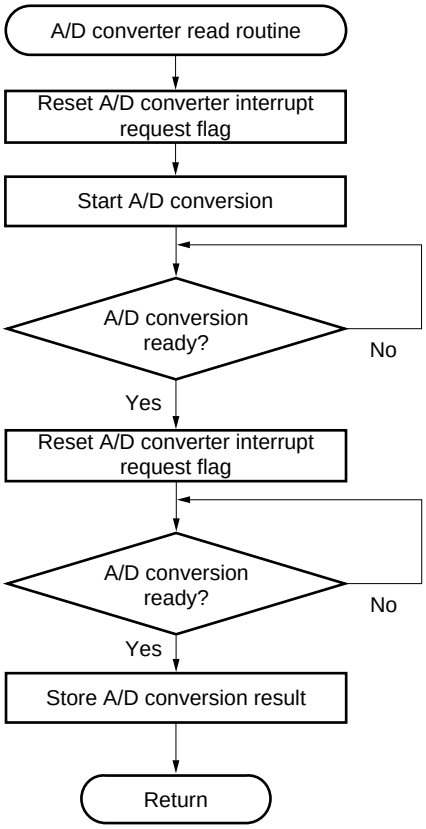
7.2 Configuration Timer TM50

Figure 8-4: Configuration Timer TM50



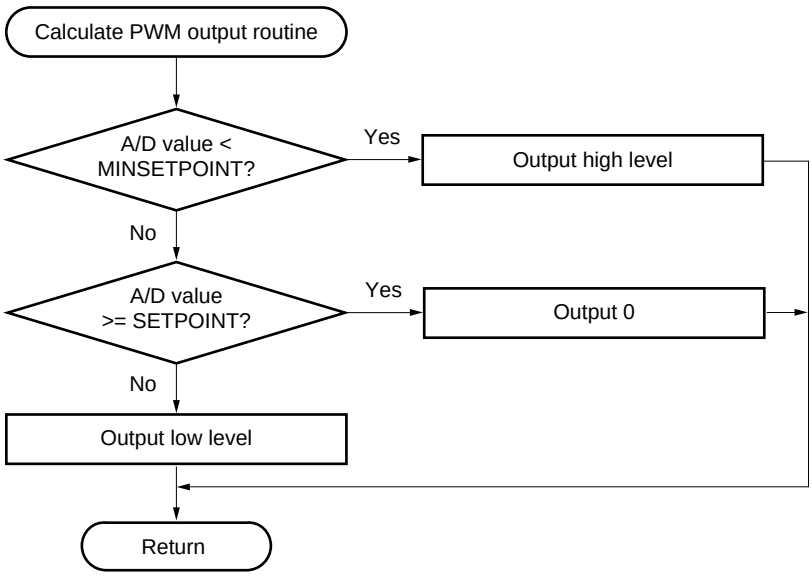
7.3 A/D Conversion

Figure 8-5: A/D Conversion



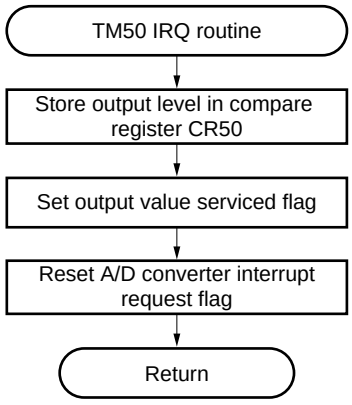
7.4 Calculate PWM Output (Control Voltage)

