To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

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

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

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

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

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

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

RENESAS

Application Note

78K Series Development Tools

Tutorial Guide

Target Devices

SP78K0 Ver.2.00
SP78K0S Ver.2.00
SP78K4 Ver.2.00

Document No. U17047EJ1V0AN00 (1st edition)
Date Published March 2004 NS CP(K)

[MEMO]

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

- **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.
Hong Kong
Tel: 2886-9318

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.
μPD780024A, 780034A, 780024AY, 780034AY Subseries	U14046E	9

Document Name	Document No	No.
μPD789046 Subseries	U13600E	9

Document Name	Document No	No.
μPD 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

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.

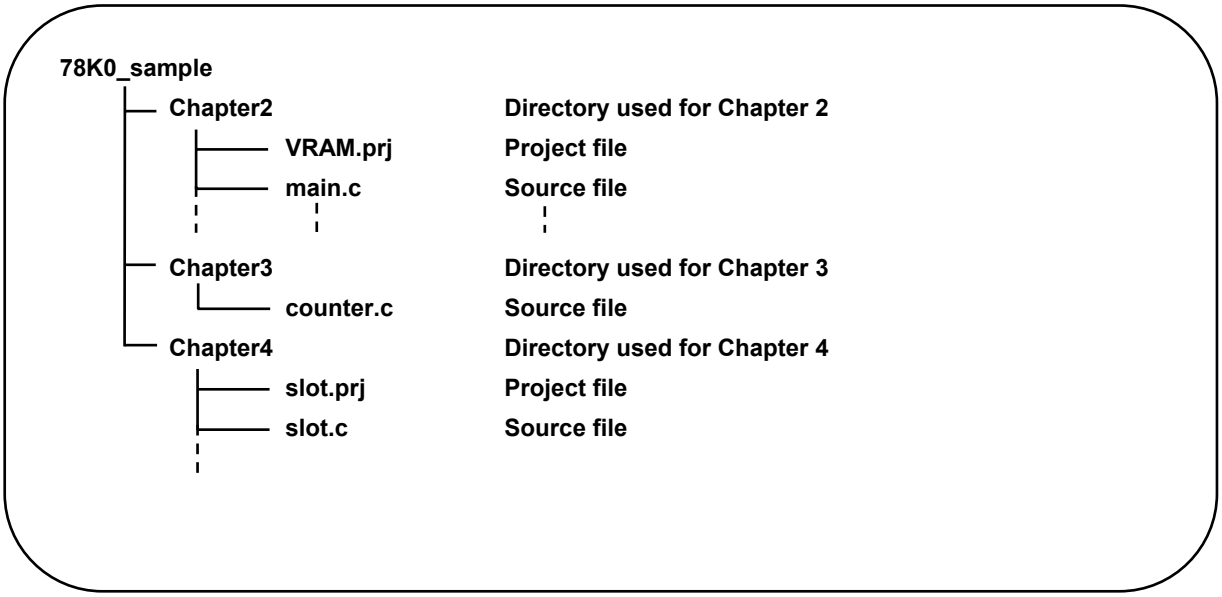
In order to be able to execute the sample programs found in this tutorial, you must install the above-mentioned development tools.
For instructions on how to install the development tools, please refer to the document "Important notes about the SP78Kxx 78K Series Software Package" included with the SP78Kxx 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.

