

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.

Application Note

78K Series Development Tools

Tutorial Guide

Target Devices

SP78K0 **Ver.2.00**

SP78K0S **Ver.2.00**

SP78K4 **Ver.2.00**

[MEMO]

Microsoft, Windows and Visual C++ are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other brand names or product names are registered trademarks or trademarks of their respective proprietors.

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

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

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

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

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

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

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

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

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

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

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

(Note)

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

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

M8E 02.11-1

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC Electronics product in your application, please contact the NEC Electronics 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.

[GLOBAL SUPPORT]

<http://www.necel.com/en/support/support.html>

NEC Electronics America, Inc. (U.S.)

Santa Clara, California
Tel: 408-588-6000
800-366-9782

NEC Electronics (Europe) GmbH

Duesseldorf, Germany
Tel: 0211-65030

NEC Electronics Hong Kong Ltd.

Hong Kong
Tel: 2886-9318

- **Sucursal en España**

Madrid, Spain
Tel: 091-504 27 87

- **Succursale Française**

Vélizy-Villacoublay, France
Tel: 01-30-67 58 00

- **Filiale Italiana**

Milano, Italy
Tel: 02-66 75 41

- **Branch The Netherlands**

Eindhoven, The Netherlands
Tel: 040-244 58 45

- **Tyskland Filial**

Taeby, Sweden
Tel: 08-63 80 820

- **United Kingdom Branch**

Milton Keynes, UK
Tel: 01908-691-133

NEC Electronics Hong Kong Ltd.

Seoul Branch
Seoul, Korea
Tel: 02-558-3737

NEC Electronics Shanghai Ltd.

Shanghai, P.R. China
Tel: 021-5888-5400

NEC Electronics Taiwan Ltd.

Taipei, Taiwan
Tel: 02-2719-2377

NEC Electronics Singapore Pte. Ltd.

Novena Square, Singapore
Tel: 6253-8311

J04.1

Introduction

— To first-time users of NEC Electronics development environments —

Welcome to the world of development environments from NEC Electronics. This tutorial introduces you to the operation of the SP78K Series software package using simple sample programs.

— To current users of NEC Electronics development environments —

In this tutorial, you will find sample programs that use a simulator, such as a virtual-screen output program and a slot-machine program. Use these programs to confirm the operation of your development environment.

Target Readers	This tutorial is intended for first-time users of 78K Series development tools. The reader should have a general knowledge of microcomputers, the C programming language, and assembly language programming, as well as a basic knowledge of Microsoft™ Windows™.
Purpose	The purpose of this tutorial is to assist the reader in understanding the basic operation of the 78K Series development tool. To gain a deeper understanding of the development tool's operation, users are encouraged to actually operate the development tool while following the tutorial examples. This document uses the 78K0 in all example explanations. Details that are unique to the 78K0S or the 78K4 are explained separately.
Organization	This tutorial consists of the following chapters:

[Chapter 1 Getting Ready](#)

This chapter contains an overview of the 78K Series development tools used in this tutorial and instructions on how to install the sample programs.

[Chapter 2 Trying Out PM plus and Simulator](#)

This chapter describes the basic operation of PM plus and system simulator using a sample program. The 78K0 and 78K4 are the target processors. The user manual documents associated with this chapter are Nos. 5, 7 and 8.

[Chapter 3 System Simulator Basics](#)

This chapter covers basic debugging with the system simulator using a sample program. The 78K0, 78K0S and 78K4 are the target processors. The user manual documents associated with this chapter are Nos. 5, 7, 8 and 9.

[Chapter 4 Programming](#)

This chapter shows how to handle CPU-specific dependencies in the C programming language for the various 78K Series CPUs using a sample program. The 78K0, 78K0S and 78K4 are the target processors. The user manual documents associated with this chapter are Nos. 5, 7, 8 and 9.

Related Documents:

Please refer to the documents listed below when using this tutorial.

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

Documents related to the development tool are stored as PDF files on the SP78Kxx installation CD.

Documents related to development tools (User's Manuals)

Document Name		Document No	No.
CC78K0 Ver.3.50 or later C compiler	Operation	U16613E	1
	Language	U14298E	2
RA78K0 Ver.3.60 or later Assembler Package	Operation	U16629E	3
	Language	U14446E	4
PM plus Ver.5.10		U16569E	5
ID78K0-NS Ver.2.52 Integrated Debugger	Operation	U16488E	6
SM78K Series Ver.2.52 System Simulator	Operation	U16768E	7
SM78K Series Ver.2.30 or later System Simulator	External Part User Open Interface Specifications	U15802E	8

Document Name		Document No	No.
CC78K0S Ver.1.50 or later C compiler	Operation	U16654E	1
	Language	U14872E	2
RA78K0S Ver.1.40 or later Assembler Package	Operation	U16656E	3
	Language	U14877E	4
PM plus Ver.5.10		U16569E	5
ID78K0S-NS Ver.2.52 Integrated Debugger	Operation	U16584E	6
SM78K Series Ver.2.52 or later System Simulator	Operation	U16768E	7
SM78K Series Ver.2.30 or later System Simulator	External Part User Open Interface Specifications	U15802E	8

Document Name		Document No	No.
CC78K4 Ver.2.40 or later C compiler	Operation	U16707E	1
	Language	U15556E	2
RA78K4 Ver.1.60 or later Assembler Package	Operation	U16708E	3
	Language	U15255E	4
PM plus Ver.5.10		U16569E	5
ID78K4-NS Ver.2.52 Integrated Debugger	Operation	U16632E	6
SM78K Series Ver.2.52 System Simulator	Operation	U16768E	7
SM78K Series Ver.2.30 or later System Simulator	External Part User Open Interface Specifications	U15802E	8

Documents related to devices (User's Manuals)

Document Name	Document No	No.
μ PD780024A, 780034A, 780024AY, 780034AY Subseries	U14046E	9

Document Name	Document No	No.
μ PD789046 Subseries	U13600E	9

Document Name	Document No	No.
μ PD 784038, 784038Y Subseries Hardware	U11316E	9

Sample programs and program execution environments described in this document are current as of January 2004 and are subject to change without notice.

When considering the use of a product, please first refer to the most up-to-date product documentation and confirm product availability by contacting an NEC Electronics sales representative.

CONTENTS

Chapter 1 Getting Ready	10
Tools Used in the Tutorial.....	11
Tutorial Sample Environment	12
Chapter 2 Trying Out PM plus and Simulator	13
Starting PM plus.....	15
Introduction to PM plus.....	16
Reading a Workspace File.....	18
Creating an Executable Program.....	20
Verifying Program Operation	22
Running the System Simulator (SM78Kxx)	23
Introduction to the System Simulator (SM78Kxx)	25
Introduction to the Input/Output Panel Window	26
Executing the Program	27
Stopping the Program.....	30
Exiting the System Simulator (SM78Kxx).....	31
Exiting PM plus	32
Chapter 3 System Simulator Basics	33
Counter Program Specifications.....	34
Starting PM plus.....	36
Creating a New Workspace.....	37
Editing the Source and Creating an Executable Program (1)	42
Running the System Simulator (SM78Kxx)	46
Setting Up the Input/Output Panel	48
Executing the Program (1).....	54
Debugging.....	57
Editing the Source and Creating an Executable Program (2)	70
Executing the Program (2).....	73
Exiting	82
Chapter 4 Programming	84
Slot Machine Program Specifications.....	85
Verifying Slot Machine Program Operation	87
Reading the Workspace File.....	88
Creating an Executable Program.....	89
Running the System Simulator (SM78Kxx).....	90
Running the Program.....	91
Stopping the Program.....	93
Comments about the Input/Output Panel	94
Exiting	99
Comments about the Program.....	101

Accessing Special-function Registers using Register Name - #pragma sfr	102
Registering an Interrupt Function #pragma interrupt or #pragma vect and __interrupt	103
Enabling/Disabling Interrupts DI(); and EI();	104
Outputting CPU Control Instructions HALT();, STOP();, BRK();, and NOP();	106
Appendix	108
Creating uoVRAM.dll	109
Counter Program Source Listing	114
Slot Machine Program Source Listing	126

Chapter 1 Getting Ready

This chapter is an overview of the development tools used in this tutorial, together with instructions on how to install the sample programs.

Note that the sample programs in this tutorial will only work with the development tools included with the SP78Kxx.

Tools Used in the Tutorial

This section gives an overview of the development tools used in the tutorial.

The name and main functions of each of the development tools are given below.

- **Device Files**

A device file contains device-specific information, which is required by the other development tools.

The sample programs in this tutorial use the following device files: the DF780034 for the 78K0, the DF789046 for the 78K0S and the DF784035 for the 78K4.

- **CC78Kxx 78K Series C compiler**

This is a highly versatile, highly portable C compiler developed to enable 78K Series embedded control programs to be written in C language.

- **RA78Kxx 78K Series Assembler Package**

This compiler generates 78K Series executable code from assembler source programs.

- **PM plus**

This is a Windows-based integrated development environment.

It integrates editing, compiling and debugging to provide an efficient and comprehensive development environment.

- **SM78Kxx 78K Series System Simulator**

Executing on the host PC, the SM78Kxx simulates the execution of 78K Series executable code.

In order to be able to execute the sample programs found in this tutorial, you must install the above-mentioned development tools.

For instructions on how to install the development tools, please refer to the document "Important notes about the SP78Kxx 78K Series Software Package" included with the SP78Kxx software package.

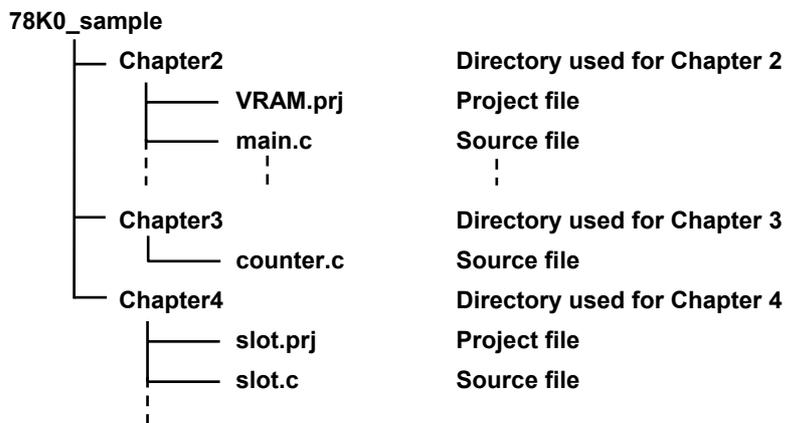
It is assumed throughout this document that the name of the program group registered in the Start Menu is the default name, "NEC Tools32".

Tutorial Sample Environment

This section describes the preparation required to run the sample programs presented in this tutorial.

- **Sample Program Main Body Directory Structure**

When you install the Sample Program Main Body, the following files are stored in the directory structure shown below, created under the directory you specified. Files stored in directories named after chapters (Chapter2, Chapter3, etc.) are explained in the corresponding chapter. The following shows the directories for the 78K0.

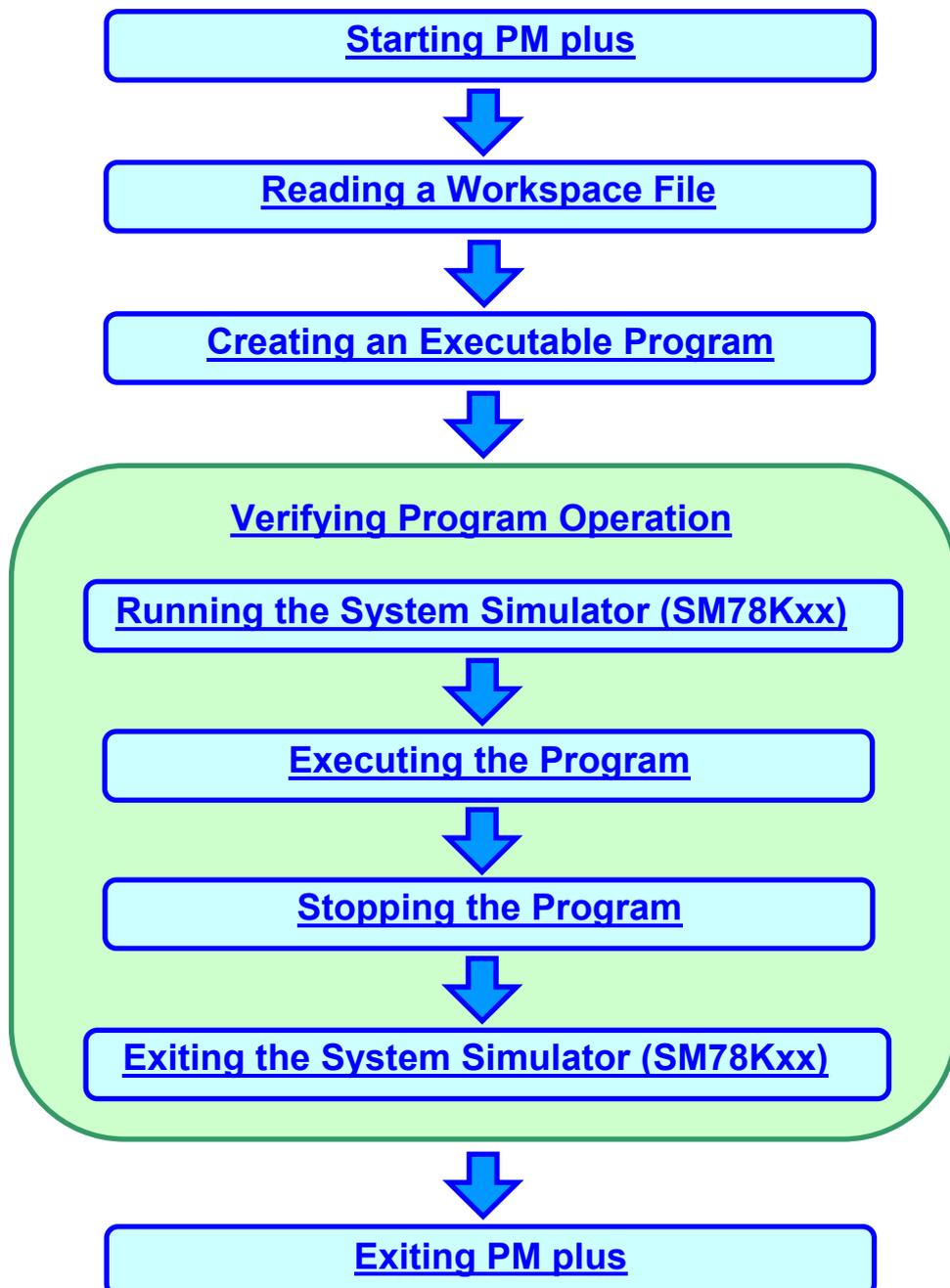


For the 78K0S, there is no Chapter2 directory.

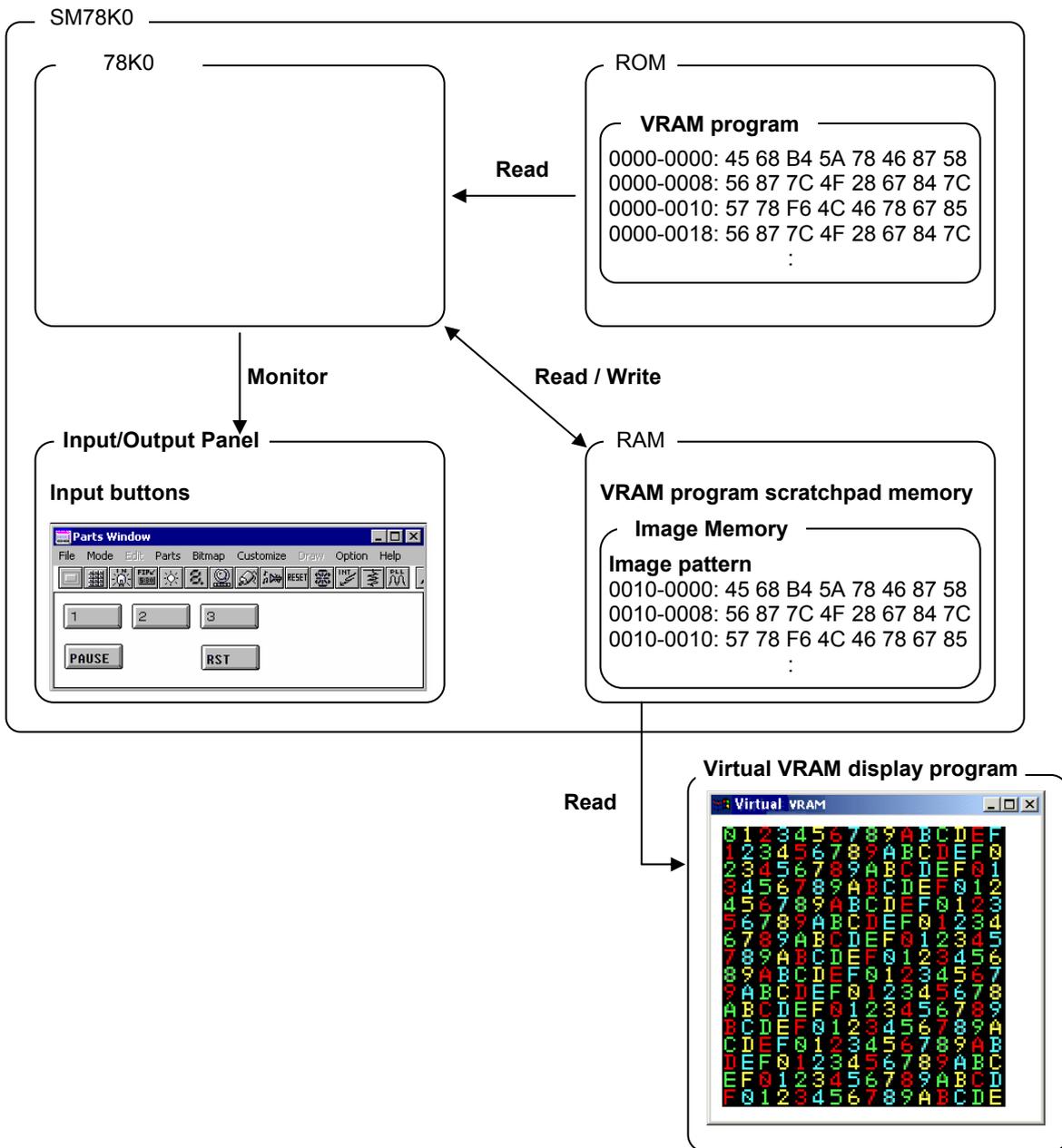
Chapter 2 Trying Out PM plus and Simulator

This chapter introduces you to the Program Manager and lets you try out the System Simulator (SM78Kxx) using a completed 78K Series program. (Note that from here on, the 78K Series processors (the K0, K0S and K4) are collectively referred to as Kxx.) The example in this chapter uses external RAM, therefore it is not compatible with the 78K0S, but it can be used with the 78K0 or the 78K4. The 78K Series program used here (called the VRAM program) writes an image pattern to Video RAM.

You will learn how to build the VRAM project, and, through operating the SM78Kxx, you will learn the basic operation of the tools (PM plus and System Simulator), as well as what is required in the project file to create an application program. The overall flow is shown here.



In this chapter, the VRAM program is executed in the following environment.



SM78Kxx: The 78Kxx is simulated, together with the RAM, ROM and input buttons.

Virtual VRAM display program:

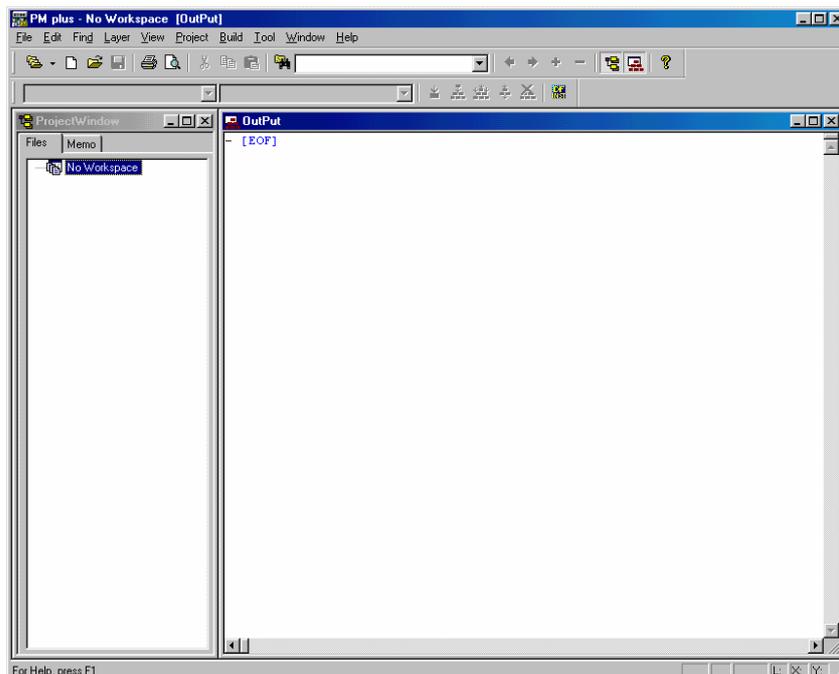
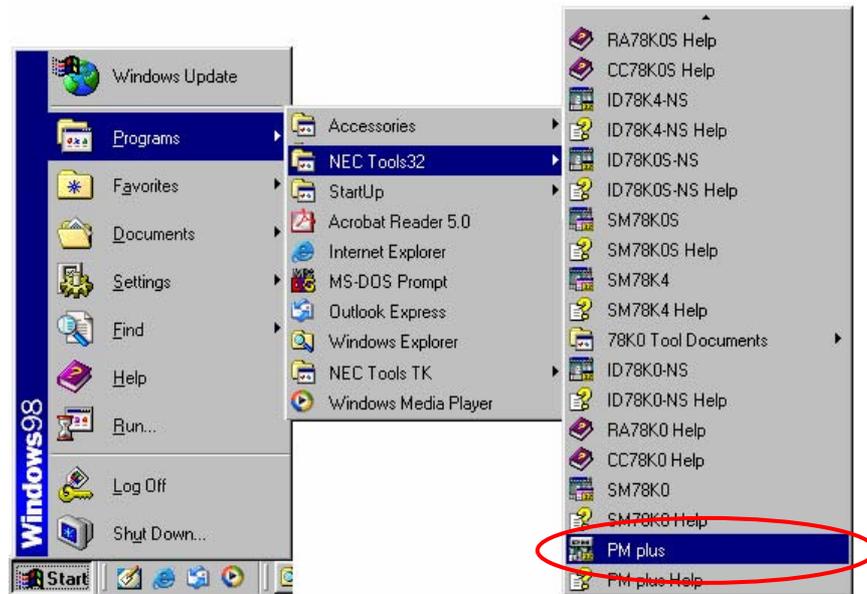
User-defined external parts for the SM78Kxx.

The SM78Kxx emulates a Video RAM display and displays the contents of the Video RAM on the screen. (This user-defined external part was constructed for use in this chapter and employs the External Part User Open Interface function. For the details on the External Part User Open Interface function, refer to the **SM78K Series System Simulator Ver.2.30 or later External Part User Open Interface Specification User's Manual (U15802E)**).

Starting PM plus

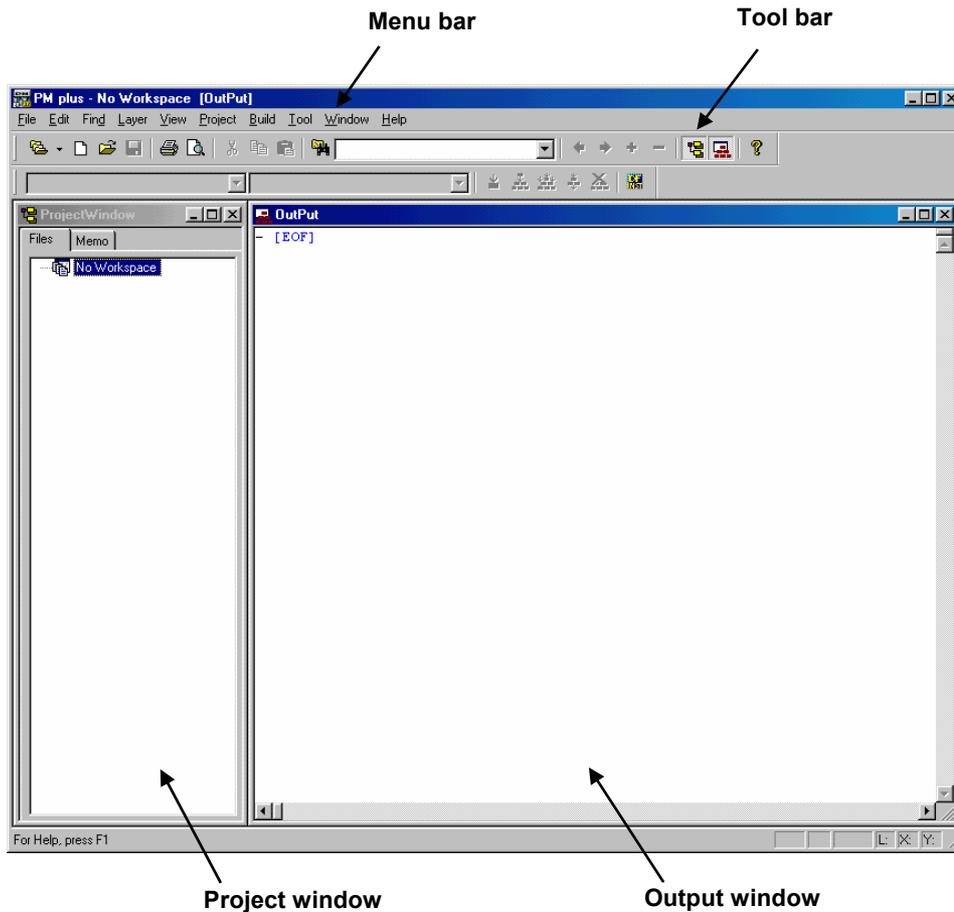
Now, let's try using each of the tools.

First, start PM plus. From the Windows Start menu, select Programs -> NEC Tools32 -> PM plus.



Introduction to PM plus

PM plus integrates all the functions required to create, edit, build, debug and manage programs, all within one programming environment. PM plus stores application program and environment settings in a single [project](#) file.



- Project window: Displays project, source, and include file names in a tree structure.
 Output window: Displays the progress of the [build](#) process.

➔ For details about the menu bar and tool bar, refer to the **PM plus Ver.5.10 User's Manual (U16569E)**.

What is a workspace ?

A workspace is the unit in which the file names of multiple project files are managed.

What is a workspace file ?

A workspace file is a file in which information such as the file names of multiple project files are stored. The file name is "xxxx.prw".

What is a project ?

A project defines the application system (all the files associated with an application program) under development using PM plus.

PM plus stores environment information in a project file.

What is a project file ?

A project file holds the environment information for a development tool or a source file within a project. The file extension used in the project file name is ".prj", such as xxxx.prj.

The project file is stored in the project directory that you specify when you create a new project.

Reading a Workspace File

PM plus stores the application program environment (directory, tool, and option information) in a project file.

Project file information is then stored in a workspace file.

Pre-created workspace and project files are used in this chapter.

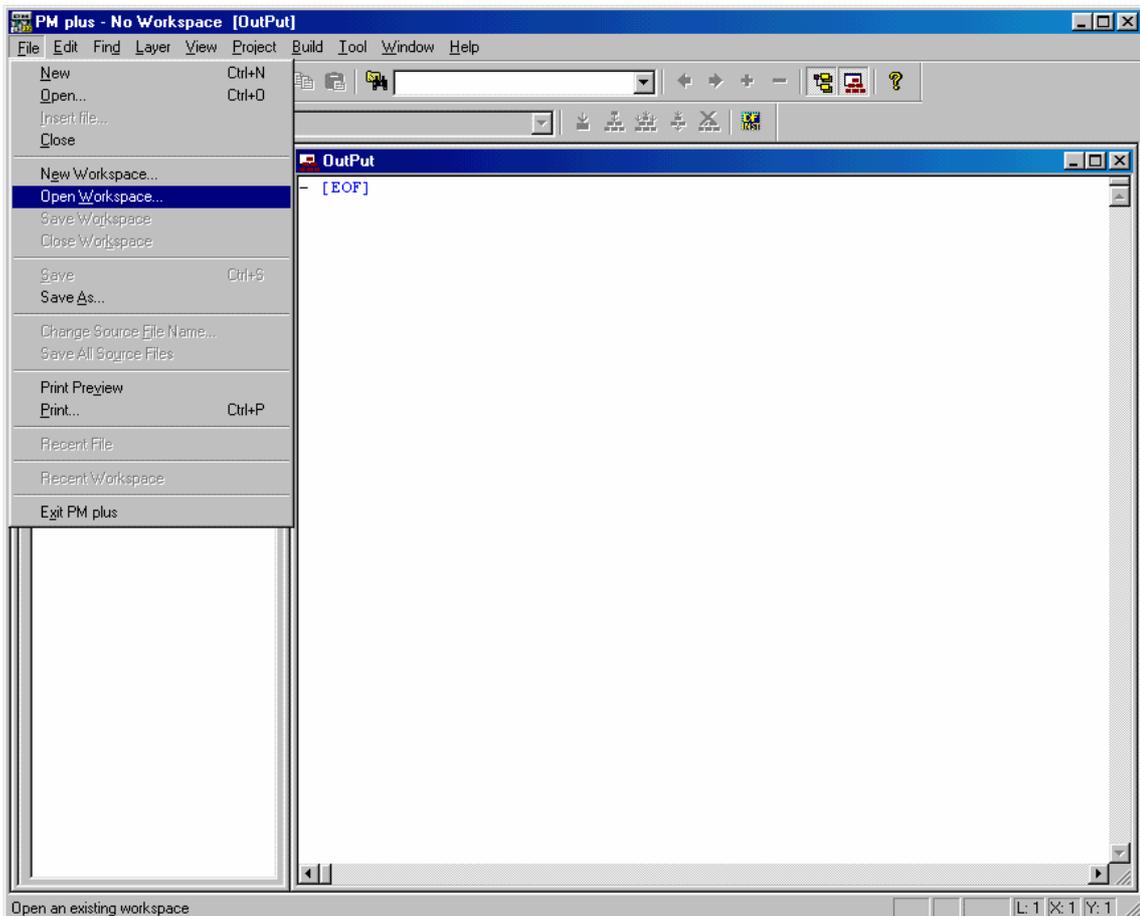
➡ Details on how to create workspace and project files are found in [Chapter 3 - System Simulator Basics](#).

The project file used in this chapter contains the completed VRAM program source file name, together with the SM78Kxx simulator settings for the 78Kxx, ROM, RAM and input buttons.

In order to start running the VRAM program, you must first open the project file in PM plus.

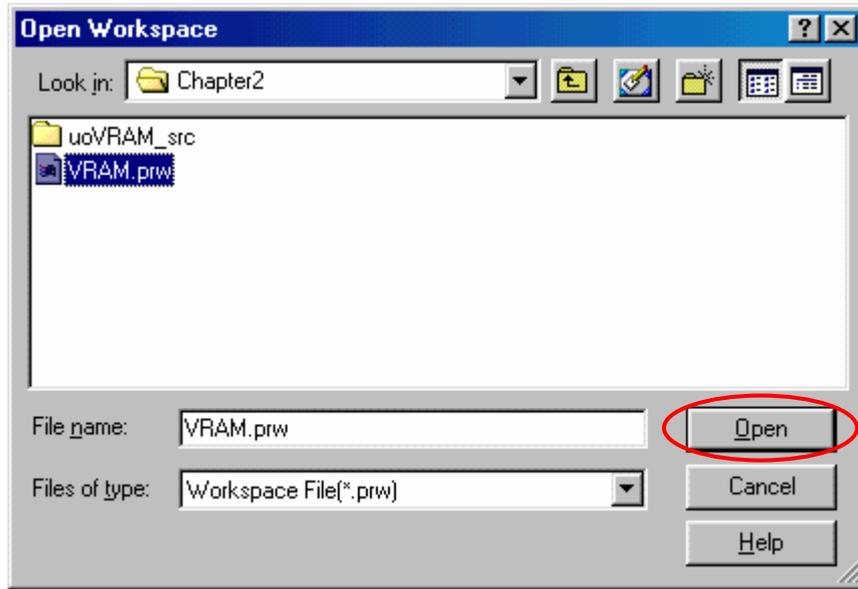
In PM plus, from the menus select File->Open Workspace... and specify the VRAM.prw workspace file.

➡ If you have not yet set up the Sample Environment, please refer to [Chapter 1 - Tutorial Sample Environment](#).





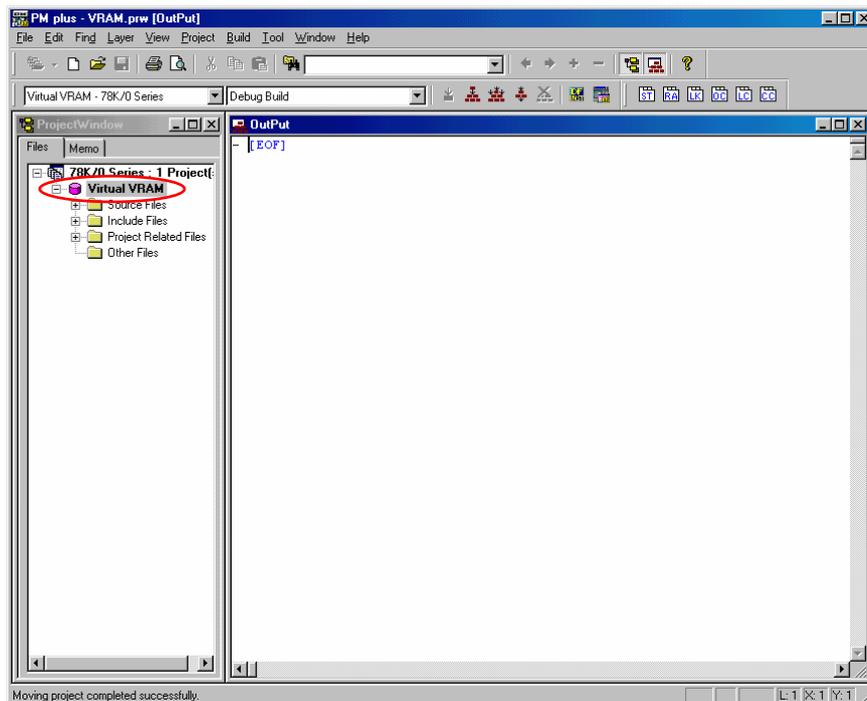
Open the Chapter2 directory.



Select VRAM.prw and click Open.



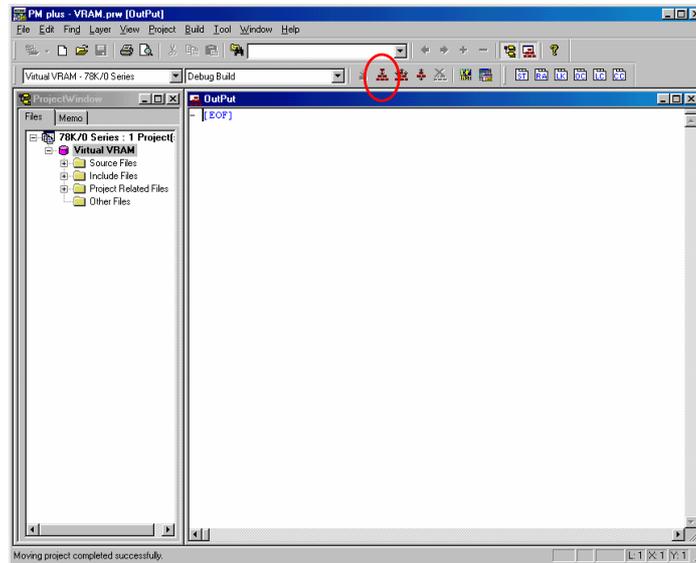
PM plus reads the workspace file "VRAM.prw."



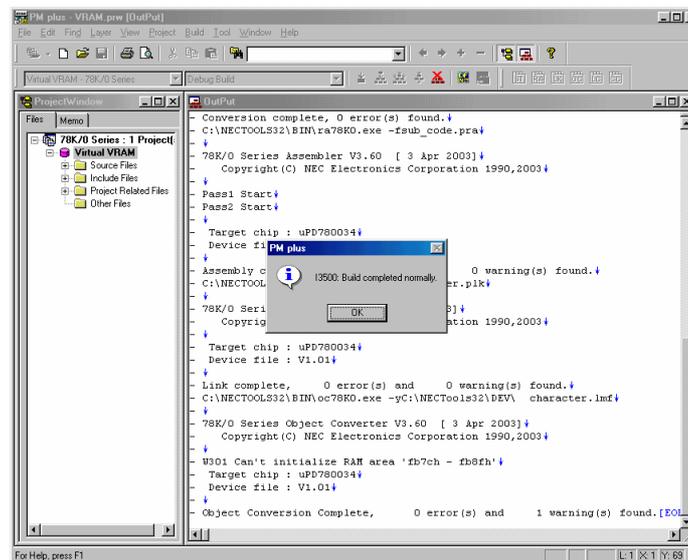
Creating an Executable Program

Next, you will create an executable program. This process is called creating a [build](#).

In PM plus, click the Build button  or select Build from the Build menu.



The build process executes.



The above dialog appears if the build process completes normally^{Note}.

Note Even though the message " Can't initialize RAM area " is displayed in the Output Window, this does not affect the operation of the VRAM program.

For details on this message, please refer to the RA78Kxx Assembler Package User's Manual.

What is a build ?

The Build function converts source files within a project into an executable program. PM plus automatically compiles, assembles, and links the program. In addition, after the first build of a project has been performed, PM plus checks if any source files have changed, and only compiles and assembles the changed files. This reduces the time required for the build.

What is a rebuild ?

For a build, only source files with changes are compiled and assembled. For a rebuild, all source files, whether they have changed or not, are compiled and assembled. When you change any compiler options or other settings, you should select rebuild instead of build. Also, there may be times when modified files are not detected. You will also need to select rebuild when:

- * You replace a modified source file with an earlier copy of the file that does not contain the modifications.
- * You adjust the clock of the host computer after a build.
- * You move the project environment to a different host computer whose clock setting is different from the previous computer.

Verifying Program Operation

NEC Electronics offers an [Integrated Debugger](#) and [System Simulator](#) execution environment for verifying the operation of a user application.

Here, we will run the System Simulator (SM78Kxx) and verify the operation of the program.

What is an integrated debugger (ID78Kxx) ?

An Integrated Debugger is a Windows-based software tool that allows you to debug a program within a development environment that consists of an in-circuit emulator connected to a target system. You can debug at the C source level or the assembly code level. With the event setting function of the in-circuit emulator, you can execute the program in real time and observe the operation.

What is a System Simulator (SM78Kxx) ?

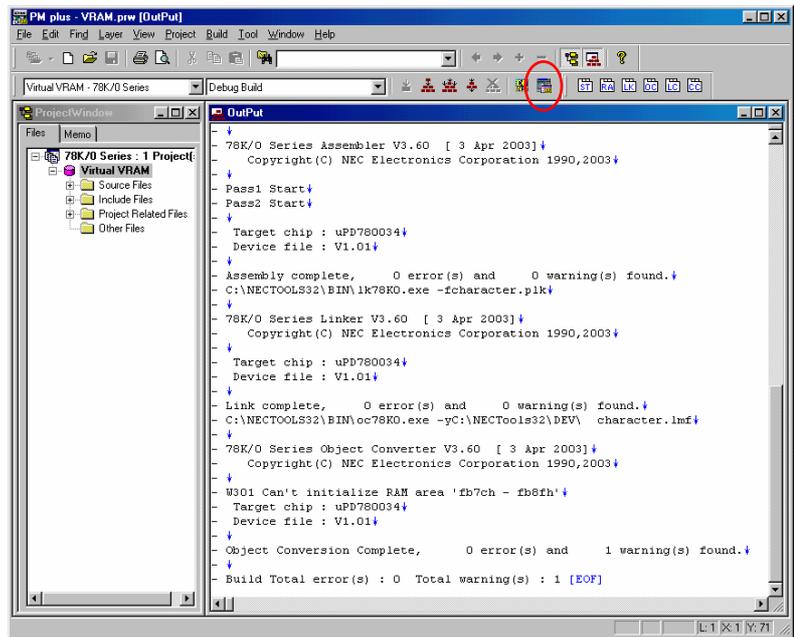
A System Simulator is a Windows-based software tool running on a host computer that simulates the operation of the target system, allowing you to run and debug your application program on the simulator. You can debug at the C source level or the assembly code level. With a System Simulator, you can separate application program logic verification from hardware development.

Running the System Simulator (SM78Kxx)

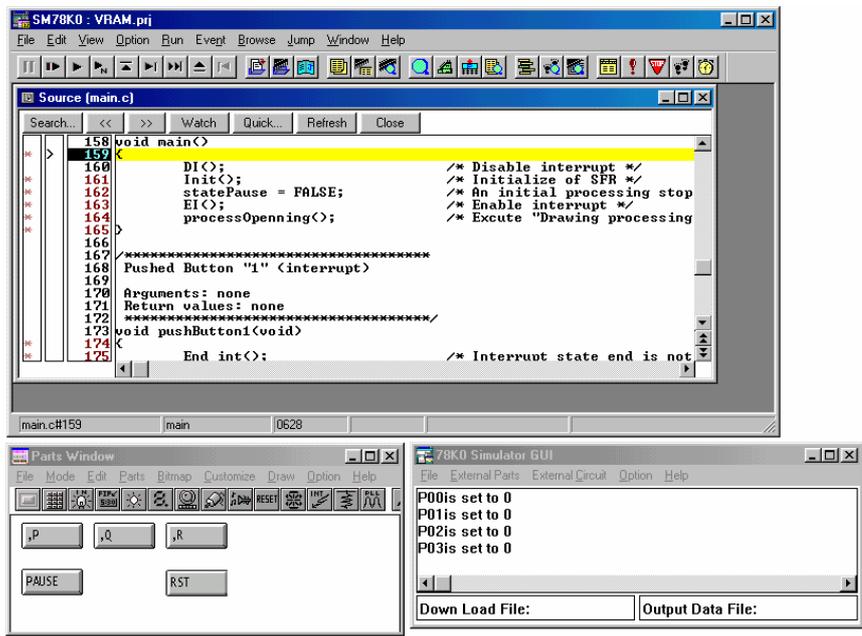
Next, you will run the System Simulator (SM78Kxx).

In PM plus, click the Debug button  or select Build ->Debug from the menu bar.