Figure 8-6: Control Voltage



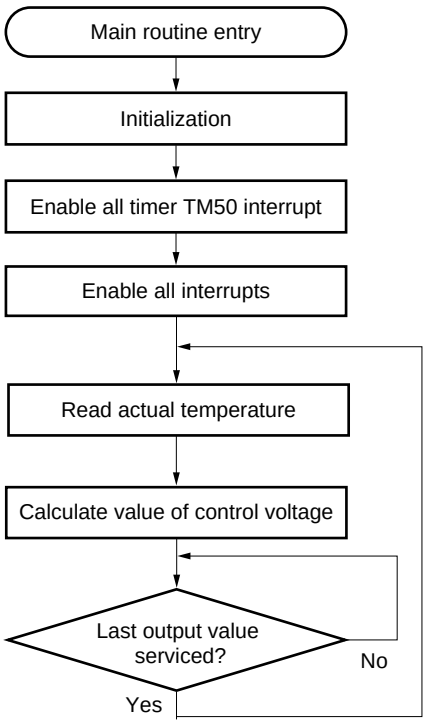
7.5 Set PWM Output

Figure 8-7: Set PWM Output



7.6 Main Routine

Figure 8-8: Main Routine



8 Sample Program

```
/*=============================================================================
** PROJECT        = IE-78001-R-A Tests
** MODULE         = AD_10Bit.c
** SHORT DESC.    = -
** DEVICE         = uPD78F0034AY
** VERSION        = 1.0
** DATE           = 26.03.1999
** LAST CHANGE    = -
** ============================================================================
** Description:  -
**
** ============================================================================
** Environment:   Device:           uPD78F0034AY
**                Assembler:        A78000               Version 1.13A
**                C-Compiler:       ICC78000             Version 3.13A
**                Linker:           XLINK                Version 4.50
**
** ============================================================================
** By:            NEC Electronics (Europe) GmbH
**                Oberrather Strasse 4
**                D-40472 Duesseldorf
**
**                DZ, NEC-EE, EAD-TPS
** ============================================================================
Changes:
** ============================================================================
*/

#pragma language = extended

#include <io78003X.h>
#include <in78000.h>

void HardwareInit(void);                        // Initialization Hardware
void vSoftwareInit(void);                       // Initialization measurement variables
void vADCInit(void);                            // Initialization A/D converter ch.
void vTM50Init(void);                           // Initialization timer 50
void vADReadData(void);                         // Read analog input channel 0
void vCalcPWMOutput(void);                      // Calculate PWM output value
                                                // Interrupt service routine timer 50
static interrupt [INTTM50_vect] void vIntTM50(void);


//————————————————————————————————————-
// General
//————————————————————————————————————-

#define TRUE  1
#define FALSE 0

#define SETPOINT 0x1FF                          // 2.50V
#define MINSETPOINT 0x1CC                       // 2.25V
#define OUTPUT_VALUE_1 0x0F7                    // 4.8V
#define OUTPUT_VALUE_2 0x014                    // 0.4V
#define OUTPUT_VALUE_3 0x00                     // 0V

//————————————————————————————————————-
// A/D converter
//————————————————————————————————————-

#define AD_START 0x80                           // Start a/d conversion
#define AD_STOP  0x7F                           // Stop a/d conversion

// Interrupt bits
//————————-

bit bADMK0 = MK1L.0;                            // A/D converter interrupt enable bit

bit bADIF0 = IF1L.0;                            // A/D converter interrupt request flag

bit bADPR0 = PR1L.0;                            // A/D converter interrupt priority flag
```

```
// A/D converter
//————————

#pragma memory = saddr

unsigned int uiADResult;                          // Result A/D conversion

#pragma memory = default

//————————————————————————————————————-
// Timer 50
//————————————————————————————————————-

#define TM50_START 0x80                           // Start timer 50
#define TM50_STOP 0x7F                            // Stop timer 50

// Ports
//———

bit bPWMPortMode   = PM7.2;                       // Port direction for signal
bit bPWMOutput     = P7.2;                        // Signal output

// Interrupt bits
//————————-

bit bTMMK50 = MK0H.6;                             // Timer 50 interrupt enable bit

bit bTMIF50 = IF0H.6;                             // Timer 50 interrupt request flag

bit bTMPR50 = PR0H.6;                             // Timer 50 interrupt priority flag

// Timer 50
//————-

#pragma memory = saddr

unsigned char uchCalcPWMOutput;                   // Calculated PWM value

#pragma memory = default

bit bNewPWMOutputValue;                           // Flag new output value serviced

/* ============================================================================
**  Module name: vHardwareInit
**
**  Description:
**                 This module is to initialize some peripheral hardware.
**
**  Operation:
**                 Sets the clock generator, the port modes and output latches
**                 and initializes the interrupts.
** ============================================================================
*/

void vHardwareInit(void)                          // Hardware inizialization
{

// clock generator setting
//————————————
    CKS  = 0x08;                                  // setup PCL frequency = fxt = 32KHz
    PCC  = 0x00;                                  // Use high speed mode (250ns @ 8MHz)
    OSTS = 0x01;                                  // 3.28ms wait after STOP release by interrupt
    IMS  = 0xC8;                                  // !!!! Select 1024 Byte RAM and 32k Byte ROM

// port setting
//————————-
    P0=0x00;                                      // Set output latch to 0
    P2=0x00;                                      // Set output latch to 0
    P3=0x00;                                      // Set output latch to 0
    P4=0x00;                                      // Set output latch to 0
    P5=0x00;                                      // Set output latch to 0
    P6=0x00;                                      // Set output latch to 0
```

```
    P7=0x00;                                    // Set output latch to 0

    PM0  = 0xF0;                                // Port 0 = output (4-bits)
    PM2  = 0xC0;                                // Port 2 = output (6-bits)
    PM3  = 0x80;                                // Port 3 = output (7-bits)
    PM4  = 0x00;                                // Port 4 = output
    PM5  = 0x00;                                // Port 5 = output
    PM6  = 0x0F;                                // Port 6 = output (4-bits)
    PM7  = 0xC0;                                // Port 7 = output (6 bits)

// interrupt setting
//————————

    IF0L = 0x00;
    IF0H = 0x00;
    IF1L = 0x00;
    MK0L = 0xFF;
    MK0H = 0xFF;
    MK1L = 0xFF;

    PR0L = 0xFF;
    PR0H = 0xFF;
    PR1L = 0xFF;
}

/* ============================================================================
**  Module name: vSoftwareInit
**
**  Description:
**             This module is to initialize the variables used by measurement.
**
**  Operation:
**                 -
** ============================================================================
*/

void vSoftwareInit(void)
{
    uiADResult = 0;                             // Reset A/D conversion result
    bNewPWMOutputValue = 0;                     // Reset flag output value serviced
    uchCalcPWMOutput = 0;                       // Reset calculated PWM value
}

/* ============================================================================
**  Module name: vADCInit
**
**  Description:
**              This module is to initialize the a/d converter control and mode
**                registers.
**
**  Operation:
**
** ============================================================================
*/

void vADCInit(void)
{
    ADM0 &= AD_STOP;                            // A/D conversion stop
    ADS0 = 0x00;                                // 00000000 = A/D conv. channel 0
                                                // ——000 - ANI0
                                                // ——001 - ANI1
                                                // ——010 - ANI2
                                                // ——011 - ANI3
                                                // ——100 - ANI4
                                                // ——101 - ANI5
                                                // ——110 - ANI6
                                                // ——111 - ANI7

    ADM0 = 0x00;                                // 00000000 = 0x00
                                                // |||||00— no edge detection
                                                // |||||01— falling edge detection
                                                // |||||10— rising edge detection
                                                // |||||11— both edges detection
                                                // ||000—— conversion time 17.1µs@8.386MHz
                                                // ||001—— conversion time 14.3µs@8.386MHz
```

```
                                        // |0———— software start
                                        // |1———— hardware start
                                        // 0———— stop conversion
                                        // 1———— enable conversion

    bADMK0 = 1;                         // Disable A/D converter interrupt
    bADIF0 = 0;                         // Reset A/D converter interrupt request flag
}

/* ============================================================================
**  Module name: vTM50Init
**
**  Description:
**               This module is to initialize the timer 50 control and mode
**                registers.
**
**  Operation:
**
** ============================================================================
*/

void vTM50Init(void)
{
    TMC50 &= TM50_STOP;                 // Timer 50 stop

    bPWMOutput   = 0;                   // Reset output latch
    bPWMPortMode = 0;                   // Set port mode for output

    CR50  = 0;

    TCL50  = 0x03;                      // 00000011 = 0x03
                                        // ——000 - TI50 falling edge
                                        // ——001 - TI50 rising edge
                                        // ——010 - fx
                                        // ——011 - fx/2^2
                                        // ——100 - fx/2^4
                                        // ——101 - fx/2^6
                                        // ——110 - fx/2^8
                                        // ——111 - fx/2^10

    TMC50  = 0x45;                      // 01000101 = 0x45
                                        // ||-||||0 - output disabled
                                        // ||-||||1 - output enabled
                                        // ||-|||0— active level high
                                        // ||-|||1— active level low
                                        // ||-|00—— no change
                                        // ||-|01—— timer output reset
                                        // ||-|10—-- timer output set
                                        // ||-0——— single mode
                                        // ||-1——— cascade mode
                                        // |0————— clear and start mode by matching
                                        // between CR50 and TM50
                                        // |1————— PWM (free-running mode)
                                        // 0————— after cleaning to 0, count
                                        //        operation disabled
                                        // 1————— count operation start

    bTMMK50 = 1;                        // Disable timer 50 interrupt
    bTMIF50 = 0;                        // Reset timer 50 interrupt request flag

   TMC50 |= TM50_START;                 // Timer 50 start
}

/* ============================================================================
**  Module name: vADReadData
**
**  Description:
**               This module is to read the input value of the a/d converter
**                channel 0.
**
**  Operation:
**
** ============================================================================
```

```
*/

void vADReadData(void)
{
    bADIF0 = 0;
    ADM0 |= AD_START;

    while(!bADIF0)                          // Just to throw away first result
     {
         _NOP();
     }

    bADIF0 = 0;

    while(!bADIF0)
     {
         _NOP();
     }

    uiADResult = ADCR0;
    uiADResult = uiADResult>>6;

}

/* ============================================================================
**  Module name: vCalcPWMOutput
**
**  Description:
**               This module is to calculate the new PWM output value.
**
**  Operation:
**               The value is stored in CR50 and output after the next
**                timer 50 interrupt.
**
** ============================================================================
*/

void vCalcPWMOutput(void)
{
    if(uiADResult < MINSETPOINT)
        uchCalcPWMOutput = OUTPUT_VALUE_1;
     else
         if(uiADResult >= SETPOINT)
             uchCalcPWMOutput = OUTPUT_VALUE_3;
          else
             uchCalcPWMOutput = OUTPUT_VALUE_2;

//    uchCalcPWMOutput = (0x3FF - uiADResult)/4;
}

/* ============================================================================
**  Module name: vIntTM50
**
**  Description:
**               This module is the interrupt service routine for the timer 50
**                interrupt operation.
**
**  Operation:
**               Interrupt source is the match between CR50 and TM50.
**
** ============================================================================
*/

static interrupt [INTTM50_vect] void vIntTM50(void)
{
    CR50 = uchCalcPWMOutput;                // Output new value
    bNewPWMOutputValue = 1;                 // New output value serviced
     bTMIF50 = 0;                           // Reset timer 50 interrupt request flag
}

/* ============================================================================
**  main function
** ============================================================================
*/
```