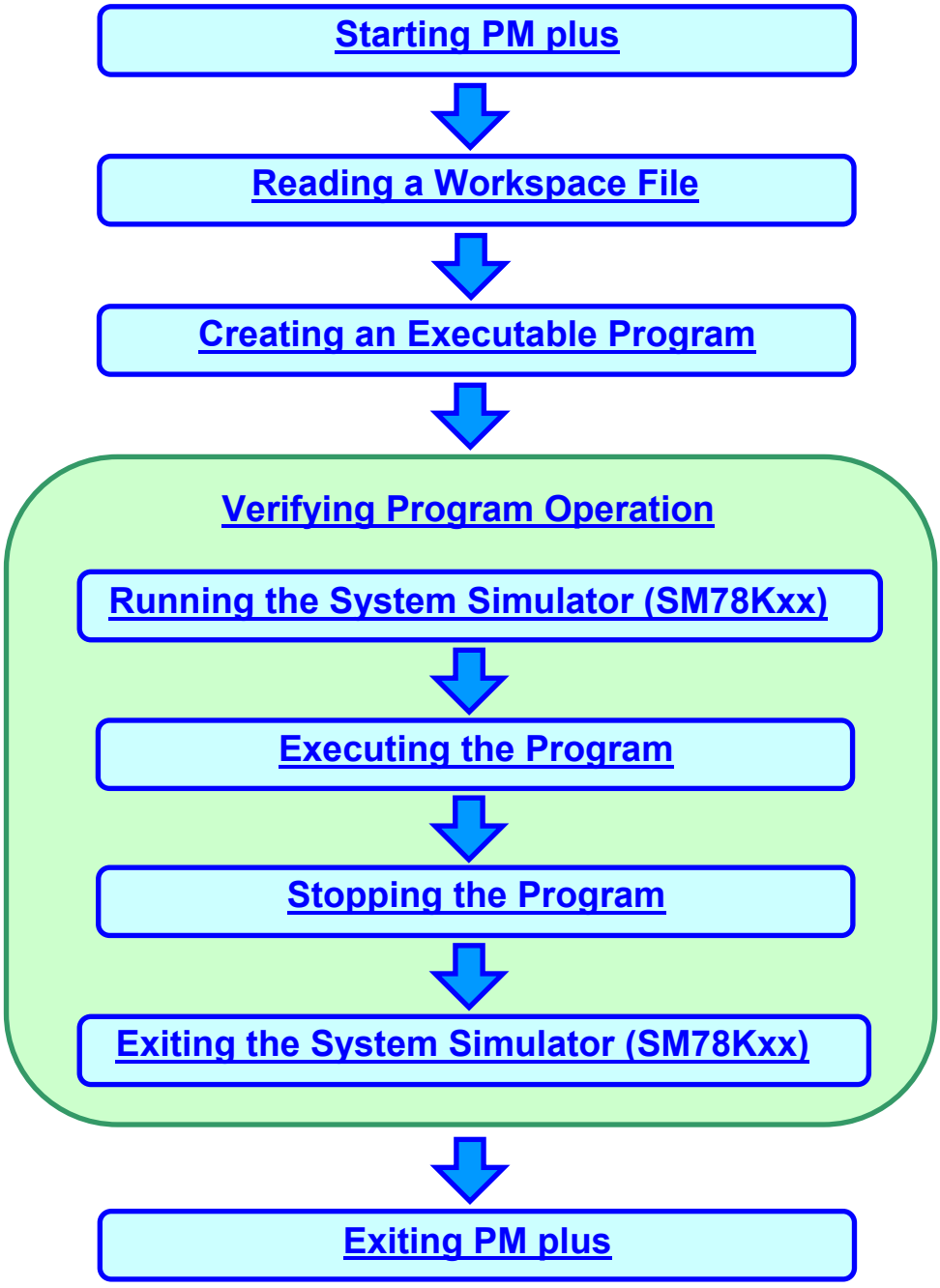


For the 78K0S, there is no Chapter2 directory.