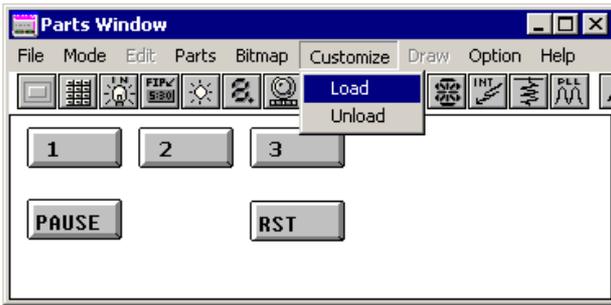
If the Debug button is not displayed, select Tool->Debugger selection->SM78Kxx System Simulator. For details on option settings, refer to [Chapter 3 - System Simulator Basics](#).



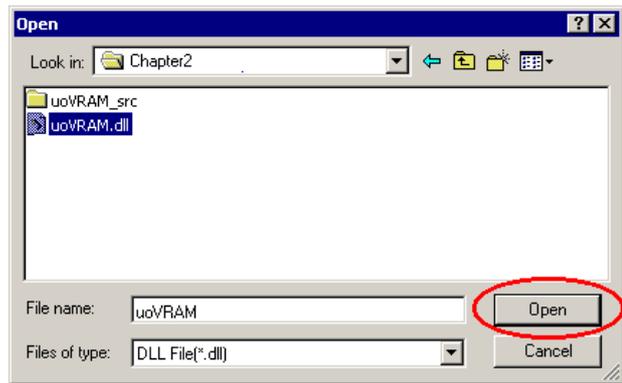
The SM78Kxx starts.



Next, from the Input/Output Panel window menus select Custom -> Load and open the "uoVRAM.dll" file.



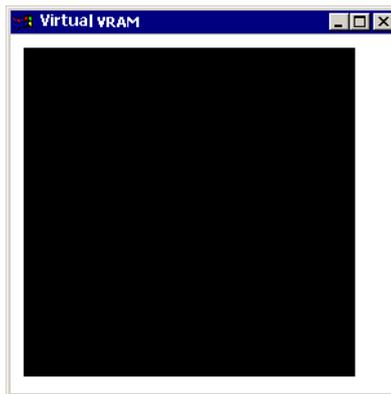
Select "uoVRAM.dll" and click Open.



If the "uoVRAM.dll" file is not displayed here, enter the name directly into the File name entry field or set Windows Explorer to be able to display files with the .dll extension.



The virtual VRAM display window opens.



The file "uoVRAM.dll" used here was created for the VRAM program, so it is not necessary to change any settings for the file. For an explanation of how to create the uoVRAM.dll from the source files, refer to [Appendix - Creating uoVRAM.dll](#). For additional details, refer to the SM78K Series System Simulator Ver.2.30 or later External Part User Open Interface Specification User's Manual (U15802E).

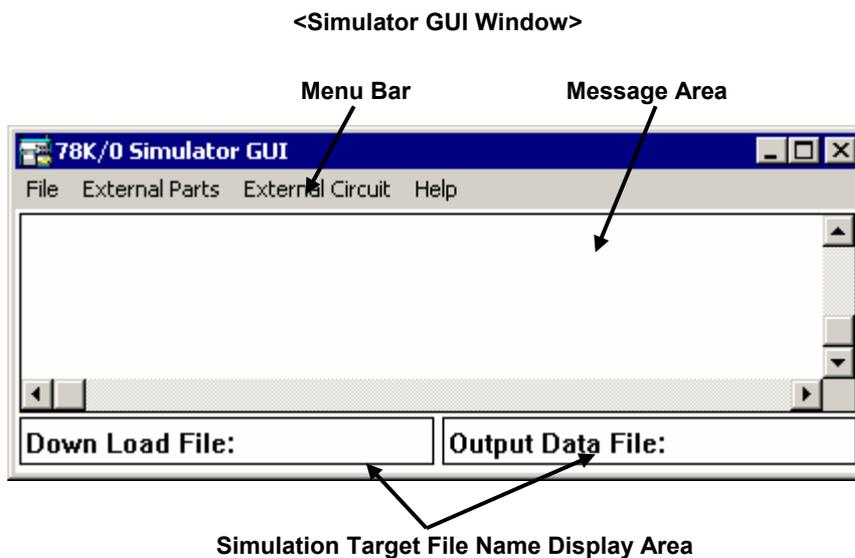
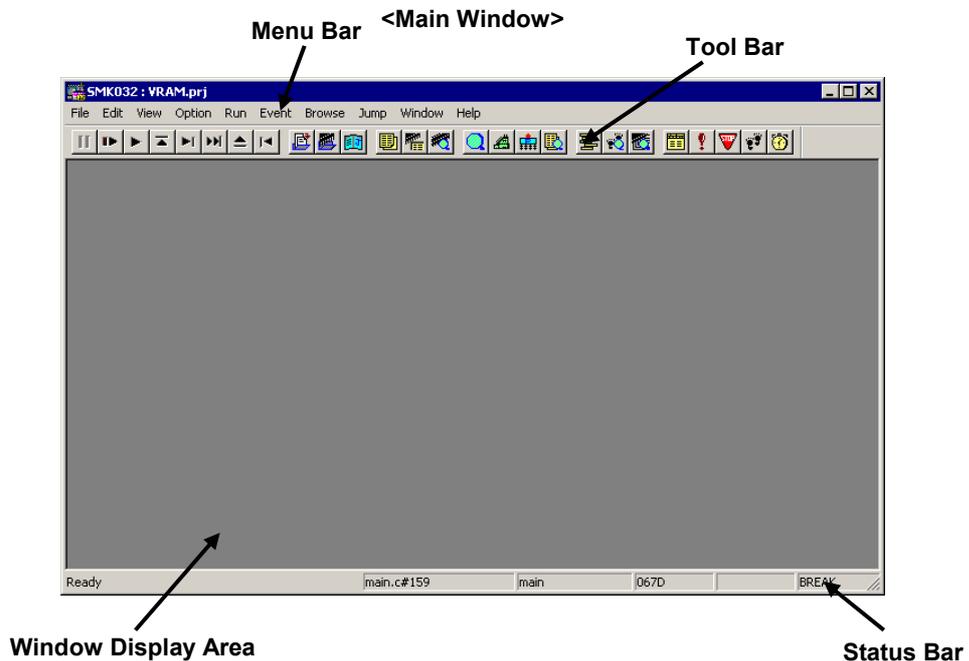
Introduction to the System Simulator (SM78Kxx)

The System Simulator (SM78Kxx) consists of a Main window and a Simulator Graphical User Interface (GUI) window.

Main window: Displays the status of the CPU core and controls the simulator execution.

Simulator GUI window: Controls external parts

The initial screen of the SM78Kxx is as follows.



➡ For details about each area, menu bar and tool bar, refer to the **SM78K Series System Simulator Ver.2.52 Operation User's Manual (U16768E)**.

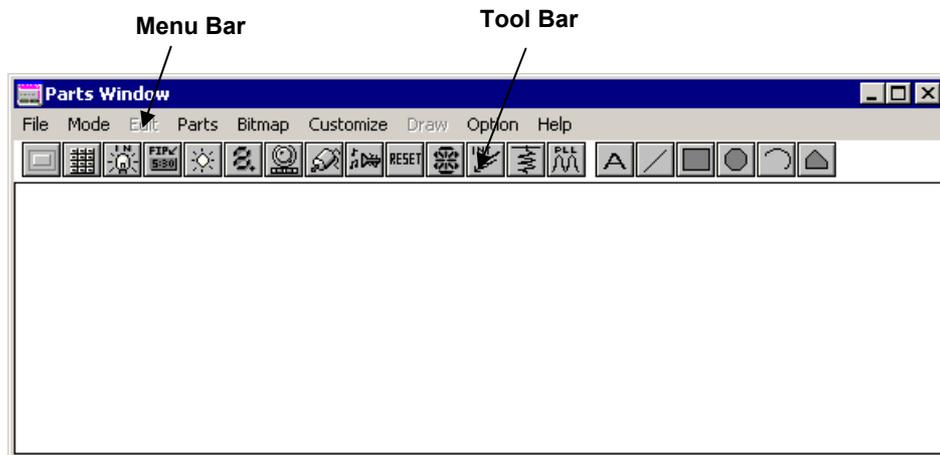
Introduction to the Input/Output Panel Window

The SM78Kxx offers standard parts, such as buttons and LEDs, as part of the simulated target system.

The Input/Output Panel window allows you to control the part settings and to operate the parts.

To open the Input/Output Panel window, from the SM78Kxx Simulator GUI window, select External Parts -> Input/Output Panel.

- ➡ For information on Input/Output Panel settings, refer to [Chapter 3 - System Simulator Basics](#).
- ➡ For details about the menu bar and tool bar, refer to the **SM78K Series System Simulator Ver.2.52 Operation User's Manual (U16768E)**.



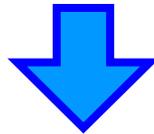
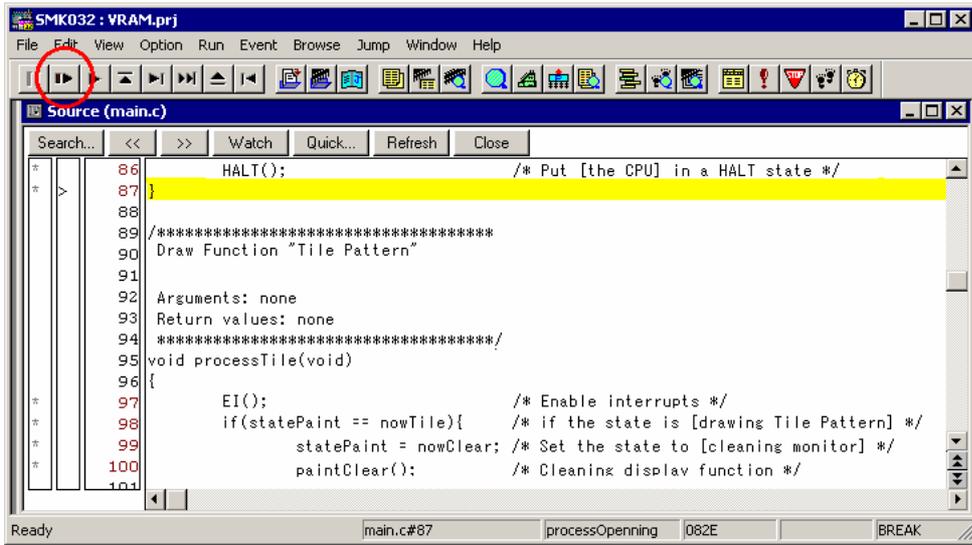
You can create additional parts as required by using Microsoft Visual C++TM, and the same procedure as that used for creating the uoVRAM.dll file.

- ➡ For details on how to create user-defined external parts, refer to the **SM78K Series System Simulator Ver.2.30 or later External Part User Open Interface Specification User's Manual (U15802E)**.

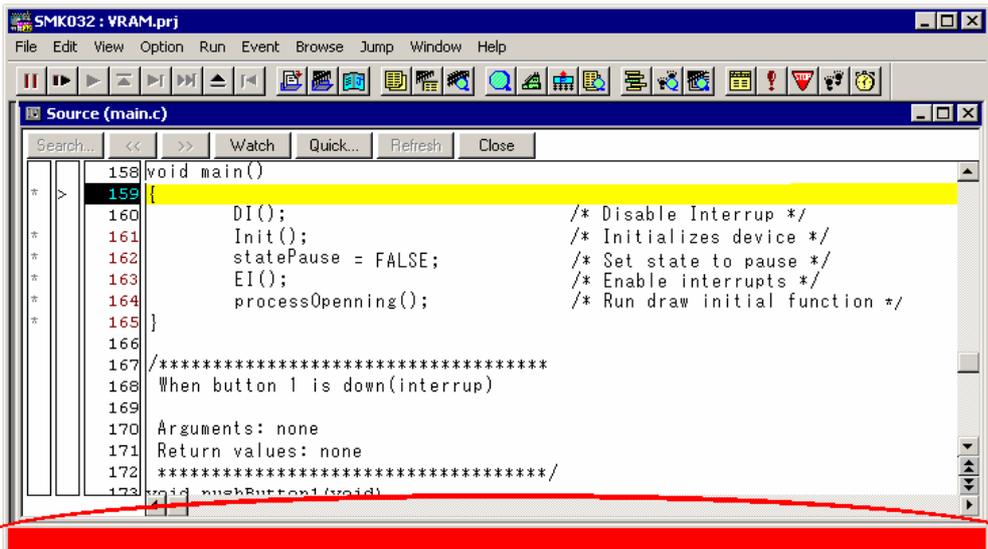
Executing the Program

Next, you will execute the program.

Click the SM78Kx Restart button  or select Run->Restart. The VRAM program will execute.

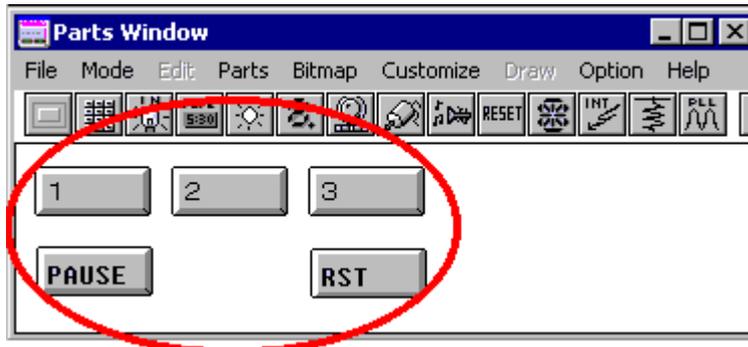


The program executes.



The color of the status bar changes to red during program execution.

Next, you will operate the VRAM program while it executes. Click each button on the Input/Output Panel, and confirm that the VRAM Display window changes accordingly.



Clicking the [1] button switches to a program that draws the pattern shown on <Screen 1>.

Clicking the [2] button switches to a program that draws the pattern shown on <Screen 2>.

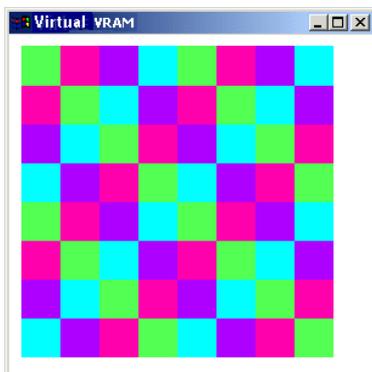
Clicking the [3] button switches to a program that draws the pattern shown on <Screen 3>.

Clicking the [RST] button [resets the target CPU](#) under simulation.

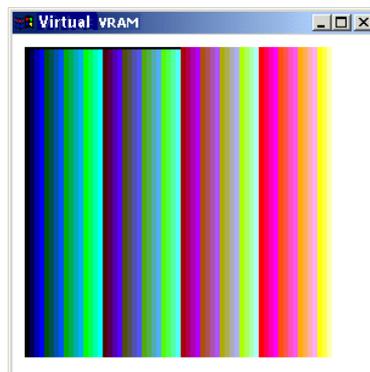
Clicking the button while a pattern is being drawn in the Display window pauses the drawing program.

Clicking the button while the program is paused resumes execution of the drawing program.

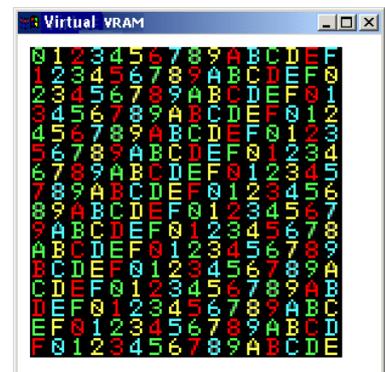
<Screen 1>



<Screen 2>



<Screen 3>



You have now confirmed that the VRAM program operates normally.

What does "resetting the target CPU" mean ?

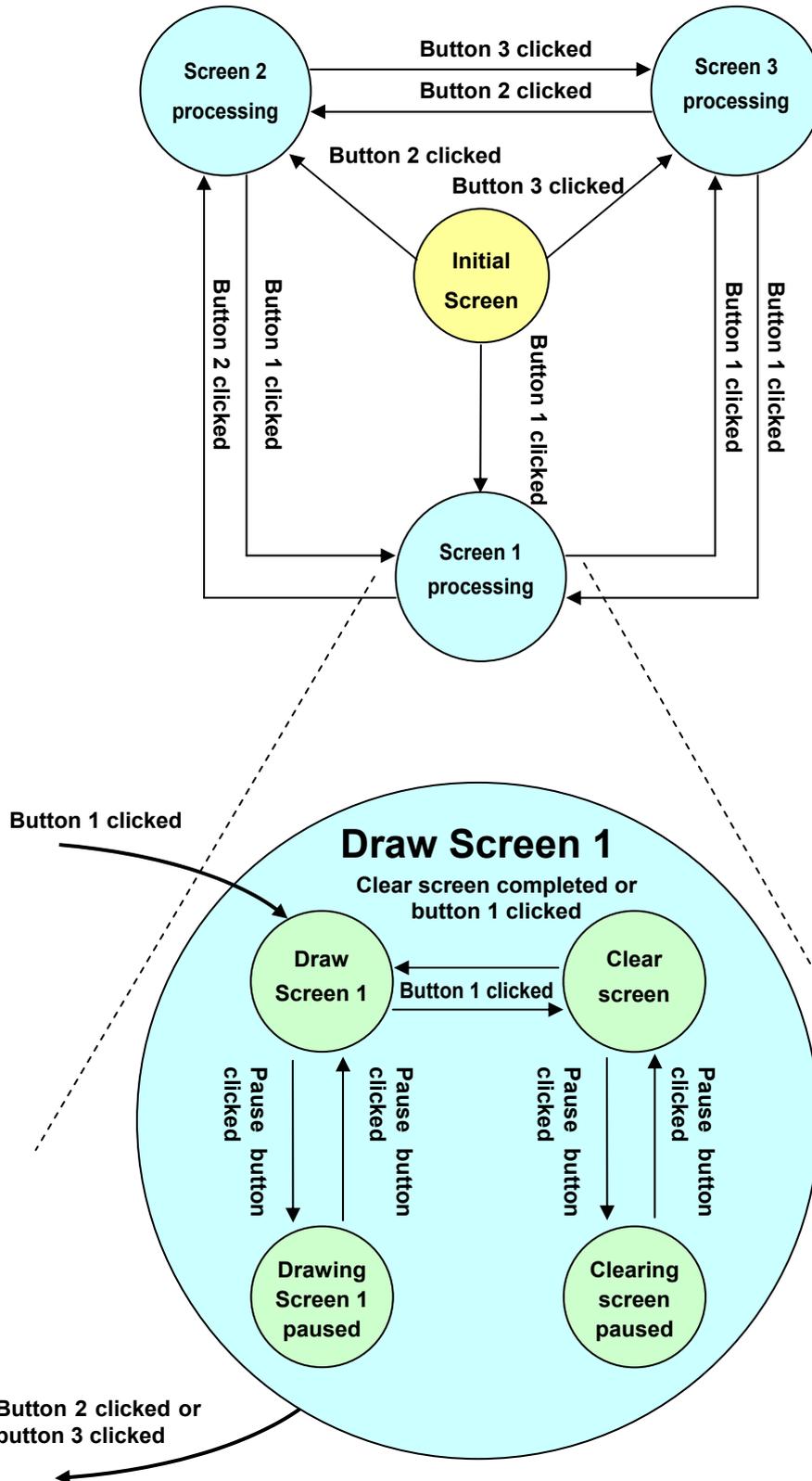
The target CPU mentioned here is a virtual μ PD780034, which the SM78K0 simulates. For the 78K4, the target CPU is the μ PD784035.

Resetting the target CPU means the SM78K0 simulates the application of a logical low signal to the RESET terminal of the virtual μ PD780034.

As a result, the VRAM program running on the virtual μ PD780034 returns to its initial state.

This operation **does not** mean that the personal computer that SM78Kxx is running on is reset.

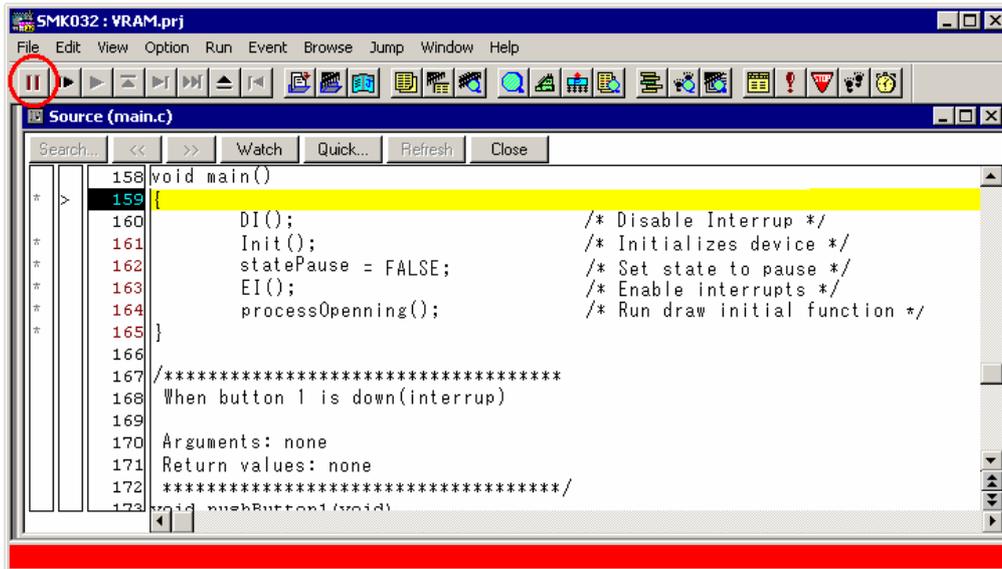
The following diagram shows the screen processing flow of VRAM program.



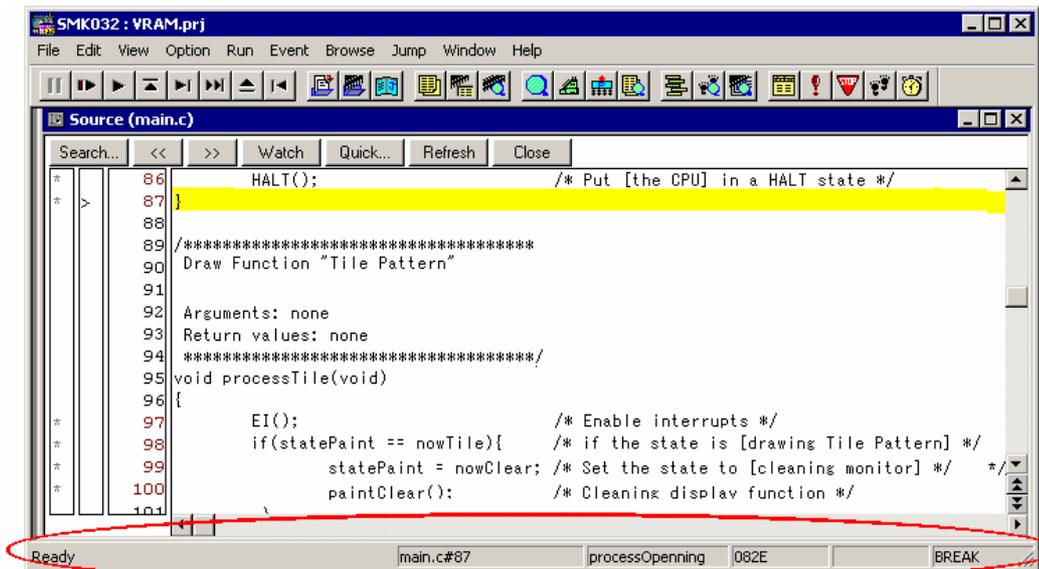
Stopping the Program

Next you will stop the execution of the program.

Click the SM78Kxx Stop button  or select Run -> Stop.



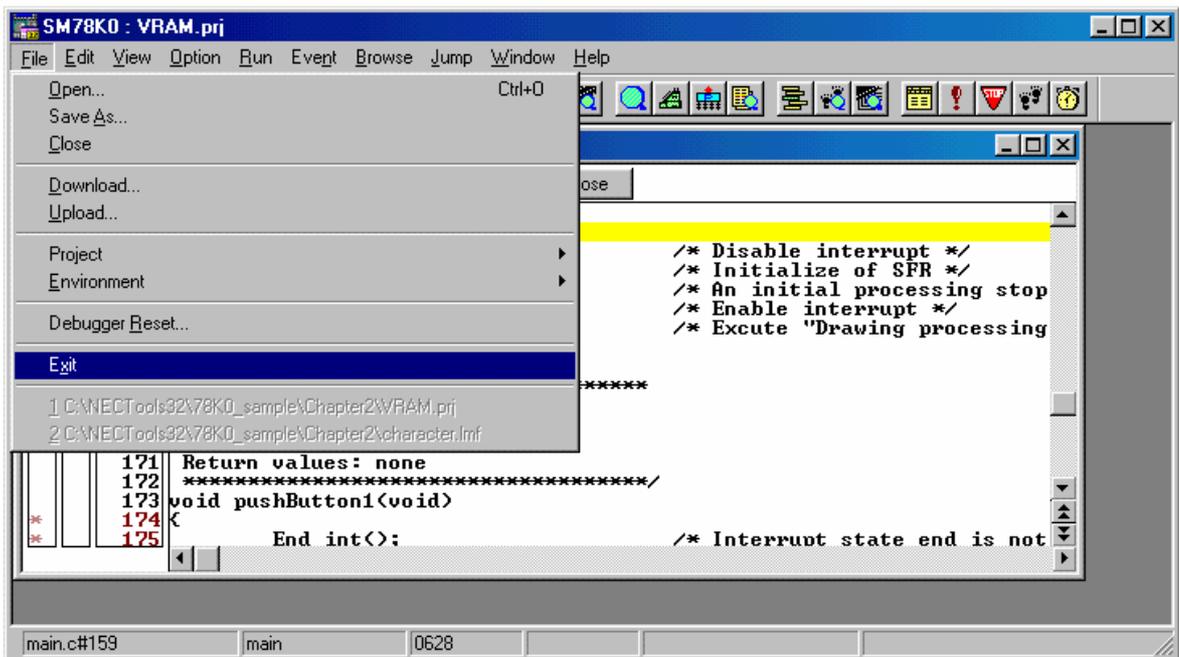
Program execution stops.



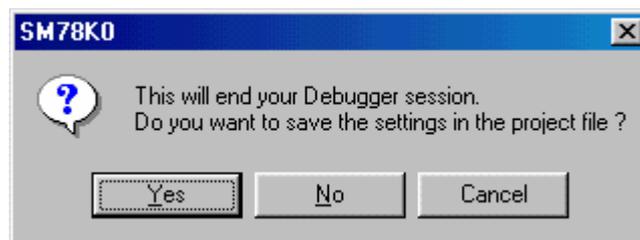
The status bar color returns to its original color when you stop the program.

Exiting the System Simulator (SM78Kxx)

To exit the System Simulator (SM78Kxx), from the menus in the SM78Kxx Main window, select File -> Exit.



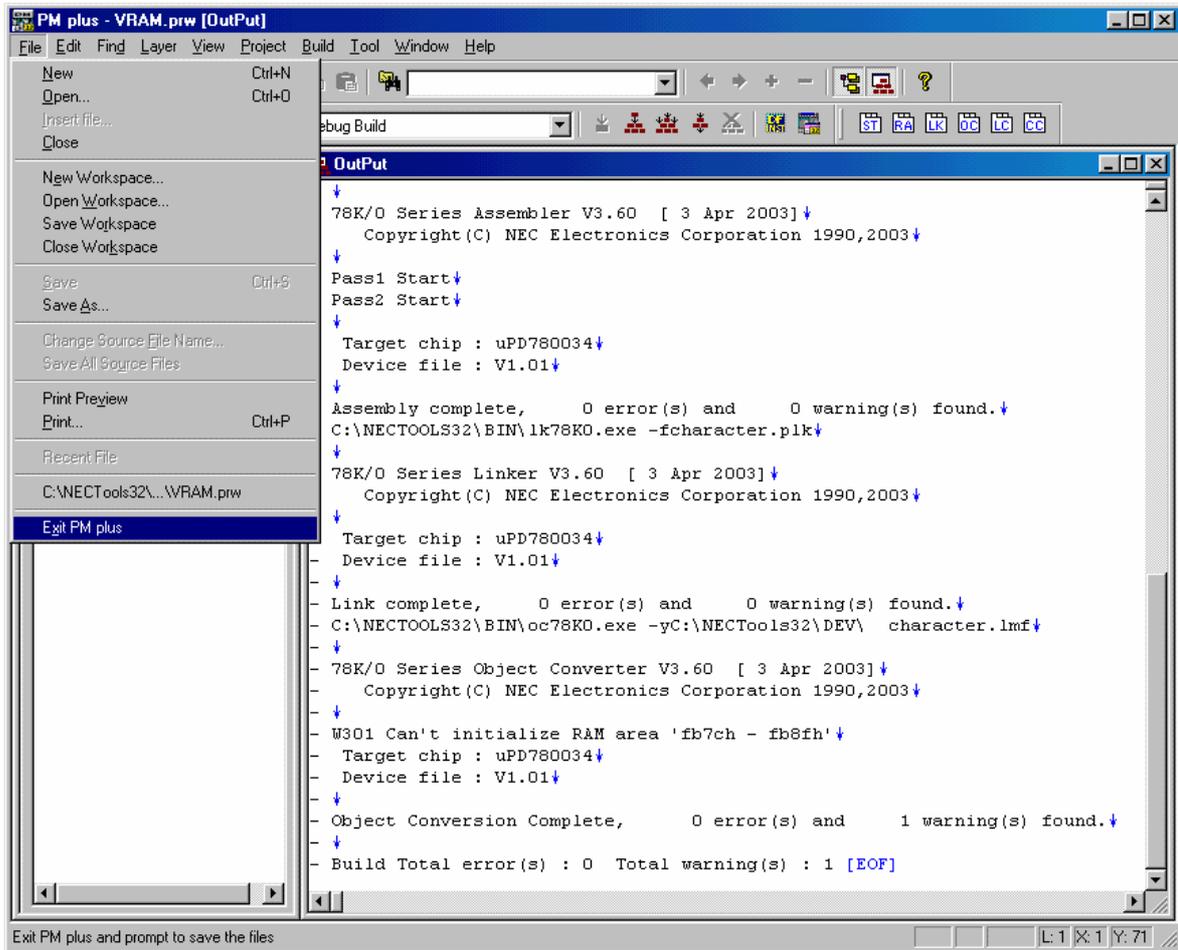
A dialog is displayed asking you if you want to exit.



Click OK button to exit the SM78Kxx.

Exiting PM plus

To exit PM plus, from the menus in PM plus window, select File -> Exit PM plus.



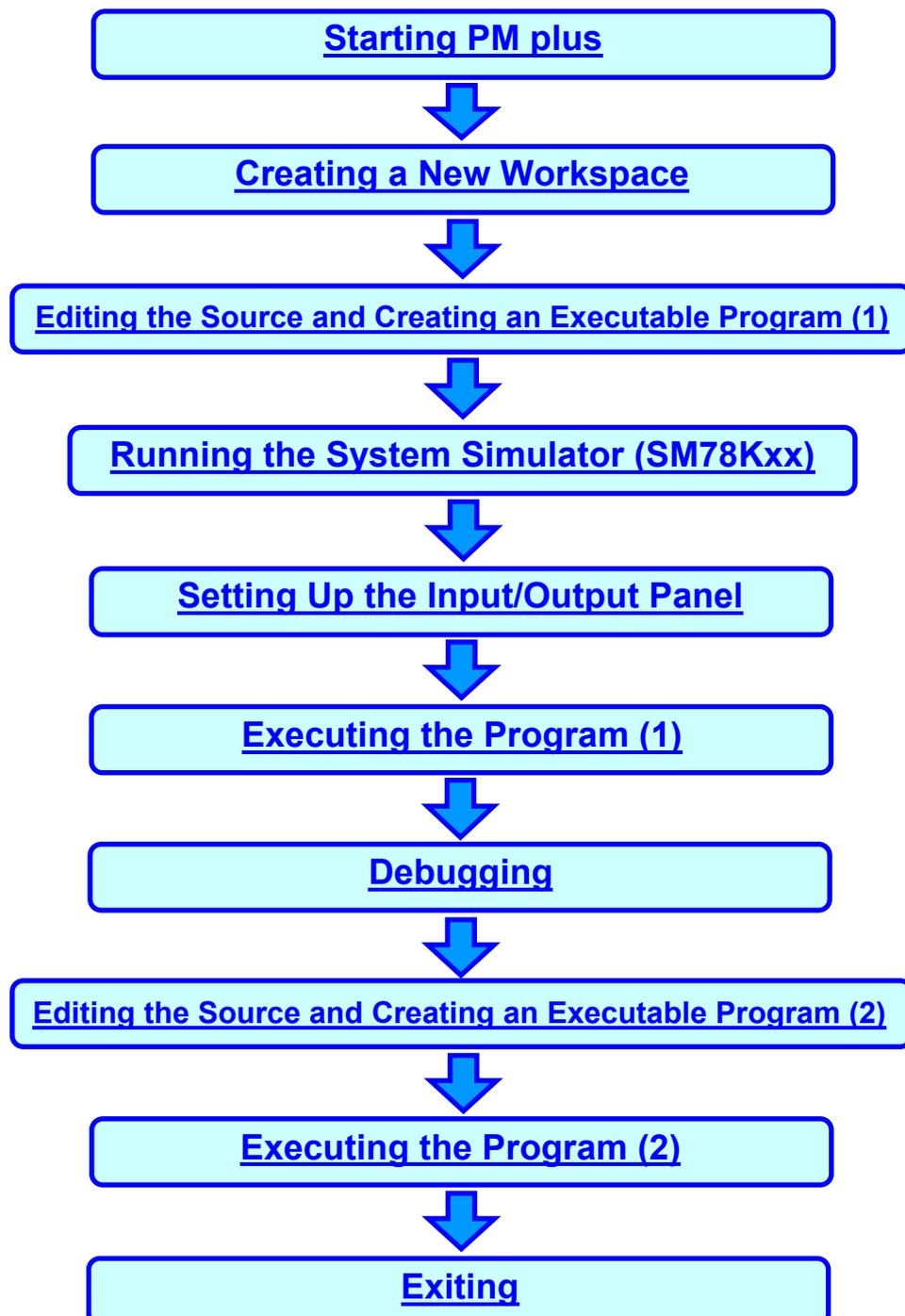
You will exit PM plus program.

Chapter 3 System Simulator Basics

This chapter explains basic debugging with the System Simulator (SM78Kxx), using a sample program. The sample program used here is a [counter program](#).

The sample counter program you will use contains several bugs that you will correct as you operate the simulator.

The overall flow is shown here.



Counter Program Specifications

Before starting to debug the counter program, you need to have a general understanding of the counter program. The basic external specifications of the program are as follows.

External Specifications

The devices specified are a button and a two-digit 7-segment display; when the button is clicked, the counter increments. (INTTMO0 is used for the 78K0S and INTC00 is used for the 78K4.)



Here you will implement an increment function and an LED display function. The main routine, which is used for debugging, takes advantage of the SM78Kxx debugging functions and handles processing such as button input and initialization.

Basic Specifications (Increment function and LED display function)

- Increment function
 - When an INTTMO0 interrupt occurs, the 2-digit decimal counter increments by one.
 - When the counter reaches 99, the next increment returns the counter to 0 (counts in a loop).
- LED display function
 - The decimal counter value is output to the 7-segment LED display.
 - An [I/O port](#) of the 78Kxx is used to control the 7-segment LED display

Main Routine Basic Specifications

- Initialize the counter to 0.
- Initialize the I/O port used for controlling the [7-segment LED](#) display.
- To simulate an INTTMO0 interrupt, the SM78K0 uses an [internal interrupt button](#) to generate a virtual internal interrupt, and implements only the part necessary to handle the internal interrupt.”

What is an I/O Port ?

Almost all 78K Series devices are equipped with I/O ports, which allow the CPU to control external components and to acquire external signals.

For details on the I/O port, refer to the user manual of the device being used.

The internal specifications are as follows.

Internal Specifications

- Store the counter value in global variables count1 and count10

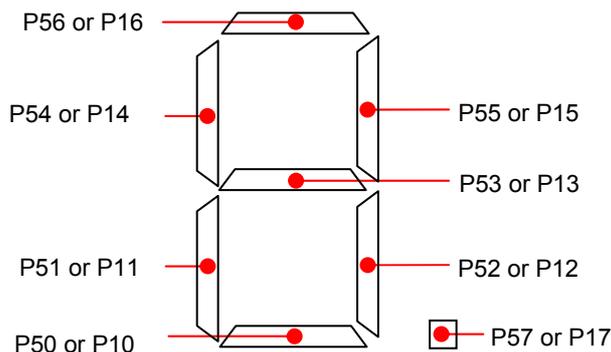
Variable	Contents
int count1	Stores the value of the first digit of the counter (1's)
int count10	Stores the value of the second digit of the counter (10's)

- The program consists of three functions: LED display, increment and Main routine for debugging.

Function	Contents
LED display void putLED()	- Displays the counter value on the LED display.
Increment void interrupt1()	- Starts on an INTTM00 interrupt. - Increments the counter value and handles digit overflow (carry and loop). - After incrementing the counter value, calls the putLED() function.
Main routine void main()	- Initializes the I/O port that controls the LED display. - Sets the conditions for accepting an INTM00 interrupt. - Initializes the counter value to 0, starts the putLED() function and displays an initial value of 0. - Puts the CPU in HALT mode.

- For the 78K0 and 78K4, I/O ports P4 and P5 are used to control the 7-segment LED display. P4 outputs the display contents, while P5 selects the digit. For the 78K0S, I/O ports P0 and P1 are used.

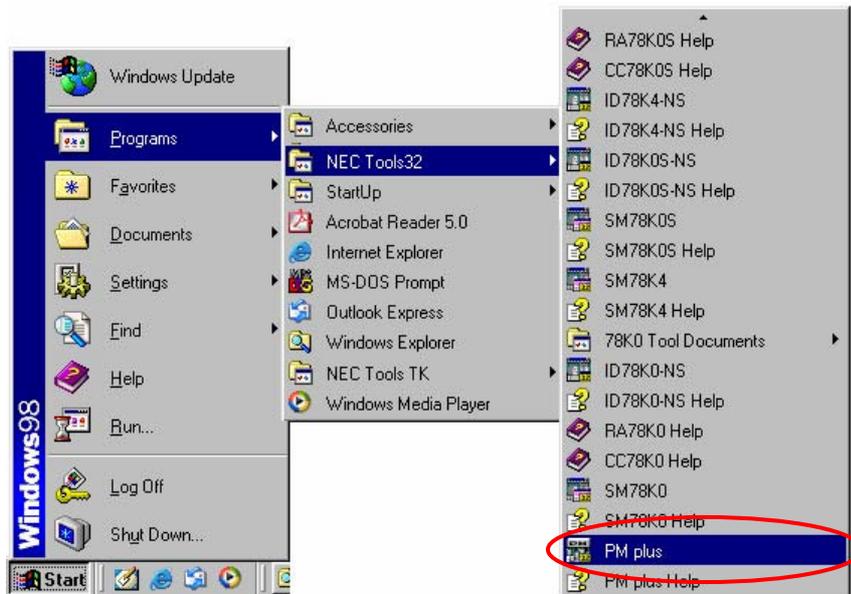
Port	Bit position	Port Address	Usage
P4 or P0	0	P40 or P00	When the state changes 0 -> 1, specifies that the contents of P1 are to be displayed on the first digit
	1	P41 or P01	When the state changes 0 -> 1, specifies that the contents of P1 are to be displayed on the second digit
P5 or P1	0	P50 or P10	Sets the state of the bottom segment of the display (1 - lit, 0 - not lit)
	1	P51 or P11	Sets the state of the lower left segment of the display (1 - lit, 0 - not lit)
	2	P52 or P12	Sets the state of the lower right segment of the display (1 - lit, 0 - not lit)
	3	P53 or P13	Sets the state of the middle segment of the display (1 - lit, 0 - not lit)
	4	P54 or P14	Sets the state of the upper left segment of the display (1 - lit, 0 - not lit)
	5	P55 or P15	Sets the state of the upper right segment of the display (1 - lit, 0 - not lit)
	6	P56 or P16	Sets the state of the top segment of the display (1 - lit, 0 - not lit)
	7	P57 or P17	Sets the state of the lower right dot of the display (1 - lit, 0 - not lit)



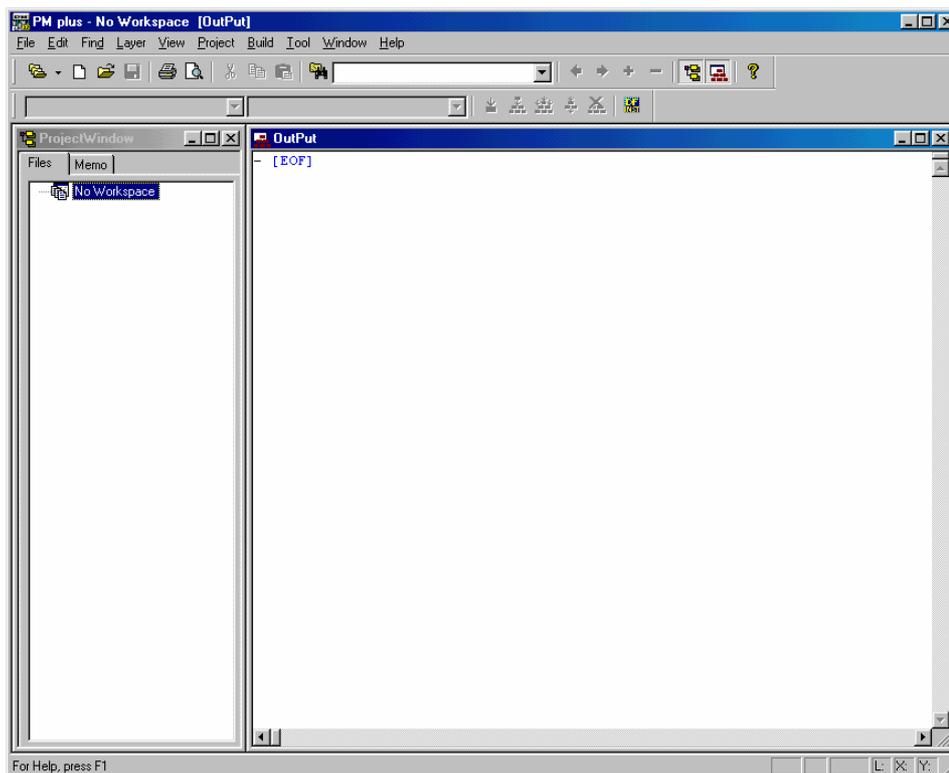
Starting PM plus

First, start PM plus.