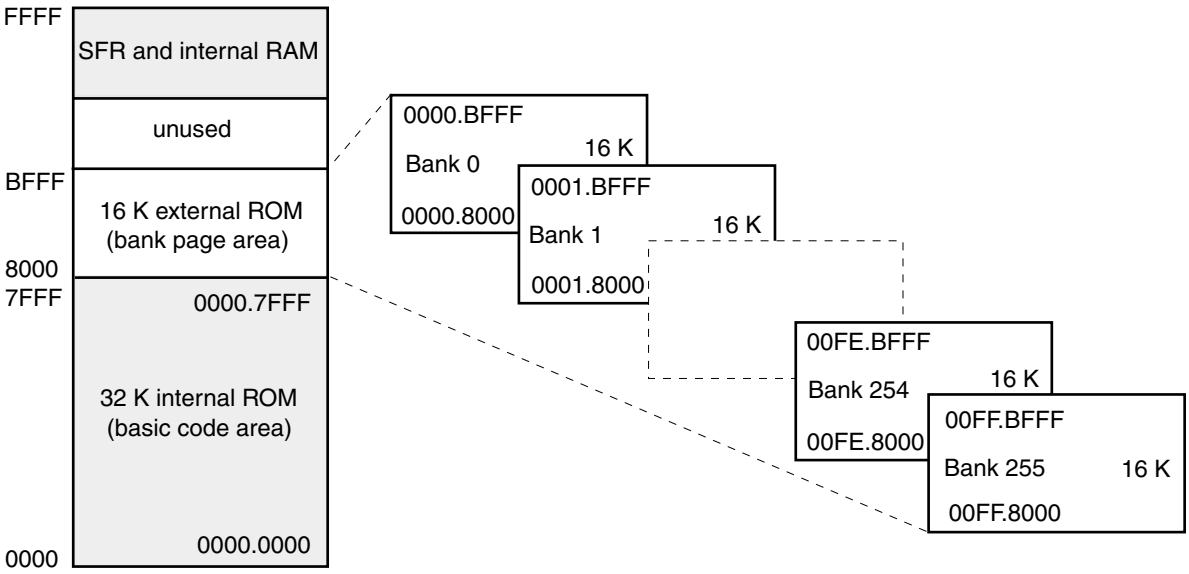
```
void main( void )
{
    _DI();                                  // Disable all interrupts
    vHardwareInit();                        // Peripheral settings
    vSoftwareInit();                        // Initialization measurement variables
    vADCInit();                             // Initialization a/d converter ch. 0
    vTM50Init();                            // Initialization 8-Bit timer 50
    bTMMK50 = 0;                            // Enable timer 50 interrupt
    _EI();                                  // Enable all interrupts

    while(TRUE)
    {
        vADReadData();                      // Read A/D converter
        vCalcPWMOutput();                   // Calculate output value
        bNewPWMOutputValue = 0;             // New output value not serviced
       while(!bNewPWMOutputValue)
             ;
    }
}
```