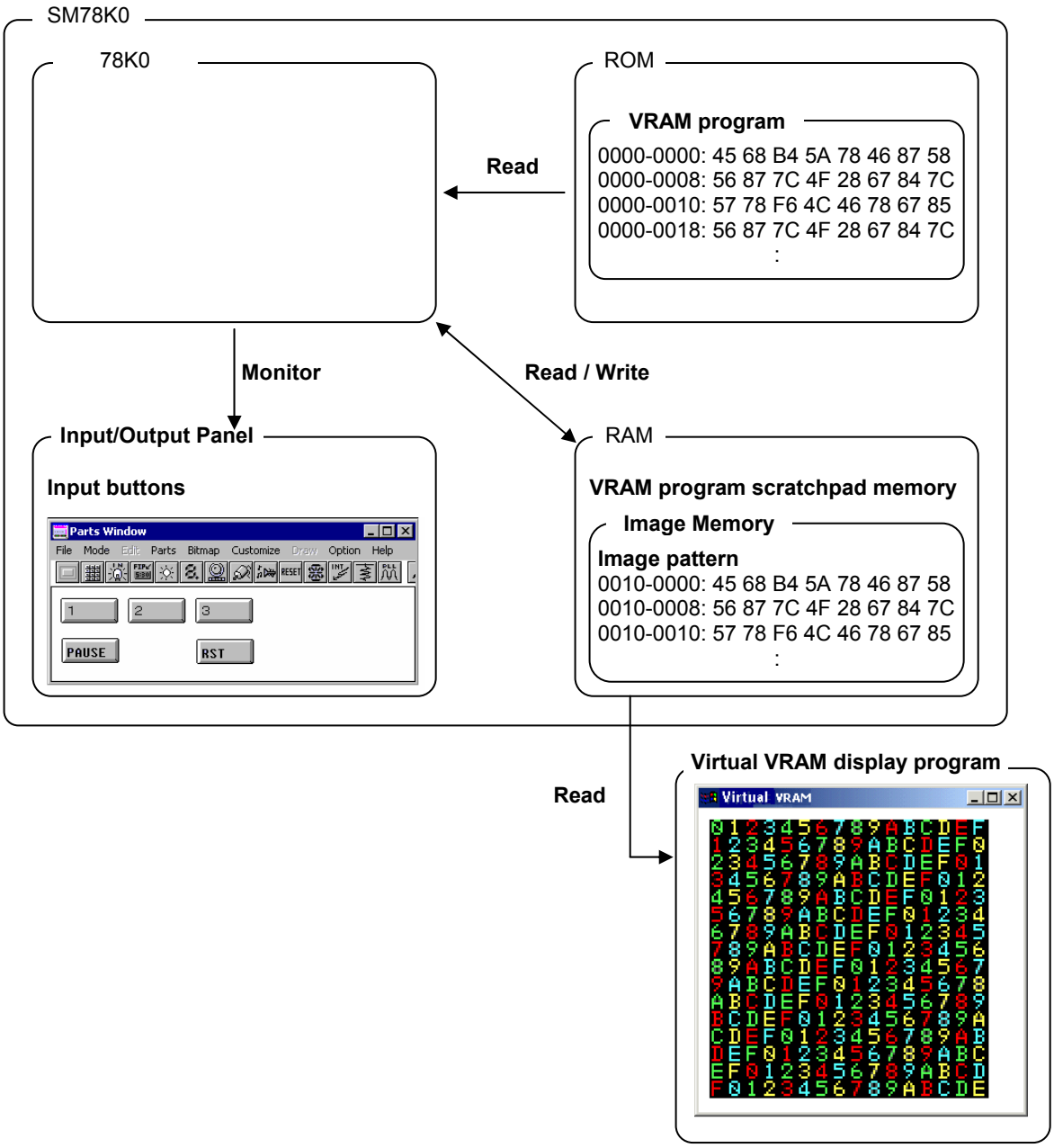
Chapter 2 Trying Out PM plus and Simulator

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:

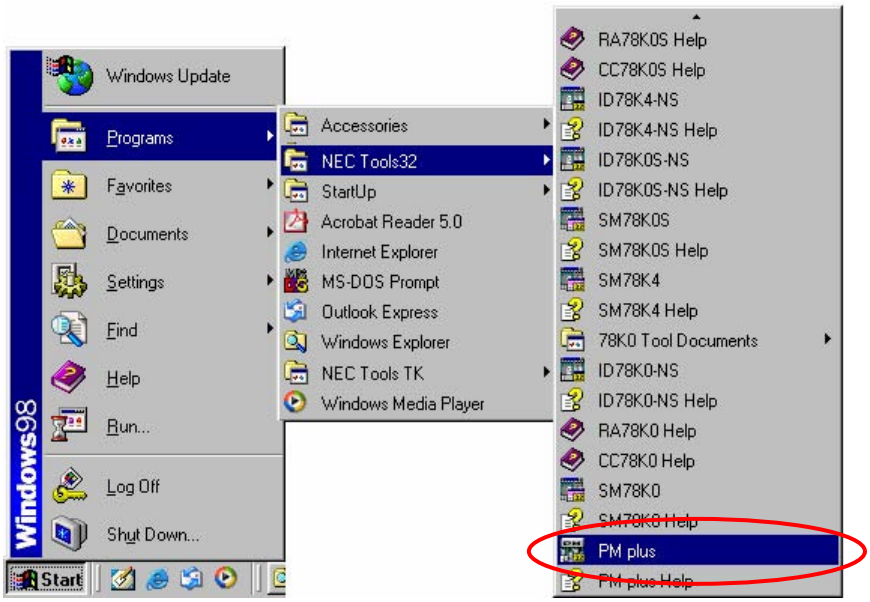
User-defined external parts for the SM78Kxx.

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)**).