From the Windows Start menu, select Programs->NEC Tools32->PM plus.



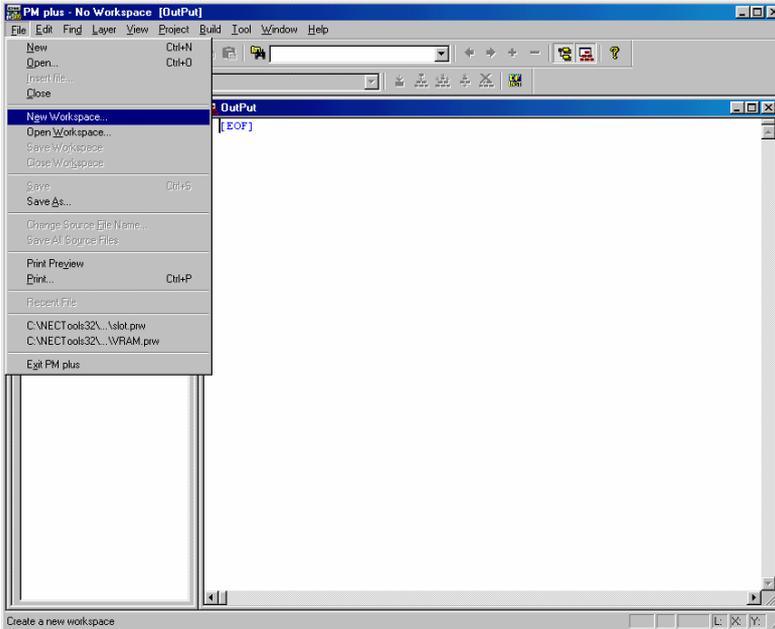
PM plus starts.



Creating a New Workspace

Next, you will create a new workspace.

From PM plus menus select File -> New Workspace...



The New Workspace dialog box opens.

Each item in the dialog box is explained below.

Workspace File Name:

- The name assigned to the file that stores the workspace information.

Folder:

- Specifies the folder in which workspace and project files are stored.
- Click the Browse... button to display a dialog box from which you can select a directory.

Project Group Name:

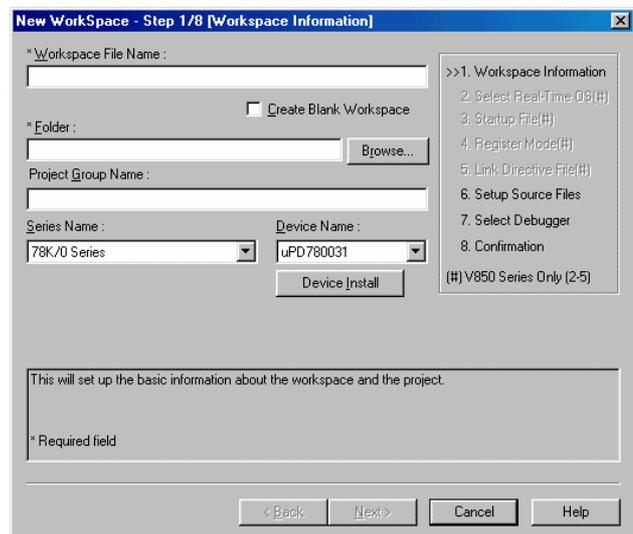
- Specifies what is displayed in the title bar of the Project Window.

Series Name:

- Specifies the series name of the device file used.

Device Name:

- Specifies the device name of the device file used.



Actual entries for the items in this dialog box are given on the next page.

Enter the following workspace information into the dialog box:

Workspace File Name:

counter

Folder:

For the 78K0:

\78k0_sample\Chapter3

(Click Browse... button and select the [sample directory](#).)

For the 78K0S:

\78k0S_sample\Chapter3

For the 78K4:

\78k4_sample\Chapter3

Project Group Name:

counter program

Series Name:

78K/0 Series (for the 78K0)

78K/0S Series (for the 78K0S)

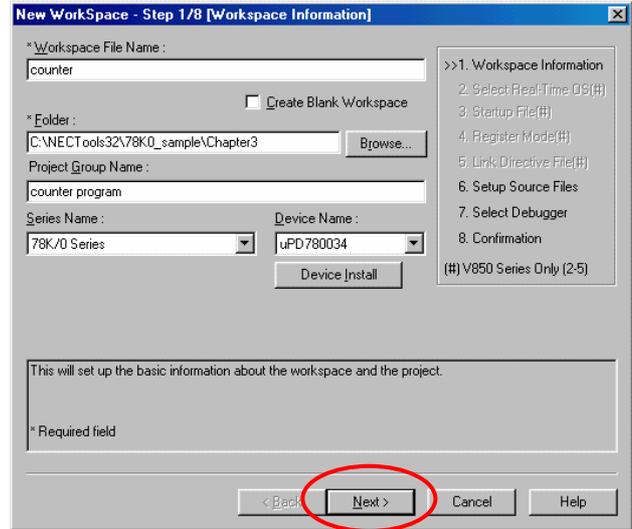
78K/4 Series (for the 78K4)

Device Name:

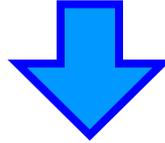
μ PD780034 (for the 78K0)

μ PD789046 (for the 78K0S)

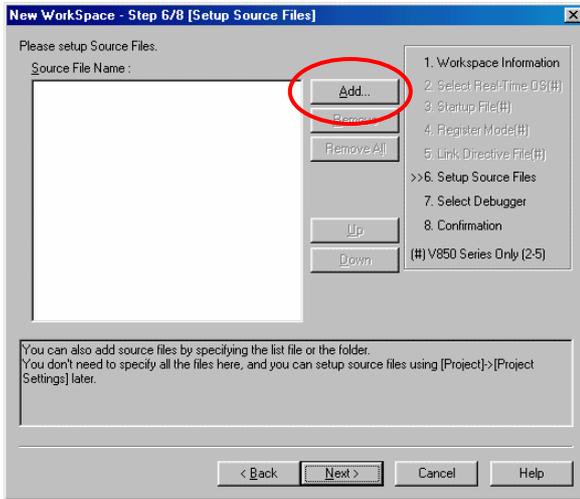
μ PD784035 (for the 78K4)



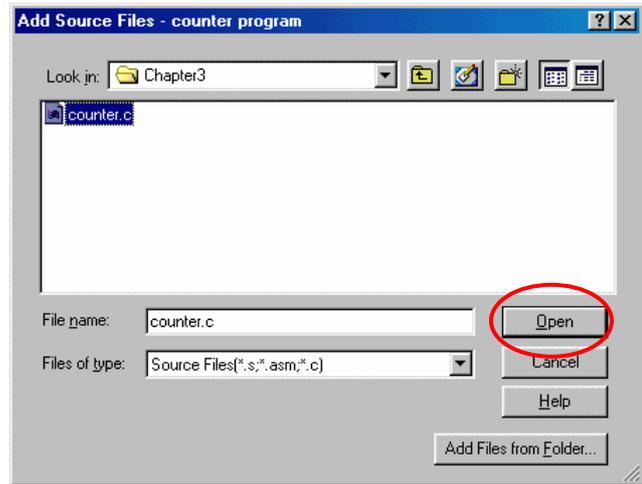
After completing the settings, click the Next button.



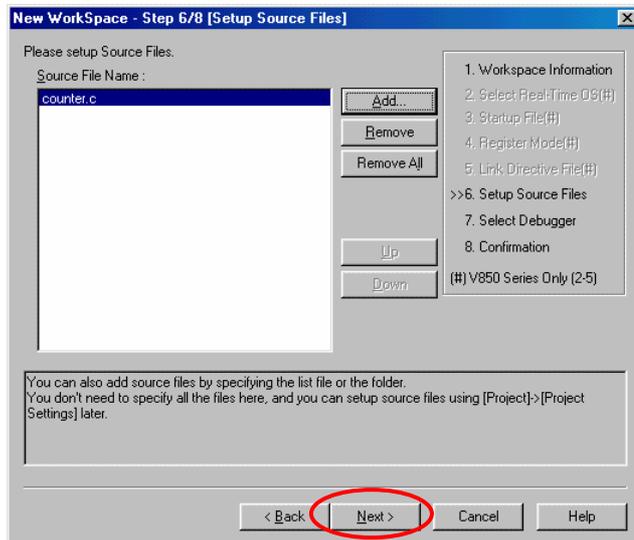
Next, you will add a source file to the project.
Click Add... button.



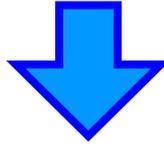
Select "counter.c" and click Open.



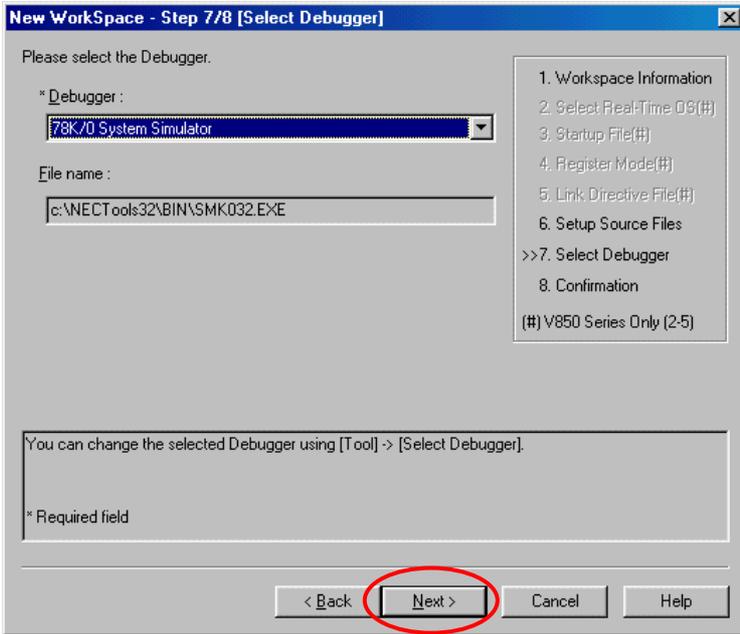
If the source file name does not appear in the list, this means that the folder position is not correctly set in the workspace information setting.
Perform the setting again, selecting the correct directory.



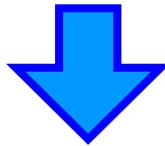
The source file "counter.c" is added to the project.



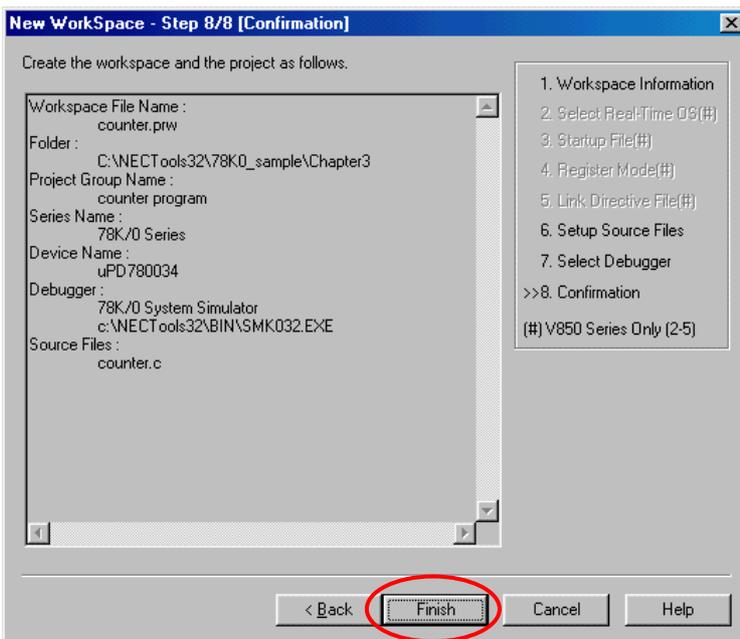
Specify the debugger to be used.



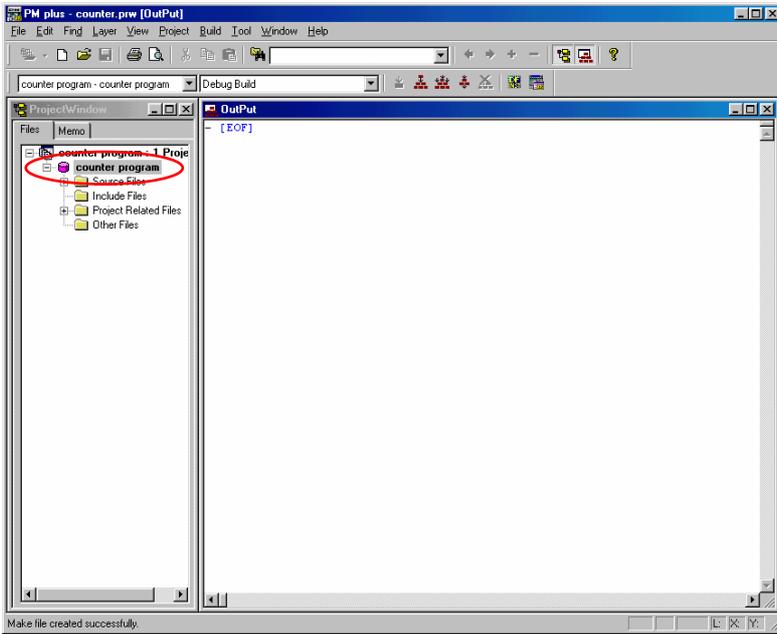
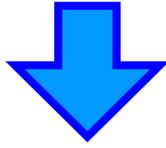
Click the Next button.



Check the set contents.



If the set contents are correct, click the Finish button.



The project "counter program" is registered in PM plus.

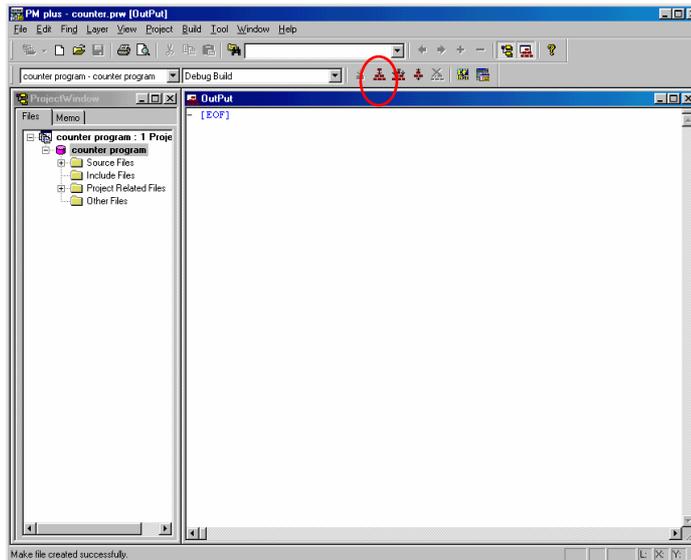
This completes the creation of the workspace.

You can add other source files at any time.

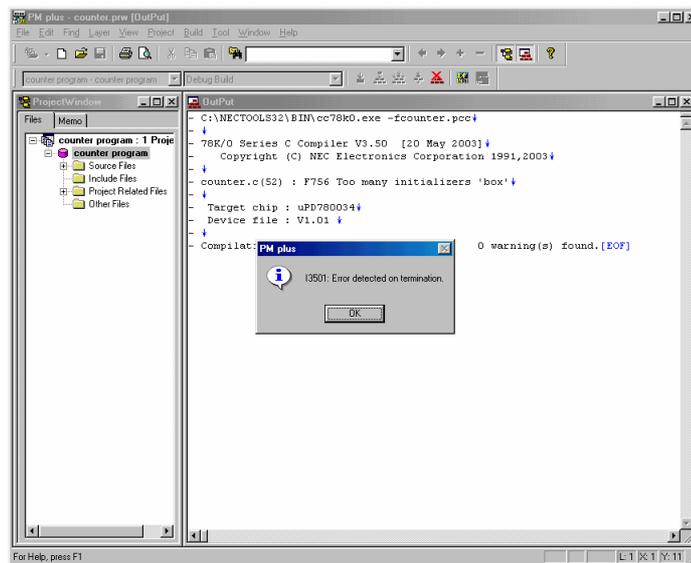
Editing the Source and Creating an Executable Program (1)

Next, you will build the project.

In PM plus, click the Build button  or, from the menus, select Build -> Build.



The build process starts.



An error is detected in the source program, and an error message is displayed on the screen.

Click OK button.

Now, let's correct the error.

To correct the error, you will edit the source.

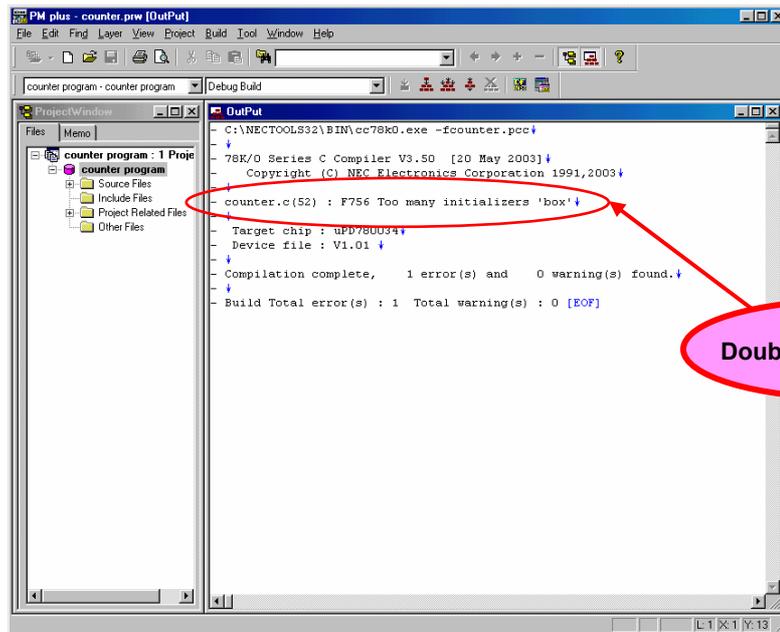
A detailed error message is displayed in the Output window.

Double-click the line where the error is indicated, "counter.c(52):F756 Too many initializers'box".

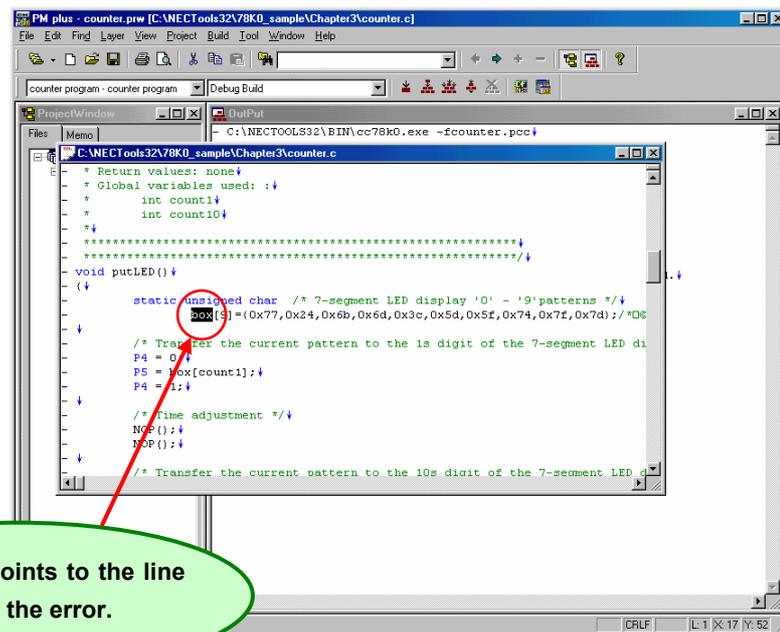
How to read the error message

"counter.c (52):F756 Too many initializers'box"

The source file name , line number and error message (cause of the error) are displayed.

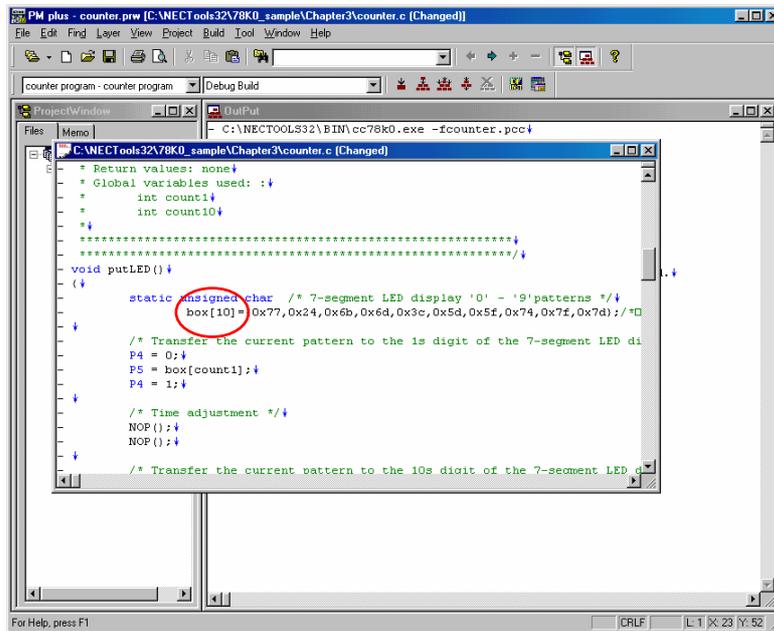


Editor opens.



The cursor points to the line that contains the error.

In line 52, the number of initializers (10) for the "box" array is larger than the specified length of the array (9). Change "box[9]" to "box[10]".



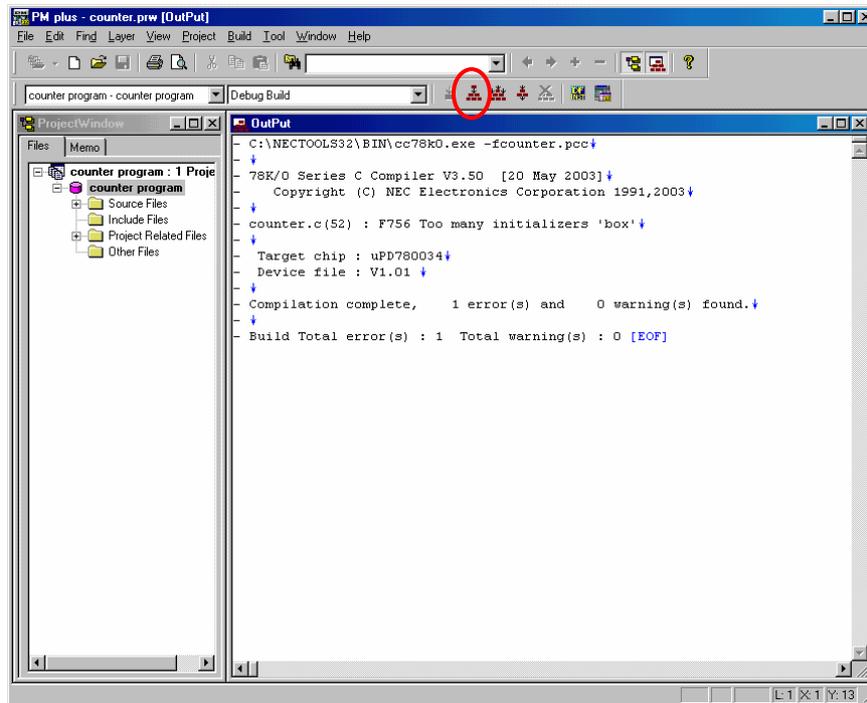
```
PM plus - counter.prw [C:\NECTools32\78K0_sample\Chapter3\counter.c (Changed)]
counter program - counter program | Debug Build
C:\NECTools32\78K0_sample\Chapter3\counter.c (Changed)
* Return values: none
* Global Variables Used: :
int count;
int count10;
*****
void putLED()
{
    static unsigned char /* 7-segment LED display '0' - '9' patterns */
    box[9] = {0x77, 0x24, 0x6b, 0x6d, 0x3c, 0x5d, 0x5f, 0x74, 0x7e, 0x7d}; /* 10
    /* Transfer the current pattern to the 1s digit of the 7-segment LED di
    P4 = 0;
    P5 = box[count1];
    P4 = 1;
    /* Time adjustment */
    NOP();
    NOP();
    /* Transfer the current pattern to the 10s digit of the 7-segment LED d
```

This completes the source editing.

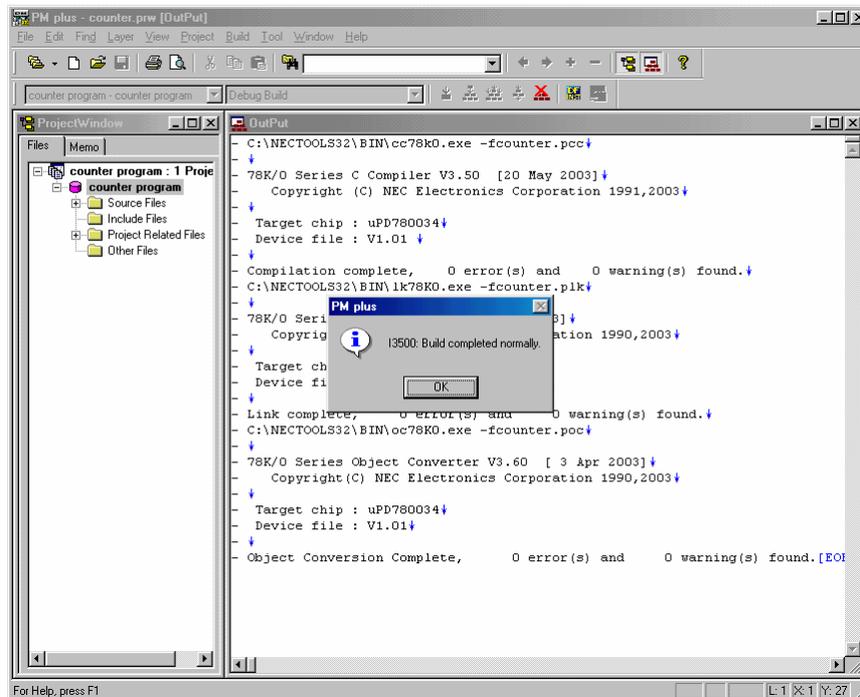
You corrected an error, so you must now rebuild the project.

In PM plus, click the Build button  or, from the menus, select Build -> Build.

When you use PM plus editor, the edited source contents are saved automatically during build.



The project is built.



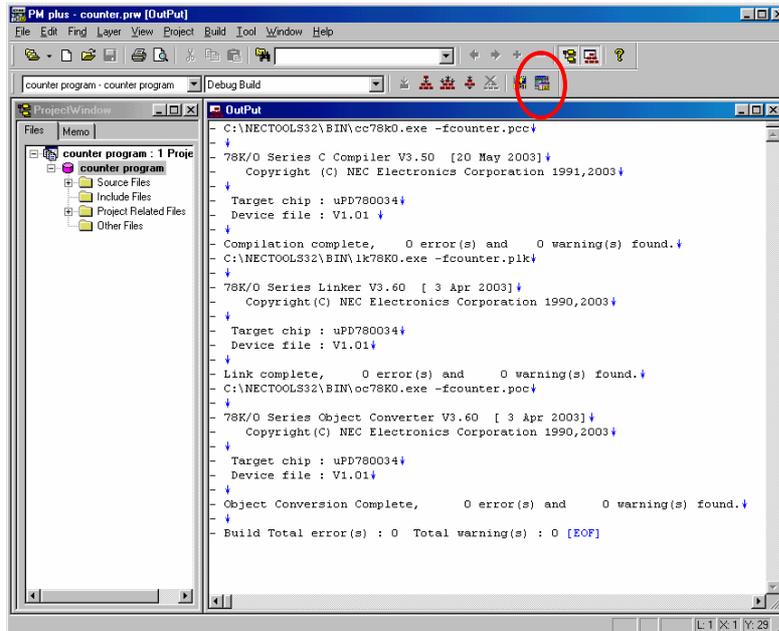
The build completes normally, and an executable program is created.

The default file name of the executable program is "source name registered first".lmc.

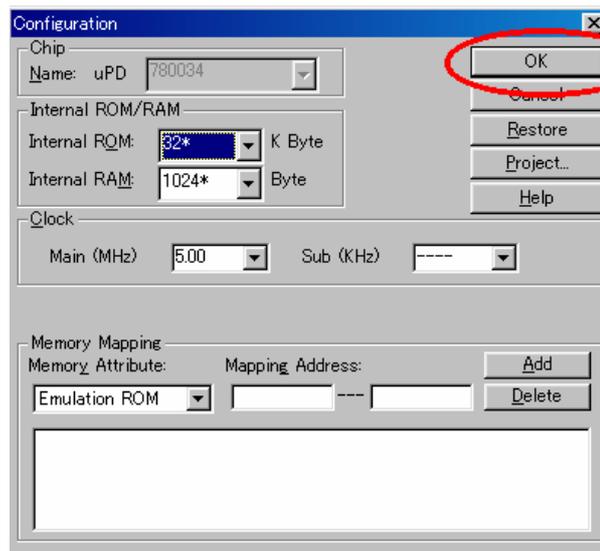
Running the System Simulator (SM78Kxx)

Start up the SM78K0.

Click the Debug button  or select [Build (B)] -> [Debug (D)] from the menu.



The Configuration dialog box opens.

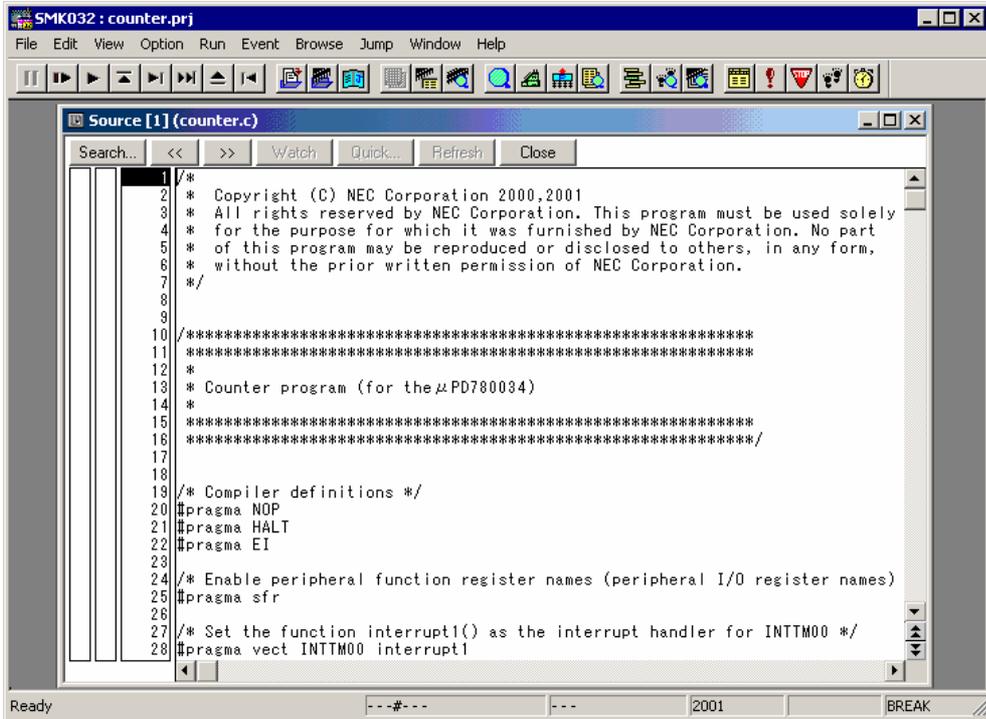


Settings such as memory mapping and microprocessor clock speed are made using this dialog box. However, since the counter program in this chapter uses the internal RAM and ROM of the μ PD78xxxx, it is not necessary to change any settings here.

Click OK button.



The Main window of the SM78Kxx opens.



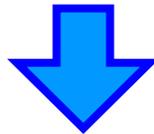
Setting Up the Input/Output Panel

Before running the counter program, you must set up the button and 7-segment LED display used by the program.

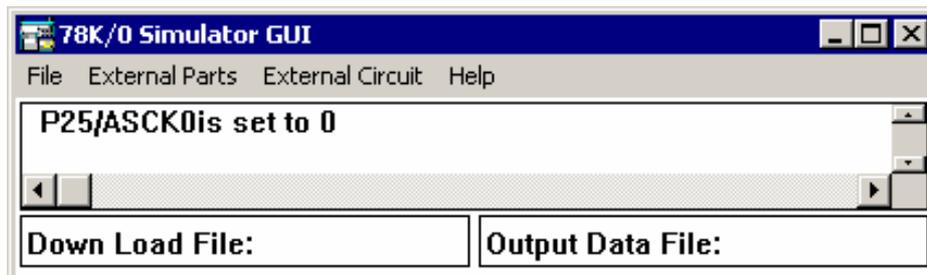
First, you will set up the button in the Input/Output Panel.

You can open the Input/Output Panel window from the 78Kxx simulator GUI window.

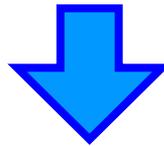
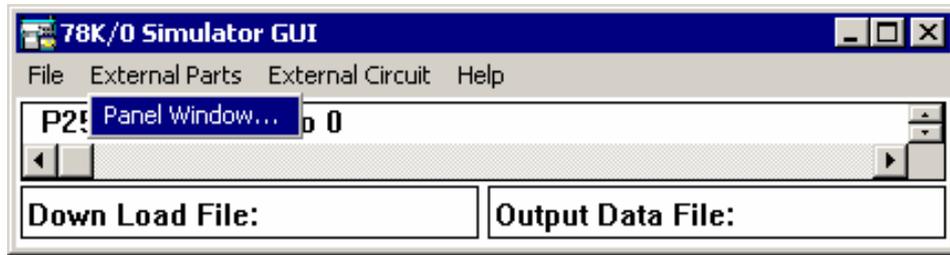
Click on "78Kxx simulator GUI" in the task bar to open the 78Kxx simulator GUI window.



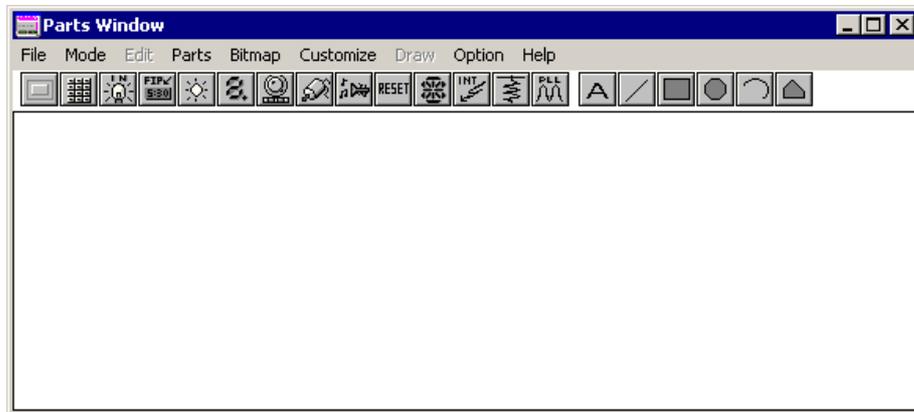
The 78Kxx simulator GUI window opens.



From the menus in the 78Kxx simulator GUI window, select External Parts -> Input/Output Panel...



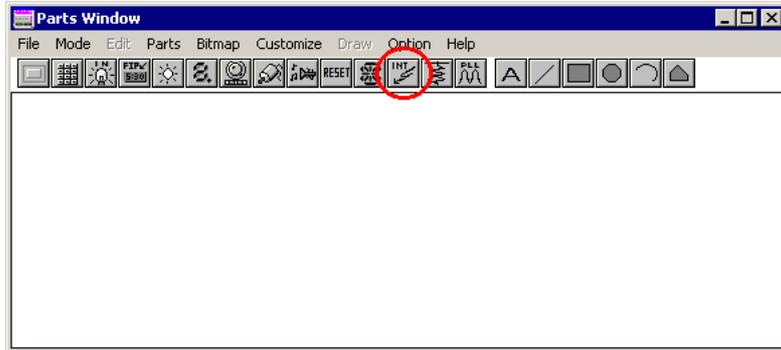
The Input/Output Panel window opens.



Next, you will set the [internal interrupt button](#).

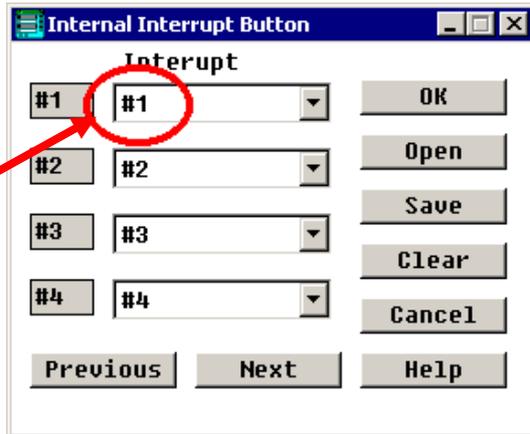
First, in the Input/Output Panel, click the Internal Interrupt button  or, from the menus, select Connection ->Internal Interrupt Button...

The Internal Interrupt Button Settings dialog box will open. Change the name of the first interrupt from "#1" to "INTTM00" for the 78K0, "INWT" for the 78K0S and "INTC00" for the 78K4.

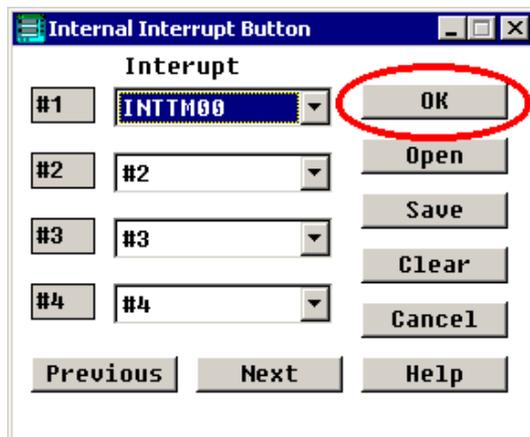


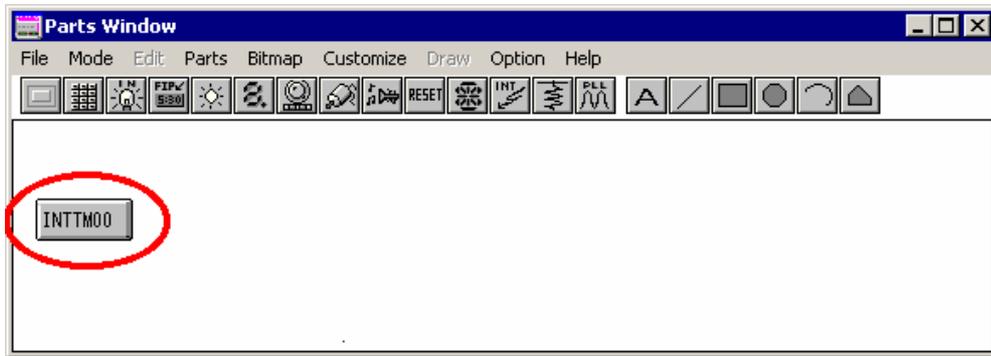
The Internal Interrupt Button Settings dialog box opens.

Change "# 1" to "INTTM00".



Click OK button.





The internal interrupt button is displayed at the top of the Input/Output Panel.

What is an internal interrupt button ?

The internal interrupt button is one of the debugging functions of the SM78Kxx. When a user clicks the button, it can generate a virtual internal interrupt that corresponds to the internal interrupt generated by a CPU peripheral function, such as a timer.

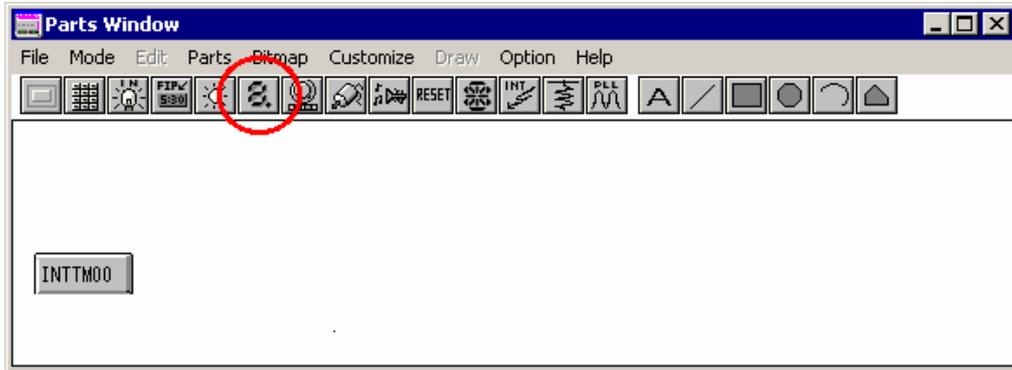
It is sometimes difficult to create the conditions required to generate an internal interrupt during debugging. You can, however, easily generate interrupts using this function to test the interrupt handling routines.

Next, you will setup the [7-segment LED](#) terminals.

Each bit of I/O ports P1 and P3 is connected to a corresponding 7-segment LED display terminal.

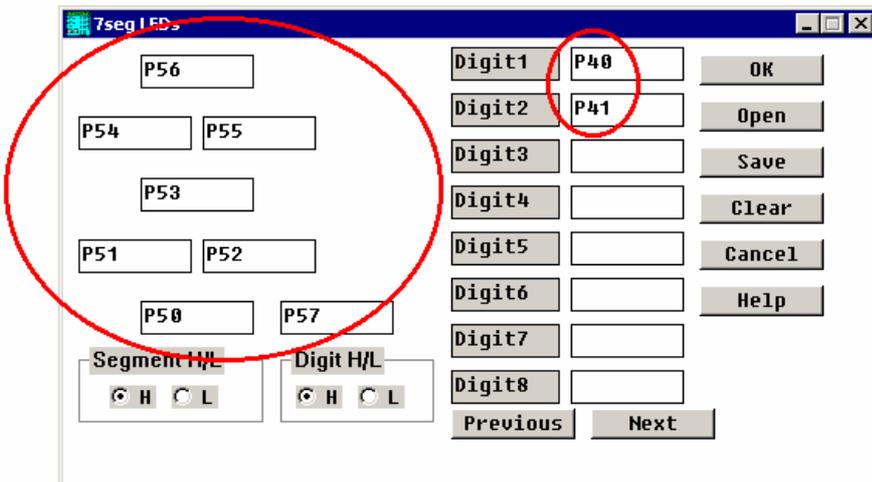
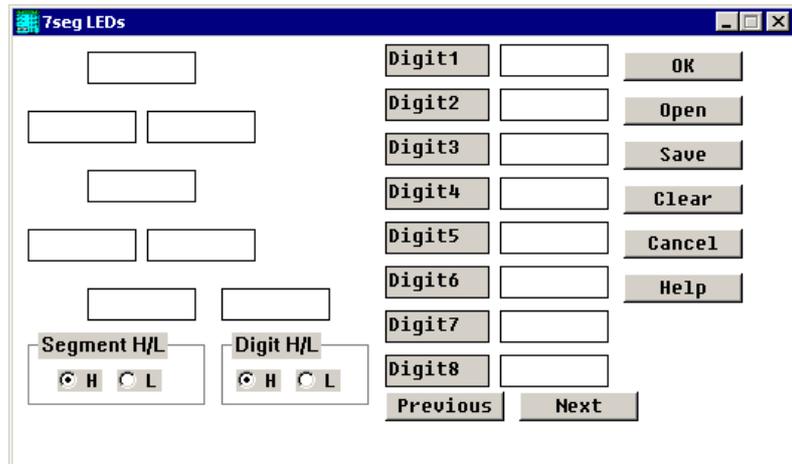
In the Input/Output Panel window, click the 7-segment LED Display Terminal Setting button  or, from the menus, select Connection->7-segment LED...