Chapter 9 Example for 78K0 Banked Code Applications

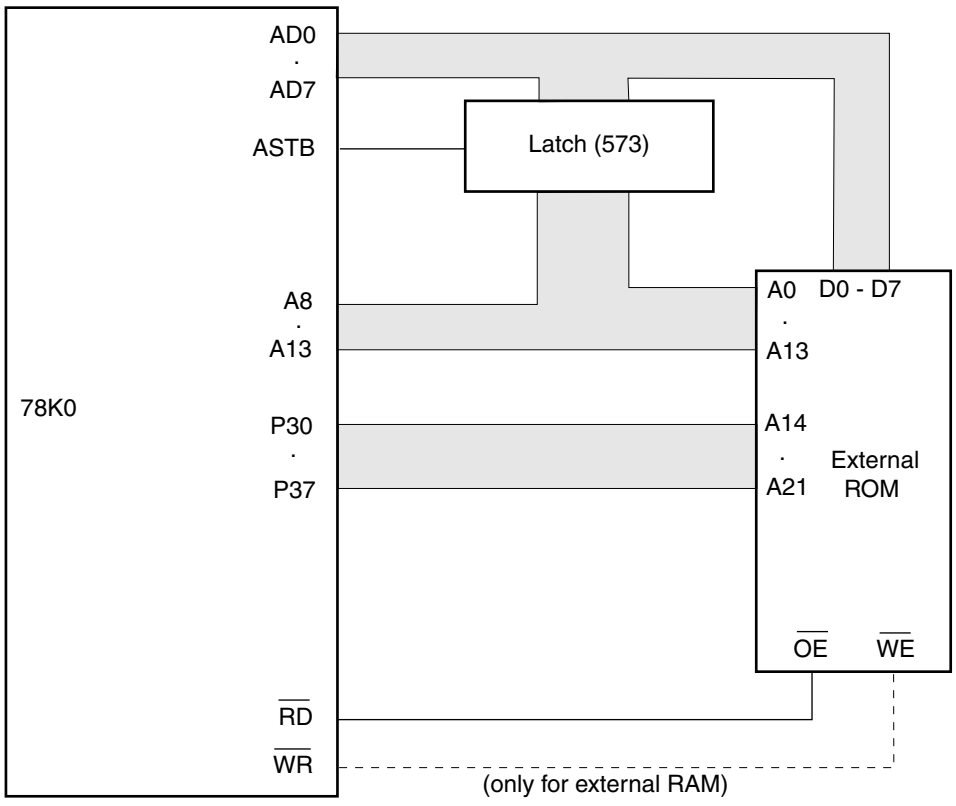
If an application needs to address more than 60k byte ROM the 78k0 series reaches the limit of direct addressable memory. To overcome this limit the complete software development system (compiler, linker and In-Circuit emulator) supports a code expansion technique called banked memory.

Figure 9-1: Banked Memory



Similar to the PC EMS technology a bank page area is defined. An external memory is connected to this area selected in pages (banks). Additional port lines do the selection out of the banks.

Figure 9-2: External Memory



Maximum 256 banks can be addressed. If a compromise is taken in between simple decoding hardware and max. banked memory a god size of bank page is 16k byte. This results to a ROM size of up to 4M byte banked memory.

Main task of the compiler and linker is to place different parts of code into the same physical address range. Each time a banked function has to be called, the correct bank has to be selected.
To make it easy it is realized by a special function call procedure. The user to the dedicated hardware can adopt this procedure (see L07.S26 file).

The below application shows an example how to realize banked code.

1. Memory Map and Organization

The internal memory area from 0000 to 7FFF should be used as basic code area (where the bank switching code is located in). The bank page area (an external EPROM) is located in the address range 8000 to BFFF of the CPU.
The program code layout is organized in following blocks:

- The memory area from 0000.0000 to 0000.7FFF is the basic code area
- The memory area from 0000.8000 to 0000.BFFF is realized as bank 0
- The memory area from 0001.8000 to 0001.BFFF is realized as bank 1.
- The external EPROM is connected to D0-D7 and A0-A13 (P5.6 and P5.7 are free to use as normal port lines)
- Because only two banks are addressed just port line P3.0 is used. P3.0 is connected to the A14 address line of the external EPROM (bank selection).

Please note, that the $\overline{RD}$ signal (here connected to the $\overline{OE}$ pin of the EPROM) and the $\overline{WR}$ signal (open) are only active (low) during 78k0 external memory read or write access. Thus these signals are used to decode the external IC's.

Table 9 -1: Memory Map

Program code address	Physical CPU address	Port P3.0 connected to A14 of the EPROM (bank select)	Physical EPROM address	Comments
0000.0000 - 0000.7FFF	0000 - 7FFF	Don t care	Not valid	Internal access of basic code area
0000.8000 - 0000.BFFF	8000 - BFFF	0	0000 - 3FFF	External access of bank0
0001.8000 - 0001.BFFF	8000 - BFFF	1	4000 - 7FFF	External access of bank1

2 Definition of the External Code Segments in XCL-File

In the linker control file the linker is informed about format, size and number of banks with following command:

-b(CODE) BANK0,BANK1=8000,4000,10000,2

Table 9 -2: Meaning of the Commands

Meaning of the command	
CODE	Memory type of all bank segments
BANK0,BANK1	List of the defined banked segments BANK0 is the name of the code segment of bank0; BANK1 is the name of the code segment of bank1. Separator ',' if each segment has to be placed in a separate bank Separator ':' if several segments may be placed in same bank
8000	HEX Start address of the bank page area
4000	HEX Size of a bank page area (8000 to BFFF = 4000)
10000	HEX Incremental factor of program bank address. It is the number multiplied with the number of the bank and added to bank page area start address to get the address of the bank. For example: Bank 0 = 0*1.0000 + 0.8000 = 0.8000 Bank 1 = 1*1.0000 + 0.8000 = 1.8000 Bank 2 = 2*1.0000 + 0.8000 = 2.8000
2	Decimal number of banks available in the logical address area.