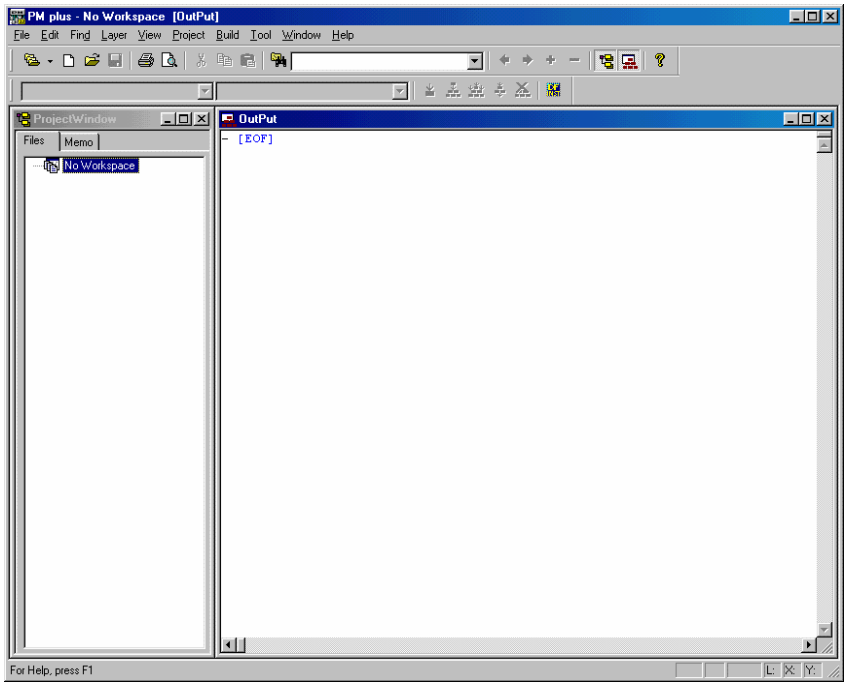
Starting PM plus

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

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

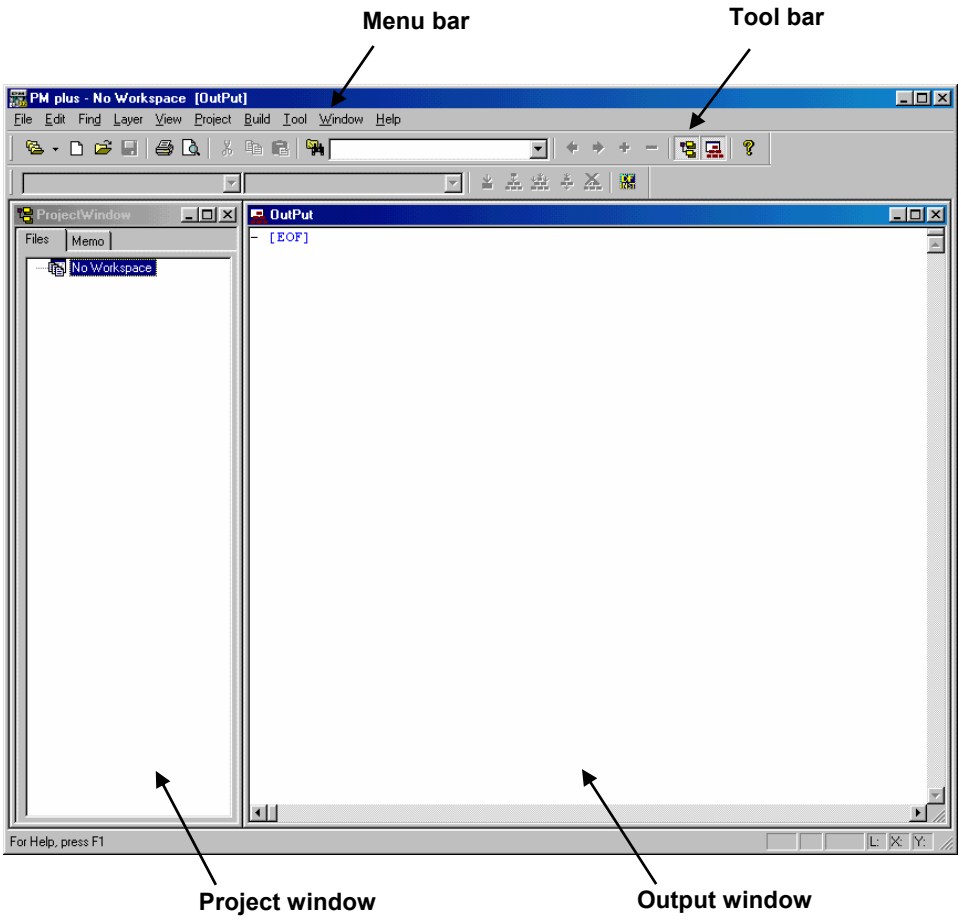


PM plus starts.



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.

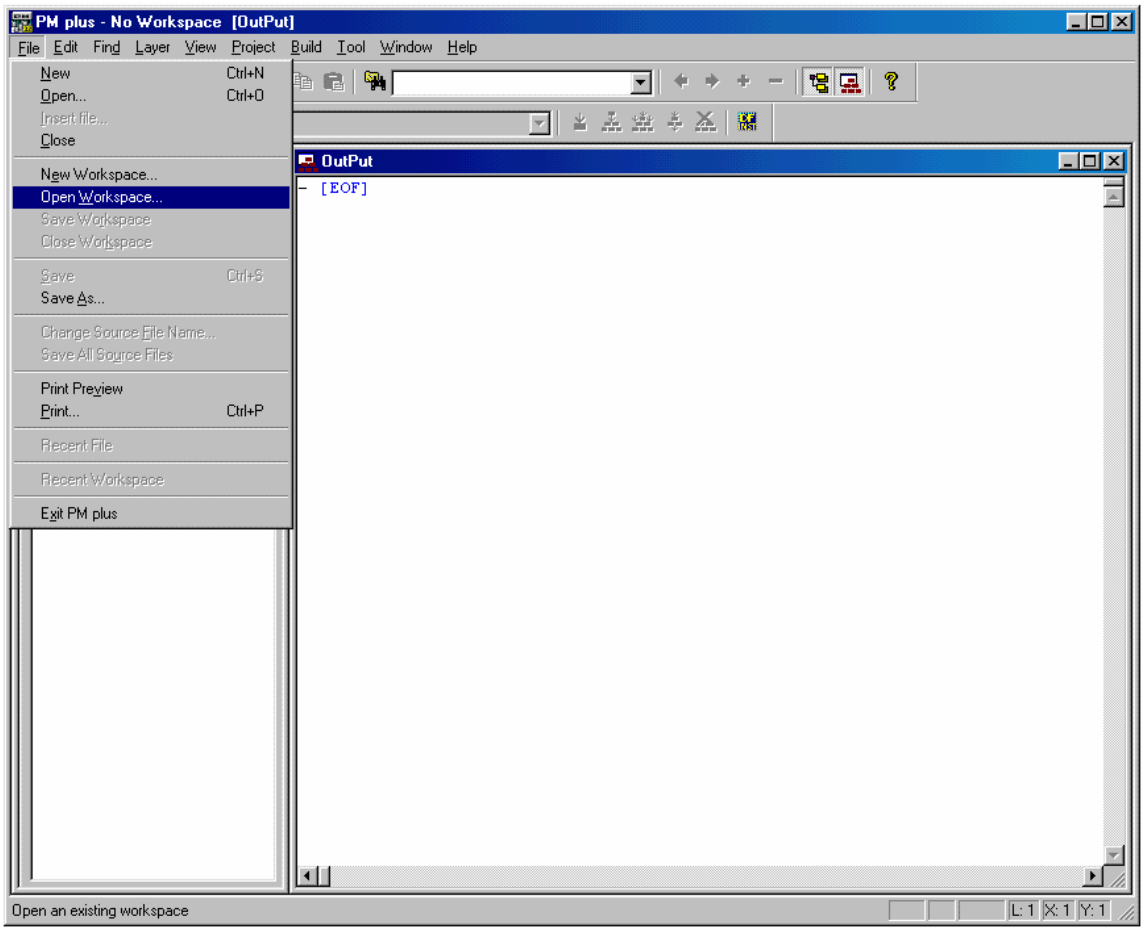
Details on how to create workspace and project files are found in Chapter 3 - System Simulator Basics.