When the 7-segment LED Terminal Setting dialog opens, enter the connection information according to the chart below.



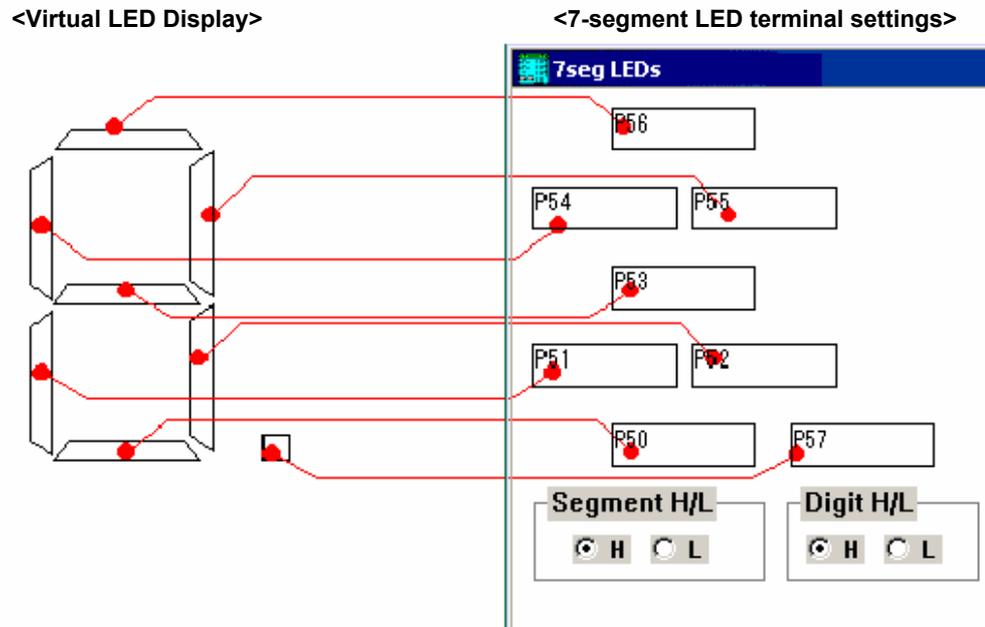
The 7-segment LED Terminal Setting dialog opens.

Enter the connection information as follows.

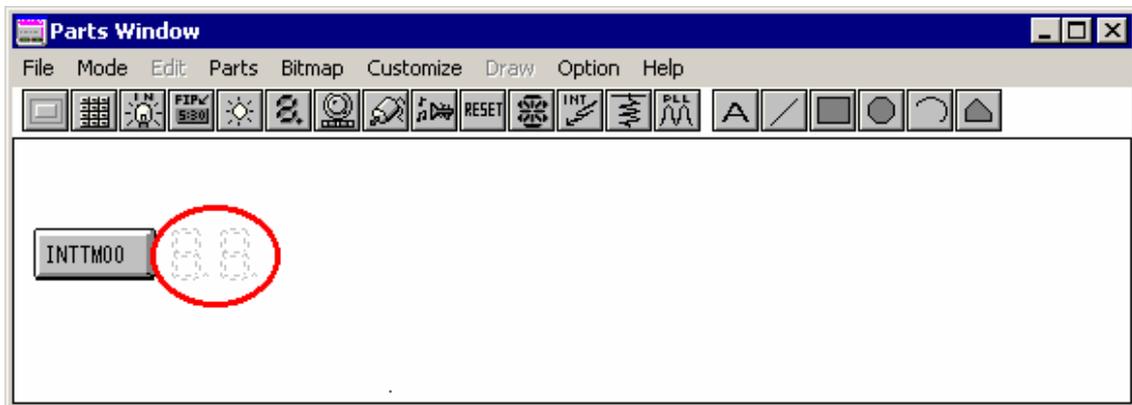


When you complete the settings, click OK button.

The following shows how the virtual LED display segments correspond to the segment signal setting entries in the 7-segment LED Terminal Setting dialog box. For the 78K0S, the settings use P1x, rather than P5x.



The virtual LED display appears inside the Input/Output Panel window.



This completes the setup of the button and 7-segment LED display.

What is a 7-segment LED display?

The 7-segment LED display described here is one of the preconfigured external parts that comes with the SM78Kxx. External parts are used to build up a virtual target system. This part is mainly used to display numerical values and is connected to an I/O port when used.

The signals used to operate the display are a set of segment signals, which are common to all digits, and an independent digit signal for each digit.

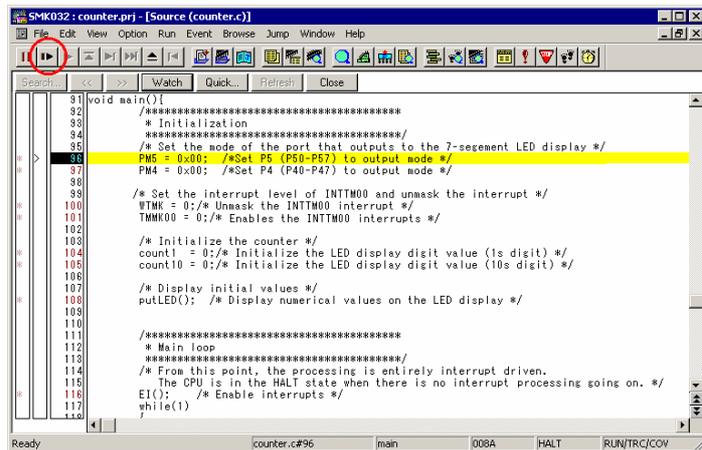
For further details, refer to the SM78K Series System Simulator Ver.2.52 Operation User's Manual (U16768E).

Executing the Program (1)

Now, you will run the [counter program](#).

In the SM78Kxx Main window, click the Restart button , or from the menus, select Run -> Restart.

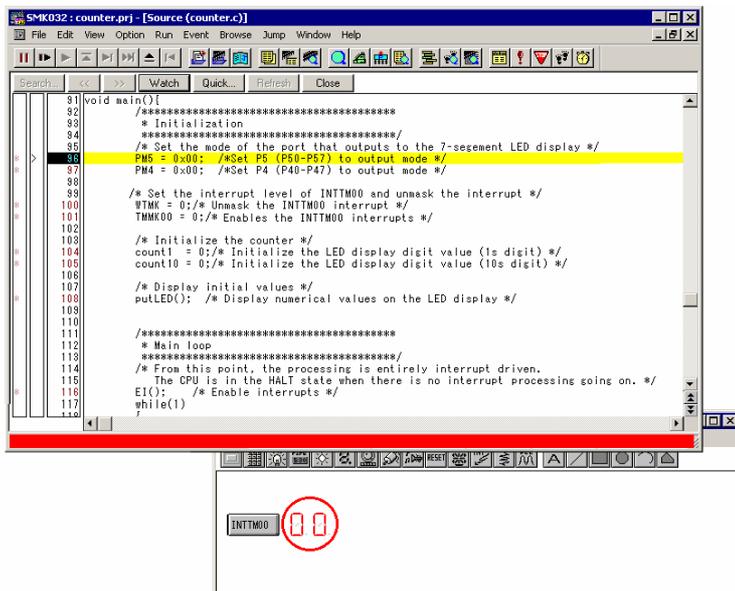
This operation resets the emulation CPU and starts program execution.



```

SM78K032 : counter.prj - [Source (counter.c)]
File Edit View Option Run Event Browse Jump Window Help
Search: << >> Watch Quick... Refresh Close
81 void main()
82 /* Initialization
83 * Initialization
84 * Set the mode of the port that outputs to the 7-segment LED display */
85 PM5 = 0x00; /*Set P5 (P50-P57) to output mode */
86 PM4 = 0x00; /*Set P4 (P40-P47) to output mode */
87
88 /* Set the interrupt level of INTTM00 and unmask the interrupt */
89 INTMK = 0; /* Unmask the INTTM00 interrupt */
90 INTMK00 = 0; /* Enables the INTTM00 interrupts */
91
92 /* Initialize the counter */
93 count1 = 0; /* Initialize the LED display digit value (1s digit) */
94 count10 = 0; /* Initialize the LED display digit value (10s digit) */
95
96 /* Display initial values */
97 putLED(); /* Display numerical values on the LED display */
98
99 /* Main loop
100 * Main loop
101 * From this point, the processing is entirely interrupt driven.
102 * The CPU is in the HALT state when there is no interrupt processing going on. */
103 EI(); /* Enable interrupts */
104 while(1)
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
Ready counter.c:#96 main 008A HALT RUN/TRC/COV
  
```

Program execution starts.



```

SM78K032 : counter.prj - [Source (counter.c)]
File Edit View Option Run Event Browse Jump Window Help
81 void main()
82 /* Initialization
83 * Initialization
84 * Set the mode of the port that outputs to the 7-segment LED display */
85 PM5 = 0x00; /*Set P5 (P50-P57) to output mode */
86 PM4 = 0x00; /*Set P4 (P40-P47) to output mode */
87
88 /* Set the interrupt level of INTTM00 and unmask the interrupt */
89 INTMK = 0; /* Unmask the INTTM00 interrupt */
90 INTMK00 = 0; /* Enables the INTTM00 interrupts */
91
92 /* Initialize the counter */
93 count1 = 0; /* Initialize the LED display digit value (1s digit) */
94 count10 = 0; /* Initialize the LED display digit value (10s digit) */
95
96 /* Display initial values */
97 putLED(); /* Display numerical values on the LED display */
98
99 /* Main loop
100 * Main loop
101 * From this point, the processing is entirely interrupt driven.
102 * The CPU is in the HALT state when there is no interrupt processing going on. */
103 EI(); /* Enable interrupts */
104 while(1)
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
INTTM00 00
  
```

The color of the status bar changes to red during program execution. The LED display on the Input/Output Panel displays "00" and the system waits for input from the INTTM00 button. The INTWT button is used for the 78K0S and the INTC00 button is used for the 78K4.

Do the following if the system does not respond as described above:

1) If the LED segments do not light:

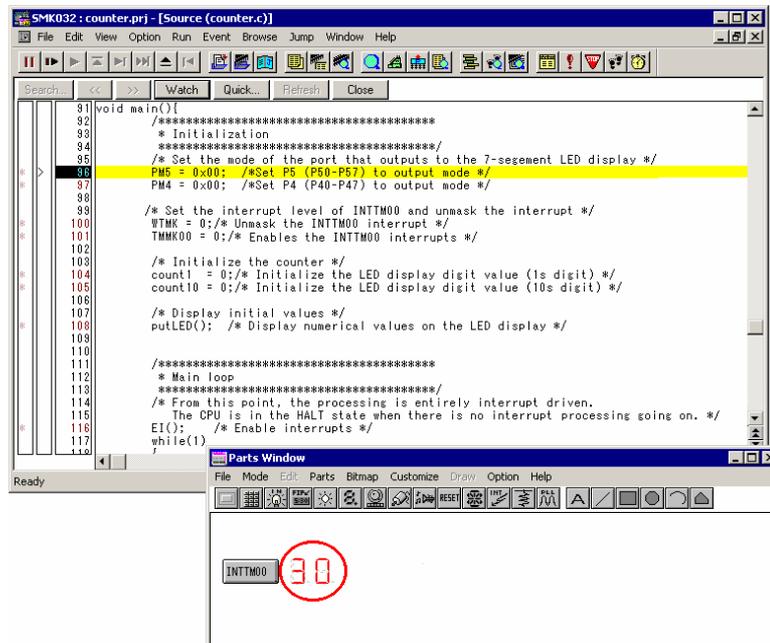
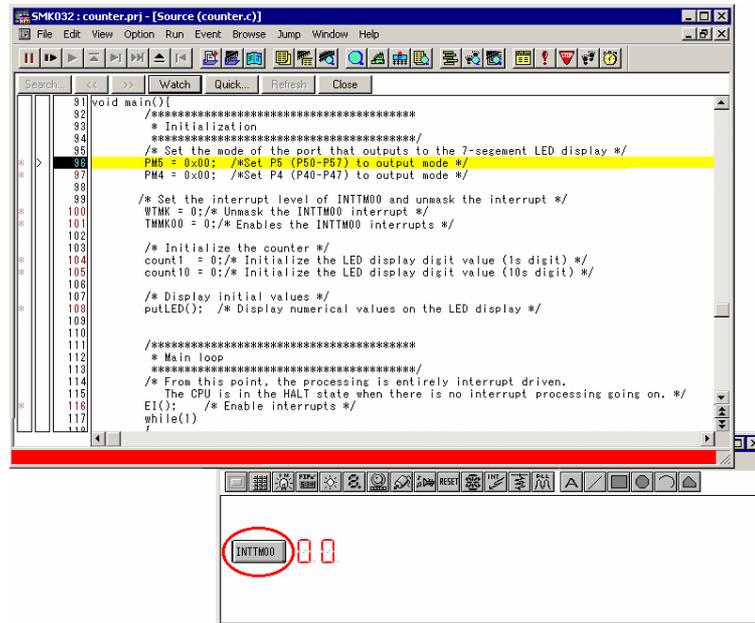
- Perform a [restart](#) operation again.

If a restart does not remedy the situation, [setup the 7-segment LED terminals](#) again.

2) If the LED display shows nothing but "00", [setup the 7-segment LED terminals](#) again.

Click the INTTM00 button several times.

Each click of the INTTM00 button should increment the counter by one, in accordance with the specifications.



Note that the count increases by 10 with each click of the INTTM00 button, while the 1s digit does not change. This shows that the program behavior does not meet the specifications.

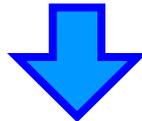
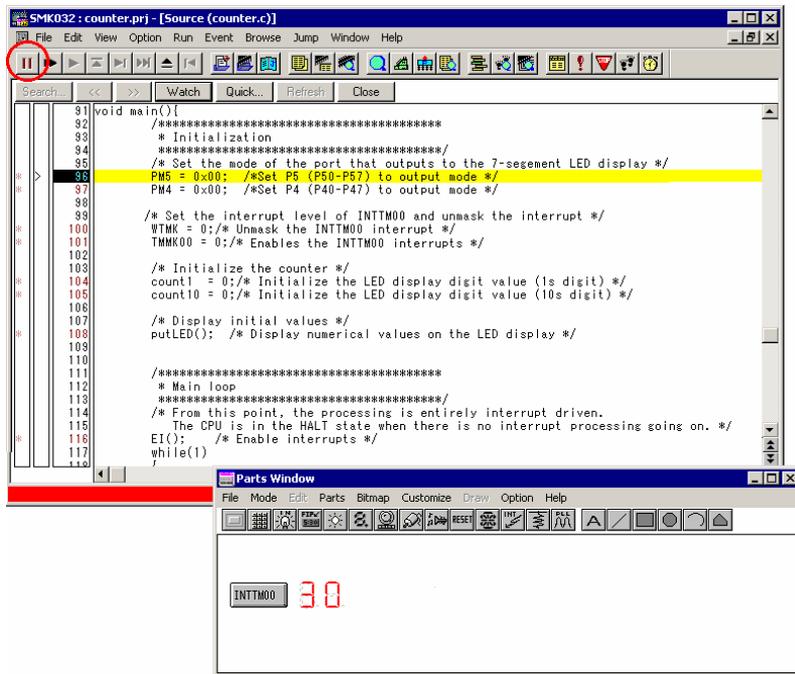
Do the following if the system does not respond as described above:

- 1) If nothing happens when you click the INTTM00 button:
 - [setup the internal interrupt button](#) again.
- 2) If the LED display does not behave as described above (counting in increments of 10),
 - [setup the 7-segment LED terminals](#) again.

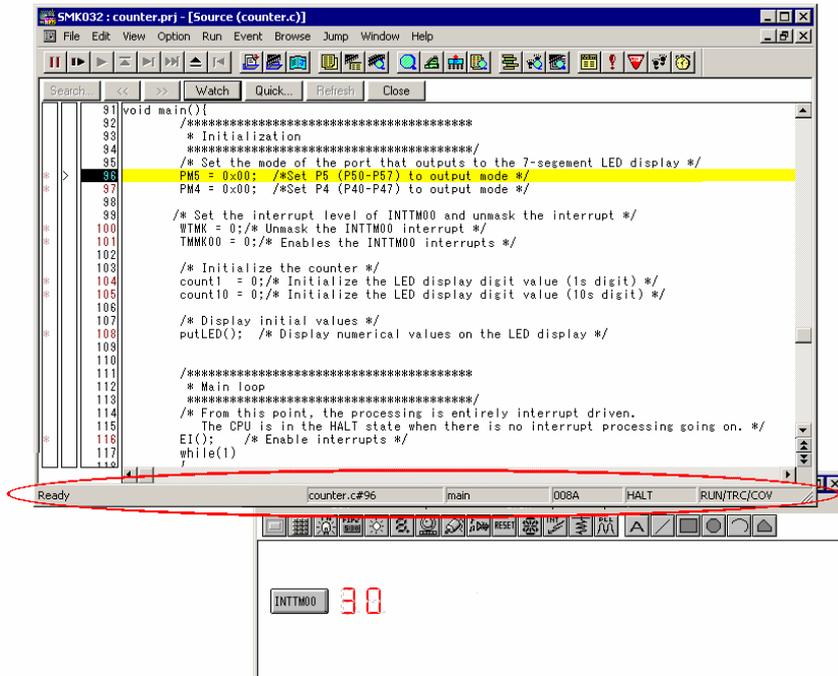
Since the counter is not counting up correctly, let's debug the program.

Stop the counter program.

In the SM78Kxx Main window, click the Stop button  , or from the menus, select Run -> Stop.



Program execution stops.



The status bar color returns to its original color when you stop the program.

Debugging

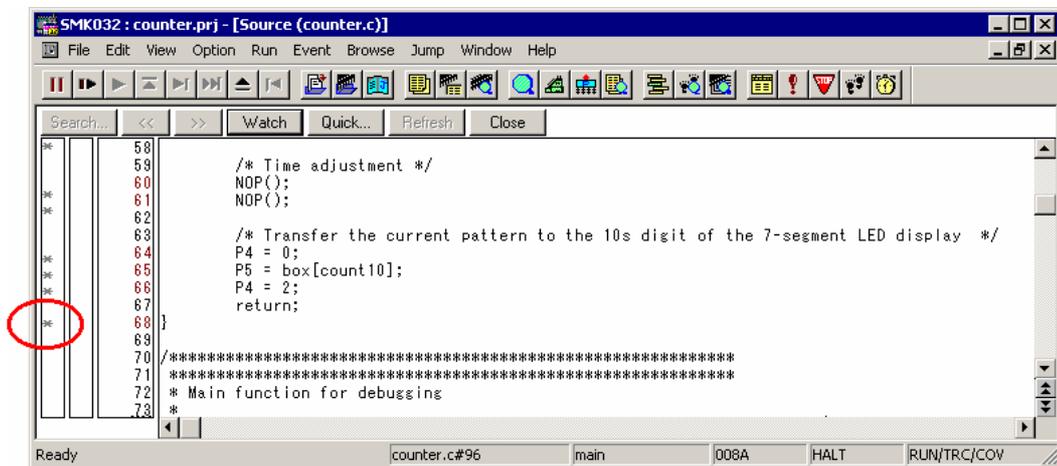
The variable count1 is associated with the 1s digit and count10 is associated with the 10s digit of the LED display.

➔ For details, refer to the section "[Counter Program Specifications](#)".

First, you will investigate what values the variables count1 and count10 take on when the LED display routine (putLED() function) is executed.

To do this, you must set a break point at line 68 for the 78K0 and 78K4, and line 69 for the 78K0S. Lines on which you can set a break point are indicated by "*" in the leftmost column.

Click the "*" on the 68th line for the 78K0 and 78K4, and the 69th line for the 78K0S.

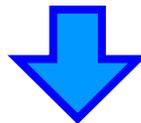
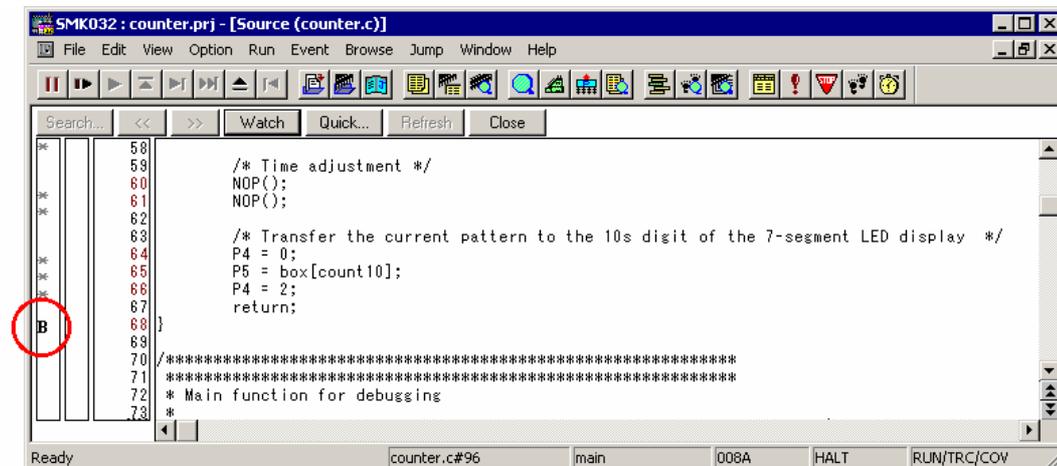


```

SMK032 : counter.prj - [Source (counter.c)]
File Edit View Option Run Event Browse Jump Window Help
Search... << >> Watch Quick... Refresh Close
58
59      /* Time adjustment */
60      NOP();
61      NOP();
62
63      /* Transfer the current pattern to the 10s digit of the 7-segment LED display */
64      P4 = 0;
65      P5 = box[count10];
66      P4 = 2;
67      return;
68 }
69
70 /*****
71 *****/
72 * Main function for debugging
73 *
Ready      counter.c#96      main      008A      HALT      RUN/TRC/COV

```

The "*" changes to "B"

```

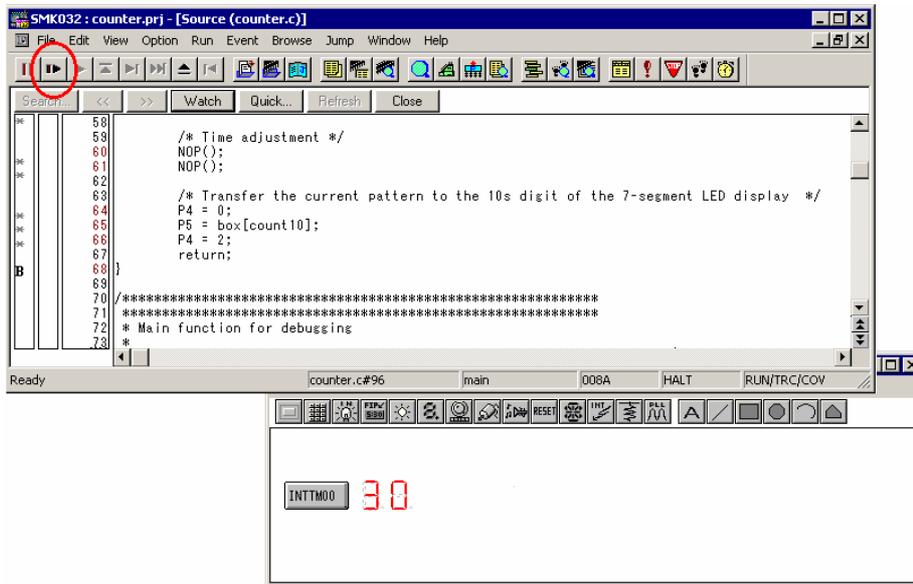
SMK032 : counter.prj - [Source (counter.c)]
File Edit View Option Run Event Browse Jump Window Help
Search... << >> Watch Quick... Refresh Close
58
59      /* Time adjustment */
60      NOP();
61      NOP();
62
63      /* Transfer the current pattern to the 10s digit of the 7-segment LED display */
64      P4 = 0;
65      P5 = box[count10];
66      P4 = 2;
67      return;
68 B }
69
70 /*****
71 *****/
72 * Main function for debugging
73 *
Ready      counter.c#96      main      008A      HALT      RUN/TRC/COV

```

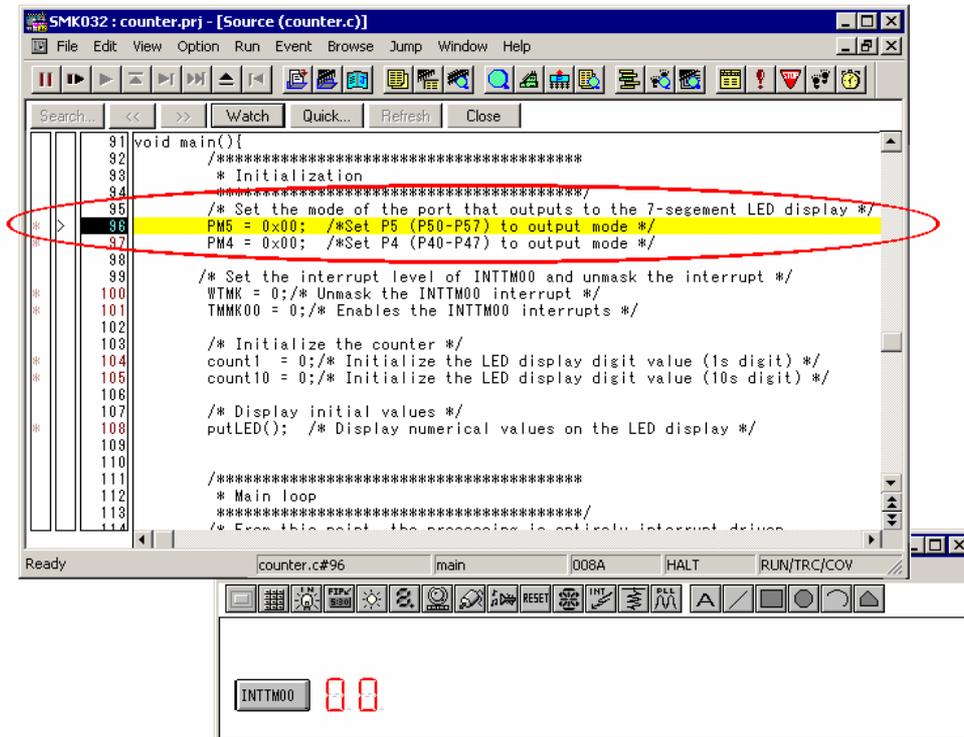
A break point is now set at line 68 (for the 78K0 and 78K4) or line 69 (for the 78K0S).

Next, you will execute the counter program.

In the SM78Kxx Main window, click the Restart button , or from the menus, select Run -> Restart.



Program execution starts.

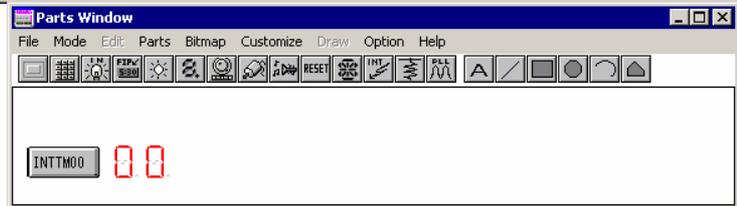
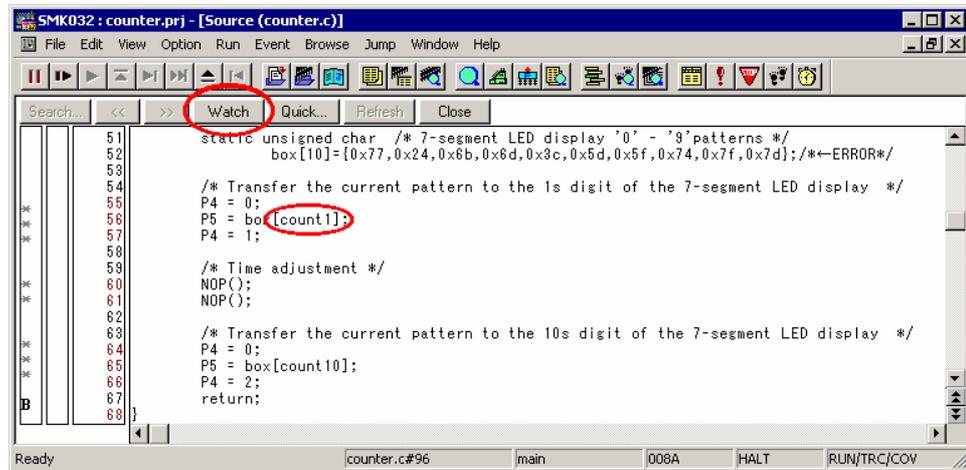


When program execution starts, the program stops almost immediately at the break point. The color of the line where the program stopped changes to yellow and a ">" symbol is displayed in the second column.

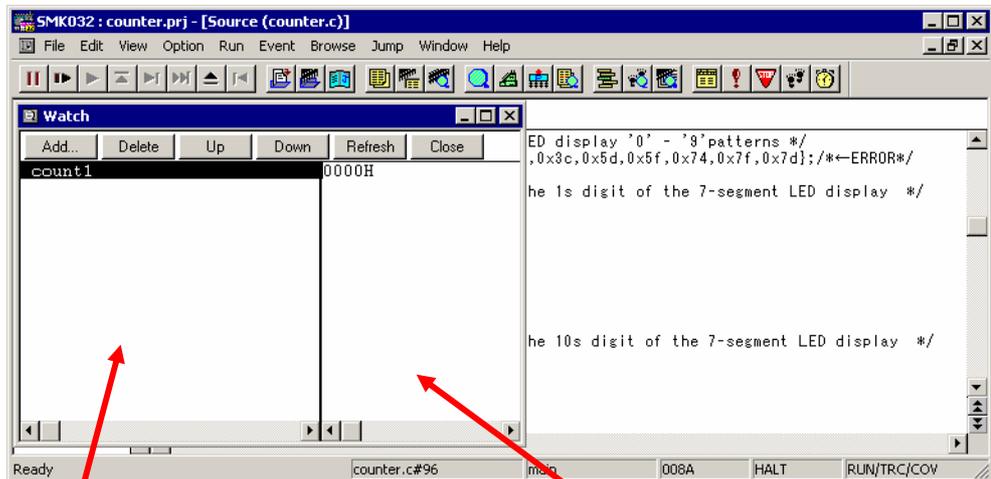
You can use the Watch window to view the value of a variable.

Let's open a Watch window and confirm the values of count1 and count10.

On line 56, double-click count1 to select it (text is highlighted) and click the Watch button.



A Watch window opens.



Symbol Name Display Area

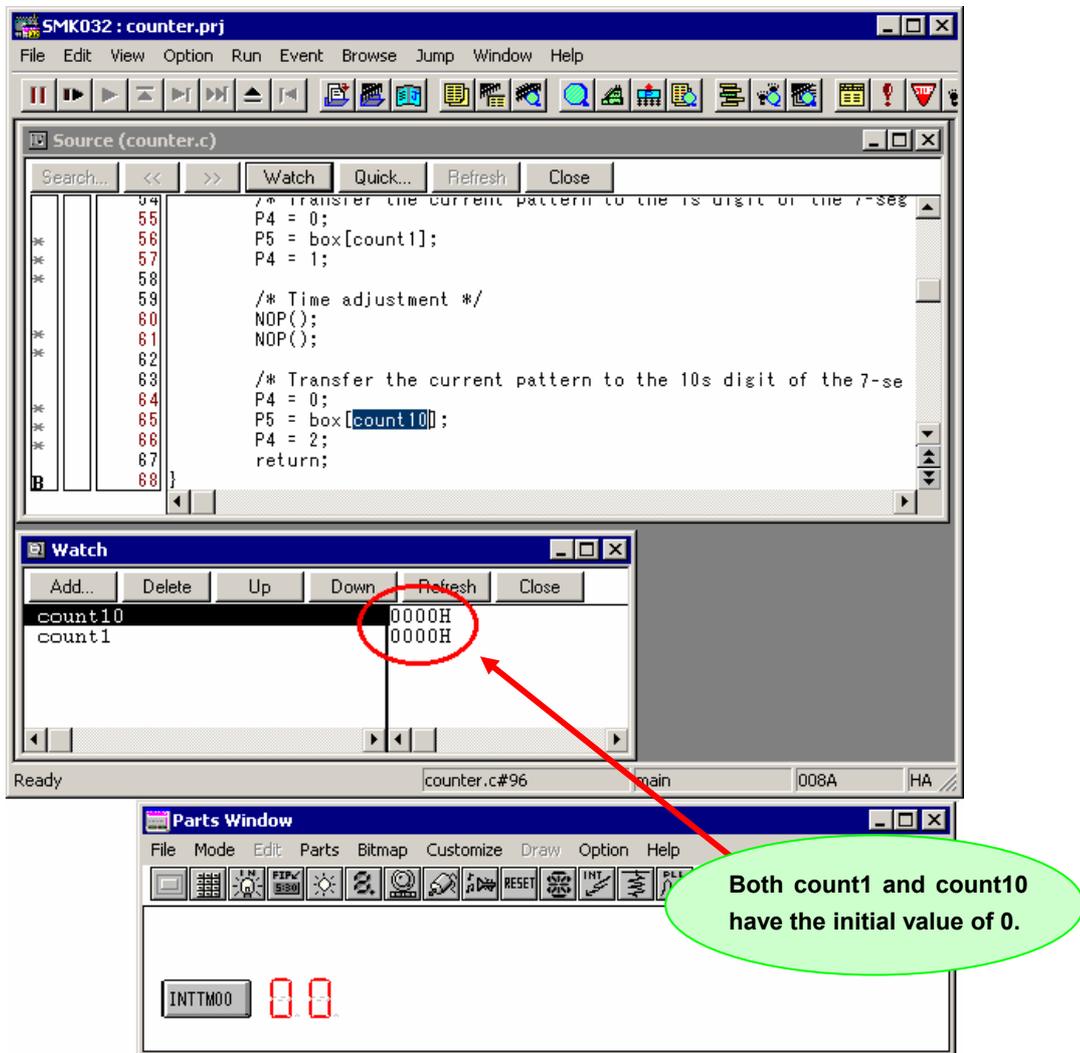
Data Value Display/Setting Area

The Watch window contains a Symbol Name Display Area and a Data Value Display/Setting Area.

➡ For further details, refer to the **SM78K Series System Simulator Ver.2.52 Operation User's Manual (U16768E)**.

Similarly, double-click count10 on line 65 in the Source Text window (Source window) to select it (text is highlighted) and click the Watch button.

The variable count10 is added to the Watch window.



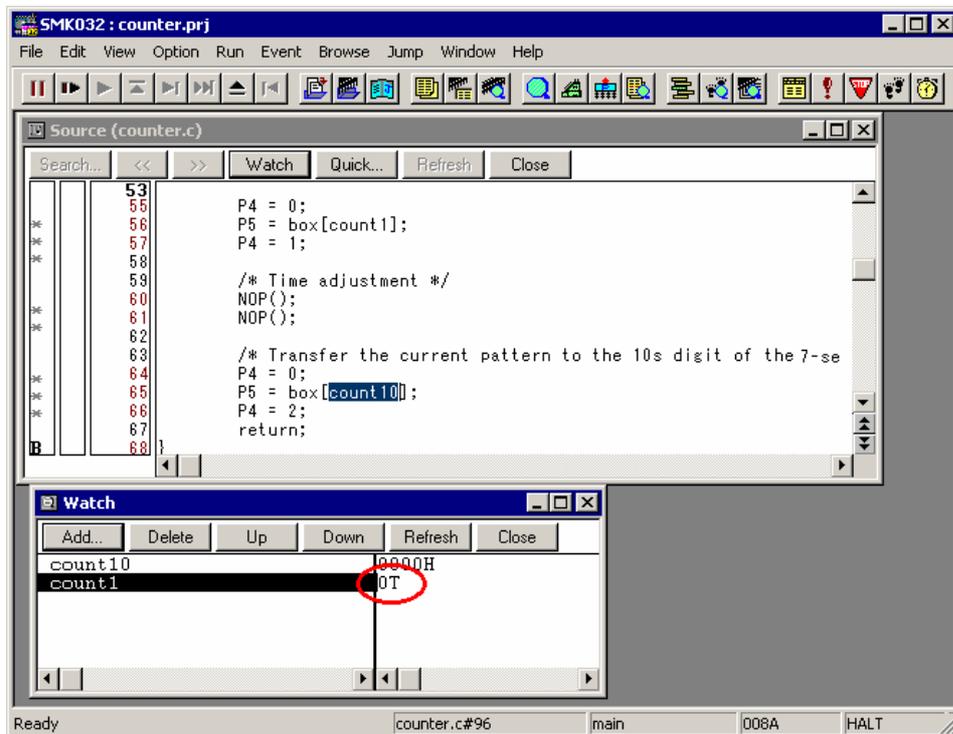
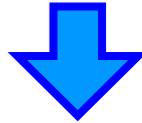
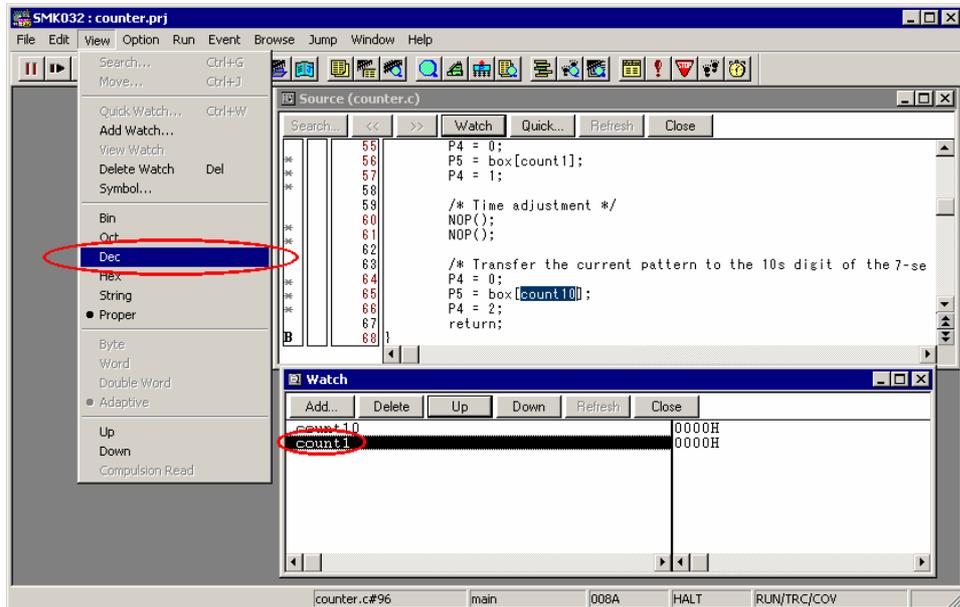
Looking at the Watch window you can confirm that both count1 and count10 have an initial value of 0.

Similarly, you can confirm that the LED display in the Input/Output Panel displays and initial value of 00.

The Data value display/setting area in the Watch window displays values in hexadecimal format (base 16), but this can be changed from the SM78Kxx View menu.

Let's change the display format of count1 to decimal format.

Click to select (highlight) count1 in the Watch window and, from the SM78Kxx Main window menus, select View -> Decimal.

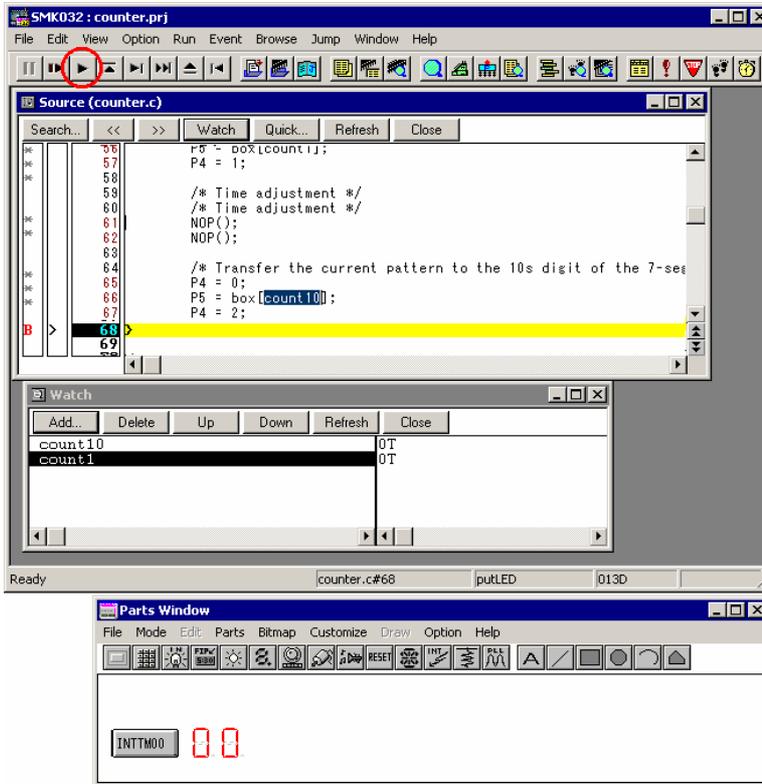


Similarly, change the format for cout10 to decimal.