3 L07.S26 Assembler Control File

This module contains the bank switching functions and has to be adopted to the hardware and also added to the project. It is each time executed if a banked function is called (see *5.Memory model selection…* for details). The address is given in the 32 bit DE:HL register pairs.

In the example the bank selection is just depending on the high address (DE-register). Here the possible values are 0000 and 0001 only, because just two banks are defined. So just the least significant bit has to be moved to the port P3.0 to select the entire bank. The HL register contains the offset inside the selected bank. So here it just represents the physical CPU address.

DE:HL	Bank	Port P3.0
DE:HL = 000**0**:8000 to 000**0**:BFFF	Bank0	P3.0=0
DE:HL = 000**1**:8000 to 000**1**:BFFF	Bank1	P3.0=1

A sample of the L07.S26 file is distributed with the IAR compiler package (see subdirectory icc78000). This file can easily be adapted to your application.
The L07.S26 adapted to this example is listed below:

```
;        TITL     'NEC 78K/0 - Banked call'
;------------------------------------------------------------------------------
;                                - L07.S26 -
;
;        This module contain support for banked function CALL and
;        RETURN for 78K/0.
;
;        Must be tailored for actual bank-switching hardware.
;
;        In the sample system P30 is used for two banks.
;        In this case the port is readable but this is not
;        needed if the current bank-number is also stored in
;        RAM.
;
;        Designed for 780034 test board with external RAM.
;        Dipl.-Ing. Ludger Lenzen, TPS, NEC Electronics (Europe) GmbH
;------------------------------------------------------------------------------
;        Archived: $Revision: 1.6 $
;        (c) Copyright IAR Systems 1997, NEC Electronics (Europe) GmbH 1999
;------------------------------------------------------------------------------
;        Entries:         ?FAR_CALL_L07
;                         ?FAR_FAST_CALL_L07
;------------------------------------------------------------------------------
         MODULE  ?EXT_FAR_CALL_L07
         RSEG    RCODE
;------------------------------------------------------------------------------
;
;        Function:        Call of a banked function
;
;------------------------------------------------------------------------------
;        Input:  AX/BC    Possible parameter value(s)
;                 D        (free)
;                 E        Bank no
;                 HL       Function address
;
;        Call:   CALL     ?FAR_CALL_L07
;
;        Output: -
;------------------------------------------------------------------------------

 sfr     P3       = 0xFF03;        ; SFR used in example, bit and byte access

         PUBLIC  ?FAR_CALL_L07

?FAR_CALL_L07:

;        Save old bank

         xch     a, d
         mov     a,#0
         push    psw              ; Save carry
         mov1    cy,P3.0          ; Get old bank number (could be in RAM)
         mov1    a.0,cy           ;
         pop     psw              ; Restore carry
         xch     a, d
         push    de               ; Always stack old bank number
```

```
;       Switch to new bank

        xch     a, e            ; Get new bank number
        push    psw             ; Save carry
        mov1    cy,a.0          ; Set new bank number (could be in RAM)
        mov1    P3.0,cy         ;
        pop     psw             ; Restore carry
        xch     a, e


;       Execute banked function

        movw    de, #far_return
        push    de              ; Return address
        push    hl
        ret                     ; Execute banked function

;-------------------------------------------------------------------------------
;
;                       Return from banked function
;
;-------------------------------------------------------------------------------
;       Input:  AX/BC   Possible return value(s)
;               DE/HL   (free)
;
;
;
;       Output: AX/BC   Possible return value(s)
;               DE/HL   (free)
;-------------------------------------------------------------------------------
;       Switch back to old bank

far_return:
        pop     hl              ; Get old bank number
        xch     a, h
        push    psw             ; Save carry
        mov1    cy,a.0          ; Set new bank number (could be in RAM)
        mov1    P3.0,cy         ;
        pop     psw             ; Restore carry
        xch     a, h

;       Return to calling function

        ret                     ; Do normal return

        ENDMOD

        MODULE  ?EXT_FAR_FAST_CALL_L07
        RSEG    RCODE
;-------------------------------------------------------------------------------
;
;       Function:       Call of a banked function, FAST
;
;-------------------------------------------------------------------------------
;       Input:  AX/BC   Possible parameter value(s)
;               D       (free)
;               E       Bank no
;               HL      Function address
;
;       Call:   CALL    ?FAR_FAST_CALL_L07
;       Output: -
;-------------------------------------------------------------------------------
```

```
        PUBLIC  ?FAR_FAST_CALL_L07

?FAR_FAST_CALL_L07:
        push    de              ; Stack old bank number = new

;       Execute banked function

        movw    de, #far_fast_return
        push    de              ; Return address
        push    hl
        ret                     ; Execute banked function

;-------------------------------------------------------------------------------
;
;                          Return from banked function, FAST
;
;-------------------------------------------------------------------------------
;       Input:  AX/BC   Possible return value(s)
;               DE/HL   (free)
;
;
;
;       Output: AX/BC   Possible return value(s)
;               DE/HL   (free)
;-------------------------------------------------------------------------------

far_fast_return:
        pop     hl              ; Get old bank number

;       Return to calling function

        ret                     ; Do normal return

        END
```

4 Required Register Settings for the Device

The device, used for a banked application, has to activate the external memory access. For the µPD78F0034A the memory expansion mode register (MEM), the memory mode register (MM) and the internal memory size register (IMS) have to be adjusted.

```
IMS = 0xC8; // for 780034A needed, select internal 32k ROM and 1k RAM
PCC = 0x00; // fastest speed
P3  = 0;    // initialise P3 latches
PM3 = 0xFE; // port P3.0 is output
MEM = 0x05; // MEM set to 16k byte expansion mode (AD0-AD7, A8-A13)
MM  = 0x10; // use one wait state (needed only for slow EPROM types)
```

5 Memory Model Selection and Related Source Design

There are two memory models possible to realize a banked code application:

Banked memory model

- If you decide to use the banked memory model, each source file has to be compiled for the banked memory model (compiler switch **-mb**). Standard memory model modules and banked memory model modules can't be mixed.
- You have to use the banked memory model library cl7801b.
- The main function and interrupt functions are fixed as NON-BANKED because they have to be located in the basic code area.
- Each function is defined as banked function by default. If a function should be located into the basic code area it has to be declared NON-BANKED using the attribute "non_banked"
  void non_banked function_name(void);
- Each function (except of main and interrupts) is executed via the L07.S26 bank switching independent, if it is a BANKED or NON-BANKED function.
- The compiler will generate a warning (mixed BANKED and NON-BANKED code) when it detects a NON-BANKED function in a module (i.e. the main function).