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.

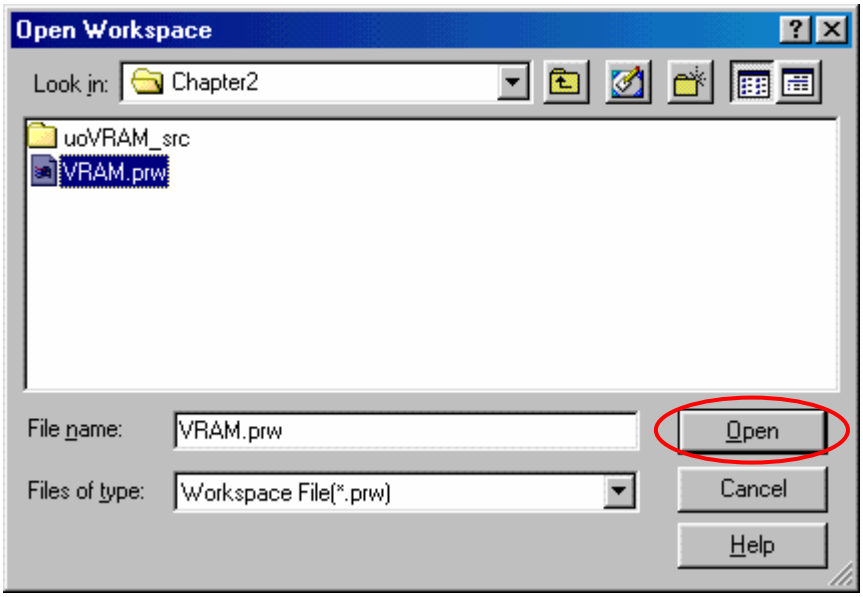
In order to start running the VRAM program, you must first open the project file in PM plus.

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.