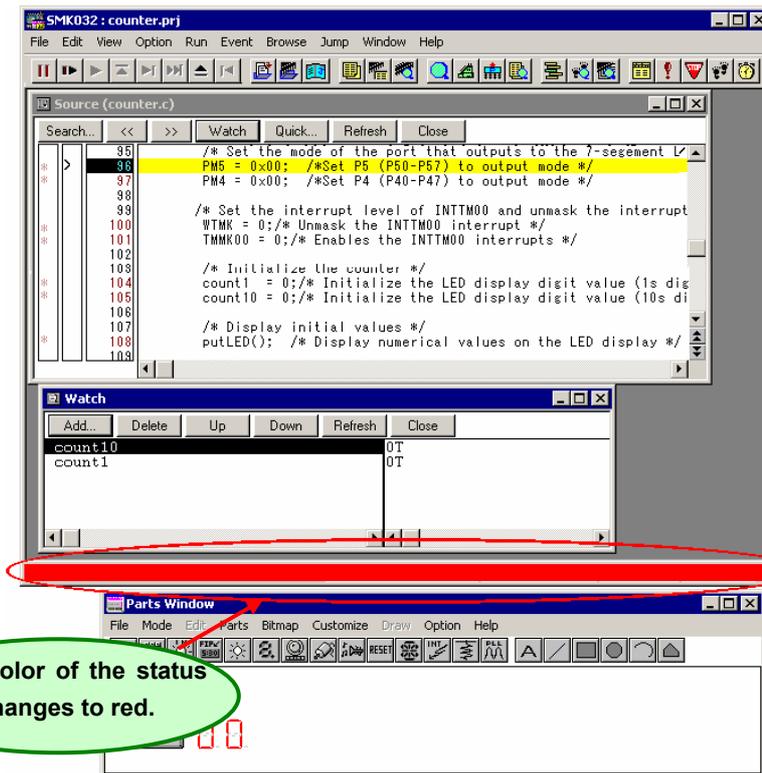
Next, you will determine if information is being correctly sent to the LED display when you click the INTTM00 button.

In the SM78Kxx Main window, click the Start button , or from the menus, select Run -> Go.

You use the Start button instead of the Restart button when you want to resume execution where you left off after the program was stopped.

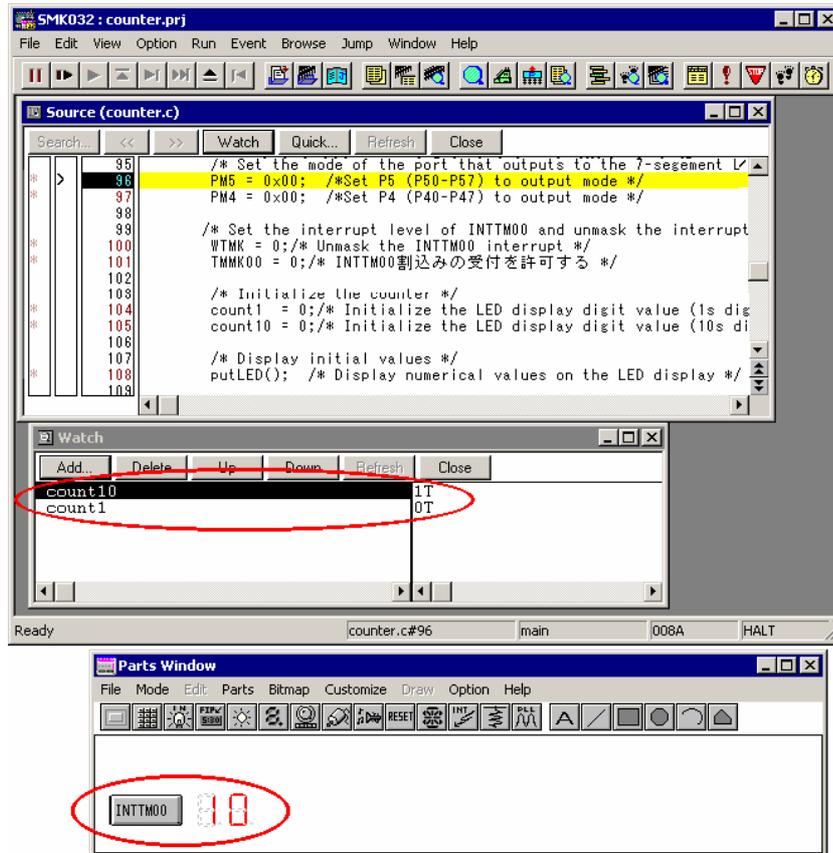


Program execution starts.



The color of the status bar changes to red.

Let's look at the execution of the LED display routine (putLED() function) when the INTTM00 button is clicked. Click the INTTM00 button and observe the contents of the Watch window.



When you click the INTTM00 button, the program stops at the break point.

Here, if the incrementation routine is working correctly, count10 should equal 0 and count1 should equal 1.

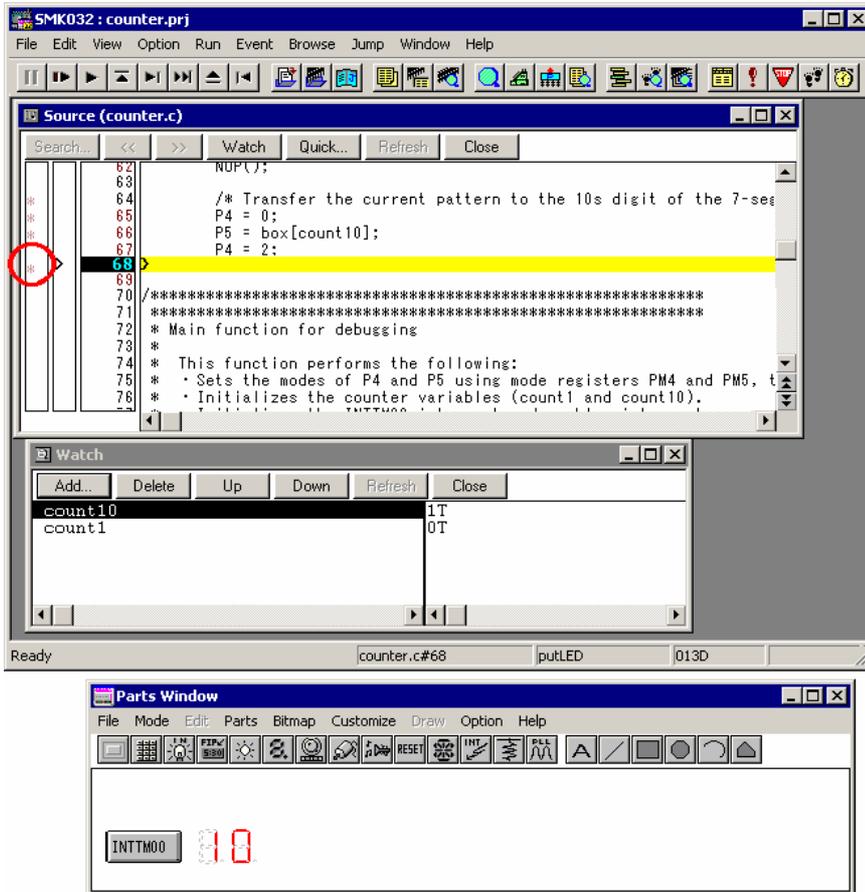
However, by observing the Watch window contents, you see that, since count10 equals 1 and count1 equals 0, the variables are not being set to the correct values.

The LED display in the Input/Output Panel, does, however, correctly reflect the variable values shown in the Watch window.

You can thereby assume that the routine that displays the values of count10 and count1 on the LED display is working correctly.

Having determined that the display routine is correct, next you will determine if the incrementation routine is correct.

Since the error does not appear to be in the LED display routine (putLED() function), you will remove the break point and look for the error in a different part of the program.



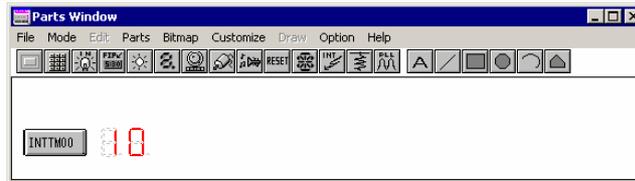
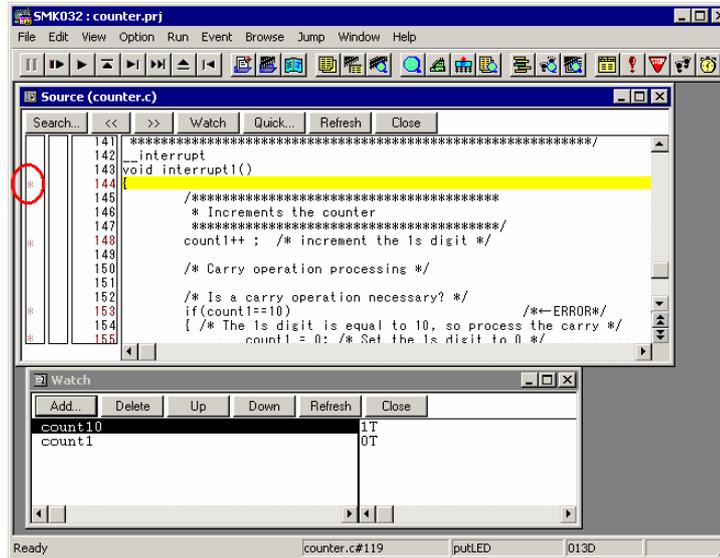
Click the "B" in the leftmost column. The "B" changes to "*" and the break point is removed.

Next, you will confirm the operation of the incrementation routine that executes when you click the INTTM00 button.

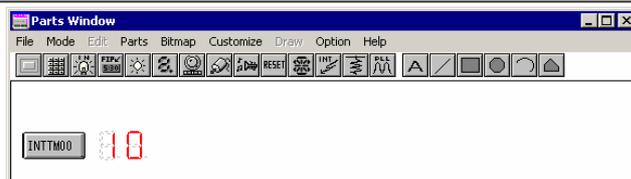
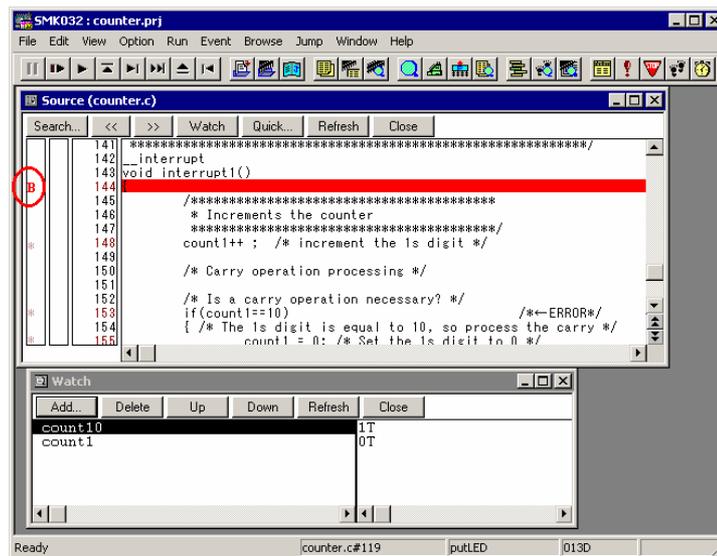
Let's investigate how the values of count1 and count10 change inside the internal interrupt handling routine (interrupt1() function) that executes when an internal interrupt occurs.

Set a break point at line 144.

Click the "*" in the leftmost column of line 144.



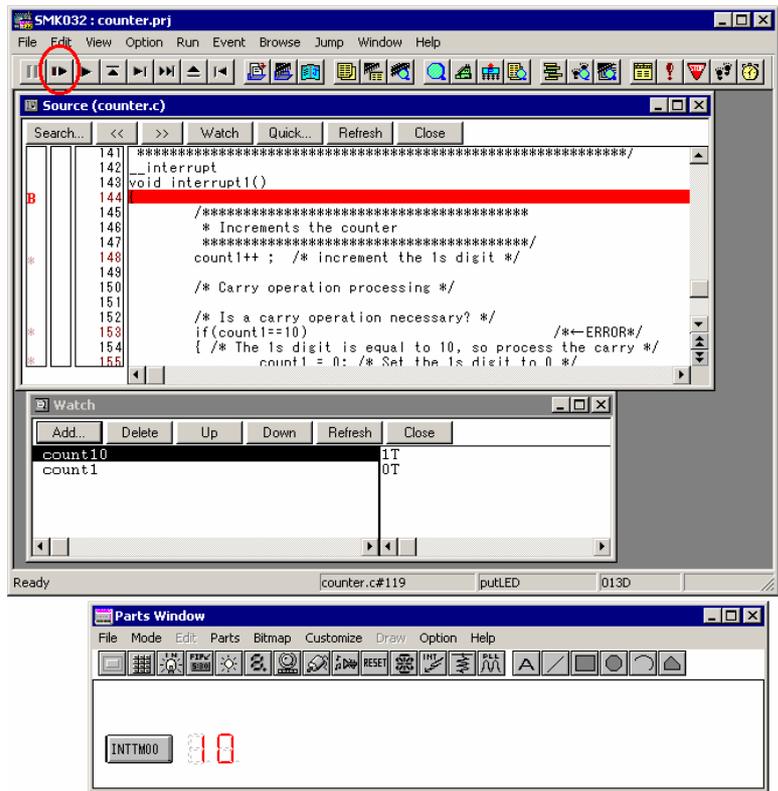
The "*" changes to "B".



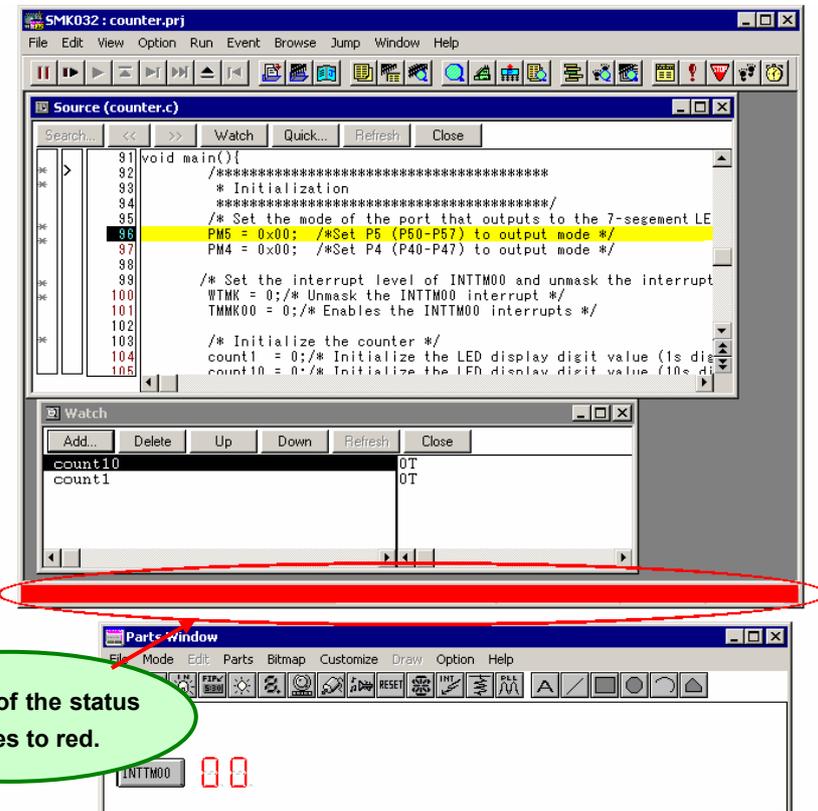
A break point is now set at line 144.

Next, perform a restart.

In the SM78Kxx Main window, click the Restart button  , or from the menus, select Run -> Restart.

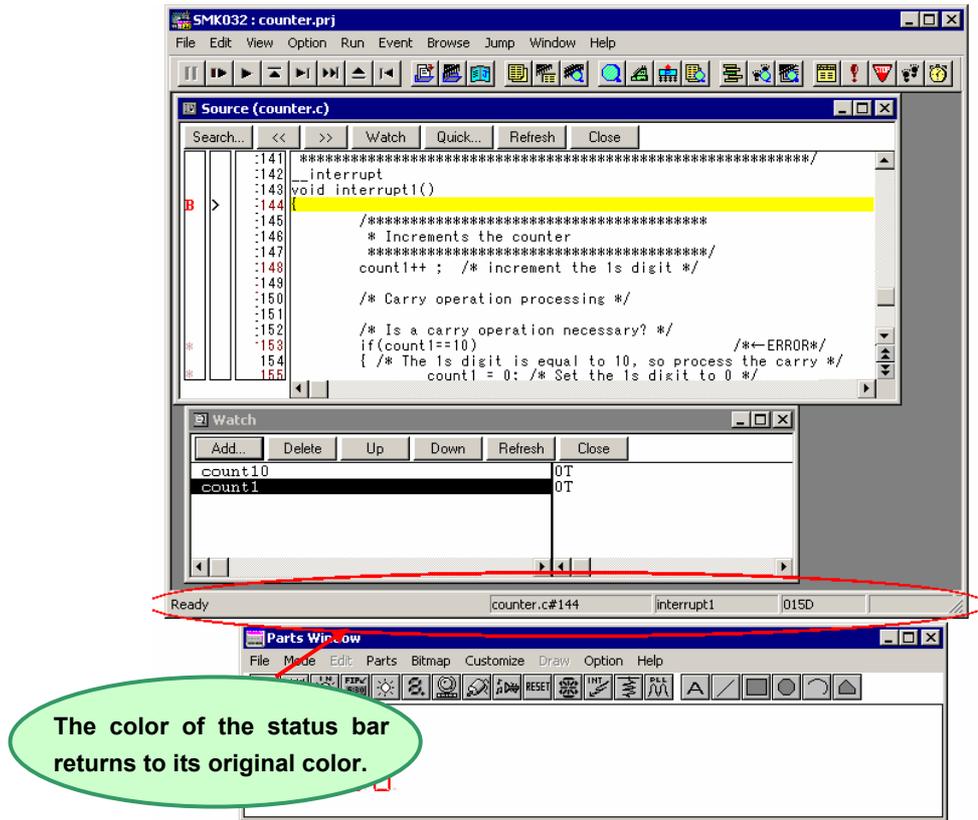


Program execution starts.



The color of the status bar changes to red.

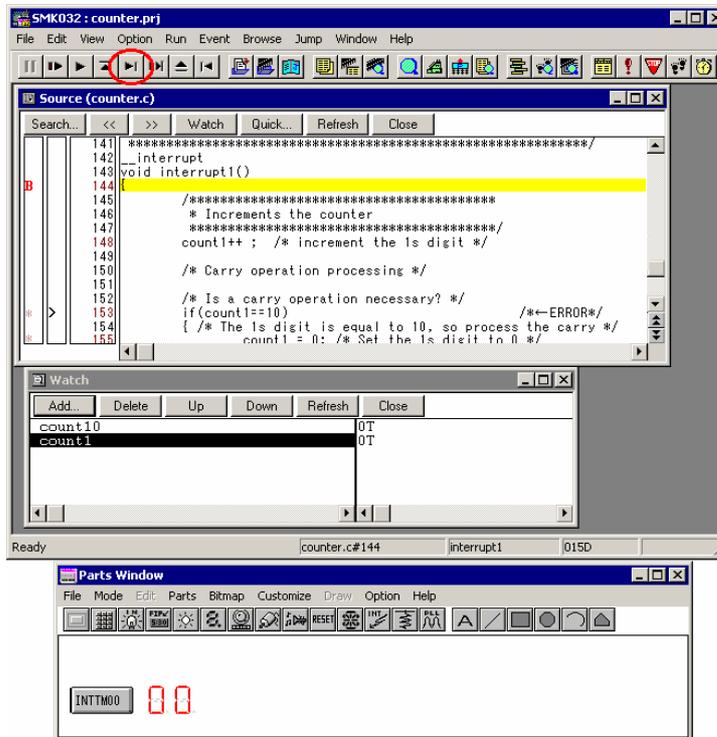
Click the INTTM00 button.



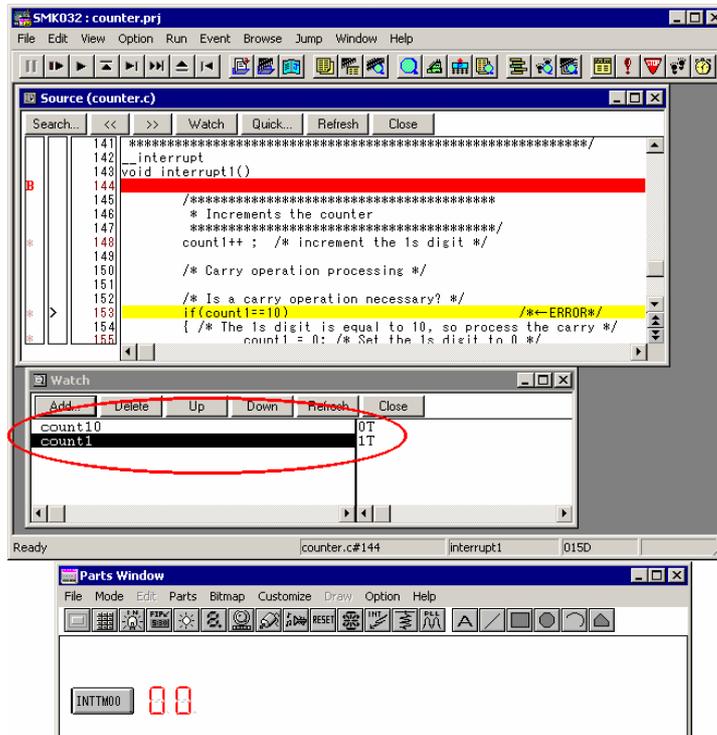
When you click the INTTM00 button, the program stops at the break point on line 144.

At this point, you can execute the program a line at a time (step) and observe the values of count1 and count10. Execute three single steps.

In the SM78Kxx Main window, click the Step-in button  three times, or from the menus, select Run -> Step-in three times.



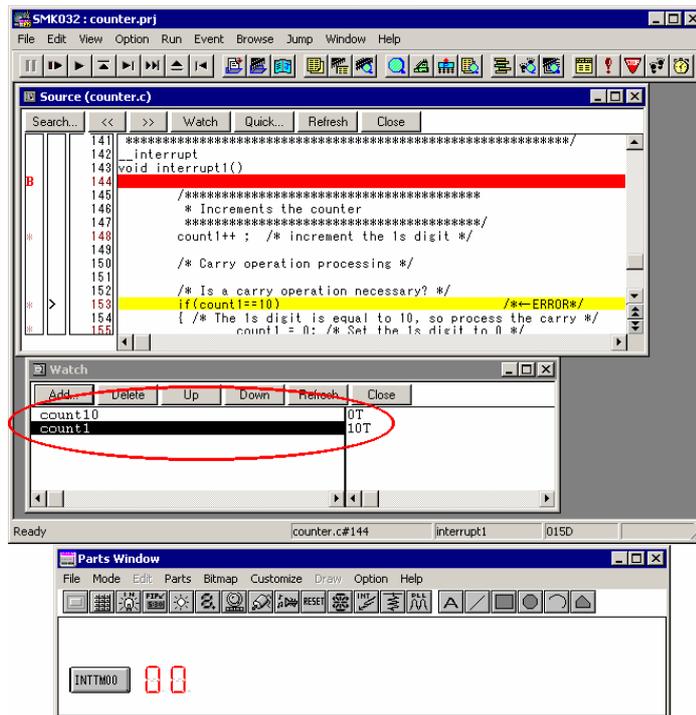
The program executes three steps.



Observe the value of count1 in the Watch window after step execution at line148 completes. Since count1 is 1, the routine is working correctly.

Continuing on, step to line153.

In the SM78Kxx Main window, click the Step-in button  , or from the menus, select Run->Step-in.



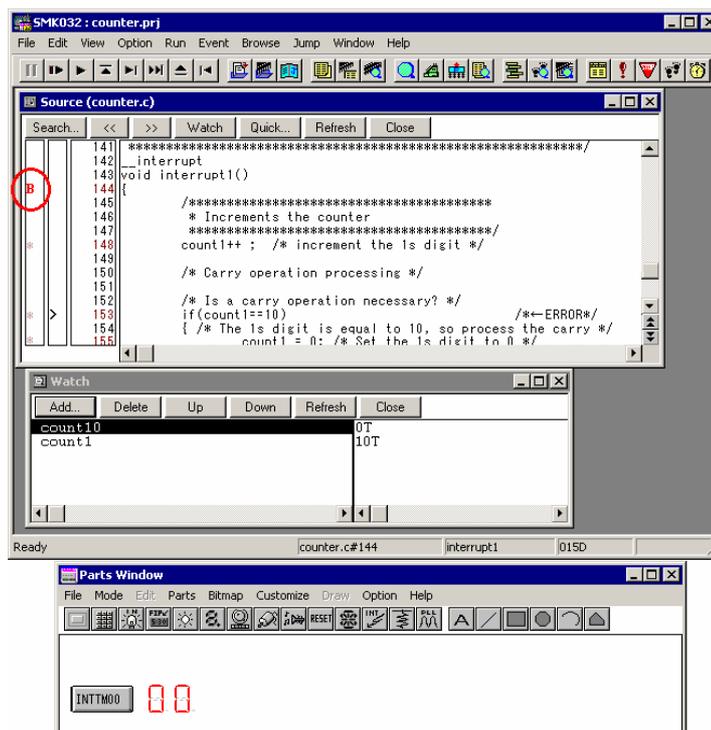
After step execution stops at line153, the value of count1 changes from 1 to 10.

Line 153 contains a conditional statement that tests if a carry to the next digit is required, so the value of count1 should not change due to this line. You can see from this that there is a problem with line 153.

Looking at line 153, in the if statement, a comparison of the value of count1 to the value 10, (count1==10), is required. However, in its place, there is an assignment statement, assigning a value of 10 to count1.

Now that you have found the location of the error, you can remove the break point.

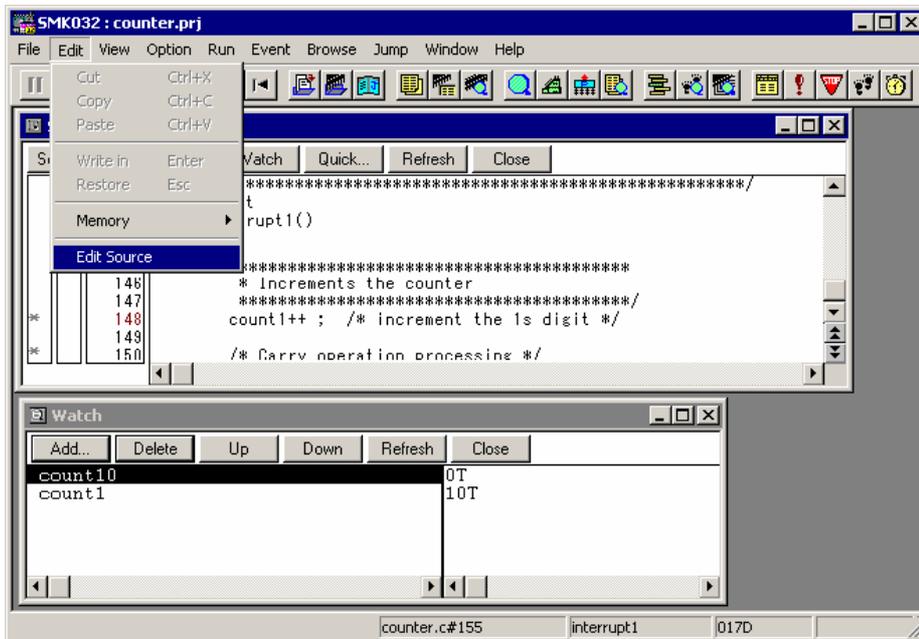
Click the "B" in the leftmost column. The "B" changes to "*" and the break point is removed.



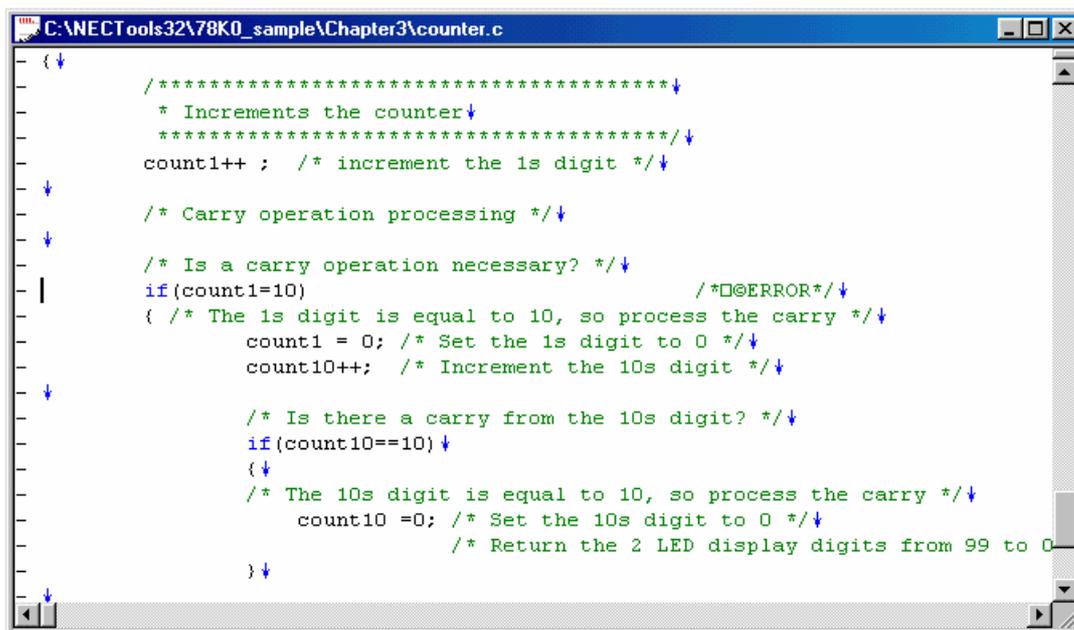
Editing the Source and Creating an Executable Program (2)

Next you will correct the error in the counter program.

In the SM78Kxx Main window, from the menus select Edit->Edit source.



Editor opens.



Correct line 153 by changing "count1=10" to "count1==10" in the if statement.

```

C:\NECTools32\78K0_sample\Chapter3\counter.c [Changed]
{
  /******↓
  * Increments the counter↓
  *****/↓
  count1++; /* increment the 1s digit */↓

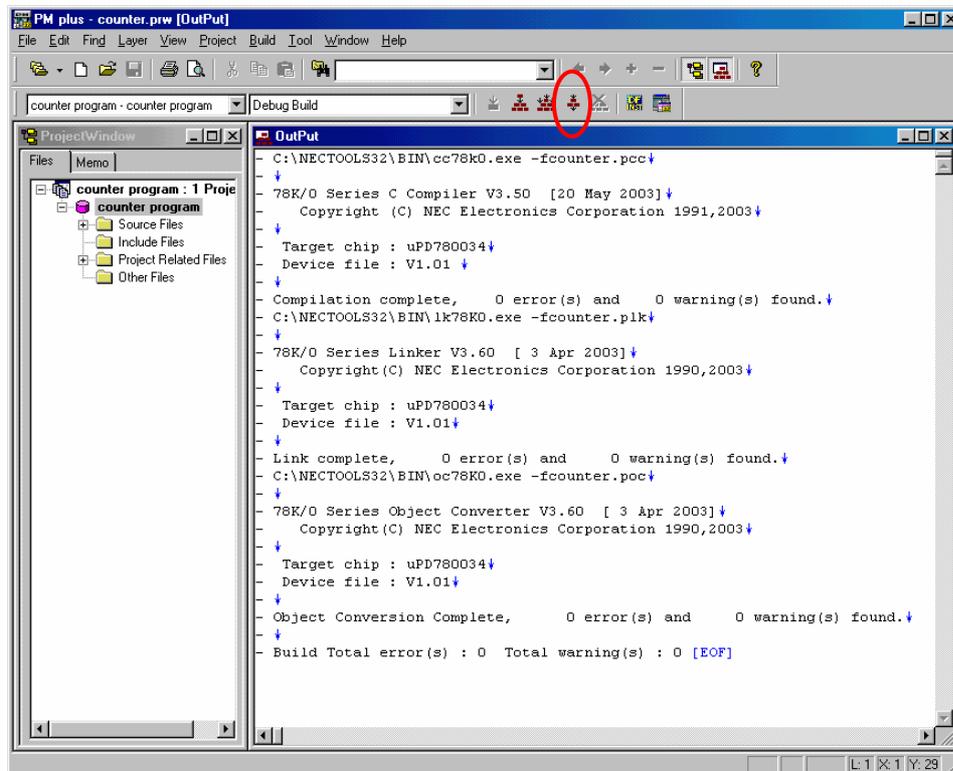
  /* Carry operation processing */↓

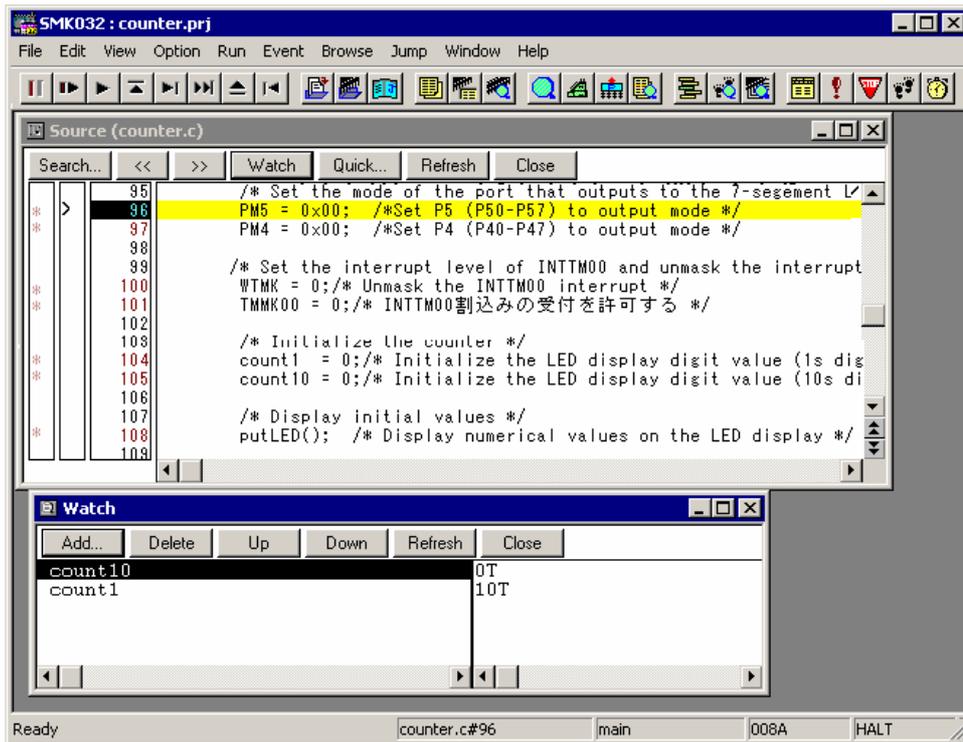
  /* Is a carry operation necessary? */↓
  if(count1=10) /*D@ERROR*/↓
  { /* The 1s digit is equal to 10, so process the carry */↓
    count1 = 0; /* Set the 1s digit to 0 */↓
    count10++; /* Increment the 10s digit */↓

    /* Is there a carry from the 10s digit? */↓
    if(count10=10)↓
    {
      /* The 10s digit is equal to 10, so process the carry */↓
      count10 =0; /* Set the 10s digit to 0 */↓
      /* Return the 2 LED display digits from 99 to 0
    }↓
  }↓
}
  
```

When you finish making the change, click the Build->Debug button  on PM plus or select [Build (B)] -> [Build->Debug (A)] from the menu.

When using PM plus editor, changes made to the source contents are automatically saved when you perform a build.



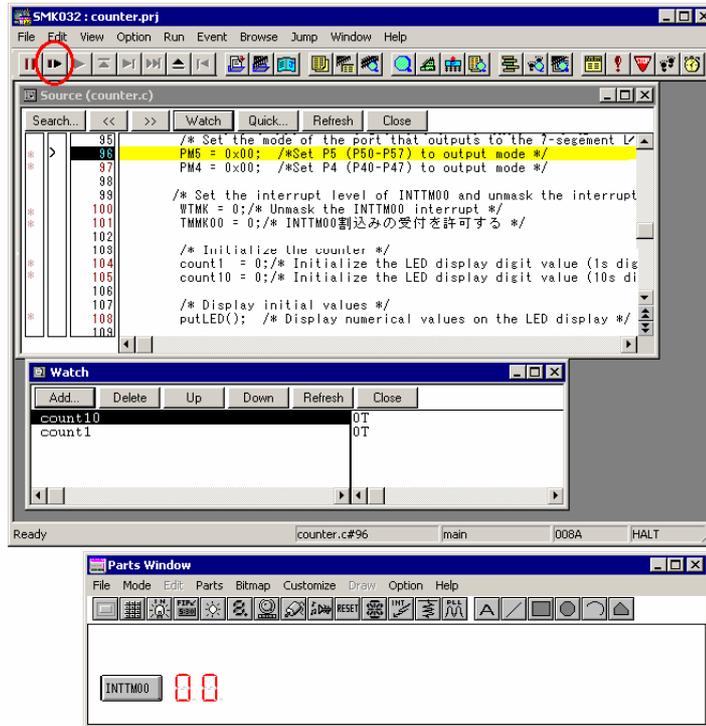


When build is completed, the SM78Kxx automatically downloads an executable program file.

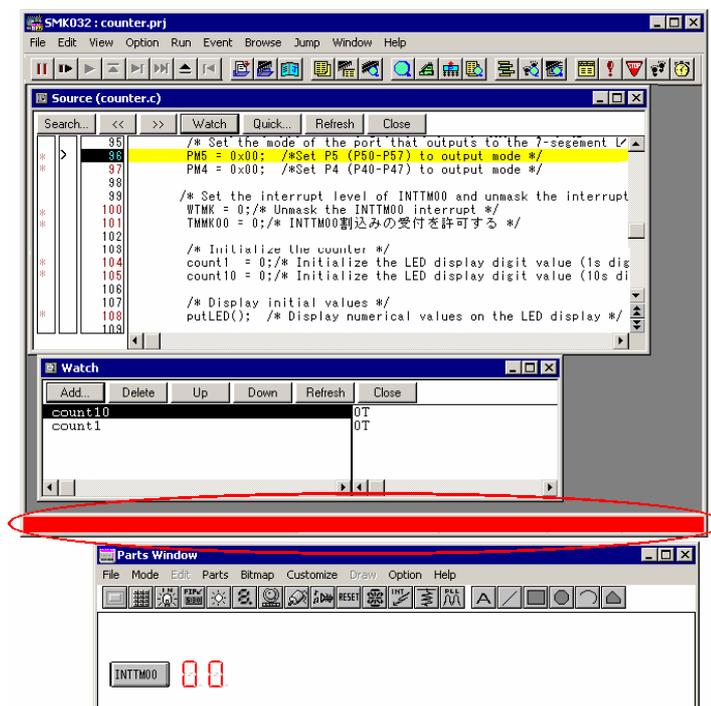
Executing the Program (2)

Now, perform a restart.

In the SM78Kxx Main window, click the Restart button , or from the menus, select Run->Restart.

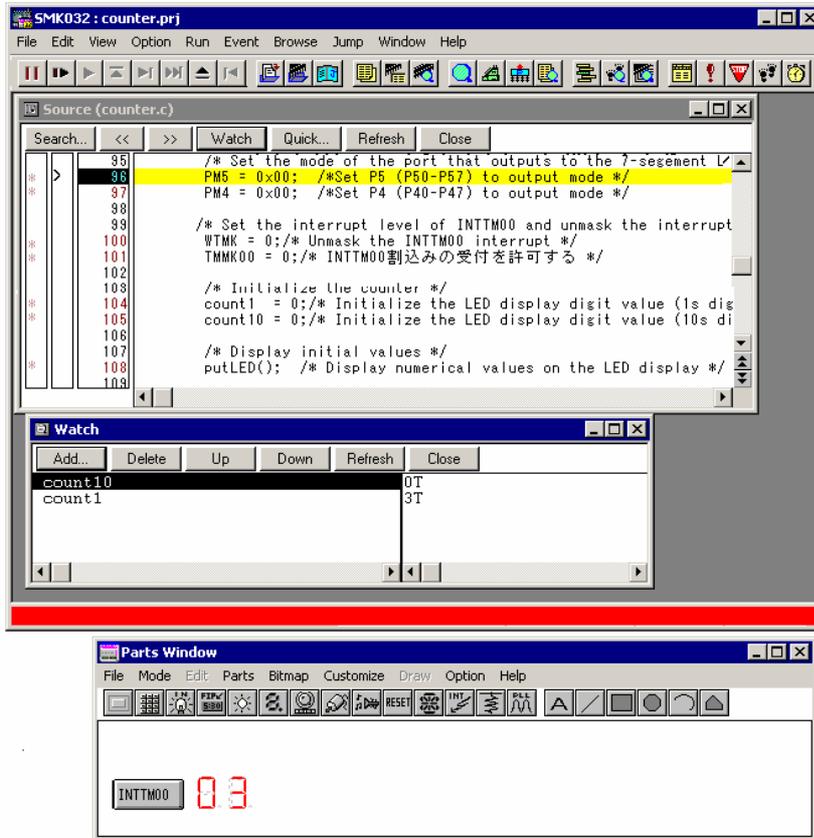


Program execution starts.



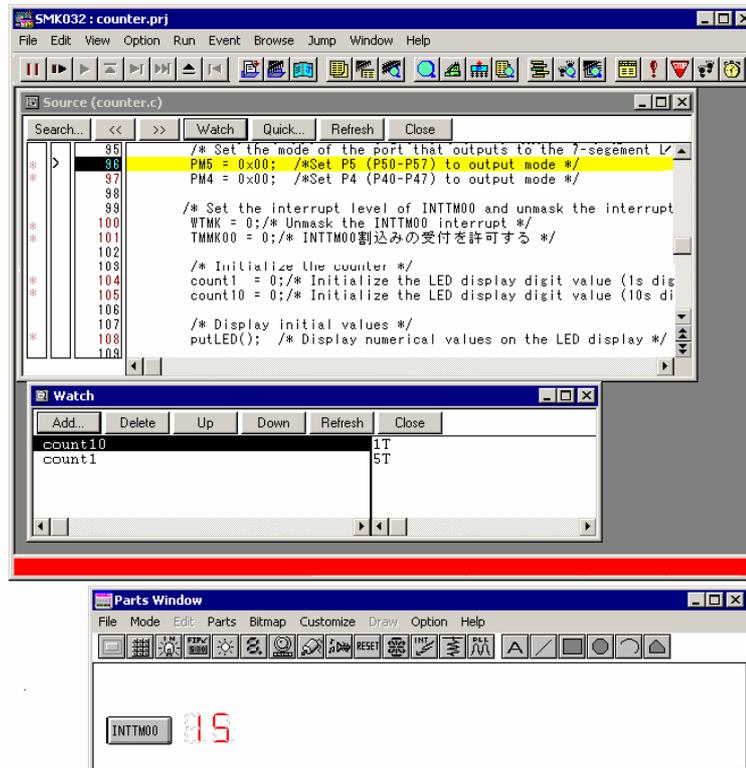
Confirm whether the incrementation works correctly.