Standard memory model

- If you decide to use the standard memory model, each source file has to be compiled for the standard memory model (compiler switch **-ms**). Standard memory model modules and banked memory model modules can't be mixed.
- You have to use the standard memory model library cl7801s.
- Each function is defined as NON-BANKED by default. If a function should be defined as BANKED it has to be declared BANKED using the attribute "banked"
  void banked function_name(void);
  The main function and interrupt functions can't be defined as BANKED because the have to be located in the basic code area.
- Only BANKED functions are executed via the L07.S26 bank switching. All NON-BANKED functions are executed via normal function call.
- The compiler will generate no warning if BANKED and NON-BANKED functions are mixed in the same module.

The standard memory model gives the advantage of a speed-optimized execution of NON-BANKED functions but the disadvantage of more complex source code writing.
It is recommended to use banked memory model only if the number of BANKED functions is high against the number of NON-BANKED one and there is no need for a speed optimization.

5.1 Banked Memory Model Source Design

Below all source files and the linker control file (XCL file) are listed designed for the banked memory model project. So all functions located in the internal basic memory (on-chip) have to be declared as ‘non_banked’. All source parts related to a banked application are printed bold.
The main and the interrupt functions are always ‘non_banked’
After compilation the compiler messages warnings, because ‘banked’ and ‘non_banked’ code is mixed.

Note that the *#pragma codeseg(name)* directive can only be used once per source module.

Source module 1 (func1b.c):

```
#pragma codeseg(BANK0)

unsigned char func1(unsigned char para1)
{
  para1+=2;
  return(para1);
}
```

Source module 2 (func2b.c):

```
#pragma codeseg(BANK1)

unsigned char func2 (unsigned char para1)
{
  para1+=2;
  para1+=2;
  return(para1);
}
```

Source module 3 (main module mainb.c):

```
// Banked application using banked memory model

unsigned char c1,c2,c3;

extern  non_banked void hardware_init( void );
extern  banked      unsigned char func1(unsigned char);
extern  banked      unsigned char func2(unsigned char);
        non_banked unsigned char func3(unsigned char);

non_banked void main (void)
{
  // device depending initialise of hardware
  hardware_init();
  while(1)
  {
    c1=func1(1);
    c2=func2(2);
    c3=func3(3);

    c1=func1(4);
    c2=func2(5);
    c3=func3(6);
  }
}
non_banked unsigned char func3(unsigned char para1)
{
  para1+=3;
  para1+=3;
  para1+=3;
  return(para1);
}
```

Source module 4 (hwinitb.c):

```
#include <in78000.h>  // include 78k0 standad header
#include <io78003x.h> // include 78k0 device related header

non_banked void hardware_init( void );

non_banked void hardware_init( void )
{
  IMS = 0xC8;  // for IE-78001-R-A with 78002x needed
  PCC = 0x00;  // fastest speed
  P3  = 0;     // init port latches
  PM3 = 0xFE;  // port 3.0 is output
  MEM = 0x05;  // MEM set to 16k adr. mode (AD0-AD7, A8-A13), P56 & P57 ports
  MM  = 0x10;  // use one wait state
}
```

XLINK control file (rsdb.xcl):

```
//————————————————————————————————————————————————————————————————————————
//            - uPD780034.xcl -
//
//  NEC 78K0 microcontroller device uPD780034.
//
//  XLINK command file template for banked appl. on 780034 test board.
//
//————————————————————————————————————————————————————————————————————————

//————————————————————————————————————————————————————————————————————————
// Define CPU.
//————————————————————————————————————————————————————————————————————————
-c78000

//————————————————————————————————————————————————————————————————————————
// Allocate interrupt vector segment.
//————————————————————————————————————————————————————————————————————————
-Z(CODE)INTVEC=0000-003F

//————————————————————————————————————————————————————————————————————————
// Allocate CALLT segments.
//————————————————————————————————————————————————————————————————————————
-Z(CODE)FLIST,IFLIST=0040-007F

//————————————————————————————————————————————————————————————————————————
// Allocate CALLF segment.
//
// Note: This entry is necessary to avoid XLINK warnings.
//       If this segment is in use, adjustments to program code
//       allocation will be necessary!
//————————————————————————————————————————————————————————————————————————
-Z(CODE)FCODE=0800-0FFF

//————————————————————————————————————————————————————————————————————————
// Allocate program code segments.
//————————————————————————————————————————————————————————————————————————
-Z(CODE)RCODE,CODE,CONST,CSTR,CCSTR,CDATA0,CDATA1,CDATA2=0080-0FFF

//————————————————————————————————————————————————————————————————————————
//      BANKED           (-b option below)
//      Address 8000 is here supposed to be start of ROM area and
//      address 3FFF is here supposed to be end of ROM area and
//      bank length is 4000 and bank increment is 10000.
```

```
//
//       -b(CODE)BANK0:BANK1=8000,4000,10000,2
//              fills the code into separated segments (banks)
//       -b(CODE)BANK0,BANK1=8000,4000,10000,2
//              fills up the segments (banks)
//————————————————————————————————————————————————————————————————————————————
-b(CODE)BANK0:BANK1=8000,4000,10000,2

//————————————————————————————————————————————————————————————————————————————
// Allocate program data segments.
//————————————————————————————————————————————————————————————————————————————
-Z(DATA)CSTACK+80,NO_INIT,IDATA2,UDATA2,ECSTR,TEMP=FB00-FE1F

//————————————————————————————————————————————————————————————————————————————
// Allocate saddr data segments.
//————————————————————————————————————————————————————————————————————————————
-Z(DATA)WRKSEG,IDATA0,IDATA1,UDATA0,UDATA1=FE20-FEDF

//————————————————————————————————————————————————————————————————————————————
// Allocate bit segment.
//————————————————————————————————————————————————————————————————————————————
-Z(BIT)BITVARS=0000

//————————————————————————————————————————————————————————————————————————————
// Select the printf/scanf formatter.
//————————————————————————————————————————————————————————————————————————————
-e_small_write=_formatted_write
-e_medium_read=_formatted_read

//————————————————————————————————————————————————————————————————————————————
// Select the ‘C’ library.
//————————————————————————————————————————————————————————————————————————————
-C cl7801b
```