First click the button several times to confirm whether the 1s digit increments correctly.



The 1s digit does, in fact, increment correctly.

Next, click the INTTM00 button more than ten times to confirm whether a carry operation (incrementing the 10s digit) is performed correctly.



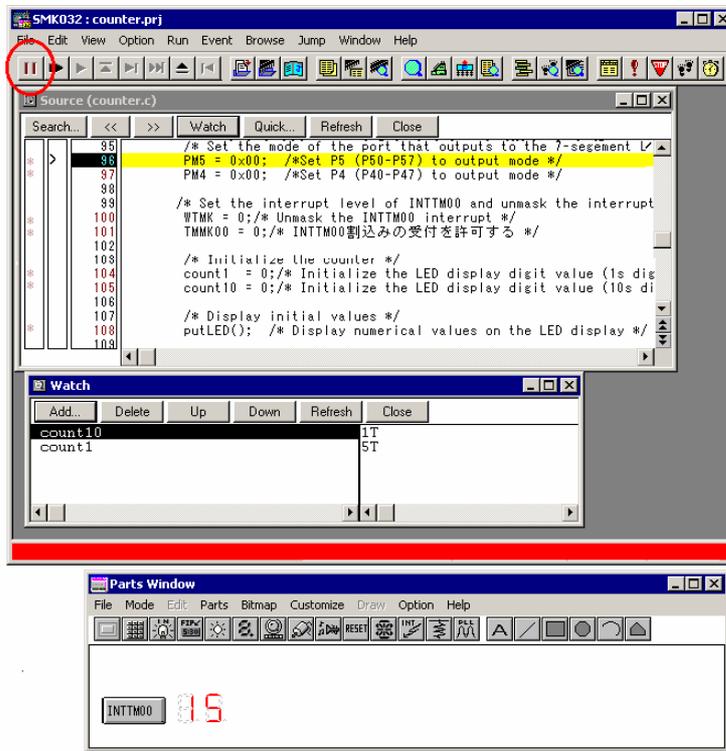
The 10s digit does, in fact, increment correctly.

This confirms that the incrementation operations for both digits are working correctly.

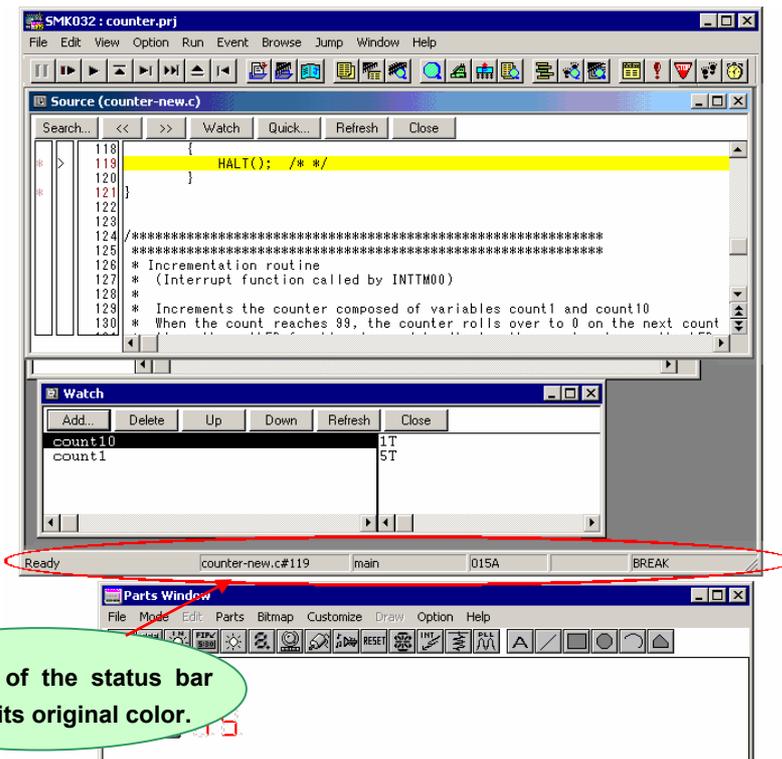
Lastly, you will confirm whether the counter overflow handling is working correctly.

You could click the INTTM00 button more than 100 times to verify the operation, but here is a simpler method.

In the SM78Kxx Main window, click the Stop button  or from the menus, select Run->Stop.



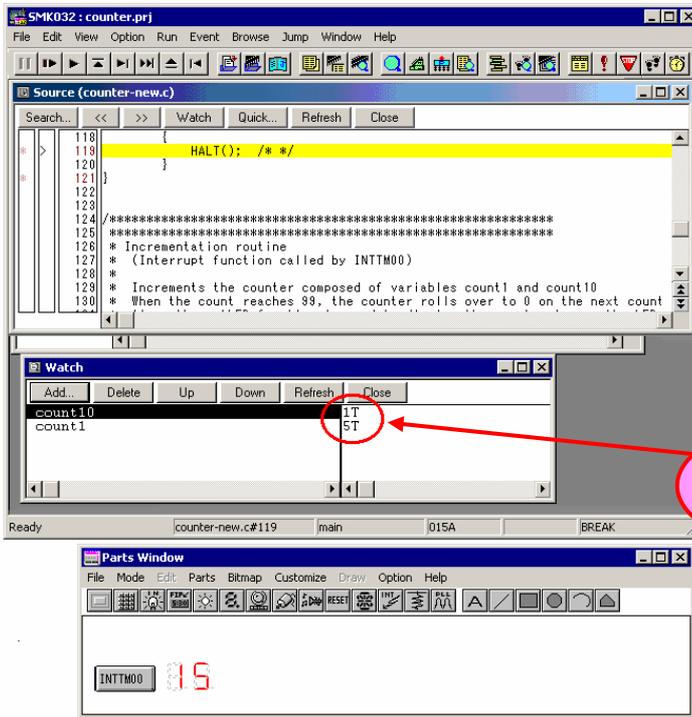
Program execution stops.



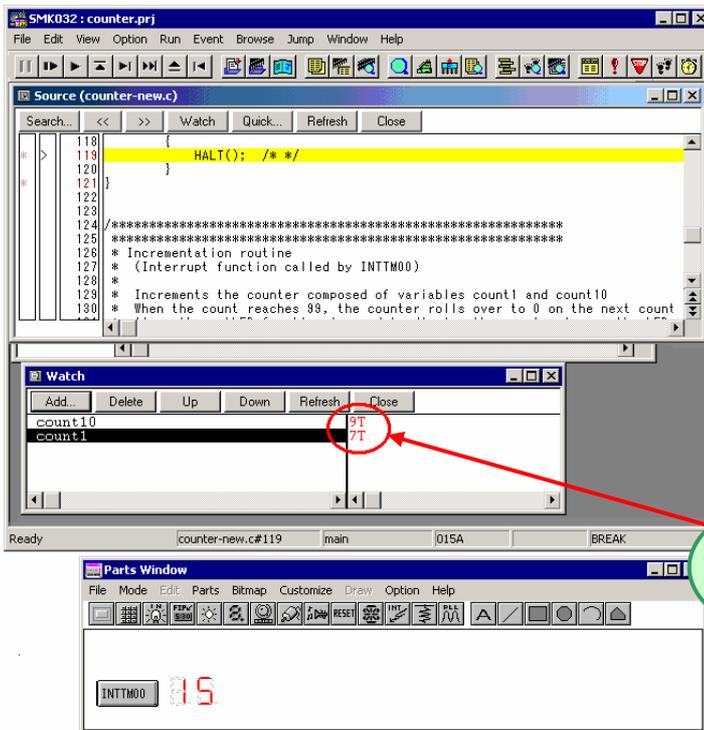
The color of the status bar returns to its original color.

Next, in the Watch window, change the values in the Data Value Display/Setting area.

Move the cursor to the Data Value Display/Setting area in the Watch window and change the value of count10 to 9 and count1 to 7.

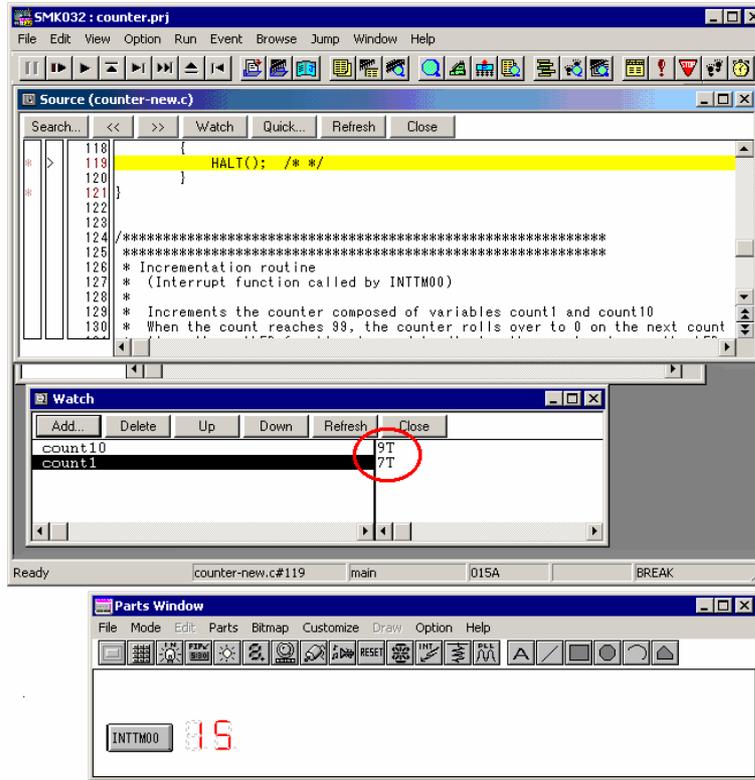


Change the value of count10 from 1 to 9 and the value of count1 from 5 to 7.



The values of count10 as 9 and count1 as 7 are displayed in red.

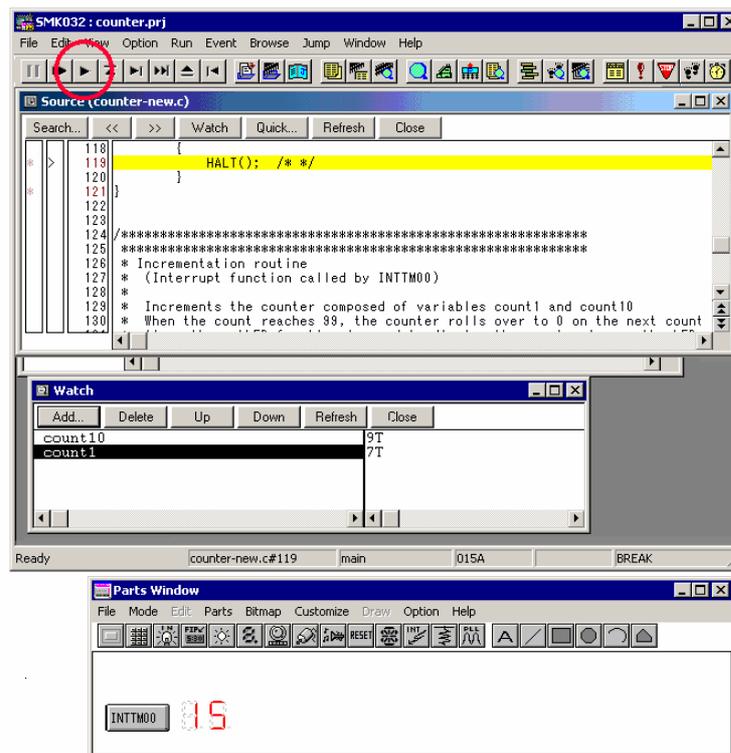
Now, with the values changed and displayed in red, press the Return key.
The values of count10 and count1 change from red to black.



At this point, count10 has become 9 and count1 has become 7.

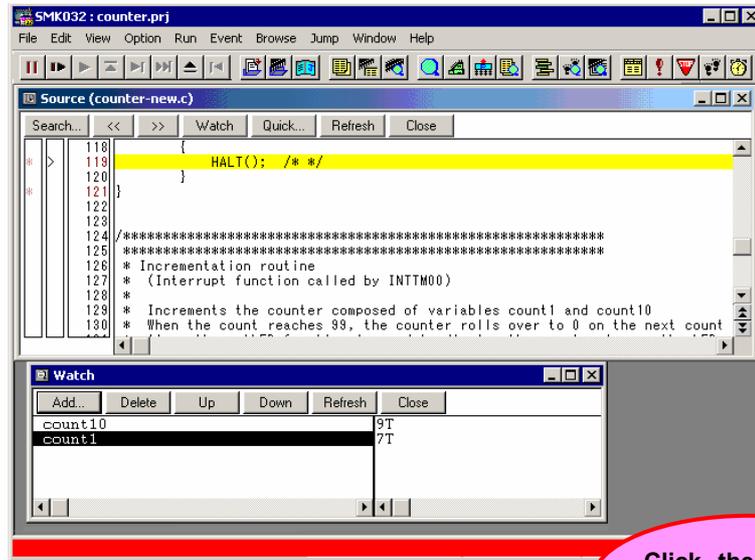
Now you can continue execution of the program.

In the SM78Kxx Main window, click the Start button  or from the menus, select Run->Go.

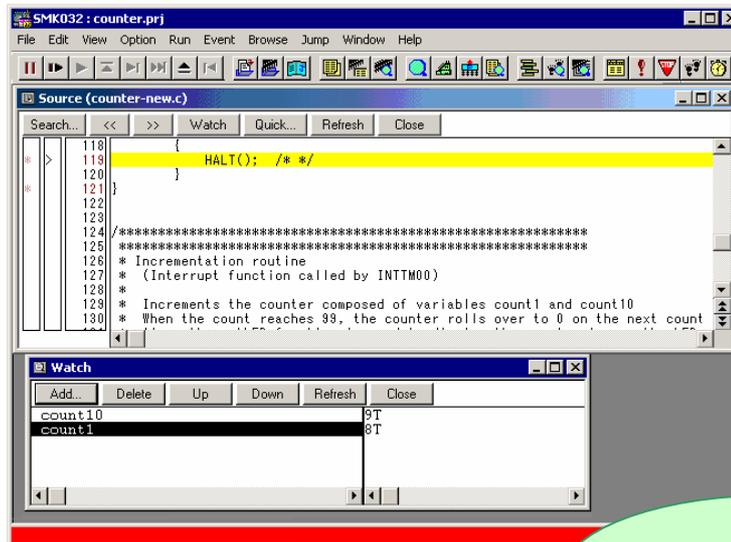
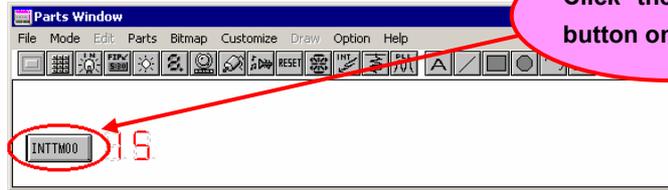




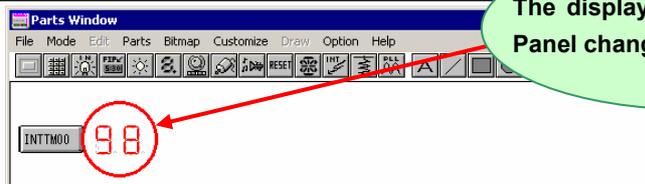
Program execution starts.



Click the INTTM00 button once.



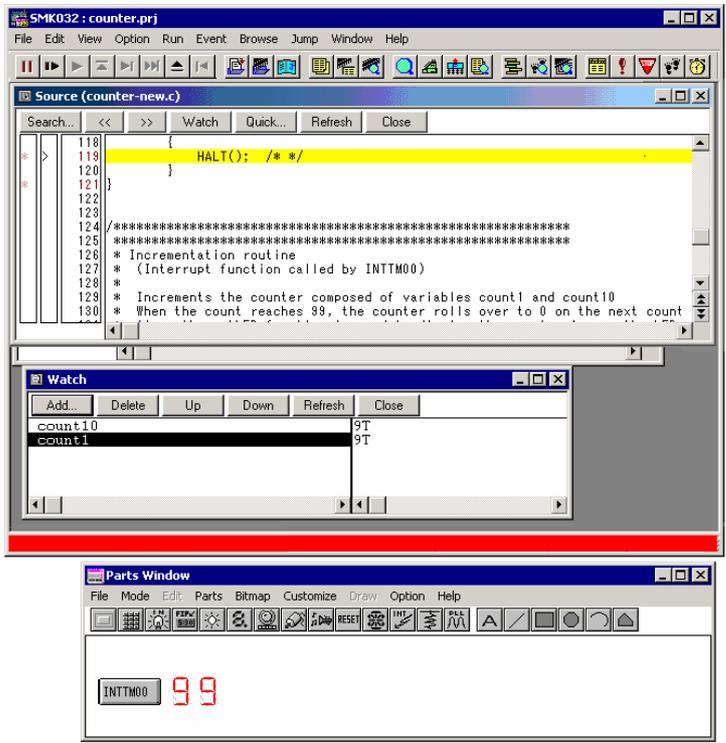
The display in the Input/Output Panel changes from 15 to 98.



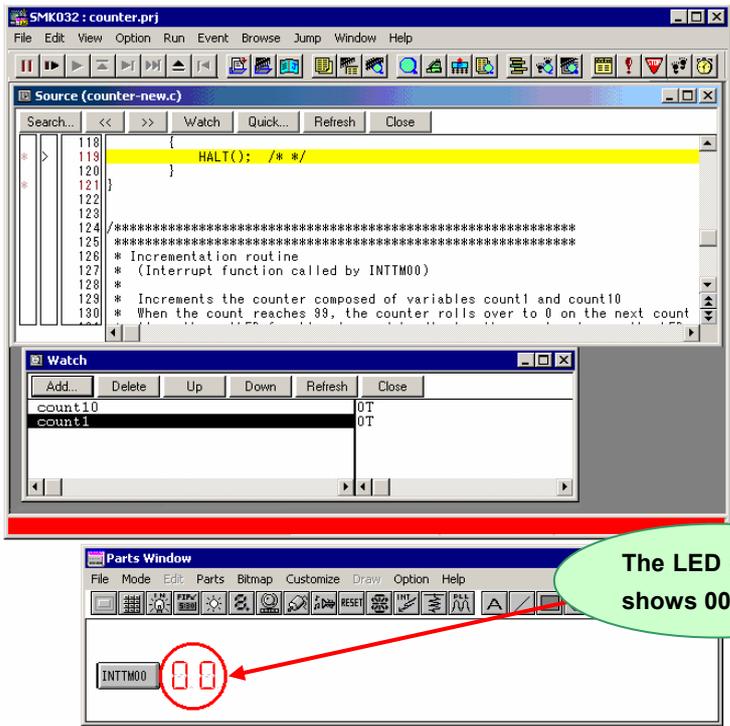
Now, the program is at the same point as if you had clicked the INTTM00 button 98 times. Continuing on, click the INTTM00 button again.



Click the INTTM00 button once.



Click the INTTM00 button once.

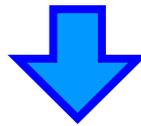
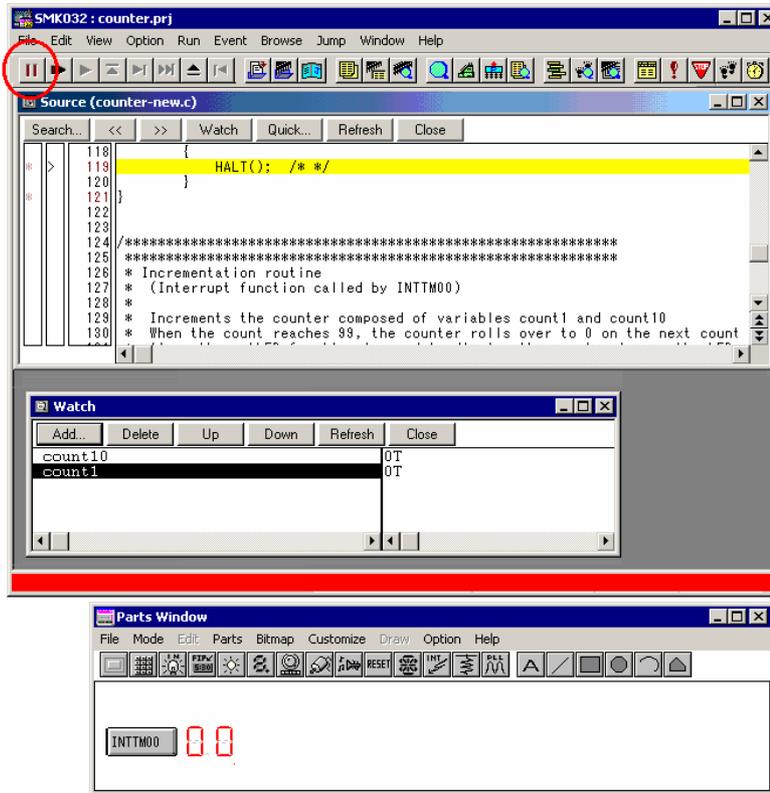


The LED display shows 00.

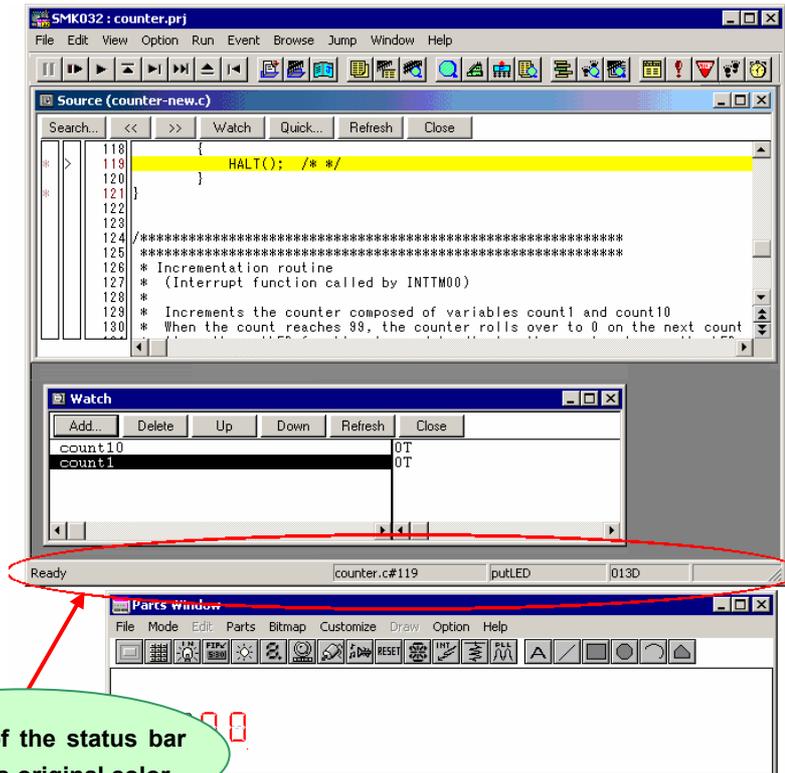
You have determined that the counter overflow handling operates correctly.

Stop the program.

In the SM78Kxx Main window, click the Stop button  , or from the menus, select Run->Stop.



Program execution stops.

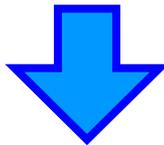
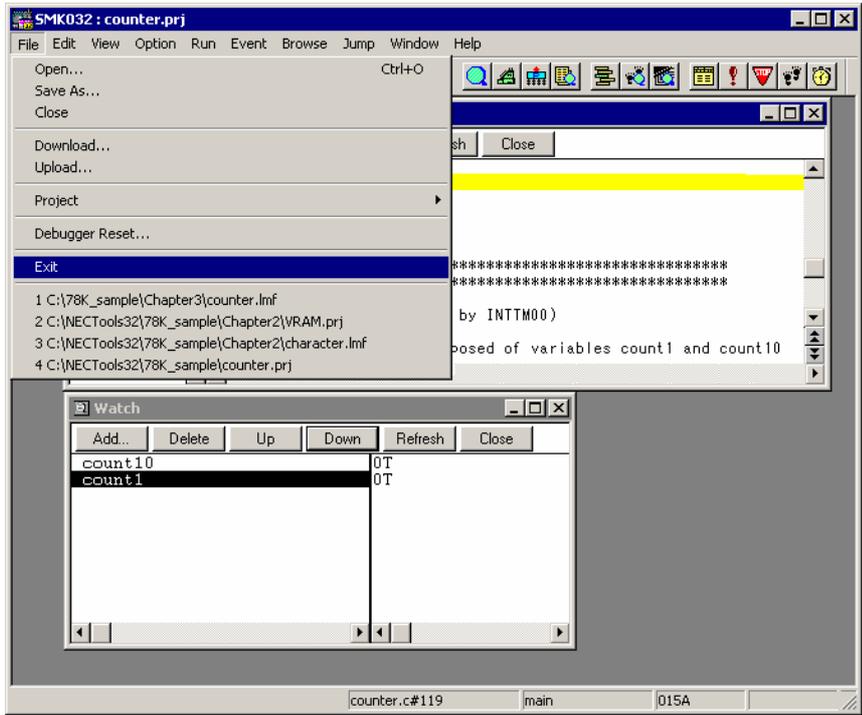


The color of the status bar returns to its original color.

Exiting

Next, you will exit the SM78Kxx.

In the SM78Kxx Main window, from the menus, select File->Exit.



A dialog is displayed asking you if you want to exit.



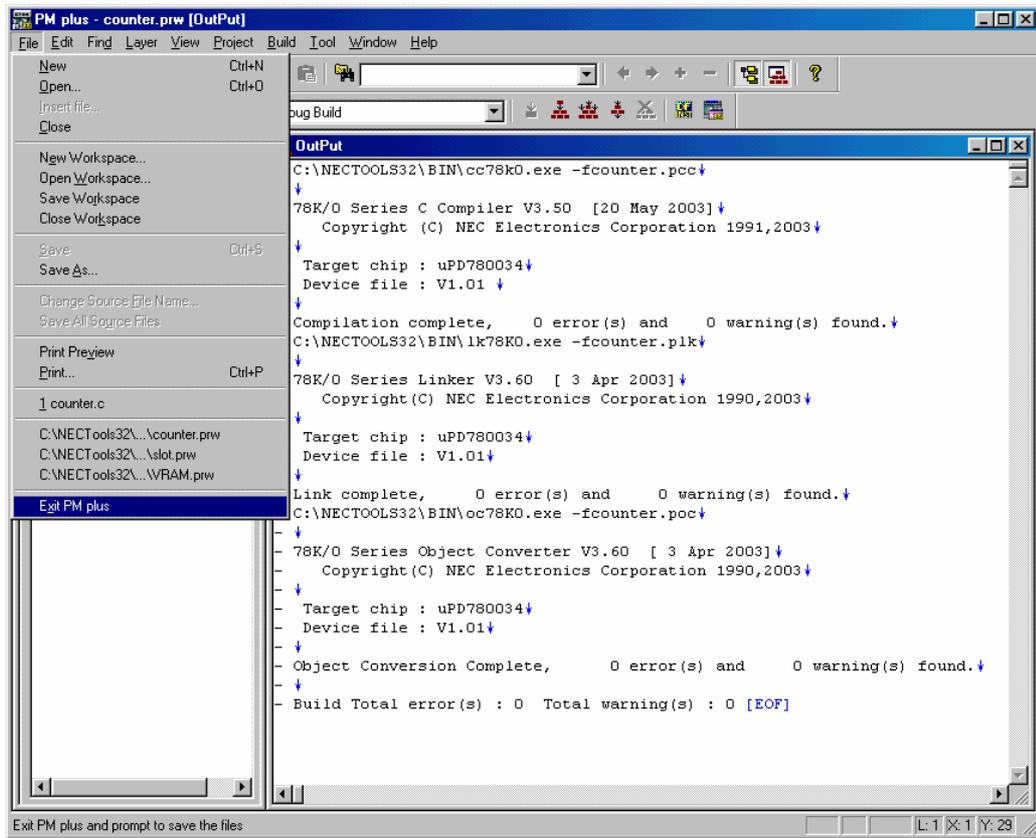
To save the settings performed in this chapter, such as the Input/Output Panel settings, click Yes button. To discard the settings, click Yes button. (To return to PM plus, click Cancel button.)

"Environment" refers to the external parts, window settings, etc.

➡ For details, refer to the **SM78K Series System Simulator Ver.2.52 Operation User's Manual (U16768E)**.

Lastly, you will exit PM plus.

In PM plus window, from the menus, select File->Exit PM plus.



Since PM plus saves project information successively, there is no confirmation dialog when you exit.

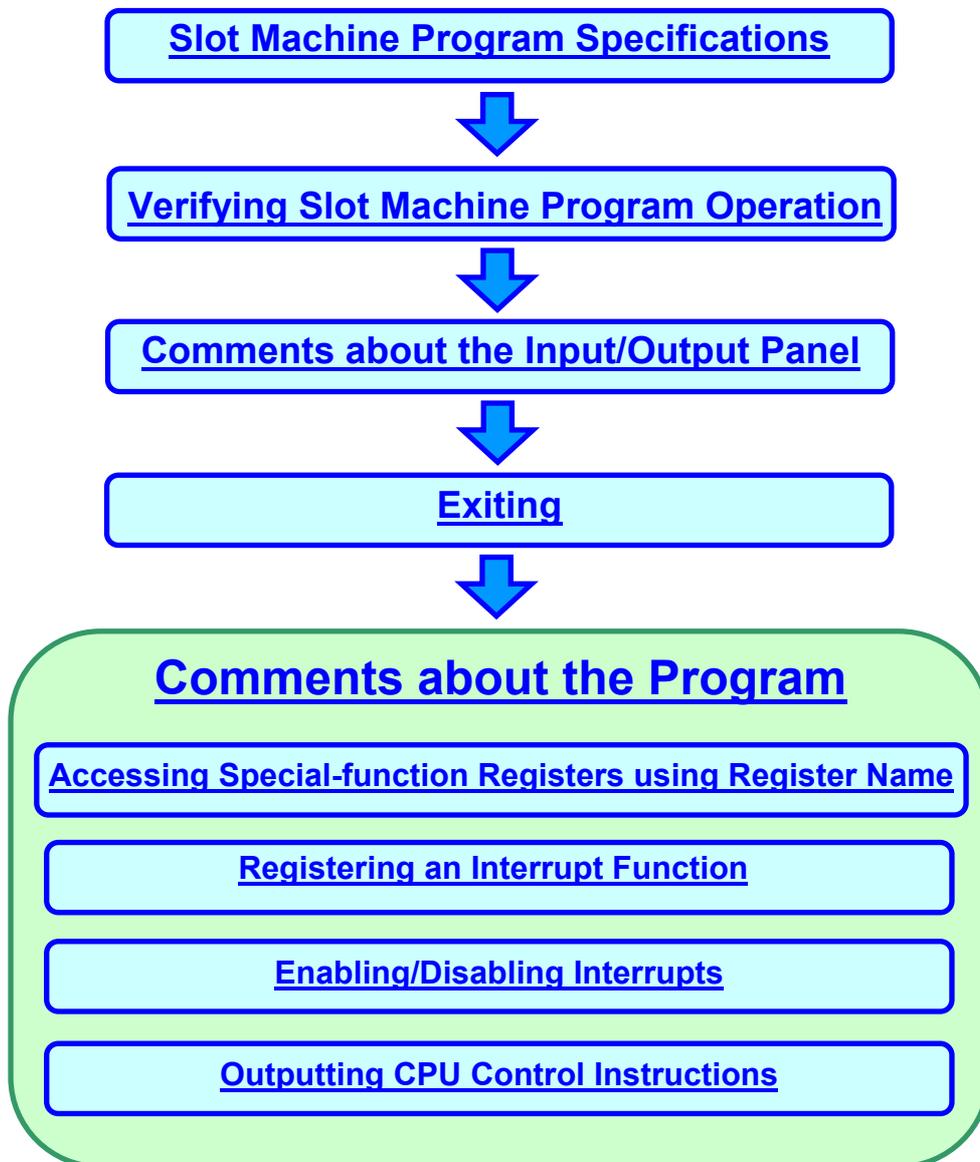
Chapter 4 Programming

This chapter shows you how to handle CPU-specific dependencies in the C programming language for the various 78K Series CPUs, using a sample program.

The sample program used here is a simple slot machine program.

The slot machine program uses extensions to the C language specification: accessing special-function registers using register name, interrupt/exception function descriptors, and enabling/disabling interrupts.

The overall flow of this chapter is as follows.



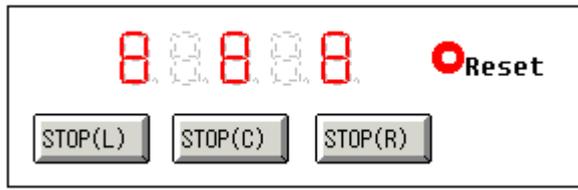
Slot Machine Program Specifications

Before running the slot machine program in this chapter, you need to have a general understanding of the program.

The external specifications are as follows.

External Specifications

- There are five 7-segment LED display digits, three square buttons and a reset button for the device.



- Every second LED display digit is used to display numbers continuously in sequence from 0 to 9, with the display cycling back to 0 after 9 is displayed.
- The square buttons are labeled from left to right as "STOP(L)", "STOP(C)", "STOP(R)", respectively and correspond to a digit. When a button is clicked, the corresponding digit stops cycling and displays the number it stopped at.
 - When the STOP(L) button is clicked, the left-most LED display digit stops.
 - When the STOP(C) button is clicked, the center LED display digit stops.
 - When the STOP(R) button is clicked, the right-most LED display digit stops.
- When the Reset button is clicked, the system returns to its initial state, where the digits resume cycling.

The basic specifications are as follows.

Basic Specifications

- Slot machine display
 - Displays sequentially incremented numbers from 0 to 9
 - When a digit reaches 9, the counter loops back to 0
- Button operation
 - When the STOP(L) button is clicked, the INTP0 interrupt is generated.
 - When the STOP(C) button is clicked, the INTP1 interrupt is generated.
 - When the STOP(R) button is clicked, the INTP2 interrupt is generated.
- Interrupt function settings
 - When the INTP0 interrupt is generated, the stop_btn_Left function is executed.
 - When the INTP1 interrupt is generated, the stop_btn_Center function is executed.
 - When the INTP2 interrupt is generated, the stop_btn_Right function is executed.
- Interrupt function processing
 - When the stop_btn_Left function is executed, the left-most digit display is frozen.
 - When the stop_btn_Center function is executed, the center digit display is frozen.
 - When the stop_btn_Right function is executed, the right-most digit display is frozen.
- Initialization of the target CPU environment
 - Initialize the ports to be used.
 - Enable the interrupts and set their priority.

The internal specifications are as follows.

Internal Specifications

- The following variables specify the LED display digit position and numerical value, respectively.

Variable	Contents
unsigned char place;	Specifies the digit position
unsigned char num_data[10];	Specifies the numerical value to be displayed

- The program consists of the main function, target CPU environment initialization, slot machine display routine, and the interrupt function.

File name	Function	Contents
slot.c	Main function void main();	- calls the target CPU environment initialization routine (init_target()) - calls the slot machine display function (slot())
	Target CPU environment initialization void init_target(void);	- initializes the target CPU environment, such as the ports and interrupt levels - enables the interrupts
	Slot machine display void slot(void);	- cycles through digits 0-9 and displays them on the LED display - accepts an interrupt during cycling
interrupt_func.h (function declaration)	STOP(L) button processing __interrupt void stp_btn_Left(void);	- triggered by an INTP0 interrupt - freezes the left-most digit display at its current value
	STOP(C) button processing __interrupt void stp_btn_Center(void);	- triggered by an INTP1 interrupt - freezes the middle digit display at its current value
interrupt_func.c (function definition)	STOP(R) button processing __interrupt void stp_btn_Right(void);	- triggered by an INTP2 interrupt - freezes the right-most digit display at its current value

- The following 3 interrupts are used

- INTP0
- INTP1
- INTP2

- The following I/O ports are used for LED display control and interrupt input:

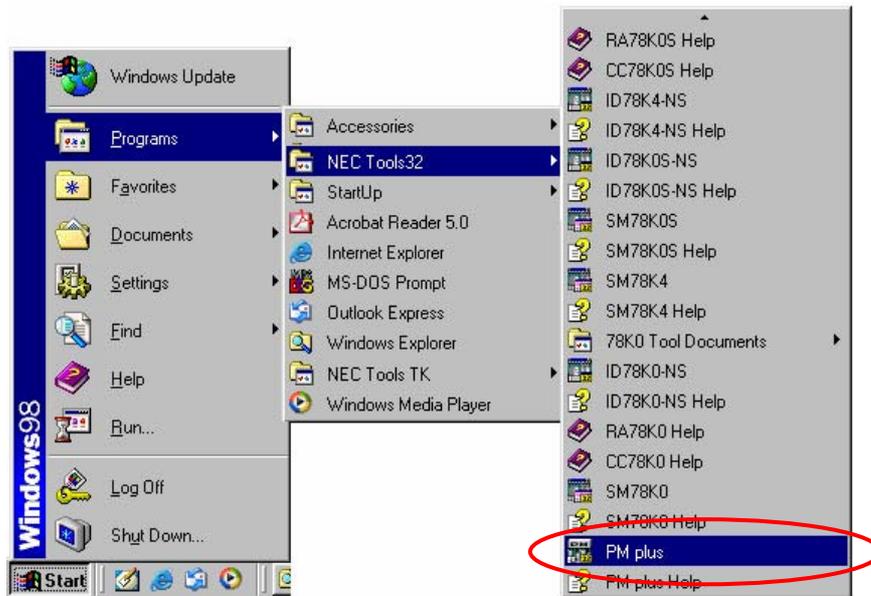
- For the 78K0: P0, P4 and P5
 - For the 78K0S: P1, P2 and P54
 - For the 78K4: P2, P4 and P5
- | | | | |
|------|-------|------|--|
| 78K0 | 78K0S | 78K4 | |
|------|-------|------|--|

- P0 P2 P2 Interrupt input
- P5 P1 P5 Lighting the 7-segment display segments
- P4 P4 P4 Selecting the 7-segment display digit

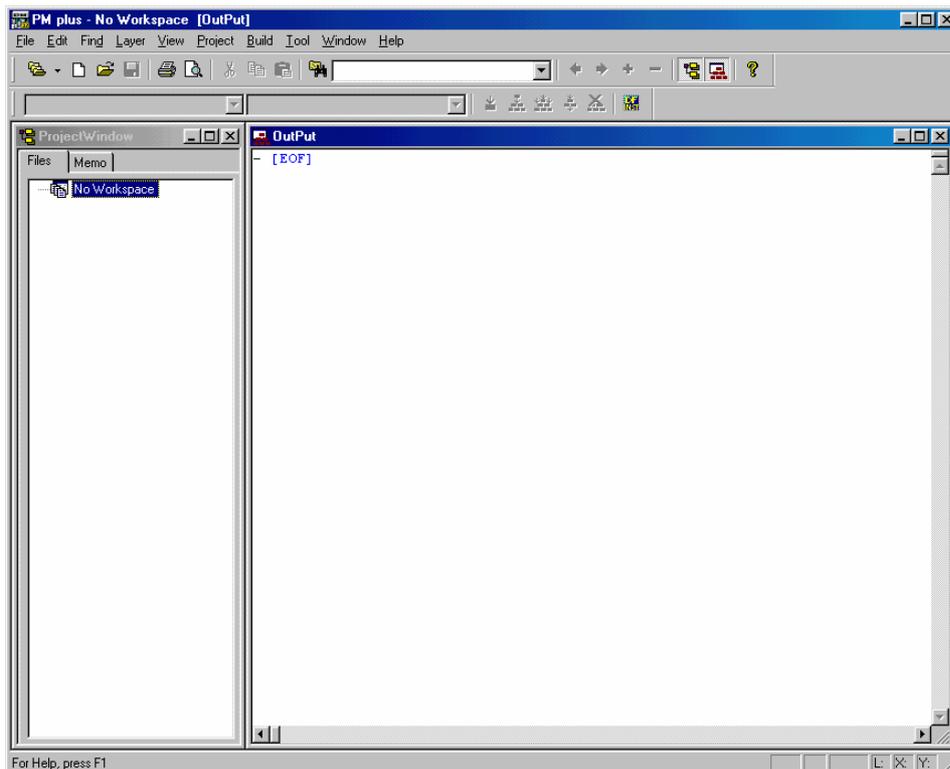
Verifying Slot Machine Program Operation

To begin verification of the slot machine program, first start PM plus.

From the Windows Start menu, select Programs->NEC Tools32->PM plus.



PM plus starts.

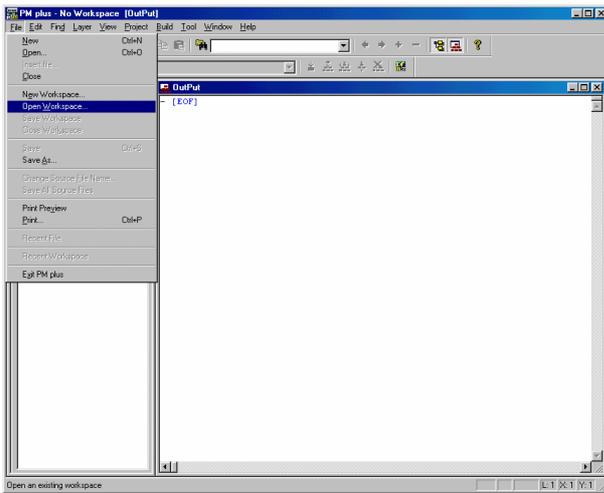


Reading the Workspace File

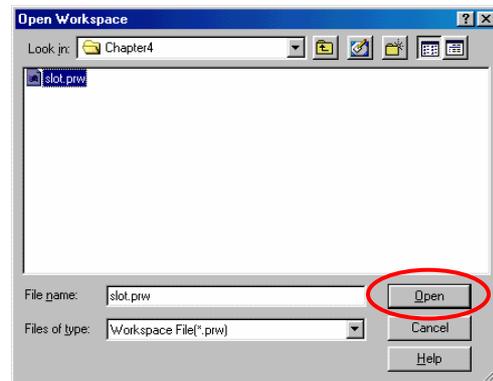
In this chapter, you will use a workspace file that has already been created.

In PM plus, from the menus, select File->Open Workspace... and specify the slot.prw workspace file.

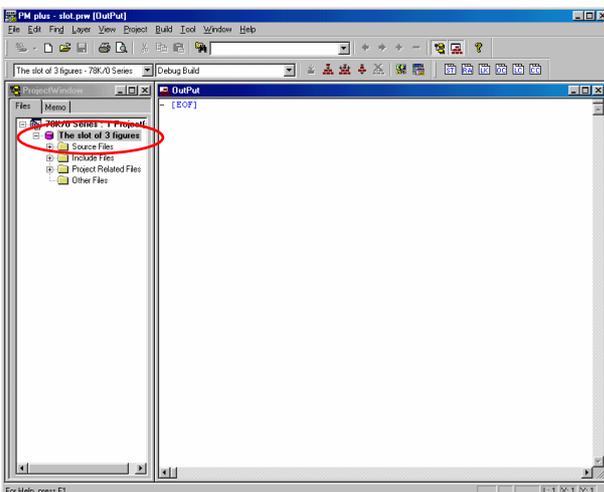
➔ If you have not yet set up the Sample Environment, please refer to [Chapter 1 - Tutorial Sample Environment](#).



Open the Chapter4 directory.



Select slot.prw and click Open.

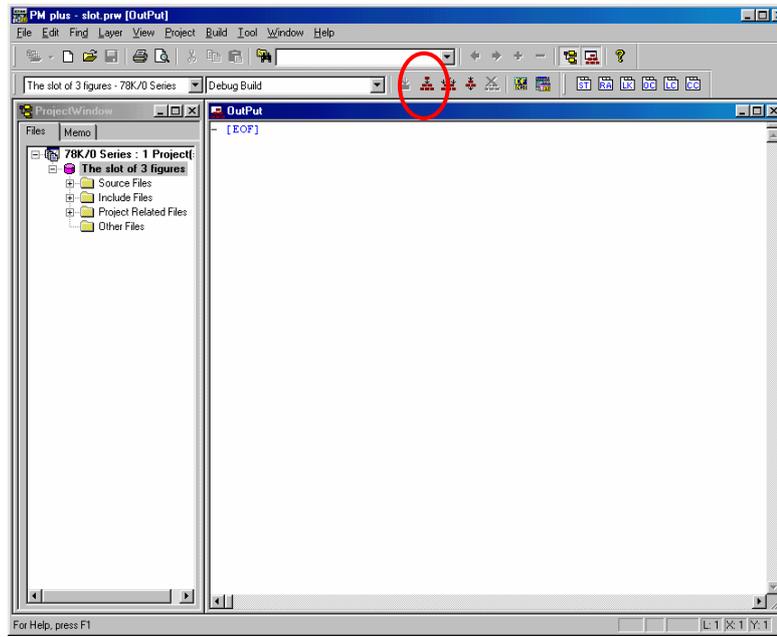


The slot.prw workspace file is read.

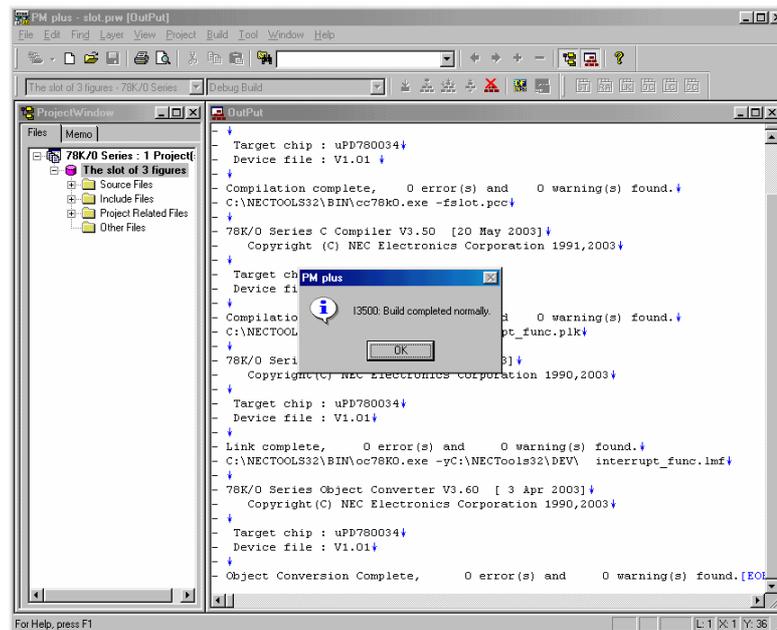
Creating an Executable Program

Next, you will create an executable program.

In PM plus, click the Build button  or, from the menus, select Build->Build.



The build process starts.

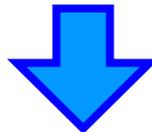
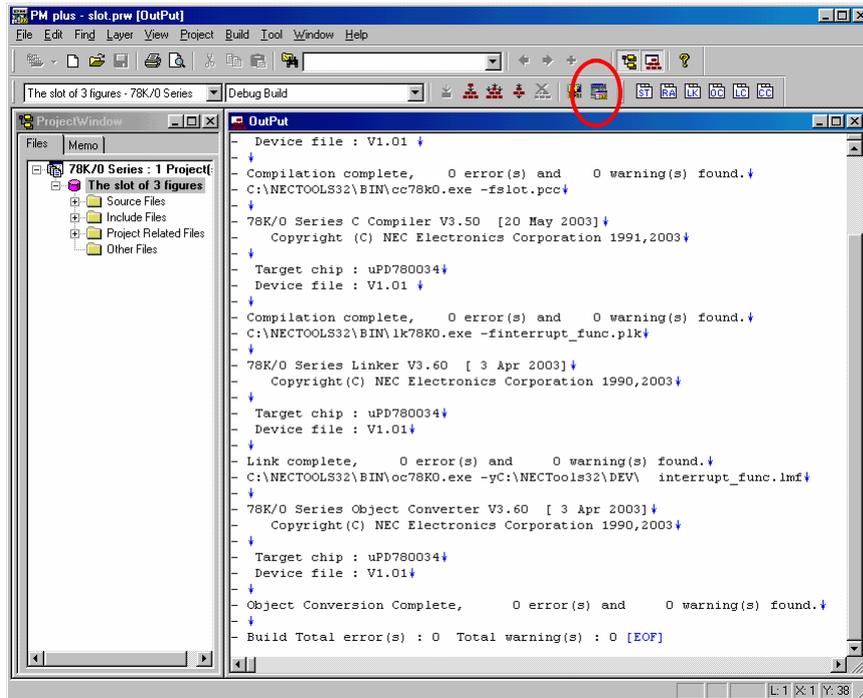


The build completes normally, and an executable program is created.

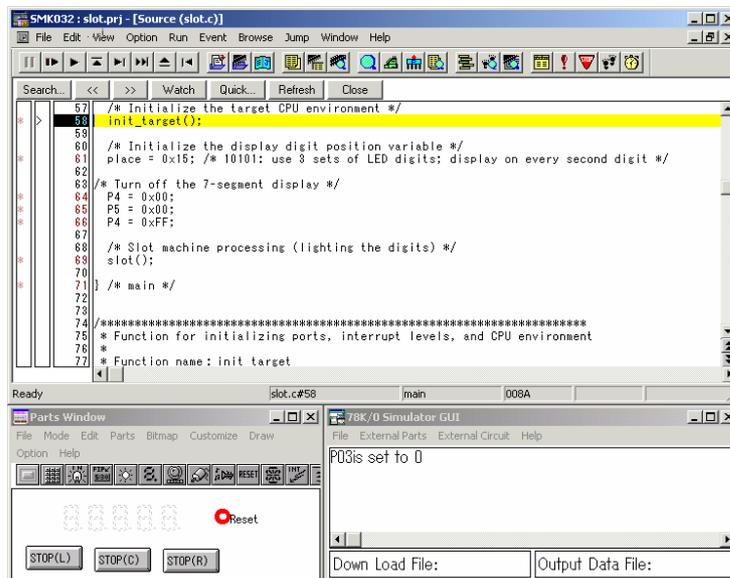
Running the System Simulator (SM78Kxx)

Next, you will run the SM78Kxx.

In the Project Manager, click the Debug button  , or, from the menus, select Build->Debug .



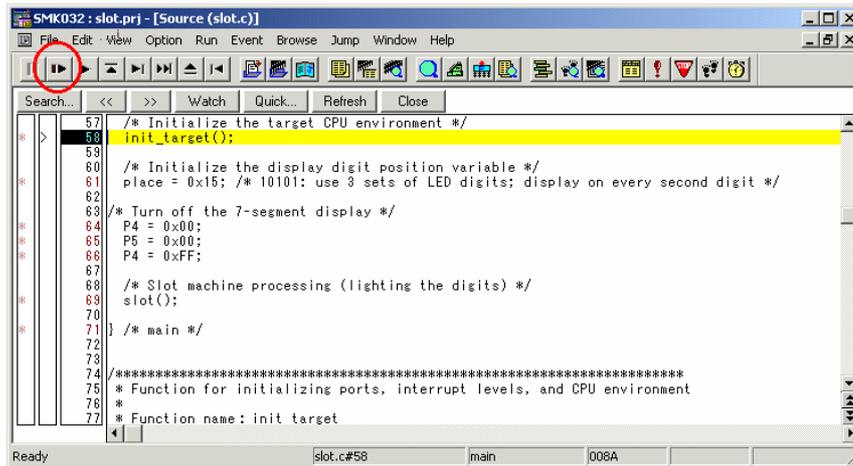
The SM78Kxx starts.



Running the Program

Now, you will run the [slot machine program](#).

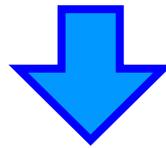
In the SM78Kxx Main window, click the Restart button  , or from the menus, select Run->Restart.



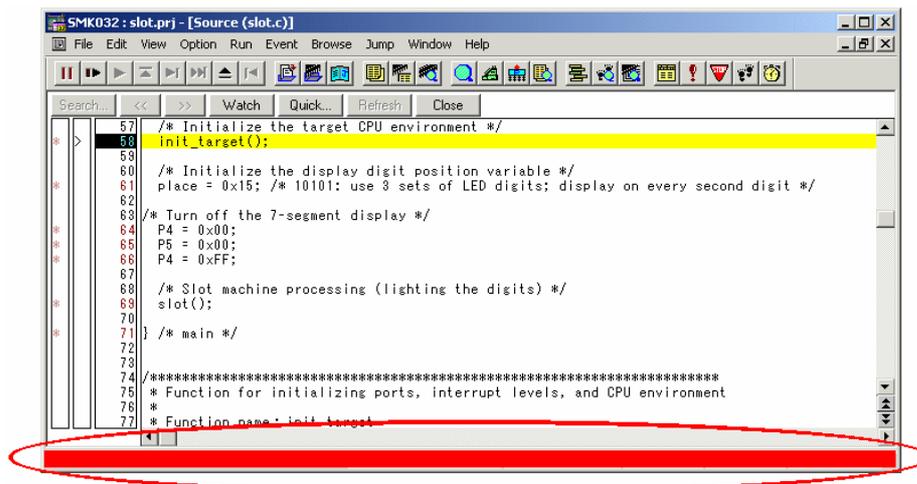
```

SMK032 : slot.prj - [Source (slot.c)]
File Edit View Option Run Event Browse Jump Window Help
Search... << >> Watch Quick... Refresh Close
57 /* Initialize the target CPU environment */
58 init_target();
59
60 /* Initialize the display digit position variable */
61 place = 0x15; /* 10101: use 3 sets of LED digits; display on every second digit */
62
63 /* Turn off the 7-segment display */
64 P4 = 0x00;
65 P5 = 0x00;
66 P4 = 0xFF;
67
68 /* Slot machine processing (lighting the digits) */
69 slot();
70
71 } /* main */
72
73
74
75 *****
76 * Function for initializing ports, interrupt levels, and CPU environment
77 *
78 * Function name: init_target
Ready slot.c#58 main 008A

```



Program execution starts.



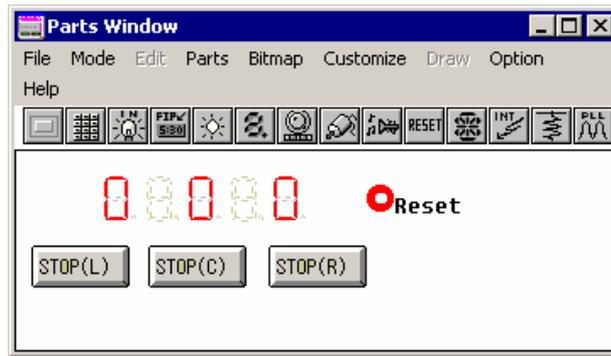
```

SMK032 : slot.prj - [Source (slot.c)]
File Edit View Option Run Event Browse Jump Window Help
Search... << >> Watch Quick... Refresh Close
57 /* Initialize the target CPU environment */
58 init_target();
59
60 /* Initialize the display digit position variable */
61 place = 0x15; /* 10101: use 3 sets of LED digits; display on every second digit */
62
63 /* Turn off the 7-segment display */
64 P4 = 0x00;
65 P5 = 0x00;
66 P4 = 0xFF;
67
68 /* Slot machine processing (lighting the digits) */
69 slot();
70
71 } /* main */
72
73
74
75 *****
76 * Function for initializing ports, interrupt levels, and CPU environment
77 *
78 * Function name: init_target

```

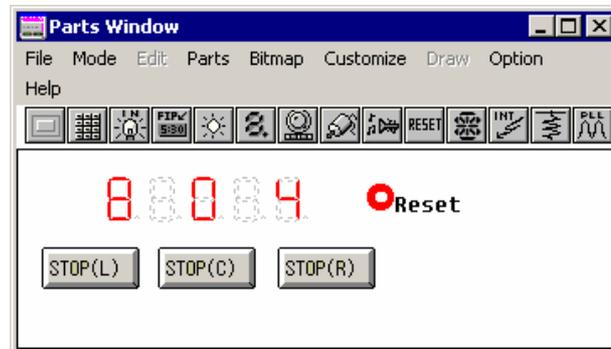
The color of the status bar changes to red during program execution.

As the program executes, the LED display digits in the Input/Output Panel cyclically increment from 0 to 9.



Now, let's try to operate the slot machine program.

Click each of the buttons in the Input/Output Panel and confirm that the digits displayed on LED display change accordingly.



-  When the STOP(L) button is clicked, the left-most LED display digit stops.
-  When the STOP(C) button is clicked, the center LED display digit stops.
-  When the STOP(R) button is clicked, the right-most LED display digit stops.

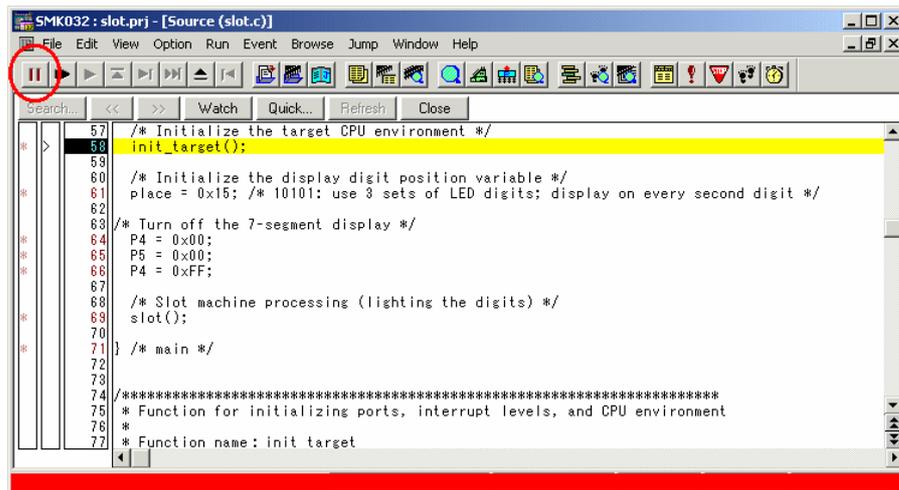
When the Reset button is clicked, the system returns to its initial state, where the digits resume cycling.

This completes the verification of the slot machine program operation.

Stopping the Program

Next, you will stop the execution of the program.

In the SM78Kxx Main window, click the Stop button , or from the menus, select Run->Stop.



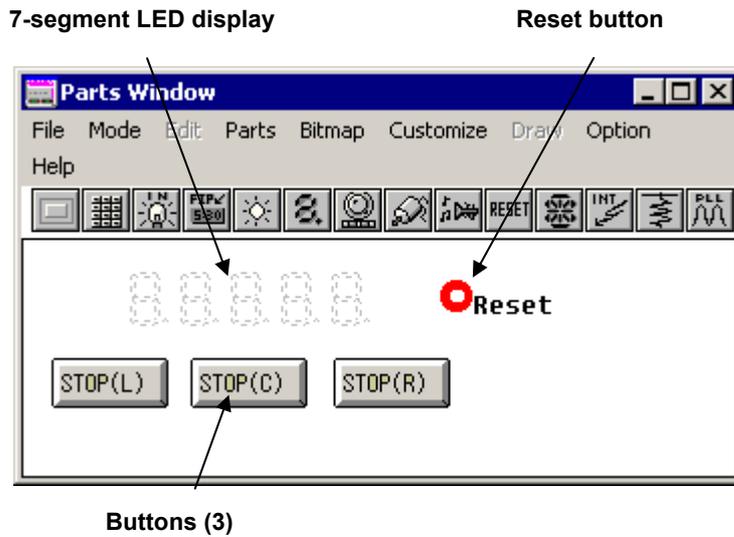
```

SMK032 : slot.prj - [Source (slot.c)]
File Edit View Option Run Event Browse Jump Window Help
Search... << >> Watch Quick... Refresh Close
67 /* Initialize the target CPU environment */
68 init_target();
69
70 /* Initialize the display digit position variable */
71 place = 0x15; /* 10101: use 3 sets of LED digits; display on every second digit */
72
73 /* Turn off the 7-segment display */
74 P4 = 0x00;
75 P5 = 0x00;
76 P4 = 0xFF;
77
78 /* Slot machine processing (lighting the digits) */
79 slot();
80
81 } /* main */
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
```

Comments about the Input/Output Panel

The devices used by the slot machine program that are displayed in the Input/Output Panel are a 7-segment LED display, 3 buttons and a reset button.

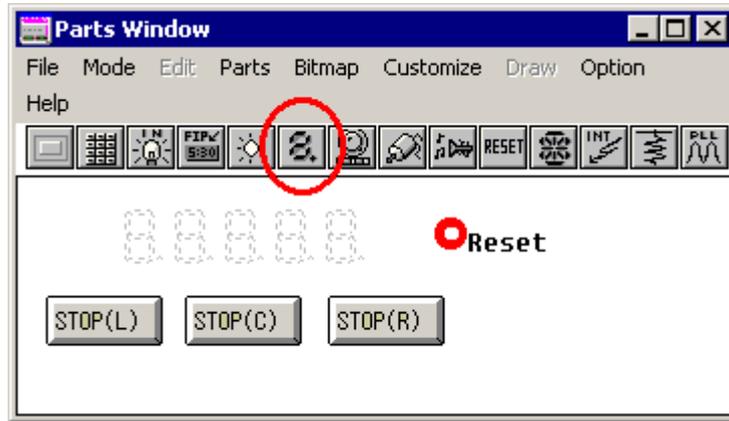
The settings for each device are described below.



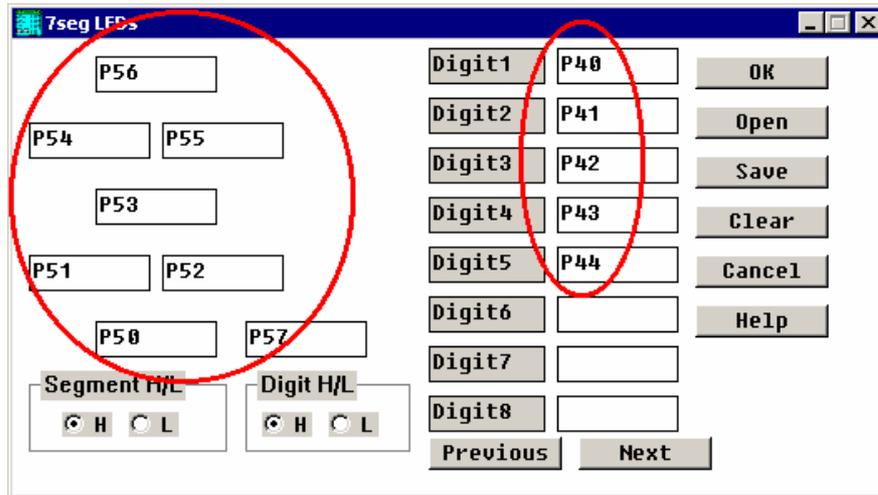
First, the 7-segment LED display terminal settings are described.

In the Input/Output Panel window, click the 7-segment LED display terminal setting button  or, from the menus, select Connection->7-segment LED...

The 7-segment LED Terminal Setting dialog opens.



The 7-segment LED Terminal Setting dialog opens.



For the 78K0 and 78K4, each bit of I/O ports P4 and P5 are connected to the 7-segment LED display terminals. For the 78K0S, each bit of I/O ports P1 and P4 are connected to the 7-segment LED display terminals.

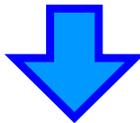
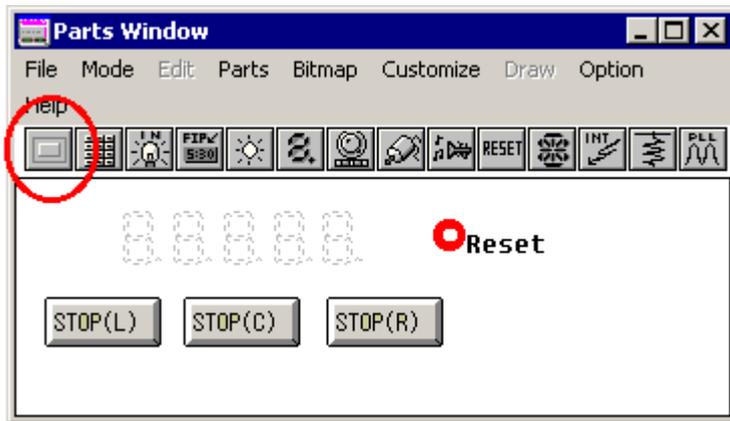
Since the slot machine program uses 5 digits, Digit Signal 1 through Digit Signal 5 are connected.

➡ For details on 7-segment LED display terminal settings, refer to [Chapter 3 - System Simulator Basics](#).

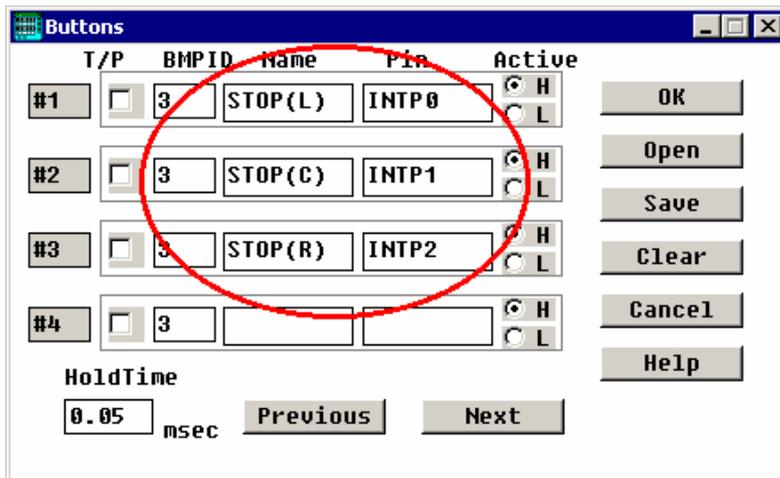
Next, the button settings are described.

In the Input/Output Panel, click the button  or, from the menus, select Connection->Button...

The Button Terminal Setting dialog opens.



The Button Terminal Setting dialog opens.



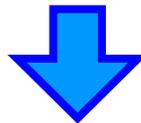
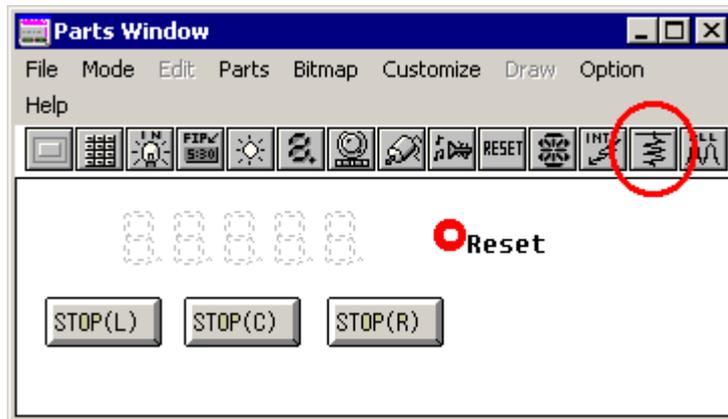
In the Button Terminal Setting dialog, you input the connection information for a button that is displayed in the Input/Output Panel window. A button can be connected to any terminal, and will apply the desired input value to the terminal when the button is clicked. For the program in this chapter, the buttons are connected to the external interrupt terminals (INTP0 to 2).

An internal interrupt button can be used to generate an internal interrupt when clicked, but in this case, the buttons are connected to the external interrupt terminals and a high state is detected on the terminal to trigger an interrupt.

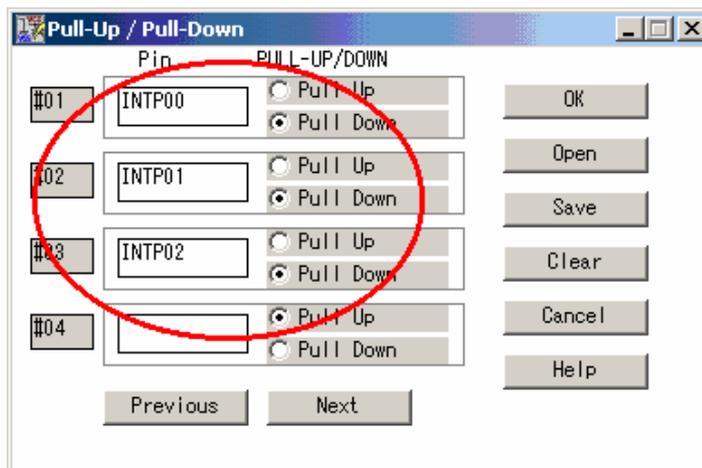
➡ For details, refer to the "[Slot Machine Program Specifications](#)" section in this chapter.

In addition to the button settings, you must perform [pull-up/pull-down settings](#) for each interrupt terminal. In the Input/Output Panel, click the Pull-up/Pull-down button  or, from the menus, select Connection -> Pull-up/Pull-down Settings...

The Pull-up/Pull-down Settings dialog box will open.



The Pull-up/Pull-down Settings dialog box opens.



For the program in this chapter, the external interrupt terminals that are connected to the buttons (INTP0 to 2) are set to pull-down.

What is pull-up/pull-down ?

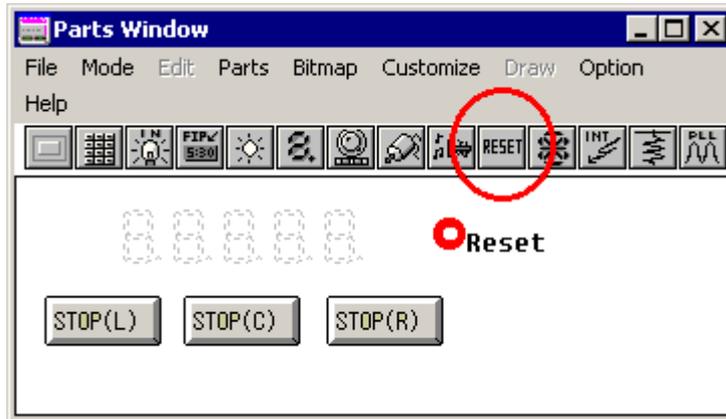
For some external parts of the SM78Kxx, the state is undefined when the part is not operating. A button is one of those parts. For such external parts, the pull-up/pull-down setting must be performed to define the state of the terminal when the button is not operating.

In addition, the pull-up/pull-down setting must be performed before setting an external part, such as a button.

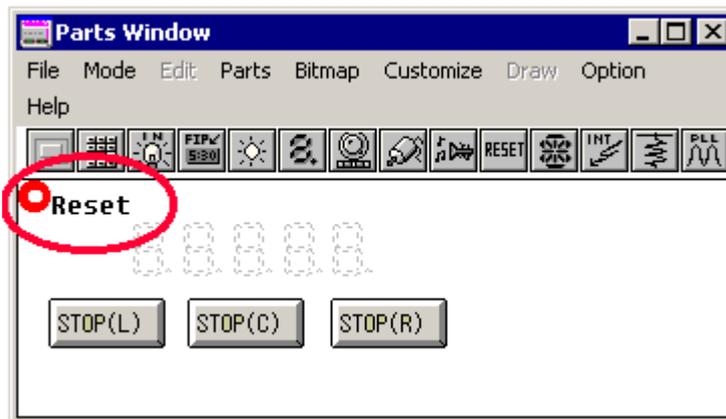
For details, refer to the SM78K Series System Simulator Ver.2.52 Operation User's Manual (U16768E).

Next, the setting of the reset button is described.

In the Input/Output Panel, click the Reset button  or, from the menus, select Connection->Reset Button. The Reset button moves to its default position.



The Reset button moves to its default position (upper-left corner).



To change the position of the reset button, from the Input/Output Panel menus, select Mode->Position and then drag and drop the reset button to the new position.

After changing the position of the reset button, from the Input/Output Panel menus, select Mode->Run to return to the Run mode.

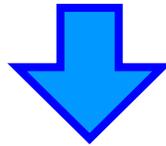
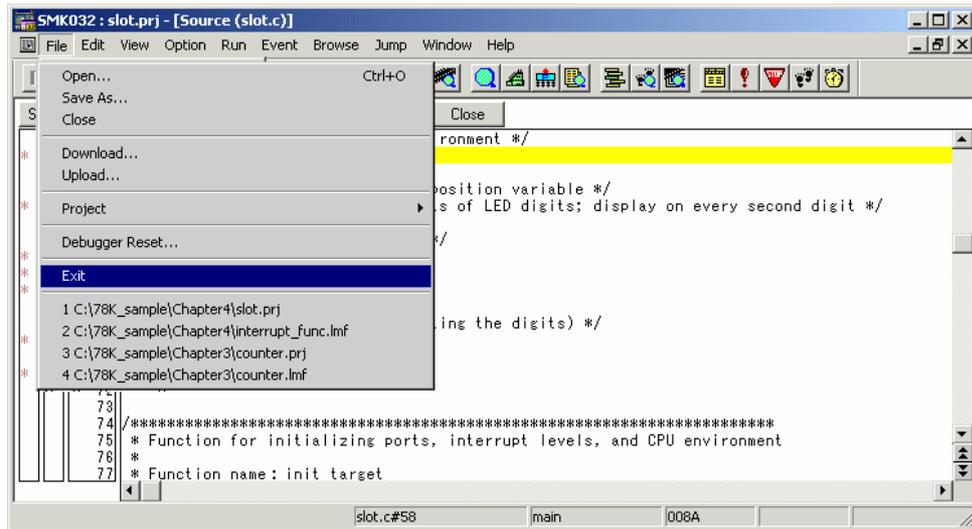
When you click the reset button during program execution, a reset signal is input to the simulator.

This completes the discussion of the various Input/Output Panel settings.

Exiting

Next, you will exit the SM78Kxx.

In the SM78Kxx Main window, from the menus, select File->Exit.



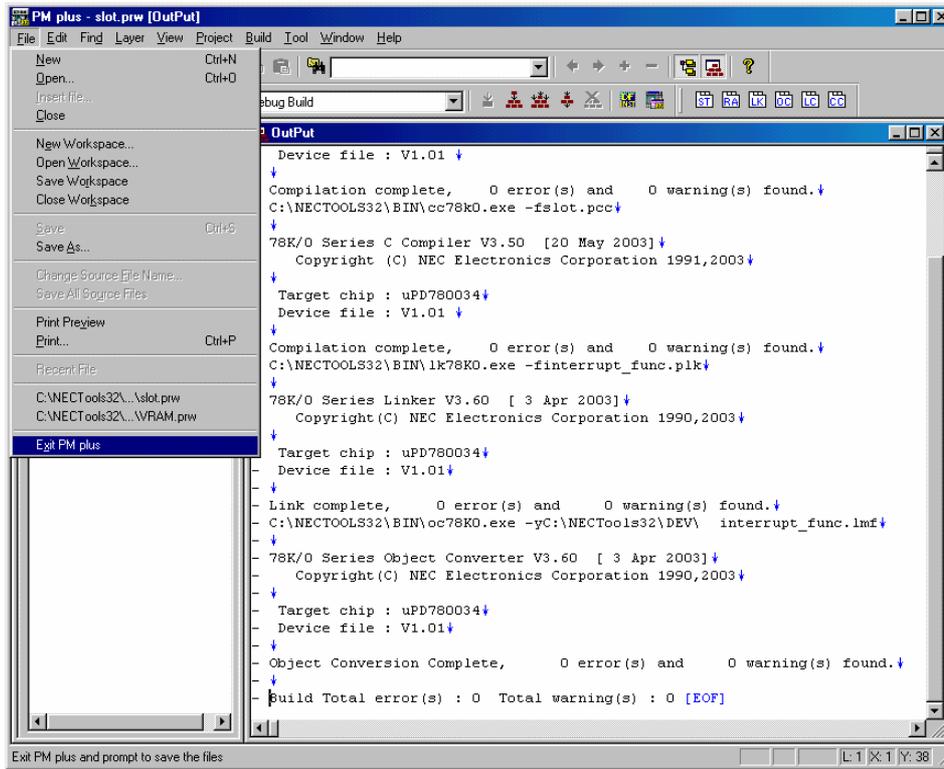
A dialog is displayed asking you if you want to exit.



To save the settings performed in this chapter, such as the Input/Output Panel settings, click Yes button. To discard the settings, click Yes button. (To return to PM plus, click Cancel button.)

Now, you will exit PM plus.

In PM plus window, from the menus, select File->Exit PM plus.



Since PM plus saves project information successively, there is no confirmation dialog when you exit.

Comments about the Program

Here, you will find a description of how to use the C programming language to implement the following functions, which are used in the slot machine program.

- Accessing special-function registers that are internal to a device
- Defining functions for when interrupts/exceptions occur
- Creating a function to control (enable/disable) interrupts.
- Outputting instructions to control the CPU

The source listing for the slot machine program is given in [Appendix - Slot Machine Program Source Listing](#). Refer to this listing when following the discussion in this chapter.

The CC78Kxx C compiler supports the ANSI standard C programming language specification.

To allow the handling of CPU-specific dependencies as much as possible using the C language, extensions to the ANSI standard specification are used.

These extensions allow such operations as interrupt handling, SFR referencing and handling of CPU-specific dependencies to be implemented in the C programming language, while maintaining efficiency of object usage, as well as improving program reusability and development efficiency.

The extensions to the specification are as follows.

- Specifying allocation to the external variable saddr area (sreg variables)
- Specifying allocation to the function argument or automatic variable saddr area, or to a register (norec, noauto functions)
- Specifying function calling for short instructions (callt function, callf function^{Note 1})
- Accessing SFRs
- Describing interrupt servicing in C language (register bank^{Note 2} switching possible)
- Outputting interrupt disable/enable instructions
- Inserting assembler descriptions in a C source program
- Outputting CPU control instructions
- Describing binary constants

For details on the language extensions, refer to the "CC78Kxx C Compiler Package Language" user manual.

Notes 1. The callf instruction is not supported by the 78K0S.

2. Not supported by the 78K0S.

Accessing Special-function Registers using Register Name - #pragma sfr

The registers for the peripheral functions incorporated in each device are called special-function registers. To access special-function registers from a C program, you need to include a "#pragma sfr" directive at the beginning of the source.

```
#pragma sfr
```

Therefore, for the slot machine program, you put a #pragma sfr directive at the beginning of slot.c.

[slot.c]

```
/* Enable special-function register name (SFR name)
#pragma sfr
```

When you use a "#pragma sfr" directive, the special-function register name can be treated as an ordinary unsigned external variable. However, without the "#pragma sfr" directive, when you try to use a special-function register name, the compiler generates an error (error: E2210: special-function register name: not defined).

Example

```
/* When there is no #pragma sfr directive* /
main() {
    /* When there is no #pragma sfr directive* /
}
```

➡ For details on a given special-function register, please refer to the user manual of the device being used.

Registering an Interrupt Function

#pragma interrupt or #pragma vect and __interrupt

An interrupt stops the currently executing program, and starts the execution of a different (interrupt) program. Once the interrupt program completes, the interrupted program resumes execution. Generating an interrupt is referred to as an interrupt request.

The processing done when an interrupt request occurs can be described as a function and, a particular function can be specified depending on the source of the interrupt. This type of function is referred to an interrupt function.

You need to perform the following to make a function into an interrupt function:

- associate the function name with the source of the interrupt (interrupt request name)
- specify the function as an interrupt function

To associate a function name with an interrupt request name, use the "#pragma interrupt" or "#pragma vect" pragma directive.

```
#pragma interrupt "interrupt request name" "function name"
```

```
#pragma vect "interrupt request name" "function name"
```

Using this pragma declaration, the function name is registered as an interrupt handler under the interrupt request name.

➡ For details on what interrupt request names can be assigned, please refer to the user manual of the device being used.

When you define a function as an interrupt function, you use the __interrupt modifier when you define the function (or declare the function).

```
__interrupt "function definition" or "function declaration"
```

A function defined as an interrupt function saves/restores both the interrupt registers and the normal registers. The function returns upon a reti instruction. A function that can be defined as an interrupt function typically has no arguments and has no return values (a "void Func(void)" type function).

For the slot machine program, the association of function name with interrupt request name is done in the source file, slot.c. The interrupt handler function setting is performed in the interrupt_func.h header file.

[slot.c]

```
- #pragma interrupt INTP0 stp_btn_Left
- #pragma interrupt INTP1 stp_btn_Center
- #pragma interrupt INTP2 stp_btn_Right
```

[interrupt_func.h]

```
- __interrupt void stp_btn_Left(void);
- __interrupt void stp_btn_Center(void);
- __interrupt void stp_btn_Right(void);
```

Enabling/Disabling Interrupts DI(); and EI();

When an interrupt occurs, the order in which the interrupt is handled depends on the interrupt's priority level. However, there are times when certain processing must proceed uninterrupted, which means that the maskable interrupts must be disabled for the duration of the processing. After the processing has completed, the interrupts are reenabled. This can be done using the C programming language.

Using interrupt control functions (DI/EI), you can enable and disable interrupts for particular sections of the program.

First, specify "#pragma DI" and "#pragma EI".

```
#pragma DI  
#pragma EI
```

```
DI();
```

The DI function disables interrupts (generates a di command).

```
EI();
```

The EI function enables interrupts (generates an ei command).

Example:

```
#pragma DI
#pragma EI

void func() {
    int i,j,k;
    .....
    DI(); /* * Disable interrupts */

    /*
    * Perform necessary processing while interrupts are disabled.
    */

    EI(); /* Enable interrupts */
    .....
    return;
}
```

For the slot machine program, an interrupt is generated when a button is clicked, so the interrupts should be enabled before the program starts cycling through the digit incrementation loop.

[slot.c]

```
/* Enable interrupts */
EI();
```

Outputting CPU Control Instructions HALT();, STOP();, BRK();, and NOP();

Instructions to control the CPU can be described in C language, in a function format.

Note that the BRK instruction is only supported by the 78K0 and 78K4.

Declare the use of these functions by using a #pragma directive.

```
#pragma HALT  
#pragma STOP  
#pragma BRK  
#pragma NOP
```

```
HALT();
```

In the 78K0 and 78K0S, the HALT function generates the halt instruction.

In the 78K4, the HALT function generates a code to manipulate STBC.

```
STOP();
```

In the 78K0 and 78K0S, the STOP function generates the stop instruction.

In the 78K4, the STOP function generates a code to manipulate STBC.

```
BRK();
```

In the 78K0 and 78K4, the BRK function generates the brk instruction.

```
NOP();
```

The NOP function generates the nop instruction.

Example:

```
#pragma HALT
#pragma STOP
#pragma BRK
#pragma NOP

void func() {
    .....
    HALT(); /* halt instruction output */
    STOP(); /* stop instruction output */
    BRK(); /* brk instruction output */
    NOP(); /* nop instruction output */

    .....
    return;
}
```

[Caution]

In the CC78K4, HALT() and STOP() check the values of CK1/CK0 in STBC and output an instruction that sets the corresponding values for HALT and STOP to STP/HLT.

(Only "MOV STBC,#value" can be set to STBC.)

As a result, an instruction that sets bits 2, 3, 6, and 7 of STBC to 0 is output.

Note that HALT() and STOP() cannot be used in devices in which bits 2, 3, 6, and 7 are not fixed to 0.

7	6	5	4	3	2	1	0
0	0	CK1	CK0	0	0	STP	HLT

Appendix

Here, as an appendix, the following topics, which were mentioned in the corresponding chapter, are described in greater detail.

- [Creating uoVRAM.dll](#)
- [Counter Program Source Listing](#)
 - [counter.c](#)
- [Slot Machine Program Source Listing](#)
 - [slot.c](#)
 - [interrupt_func.h](#)
 - [interrupt_func.c](#)

Creating uoVRAM.dll

This appendix describes how to create an external part (uoVRAM.dll) for the virtual VRAM program given in Chapter 2 using Microsoft Visual C++ (from here on referred to as VC++).

Note that a completed uoVRAM.dll is included in the Chapter 2 sample environment, so you do not have to perform the instructions here in order to follow the discussion in Chapter 2.