5.2 Standard Memory Model Source Design

Below all source files and the linker control file (XCL file) are listed designed for the standard memory model project. So all functions located in the entire banks have to be declared as 'banked'.

Note that the *#pragma codeseg(name)* directive can only be used once per source module.

Source module 1 (func1s.c):

```
#pragma codeseg(BANK0)

banked unsigned char func1(unsigned char para1)
{
  para1+=2;
  return(para1);
}
```

Source module 2 (func2s.c):

```
#pragma codeseg(BANK1)

banked unsigned char func2 (unsigned char para1)
{
  para1+=2;
  para1+=2;
  return(para1);
}
```

Source module 3 (main module mains.c):

```
// Banked application using standard memory model

unsigned char c1,c2,c3;

extern   banked unsigned char func1(unsigned char);
extern   banked unsigned char func2(unsigned char);
                unsigned char func3(unsigned char);

extern void hardware_init( void );

void main (void)
{
  // device depending initialise of hardware
  hardware_init();
  while(1)
  {
    c1=func1(1);
    c2=func2(2);
    c3=func3(3);

    c1=func1(4);
    c2=func2(5);
    c3=func3(6);
  }
}

unsigned char func3(unsigned char para1)
{
  para1+=3;
  para1+=3;
  para1+=3;
  return(para1);
}
```

Source module 4 (hwinits.c):

```
#include <in78000.h>  // include 78k0 standad header
#include <io78003x.h> // include 78k0 device related header

void hardware_init( void );

void hardware_init( void )
{
  IMS = 0xC8;  // for IE-78001-R-A with 78002x needed
  PCC = 0x00;  // fastest speed
  P3  = 0;     // init port latches
  PM3 = 0xFE;  // port 3.0 is output
  MEM = 0x05;  // MEM set to 16k adr. mode (AD0-AD7, A8-A13), P56 & P57 ports
  MM  = 0x10;  // use one wait state
}
```

XLINK control file (rsds.xcl):

```
//————————————————————————————————————————————————————————————————————————————
//             - uPD780034.xcl -
//
//  NEC 78K0 microcontroller device uPD780034.
//
//  XLINK command file template for banked appl. on 780034 test board.
//
//————————————————————————————————————————————————————————————————————————————

//————————————————————————————————————————————————————————————————————————————
// Define CPU.
//————————————————————————————————————————————————————————————————————————————
-c78000

//————————————————————————————————————————————————————————————————————————————
// Allocate interrupt vector segment.
//————————————————————————————————————————————————————————————————————————————
-Z(CODE)INTVEC=0000-003F

//————————————————————————————————————————————————————————————————————————————
// Allocate CALLT segments.
//————————————————————————————————————————————————————————————————————————————
-Z(CODE)FLIST,IFLIST=0040-007F

//————————————————————————————————————————————————————————————————————————————
// Allocate CALLF segment.
//
// Note: This entry is necessary to avoid XLINK warnings.
//        If this segment is in use, adjustments to program code
//        allocation will be necessary!
//————————————————————————————————————————————————————————————————————————————
-Z(CODE)FCODE=0800-0FFF

//————————————————————————————————————————————————————————————————————————————
// Allocate program code segments.
//————————————————————————————————————————————————————————————————————————————
-Z(CODE)RCODE,CODE,CONST,CSTR,CCSTR,CDATA0,CDATA1,CDATA2=0080-0FFF

//————————————————————————————————————————————————————————————————————————————
//       BANKED           (-b option below)
//       Address 8000 is here supposed to be start of ROM area and
//       address 3FFF is here supposed to be end of ROM area and
//       bank length is 4000 and bank increment is 10000.
//
```

```
//       -b(CODE)BANK0:BANK1=8000,4000,10000,2
//              fills the code into separated segments (banks)
//       -b(CODE)BANK0,BANK1=8000,4000,10000,2
//              fills up the segments (banks)
//————————————————————————————————————————————————————————————————————————————
-b(CODE)BANK0:BANK1=8000,4000,10000,2

//————————————————————————————————————————————————————————————————————————————
// Allocate program data segments.
//————————————————————————————————————————————————————————————————————————————
-Z(DATA)CSTACK+80,NO_INIT,IDATA2,UDATA2,ECSTR,TEMP=FB00-FE1F

//————————————————————————————————————————————————————————————————————————————
// Allocate saddr data segments.
//————————————————————————————————————————————————————————————————————————————
-Z(DATA)WRKSEG,IDATA0,IDATA1,UDATA0,UDATA1=FE20-FEDF

//————————————————————————————————————————————————————————————————————————————
// Allocate bit segment.
//————————————————————————————————————————————————————————————————————————————
-Z(BIT)BITVARS=0000

//————————————————————————————————————————————————————————————————————————————
// Select the printf/scanf formatter.
//————————————————————————————————————————————————————————————————————————————
-e_small_write=_formatted_write
-e_medium_read=_formatted_read

//————————————————————————————————————————————————————————————————————————————
// Select the ‘C’ library.
//————————————————————————————————————————————————————————————————————————————
-C cl7801s
```

6 Testing

During the program development it is recommended to test the program using the 78k0 In-Circuit emulator. It is also recommended to emulate the external program memory device by an external static RAM. This prevents a reprogramming of an external memory in case of errors. The development tool environment supports all required functions including the RAM download.
For the final program test the real external program memory (i.e. an EPROM) may be tested also using the In-Circuit emulator.