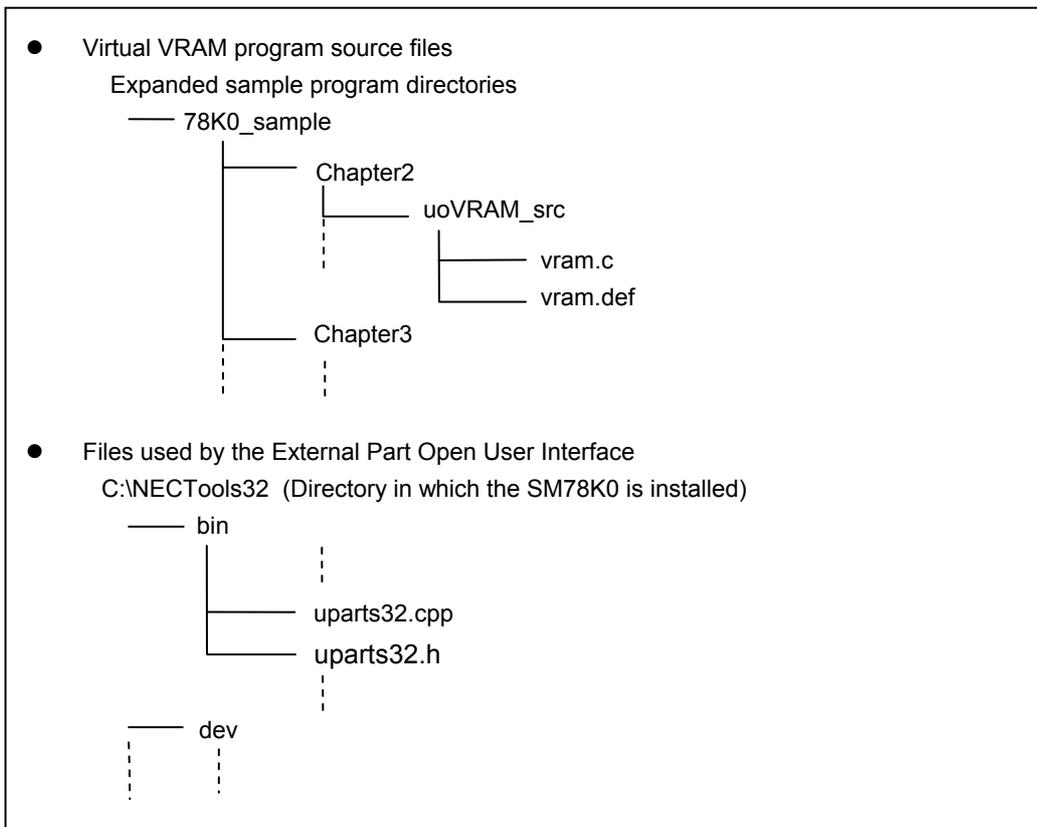
➔ This appendix provides you with a reference on how you can create external parts that use the External Part User Open Interface.

For additional details on the External Part User Open Interface, refer to either the **SM78K Series System Simulator Ver.2.30 or later External Part User Open Interface Specification User's Manual (U15802E)**.

The uoVRAM.dll source file is stored together with the other sample programs.

<Files used>

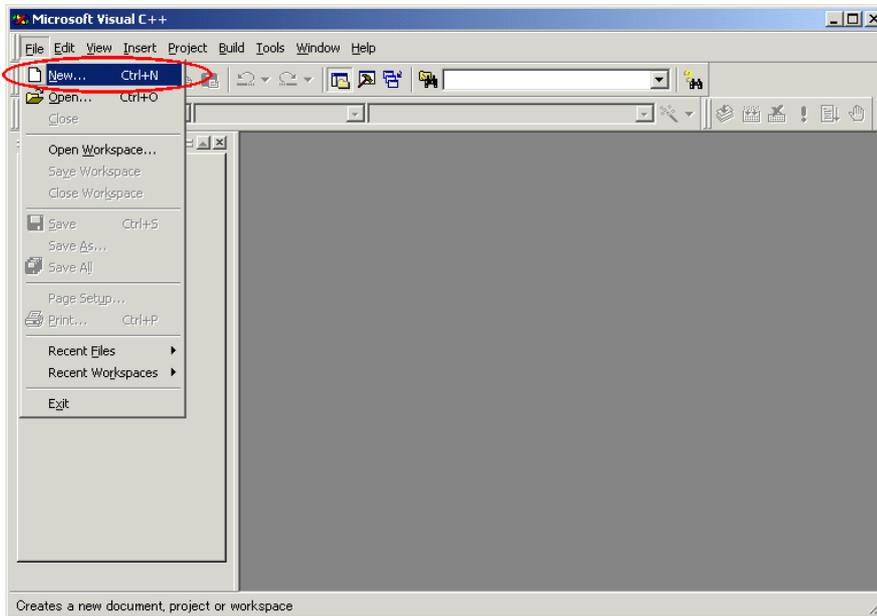
The following files are used to create uoVRAM.dll: source files vram.c and vram.def, found in the self-extracting compressed file; files used by the External Part Open User Interface, uparts32.cpp and uparts32.h, which are installed together with the SM78K0.



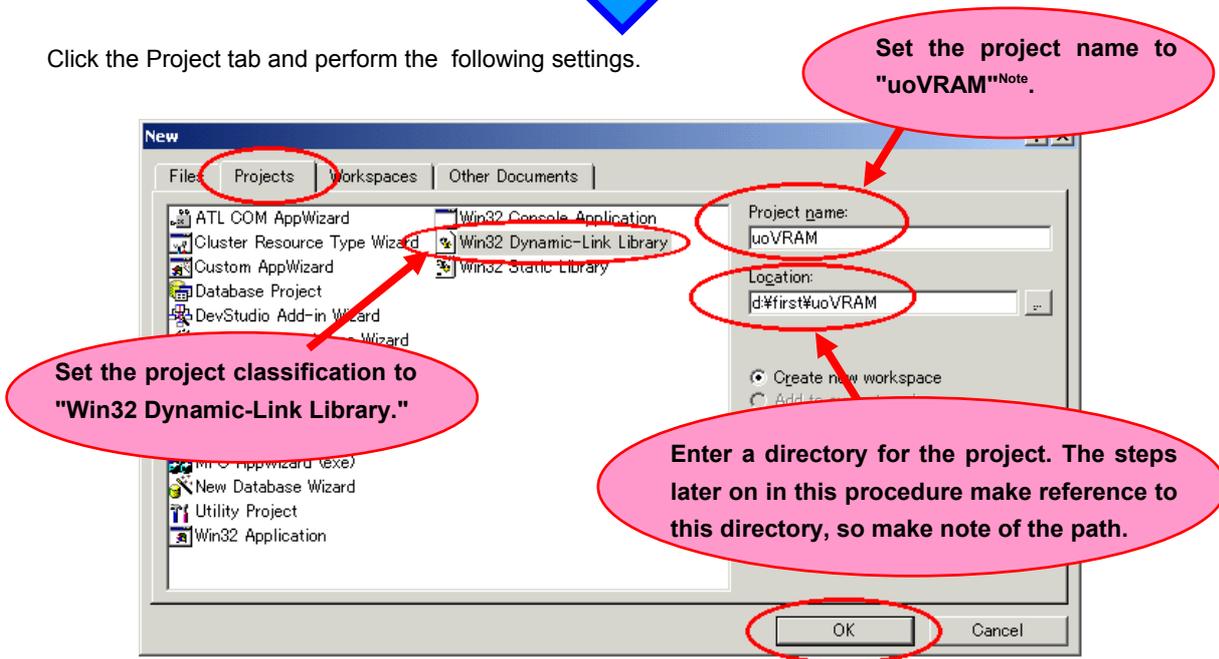
< Procedure for creating uoVRAM.dll >

Use the following procedure to build uoVRAM.dll using VC++ Ver.6.

1. Start VC++ and create a new "Win32 Dynamic-Link Library" project.
First, from the menus, select File->New...



Click the Project tab and perform the following settings.



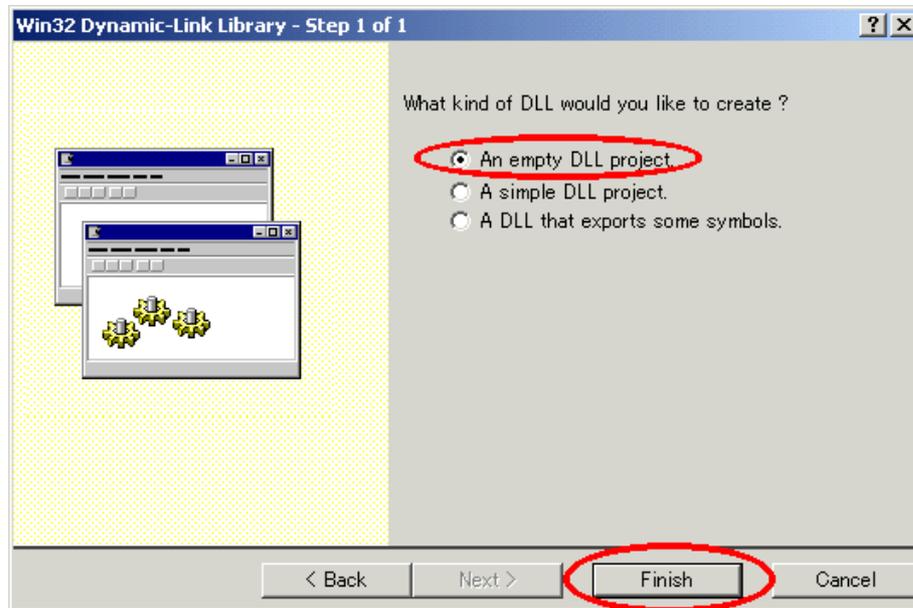
When you have completed the settings, click OK button.

Note Use "uoVRAM" for the project name. If you use a different project name, make sure to set the options in VC++ to name the output DLL file "uoVRAM.dll". For the SM78Kxx External Component Open User Interface, the DLL file name and the function name exported from the DLL must correspond. If you use the supplied source file as it is, then if the DLL file name is not uoVRAM.dll, the SM78Kxx will not be able to read the DLL properly. For additional details, refer to the **SM78K Series System Simulator Ver.2.30 or later External Part User Open Interface Specification User's Manual (U15802E)**.



Next, select the type of DLL.

Select "Empty DLL project."

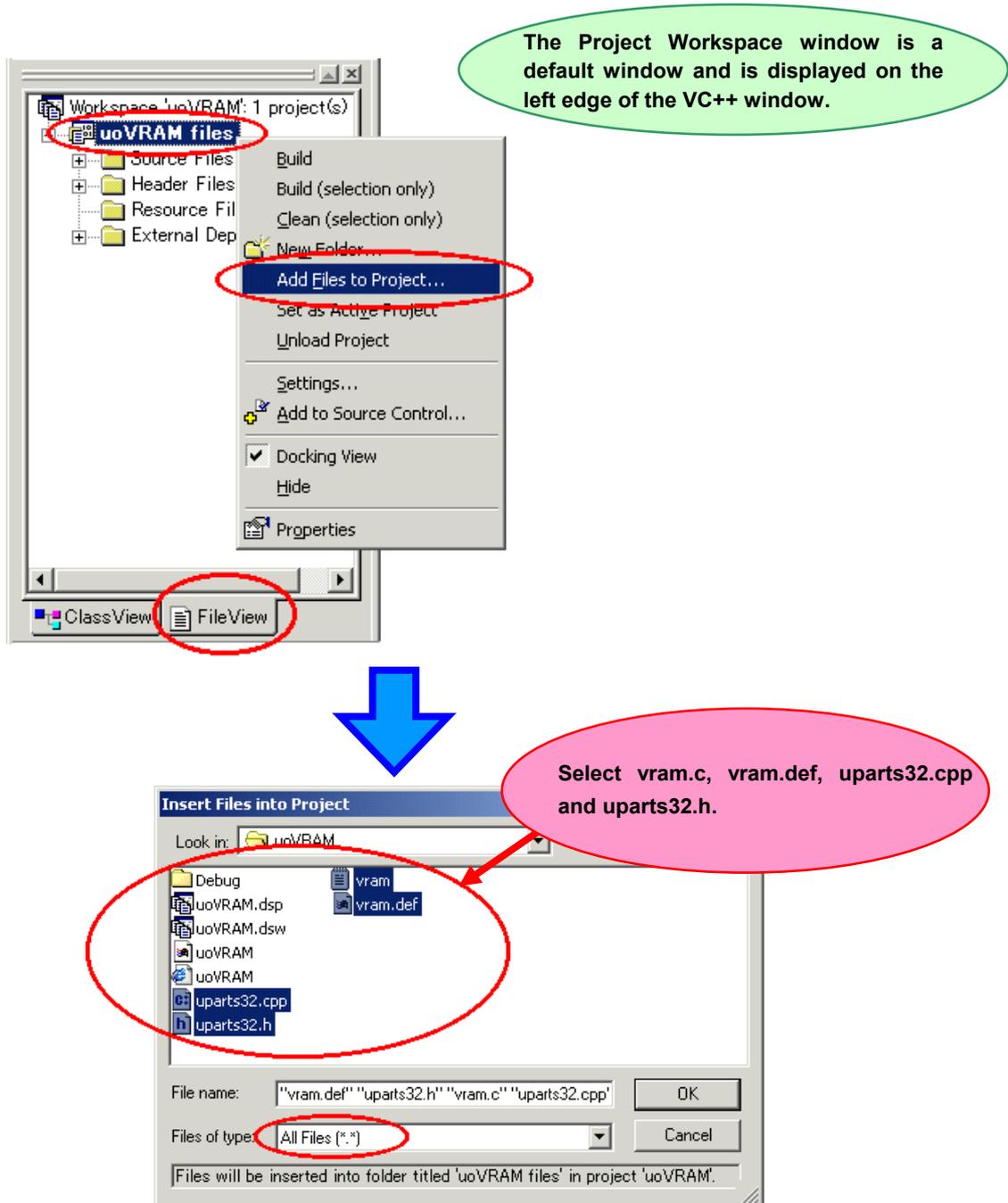


After making the selection, click Finish button.

2. Copy the virtual VRAM program source files (vram.c and vram.def) and the files used by the External Part User Open Interface (uparts32.cpp, uparts32.h) to the project directory you created in step 1 (for the locations of the files, refer to the "[Files used](#)" section above).
3. Next, you will register the files copied in step 2 (vram.c, vram.def, uparts32.cpp, uparts32.h) with the project you created in step 1. You will perform the operations in the VC++ Project Workspace window inside the File view tab.

First, click on the File view tab.

Then, right click on uoVRAM file and, from the pop-up menu, select Add Files to project.

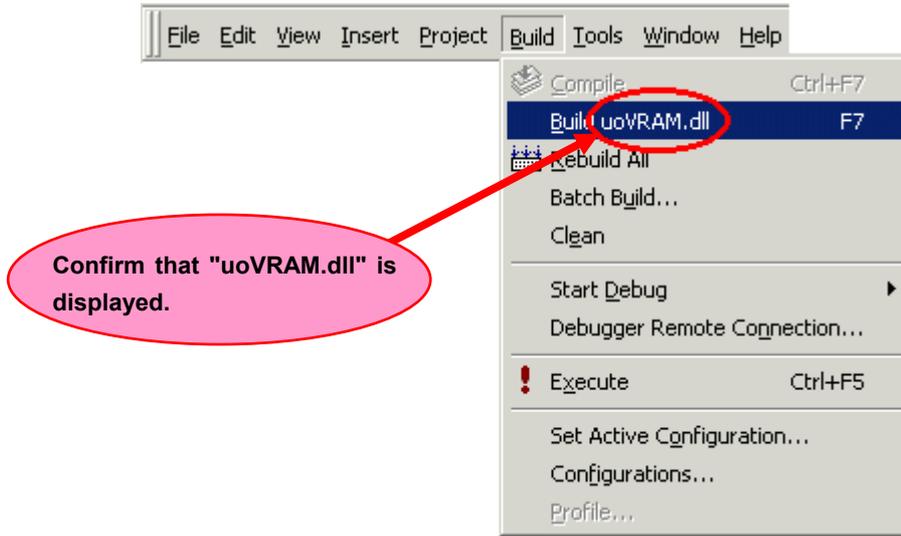


By setting the File type to All files, you can select all four files together.

After selecting the files, click OK button to register the files.

Next, you build the project.

From the menus, select Build->Build.



This completes the procedure for creating uoVRAM.dll.

Counter Program Source Listing

[counter.c]

78K0 Source Listing

(1/4)

```

/*
 * Copyright (C) NEC Electronics Corporation 2000,2004
 * All rights reserved by NEC Electronics Corporation. This program must be used solely
 * for the purpose for which it was furnished by NEC Electronics Corporation. No part
 * of this program may be reproduced or disclosed to others, in any form,
 * without the prior written permission of NEC Electronics Corporation.
 */

/*****
*****
*
* Counter program (for the uPD780034)
*
*****
*****/

/* Compiler definitions */
#pragma NOP
#pragma HALT
#pragma EI

/* Enable special-function register names (SFR names) */
#pragma sfr

/* Set the function interrupt1() as the interrupt function for INTTM00 */
#pragma vect INTTM00 interrupt1

/* Variables for storing counter values */
volatile int count1; /* Numerical value displayed on the LED (1s digit) */
volatile int count10; /* Numerical value displayed on the LED (10s digit)*/

/*****
*****
* Displays a numerical value on the 7-segment LED display.
*
* count1 stores the 1s digit and is displayed on the first LED digit.
* count10 stores the 10s digit and is displayed on the second LED digit.

```

```

* Function name: putLED
* Arguments: none
* Return values: none
* Global variables used:
*   int count1
*   int count10
*
*****
*****/
void putLED()
{
    static unsigned char /* 7-segment LED display '0' - '9'patterns */
        box[10]={0x77,0x24,0x6b,0x6d,0x3c,0x5d,0x5f,0x74,0x7f,0x7d};/*←ERROR*/

    /* Transfer the current pattern to the 1s digit of the 7-segment LED display */
    P4 = 0;
    P5 = box[count1];
    P4 = 1;

    /* Time adjustment */
    NOP();
    NOP();

    /* Transfer the current pattern to the 10s digit of the 7-segment LED display */
    P4 = 0;
    P5 = box[count10];
    P4 = 2;
    return;
}
/*****
*****
* Main function for debugging
*
* This function performs the following:
* ● Sets the modes of P4 and P5 using mode registers PM4 and PM5, to display on the
*   LED display.
* ● Initializes the counter variables (count1 and count10).
* ● Initializes the INTTM00 interrupt and enables interrupts.
* ● Puts the CPU in a HALT state until an INTTM00 interrupt is generated.
*   When the INTTM00 interrupt processing completes, puts the CPU in a HALT state
*   and waits for the next INTTM00 interrupt.
*
* Function name: main
* Arguments: none
* Return values: none
* Global variables used:
*   int count1
*   int count10
*****

```

```

*****/
void main(){
  /*****
   * Initialization
   *****/

  /* Set the mode of the port that outputs to the 7-segment LED display */
  PM5 = 0x00; /*Set P5 (P50-P57) to output mode */
  PM4 = 0x00; /*Set P4 (P40-P47) to output mode */

  /* Set the interrupt level of INTTM00 and unmask the interrupt */
  WTMK = 0; /* Unmask the INTTM00 interrupt */
  /* Initialize the counter */
  count1 = 0; /* Initialize the LED display digit value (1s digit) */
  count10 = 0; /* Initialize the LED display digit value (10s digit) */
  /* Display initial values */
  putLED(); /* Display numerical values on the LED display */
  /*****

   * Main loop
   *****/

  /* From this point, the processing is entirely interrupt driven.
   The CPU is in the HALT state when there is no interrupt processing going on. */
  EI(); /* Enable interrupts */
  while(1)
  {
    HALT(); /* */
  }
}
/*****
*****

* Incrementation routine
* (Interrupt function called by INTTM00)
*
* Increments the counter composed of variables count1 and count10
* When the count reaches 99, the counter rolls over to 0 on the next count.
* Also, the putLED function is used to display the count value on the LED display.
*
* Function name: interrupt1
* Arguments: none
* Return values: none
* Global variables used:
*   int count1
*   int count10
*
*****
*****/

__interrupt
void interrupt1()
{

```

```
/******  
 * Increments the counter  
*****/  
count1++; /* increment the 1s digit */  
  
/* Carry operation processing */  
  
/* Is a carry operation necessary? */  
if(count1=10) /*←ERROR*/  
{ /* The 1s digit is equal to 10, so process the carry */  
    count1 = 0; /* Set the 1s digit to 0 */  
    count10++; /* Increment the 10s digit */  
  
    /* Is there a carry from the 10s digit? */  
    if(count10==10)  
    {  
        /* The 10s digit is equal to 10, so process the carry */  
        count10 =0; /* Set the 10s digit to 0 */  
        /* Return the 2 LED display digits from 99 to 0 */  
    }  
}  
  
}  
  
/******  
 * Display the count value on the LED display  
*****/  
putLED(); /* Display the count value on the LED display */  
return;  
}
```

```

/*
 * Copyright I NEC Electronics Corporation 2000,2004
 * All rights reserved by NEC Electronics Corporation. This program must be used
 * solely for the purpose for which it was furnished by NEC Electronics Corporation. No
 * part of this program may be reproduced or disclosed to others, in any form,
 * without the prior written permission of NEC Electronics Corporation.
 */

/*****
*****
 *
 * Counter program (for the uPD789046)
 *
*****
*****/

/* Compiler definitions */
#pragma NOP
#pragma HALT
#pragma EI

/* Enable special-function register names (SFR names) */
#pragma sfr

/* Set the function interrupt1() as the interrupt function for INTWT */
#pragma vect INTWT interrupt1

/* Variables for storing counter values */
volatile int count1; /* Numerical value displayed on the LED (1s digit) */
volatile int count10; /* Numerical value displayed on the LED (10s digit)*/

/*****
*****
 * Displays a numerical value on the 7-segment LED display
 *
 * count1 stores the 1s digit and is displayed on the first LED digit.
 * count10 stores the 10s digit and is displayed on the second LED digit.
 * Function name: putLED
 * Arguments: none
 * Return values: none
 * Global variables used:
 * int count1
 * int count10
 * *****/
void putLED()
{
    static unsigned char /* 7-segment LED display '0' – '9' patterns */
        box[9]={0x77,0x24,0x6b,0x6d,0x3c,0x5d,0x5f,0x74,0x7f,0x7d};/*←ERROR*/
    /* Transfer the current pattern to the 1s digit of the 7-segment LED display */

```

```

    P0 = 0;
    P1 = box[count1];
    P0 = 1;

    /* Time adjustment */
    NOP();
    NOP();

    /* Transfer the current pattern to the 10s digit of the 7-segment LED display */
    P0 = 0;
    P1 = box[count10];
    P0 = 2;
    return;
}
/*****
*****
* Main function for debugging
*
* This function performs the following:
* ● Sets the modes of P0 and P1 using mode registers PM0 and PM1, to display on the
*   LED display.
* ● Initializes the counter variables (count1 and count10).
* ● Initializes the INTWT interrupt and enables interrupts.
* ● Puts the CPU in a HALT state until an INTWT interrupt is generated.
*   When the INTWT interrupt processing completes, puts the CPU in a HALT state
*   and waits for the next INTWT interrupt.
*
* Function name: main
* Arguments: none
* Return values: none
* Global variables used:
*   int count1
*   int count10
*
*****
*****/
void main(){
    /*****
    * Initialization
    *****/

    /* Set the mode of the port that outputs to the 7-segement LED display */
    PM1 = 0x00; /*Set P1 (P10-P17) to output mode */
    PM0 = 0x00; /*Set P0 (P00-P07) to output mode */

    /* Set the interrupt level of INTWT and unmask the interrupt */
    WTMK = 0; /* Unmask the INTWT interrupt */

```

```

/* Initialize the counter */
count1 = 0; /* Initialize the LED display digit value (1s digit) */
count10 = 0; /* Initialize the LED display digit value (10s digit) */

/* Display initial values */
putLED(); /* Display numerical values on the LED display */
/*****

* Main loop
*****/

/* From this point, the processing is entirely interrupt driven.
The CPU is in the HALT state when there is no interrupt processing. */
EI(); /* Enable interrupts */
while(1)
{
    HALT(); /* */
}
}
/*****
*****/

* Incrementation routine
* (Interrupt function called by INTWT)
*
* Increments the counter composed of variables count1 and count10.
* When the count reaches 99, the counter rolls over to 0 on the next count.
* Also, the putLED function is used to display the count value on the LED display.
*
* Function name: interrupt1
* Arguments: none
* Return values: none
* Global variables used:
*   int count1
*   int count10
*
*****/

__interrupt
void interrupt1()
{
    /*****
    * Increments the counter
    *****/
    count1++; /* increment the 1s digit */

    /* Carry operation processing */

```

```
/* Is a carry operation necessary? */
if(count1=10)                /*←ERROR*/
{ /* The 1s digit is equal to 10, so process the carry */
    count1 = 0; /* Set the 1s digit to 0 */
    count10++; /* Increment the 10s digit */

    /* Is there a carry from the 10s digit? */
    if(count10==10)
    {
        /* The 10s digit is equal to 10, so process the carry */
        count10 =0; /* Set the 10s digit to 0 */
        /* Return the 2 LED display digits from 99 to 0 */
    }
}

/*****
 * Display the count value on the LED display
 *****/
putLED(); /* Display the count value on the LED display */
return;
}
```

```

/*
 * Copyright (C) NEC Electronics Corporation 2000,2004
 * All rights reserved by NEC Electronics Corporation. This program must be used solely
 * for the purpose for which it was furnished by NEC Electronics Corporation. No part
 * of this program may be reproduced or disclosed to others, in any form,
 * without the prior written permission of NEC Electronics Corporation.
 */

/*****
*****
 *
 * Counter program (for the uPD784035)
 *
*****
*****/

/* Compiler definitions */
#pragma NOP
#pragma HALT
#pragma EI

/* Enable special-function register names (SFR names) */
#pragma sfr

/* Set the function interrupt1() as the interrupt function for INTC00 */
#pragma vect INTC00 interrupt1

/* Variables for storing counter values */
volatile int count1; /* Numerical value displayed on the LED (1s digit) */
volatile int count10; /* Numerical value displayed on the LED (10s digit)*/

/*****
*****
 * Displays a numerical value on the 7-segment LED display
 *
 * count1 stores the 1s digit and is displayed on the first LED digit.
 * count10 stores the 10s digit and is displayed on the second LED digit.
 * Function name: putLED
 * Arguments: none
 * Return values: none
 * Global variables used:
 *   int count1
 *   int count10
 *
*****
*****/

```

```

*****/
void putLED()
{
    static unsigned char /* 7-segment LED display '0' - '9'patterns */
        box[9]={0x77,0x24,0x6b,0x6d,0x3c,0x5d,0x5f,0x74,0x7f,0x7d};/*←ERROR*/

    /* Transfer the current pattern to the 1s digit of the 7-segment LED display */
    P4 = 0;
    P5 = box[count1];
    P4 = 1;
    /* Time adjustment */
    NOP();
    NOP();

    /* Transfer the current pattern to the 10s digit of the 7-segment LED display */
    P4 = 0;
    P5 = box[count10];
    P4 = 2;
    return;
}
/*****
*****
* Main function for debugging
*
* This function performs the following:
* ●Sets the modes of P4 and P5 using mode registers PM4 and PM5, to display on the
* LED display.
* ●Initializes the counter variables (count1 and count10).
* ●Initializes the INTC00 interrupt and enables interrupts.
* ●Puts the CPU in a HALT state until an INTTM00 interrupt is generated.
* When theINTC00 interrupt processing completes, puts the CPU in a HALT state
* and waits for the next INTC00 interrupt.
*
* Function name: main
* Arguments: none
* Return values: none
* Global variables used:
*     int count1
*     int count10
*
*****
*****/
void main(){
    /*****
    * Initialization
    *****/

    /* Set the mode of the port that outputs to the 7-segement LED display */
    PM5 = 0x00; /*Set P5 (P50-P57) to output mode */
    PM4 = 0x00; /*Set P4 (P40-P47) to output mode */

```

```

/* Set the interrupt level of INTC00 and unmask the interrupt */
CIC00 = CIC00 & 0xF0; /* Make the INTC00 interrupt level high priority (CPR001=0
and CPR000=0) */
CMK00 = 0; /* Unmask the INTC00 interrupt */

/* Initialize the counter */
count1 = 0; /* Initialize the LED display digit value (1s digit) */
count10 = 0; /* Initialize the LED display digit value (10s digit) */

/* Display initial values */
putLED(); /* Display numerical values on the LED display */

/*****
* Main loop
*****/

/* From this point, the processing is entirely interrupt driven.
The CPU is in the HALT state when there is no interrupt processing going on. */
EI(); /* Enable interrupts */
while(1)
{
    HALT(); /* */
}
}

/*****
*****/

* Incrementation routine
* (Interrupt function called by INTC00)
*
* Increments the counter composed of variables count1 and count10
* When the count reaches 99, the counter rolls over to 0 on the next count.
* Also the putLED function is used to display the count value on the LED display.
*
* Function name: interrupt1
* Arguments: none
* Return values: none
* Global variables used:
*   int count1
*   int count10
*
*****/

```

```
__interrupt
void interrupt1()
{
    /******
    * Increments the counter
    *****/
    count1++; /* increment the 1s digit */

    /* Carry operation processing */

    /* Is a carry operation necessary? */
    if(count1==10) /*←ERROR*/
    { /* The 1s digit is equal to 10, so process the carry */
        count1 = 0; /* Set the 1s digit to 0 */
        count10++; /* Increment the 10s digit */

        /* Is there a carry from the 10s digit? */
        if(count10==10)
        {
            /* The 10s digit is equal to 10, so process the carry */
            count10 = 0; /* Set the 10s digit to 0 */
            /* Return the 2 LED display digits from 99 to 0 */
        }
    }

    /******
    * Display the count value on the LED display
    *****/
    putLED(); /* Display the count value on the LED display */
    return;
}
```

Slot Machine Program Source Listing

[slot.c]

78K0 Source Listing

(1/4)

```

/*
 * Copyright (C) NEC Electronics Corporation 2000,2004
 * All rights reserved by NEC Electronics Corporation. This program must be used solely
 * for the purpose for which it was furnished by NEC Electronics Corporation. No part
 * of this program may be reproduced or disclosed to others, in any form,
 * without the prior written permission of NEC Electronics Corporation.
 */

/*****
 * Slot machine program (for the uPD780034)
 *****/

/* Compiler definitions */
#pragma EI

/* Enable special-function register names (SFR names) */
#pragma sfr

/* Assign the functions stp_btn_Left(), stp_btn_Center(), and stp_btn_Right()
 * as interrupt functions for INTP0, INTP1, and INTP2 interrupts, respectively */
#pragma interrupt INTP0 stp_btn_Left
#pragma interrupt INTP1 stp_btn_Center
#pragma interrupt INTP2 stp_btn_Right

#include "interrupt_func.h" /* Interrupt function declaration */

/* Position (LED display digit to be lit) of the display */
unsigned char place;

/* Numerical data for the display (for lighting the LED display segments) */
unsigned char num_data[10]
    = { 0x77, 0x24, 0x6b, 0x6d, 0x3c, 0x5d, 0x5f, 0x74, 0x7f, 0x7d, };
    /* '0' '1' '2' '3' '4' '5' '6' '7' '8' '9' */

/* Function for initializing ports, interrupt levels, and CPU environment */
void init_target(void);

/* Slot machine display function */
void slot(void);

```

```

/*****
* Slot machine main function
* Loop through the display digits (0-9), displaying each digit on the LED display.
* When an interrupt occurs inside the loop, call the proper interrupt function
* and freeze the current display digit.
*
* Function name: main
* Arguments: none
* Return values: none
* Global variables used:
*   unsigned char place;
*****/
void main(void)
{

    /* Initialize the target CPU environment */
    init_target();

    /* Initialize the display digit position variable */
    place = 0x15; /* 10101: use 3 sets of LED digits; display on every second digit */

    /* Turn off the 7-segment display */
    P4 = 0x00;
    P5 = 0x00;
    P4 = 0xFF;

    /* Slot machine processing (lighting the digits) */
    slot();

} /* main */

/*****
* Function for initializing ports, interrupt levels, and CPU environment
*
* Function name: init_target
* Arguments: none
* Return values: none
* Global variables used: none
*****/
void init_target(void){
    /*
    * Use Port0 for the interrupt input.
    * Use Port5 for lighting the LED display digit.
    * Use Port4 for specifying the digit position.
    */

    /* Set all Port0 bits to input mode */
    PM0 = 0xFF; /* Set all mode register (PM0) bits to input (1) */

```

```

/* Set all Port5 bits to output mode */
PM5 = 0x00; /* Set all mode register (PM5) bits to output (0) */

```

```

/* Set all Port4 bits to output mode */
PM4 = 0x00; /* Set all mode register (PM4) bits to output (0) */

```

```

/* To be able to use Port4 and Port5 in output mode,
 * set the extended memory mode register to port mode */
MEM = 0x00;

```

```

/*
 * The active edge for triggering an external input on the external input terminals
 * is set to a positive edge by the external interrupt positive edge enable register (EGP)
 * or to a negative edge by the external interrupt negative edge enable register (EGN).
 *
 * In this program, to make the external interrupt requests work with INTP0, INTP1 and
 * INTP2, each interrupt is set to be positive edge triggered using EGP and
 * the negative edge triggers are disabled (using EGN).
 */

```

```

EGP = 0x07; /* 0x07 = X X X X 0 1 1 1
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */

```

```

EGN = 0x00; /* 0x00 = X X X X 0 0 0 0
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */

```

```

* EGP | EGN |
* -----
* 0 | 0 | Interrupt disabled
* 0 | 1 | Negative edge
* 1 | 0 | Positive edge
* 1 | 1 | Positive and negative edge
*/

```

```

PMK0 = 0; /* Unmask INTP0 interrupt */
PMK1 = 0; /* Unmask INTP1 interrupt */
PMK2 = 0; /* Unmask INTP2 interrupt */

} /* init_target */

/*****
 * Slot machine display function
 * Loop through the display digits (0-9), displaying each digit on the LED display.
 * When an interrupt occurs inside the loop, call the proper interrupt function
 * and freeze the current display digit.
 *
 * Function name: slot
 * Arguments: none
 * Return values: none
 * Global variables used:
 *   unsigned char place;
 *   unsigned int num_data[];
 *****/
void slot(void) {
    /*
     * Loop through the display digits (0-9).
     * The place variable specifies the display digit.
     */

    /* Index for the display value (num_data) */
    int num_idx = 0;

    /* Enable interrupts */
    EI();

    while (1) { /* Infinite loop */

        /* Display the digit */
        P5 = num_data[num_idx];
        P4 = place;

        num_idx++;

        /* There are 10 elements in num_data; when num_idx reaches 10
         * the value of the index must be set back to 0. */
        if( num_idx >= 10 ) {
            num_idx = 0;
        }

    } /* While */

} /* slot */

```

```

/*
 * Copyright (C) NEC Electronics Corporation 2000,2004
 * All rights reserved by NEC Electronics Corporation. This program must be used solely
 * for the purpose for which it was furnished by NEC Electronics Corporation. No part
 * of this program may be reproduced or disclosed to others, in any form,
 * without the prior written permission of NEC Electronics Corporation.
 */

/*****
 * Slot machine program (for the uPD789046)
 *****/

/* Compiler definitions */
#pragma EI

/* Enable special-function register names (SFR names) */
#pragma sfr

/* Assign the functions stp_btn_Left(), stp_btn_Center(), and stp_btn_Right()
 * as interrupt functions for INTP0, INTP1, and INTP2 interrupts, respectively */
#pragma interrupt INTP0 stp_btn_Left
#pragma interrupt INTP1 stp_btn_Center
#pragma interrupt INTP2 stp_btn_Right

#include "interrupt_func.h" /* Interrupt function declaration */

/* Position (LED display digit to be lit) of the display */
unsigned char place;

/* Numerical data for the display (for lighting the LED display segments) */
unsigned char num_data[10]
    = { 0x77, 0x24, 0x6b, 0x6d, 0x3c, 0x5d, 0x5f, 0x74, 0x7f, 0x7d, };
    /* '0' '1' '2' '3' '4' '5' '6' '7' '8' '9' */

/* Function for initializing ports, interrupt levels, and CPU environment */
void init_target(void);

/* Slot machine display function */
void slot(void);

/*****
 * Slot machine main function
 * Loop through the display digits (0-9), displaying each digit on the LED display.
 * When an interrupt occurs inside the loop, call the proper interrupt function
 * and freeze the current display digit.
 *
 * Function name: main
 * Arguments: none
 * Return values: none
 *****/

```

```

* Global variables used:
*   unsigned char place;
*****/
void main(void)
{

/* Initialize the target CPU environment */
init_target();

/* Initialize the display digit position variable */
place = 0x15; /* 10101: use 3 sets of LED digits; display on every second digit */

/* Turn off the 7-segment display */
P0 = 0x00;
P1 = 0x00;
P0 = 0xFF;

/* Slot machine processing (lighting the digits) */
slot();

} /* main */

/*****
* Function for initializing ports, interrupt levels, and CPU environment
*
* Function name: init_target
* Arguments: none
* Return values: none
* Global variables used: none
*****/
void init_target(void){
/*
* Use Port2 for the interrupt input.
* Use Port1 for lighting the LED display digit.
* Use Port0 for specifying the digit position.
*/

/* Set all Port2 bits to input mode */
PM2 = 0xFF; /* Set all mode register (PM2) bits to input (1) */

/* Set all Port1 bits to output mode */
PM1 = 0x00; /* Set all mode register (PM1) bits to output (0) */

/* Set all Port0 bits to output mode */
PM0 = 0x00; /* Set all mode register (PM0) bits to output (0) */

```



```
/* Index for the display value (num_data) */
int num_idx = 0;

/* Enable interrupts */
EI();

while (1) { /* Infinite loop */

    /* Display the digit */
    P0 = 0x00;
    P1 = num_data[num_idx];
    P0 = place;

    num_idx++;

    /* There are 10 elements in num_data; when num_idx reaches 10
    * the value of the index must be set back to 0. */
    if( num_idx >= 10 ) {
        num_idx = 0;
    }

} /* While */

} /* slot */
```

```

/*
 * Copyright (C) NEC Electronics Corporation 2000,2004
 * All rights reserved by NEC Electronics Corporation. This program must be used solely
 * for the purpose for which it was furnished by NEC Electronics Corporation. No part
 * of this program may be reproduced or disclosed to others, in any form,
 * without the prior written permission of NEC Electronics Corporation.
 */

/*****
 * Slot machine program (for the uPD784035)
 *****/

/* Compiler definitions */
#pragma EI

/* Enable special-function register names (SFR names) */
#pragma sfr

/* Assign the functions stp_btn_Left(), stp_btn_Center(), and stp_btn_Right()
 * as interrupt functions for INTP0, INTP1, and INTP2 interrupts, respectively */
#pragma interrupt INTP0 stp_btn_Left
#pragma interrupt INTP1 stp_btn_Center
#pragma interrupt INTP2 stp_btn_Right

#include "interrupt_func.h" /* Interrupt function declaration */

/* Position (LED display digit to be lit) of the display */
unsigned char place;

/* Numerical data for the display (for lighting the LED display segments) */
unsigned char num_data[10]
    = { 0x77, 0x24, 0x6b, 0x6d, 0x3c, 0x5d, 0x5f, 0x74, 0x7f, 0x7d, };
    /* '0' '1' '2' '3' '4' '5' '6' '7' '8' '9' */

/* Function for initializing ports, interrupt levels, and CPU environment */
void init_target(void);

/* Slot machine display function */
void slot(void);

/*****
 * Slot machine main function
 * Loop through the display digits (0-9), displaying each digit on the LED display.
 * When an interrupt occurs inside the loop, call the proper interrupt function
 * and freeze the current display digit.
 *
 * Function name: main
 * Arguments: none
 * Return values: none
 *****/

```

```

* Global variables used:
*   unsigned char place;
*****/
void main(void)
{

/* Initialize the target CPU environment */
init_target();

/* Initialize the display digit position variable */
place = 0x15; /* 10101: use 3 sets of LED digits; display on every second digit */

/* Turn off the 7-segment display */
P4 = 0x00;
P5 = 0x00;
P4 = 0xFF;

/* Slot machine processing (lighting the digits) */
slot();

} /* main */

/*****
* Function for initializing ports, interrupt levels, and CPU environment
*
* Function name: init_target
* Arguments: none
* Return values: none
* Global variables used: none
*****/
void init_target(void){
/*
* Use Port2 for the interrupt input.
* Use Port5 for lighting the LED display digit.
* Use Port4 for specifying the digit position.
*/

/* Since Port2 is a dedicated input port, no mode setting is required. */

/* Set all Port5 bits to output mode */
PM5 = 0x00; /* Set all mode register (PM5) bits to output (0) */

/* Set all Port4 bits to output mode */
PM4 = 0x00; /* Set all mode register (PM4) bits to output (0) */

/* To be able to use Port4 and Port5 in output mode
* set the extended memory mode register to port mode */
MM = 0x00;

```



```
/* Index for the display value (num_data) */
int num_idx = 0;

/* Enable interrupts */
EI();

while (1) { /* Infinite loop */

    /* Display the digit */
    P4 = 0x00;
    P5 = num_data[num_idx];
    P4 = place;

    num_idx++;

    /* There are 10 elements in num_data; when num_idx reaches 10
    * the value of the index must be set back to 0. */
    if( num_idx >= 10 ) {
        num_idx = 0;
    }

} /* While */

} /* slot */
```

[interrupt_func.h]

```
/*
 * Copyright (C) NEC Electronics Corporation 2000,2004
 * All rights reserved by NEC Electronics Corporation. This program must be used solely
 * for the purpose for which it was furnished by NEC Electronics Corporation. No part
 * of this program may be reproduced or disclosed to others, in any form,
 * without the prior written permission of NEC Electronics Corporation.
 */

/* Function to stop the slot machine digits */

/* The __interrupt modifier declares
 * stp_btn_Left(), stp_btn_Center(), and stp_btn_Right()
 * as interrupt functions.
 */
__interrupt void stp_btn_Left(void);
__interrupt void stp_btn_Center(void);
__interrupt void stp_btn_Right(void);
```

[interrupt_func.c]

```

/*
 * Copyright (C) NEC Electronics Corporation 2000,2004
 * All rights reserved by NEC Electronics Corporation. This program must be used solely
 * for the purpose for which it was furnished by NEC Electronics Corporation. No part
 * of this program may be reproduced or disclosed to others, in any form,
 * without the prior written permission of NEC Electronics Corporation.
 */

#include "interrupt_func.h"

extern unsigned char place; /* Specify the display digit position */

/* For the 7-segment LED display, when the outputs of the terminals
 * allocated to the digit signals (in this program port5) are active
 * the corresponding LED can be turned on or off.
 * The terminal output information (value to be displayed) is received, and the value is
 * displayed
 * until the value changes.
 */

void stp_btn_Left(void) {
    /*
     * Freeze the display of the left-most digit with its current value.
     * (Make the number on the Input/Output Panel appear to have stopped.)
     */

    /* Indicate which digit to stop by setting
     * the corresponding bit of the digit position variable (place) to 0. */
    place = place & 0xEF; /* 0xEF = 1110 1111 */
}

void stp_btn_Center(void) {
    /* Freeze the display of the middle digit with its current value. */
    place = place & 0xFB; /* 0xFB = 1111 1011 */
}

void stp_btn_Right(void) {
    /* Freeze the display of the right-most digit with its current value. */
    place = place & 0xFE; /* 0xFE = 1111 1110 */
}

*/

```