6.1 Testing Using Development Tool and External RAM

If you test a banked program using the In-Circuit emulator by emulating the external program memory using an external static RAM the hardware has to be changed a bit. In opposite to an external program memory the In-Circuit emulator needs write access to that external static RAM to download the banked code.

Proceed as follow:

♦ Generate the lnk-File (xcoff78k format) of the project. The lnk-file includes the complete code including the banked code. How to generate the file depends on your tool environment:

Tool environment	Memory Model	File to use
Using the DOS batch file	Both memory models	Built.bat
Using the IAR workbench	Banked memory model	Bnklnkb.prj
	Standard memory model	Bnklnks.prj

♦ The generated filename of the lnk-file depends on the selected memory model:

Memory Model	Filename
Banked memory model	Bnklnkb.lnk
Standard memory model	Bnklnks.lnk

♦ Power on your target hardware and start the In-Circuit emulator and debugger software
♦ In the debugger configuration select the same settings for the memory banking you have chosen in the linker command file (XCL).
For the given example set:
port 0 = P30, bank range = 8000-BFFF, bank increment = 10000, MM=10
♦ Set the SFR's required for the external memory expansion (see note below).
For the given example set:
IMS = C8, MEM = 05, MM = 10, PM3 = FE, P3 = 00
♦ Download the lnk-file and execute/debug the program.

Note for the download:

You have to set the correct values needed for external memory mode before downloading the lnk-file. This is needed because the "after-reset" status of the device is the single chip mode. To make it easy it is also possible to download the lnk-file first, execute it once (initializes the registers) and download the lnk-file a second time **without** chip-reset.

Note for the IE-78001-R-A:

In case of using the IE-78001-R-A you need a hardware level F (serial no. EF…) or later to have a break board (IE-78001-R-BK) version V1.43 or newer. Previous versions of the break board do not support external code fetch.

6.2 Testing Using the Emulator and an External EPROM

To test your banked program using the emulator and an external EPROM proceed as follow:

- First generate the lnk-File (xcoff78k format) of the project. The lnk-file is used for the emulator download of the base code. The also including banked code is ignored by the emulator if the external EPROM contains the same data. How to generate the file depends on your tool environment:

Tool environment	Memory Model	File to use
Using the DOS batch file	Both memory models	Built.bat
Using the IAR workbench	Banked memory model	Bnklnkb.prj
	Standard memory model	Bnklnks.prj

- The generated filename of the lnk-file depends on the selected memory model:

Memory Model	Filename
Banked memory model	Bnklnkb.lnk
Standard memory model	Bnklnks.lnk

- Next generate a hex-file of the project either in the Motorola record format or in the extended Intel-hex format. The NEC2 or the standard Intel-hex format can't be used because the code exceeds the 64k-address range. If you generate these files using the prepared build files the Motorola record format is used because it is easier to understand. The build file depends on your tool environment:

Tool environment	Memory Model	File to use
Using the DOS batch file	Both memory models	Built.bat
Using the IAR workbench	Banked memory model	Bnkrecb.prj
	Standard memory model	Bnkrecs.prj

- The generated filename of the record file depends on the selected memory model:

Memory Model	Filename
Banked memory model	Bnkrecb.rec
Standard memory model	Bnkrecs.rec

- Use i.e. the PG1500 to program the external EPROM (the PG1500 supports both recommended formats)

Program the code address area 0.8000-0.BFFF to the EPROM address area 0000-3FFF (bank0)
Program the code address area 1.8000-1.BFFF to the EPROM address area 4000-7FFF (bank1)

- Plug-in the programmed EPROM in your target hardware
- Power on your target hardware and start the In-Circuit emulator and debugger software
- In the debugger configuration select the same settings for the memory banking you have chosen in the linker command file (XCL).
  For the given example set:
  port 0 = P30, bank range = 8000-BFFF, bank increment = 10000, MM=10
- Set the SFR's required for the external memory expansion (see note below).
  For the given example set:
  IMS = C8, MEM = 05, MM = 10, PM3 = FE, P3 = 00
- Download the lnk-file and execute/debug the program.

6.3 Testing Using the Flash Device and an External EPROM

To test your banked program using a real flash device and an external EPROM proceed as follow:

- Generate a hex-file of the project either in the Motorola record format or in the extended Intel-hex format. The NEC2 or the standard Intel-hex format can't be used because the code exceeds the 64k-address range. If you generate these files using the prepared build files the Motorola record format is used because it is easier to understand. The build file depends on your tool environment:

Tool environment	Memory Model	File to use
Using the DOS batch file	Both memory models	Built.bat
Using the IAR workbench	Banked memory model	Bnkrecb.prj
	Standard memory model	Bnkrecs.prj

- The generated filename of the record file depends on the selected memory model:

Memory Model	Filename
Banked memory model	Bnkrecb.rec
Standard memory model	Bnkrecs.rec

- Use i.e. the PG1500 to program the external EPROM (the PG1500 supports both recommended formats)

Program the code address area 0.8000-0.BFFF to the EPROM address area 0000-3FFF (bank0)
Program the code address area 1.8000-1.BFFF to the EPROM address area 4000-7FFF (bank1)

- The record file can be used also to program the 78F0034AY flash device using your flash-programmer (i.e. the FlashMaster).
- Plug-in the programmed devices in your target hardware
- Power on your target hardware and start the software test.

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

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

Thank you for your kind support.

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

Europe
NEC Electronics (Europe) GmbH
Market Communication Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6462-6829

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

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: +82-2-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: +886-2-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

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

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

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

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

[MEMO]