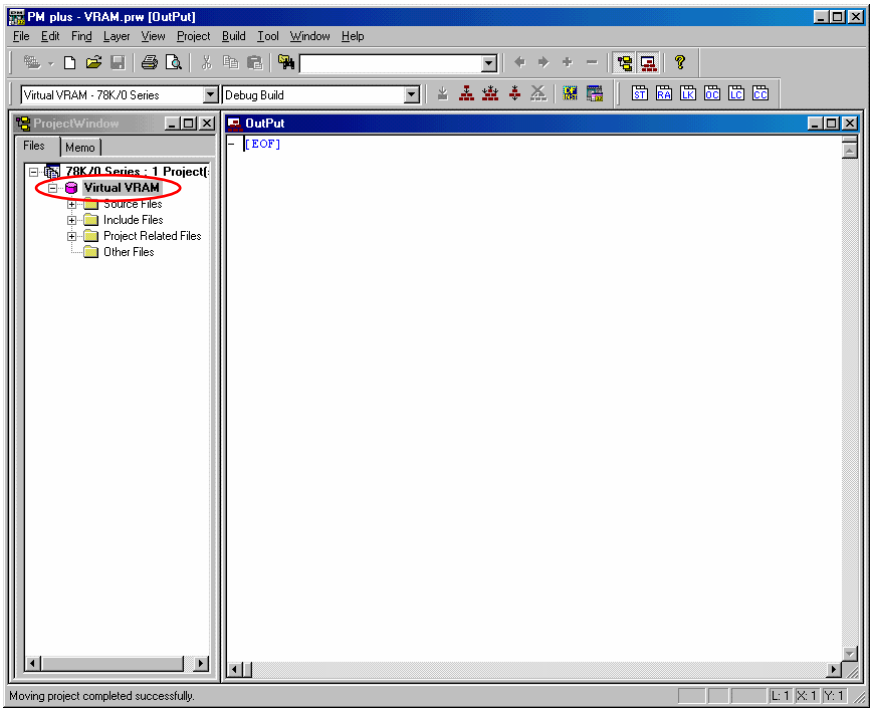


Open the Chapter2 directory.



Select VRAM.prw and click Open.

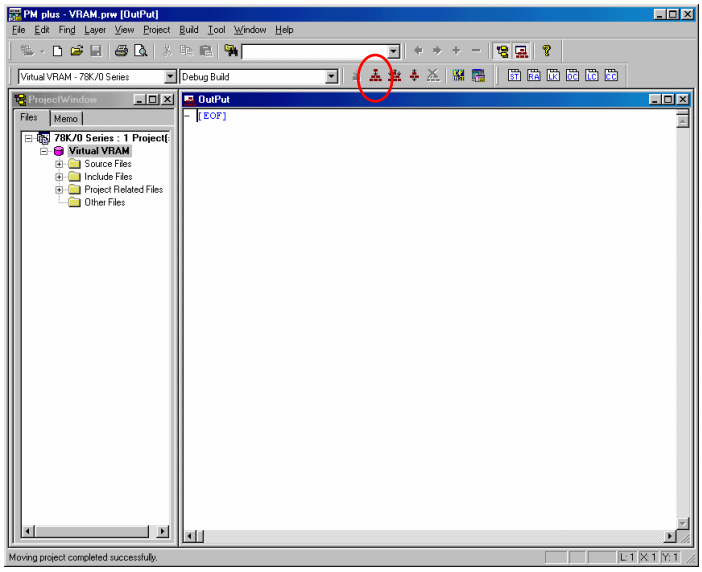
PM plus reads the workspace file "VRAM.prw."



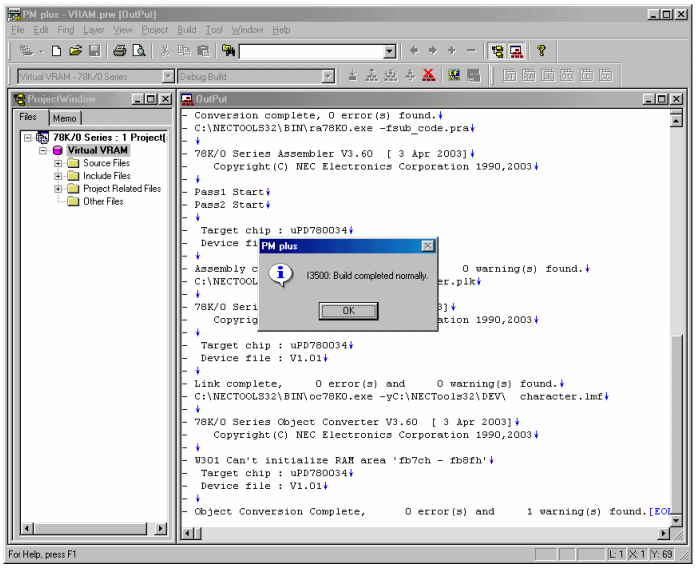
Creating an Executable Program

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) ?

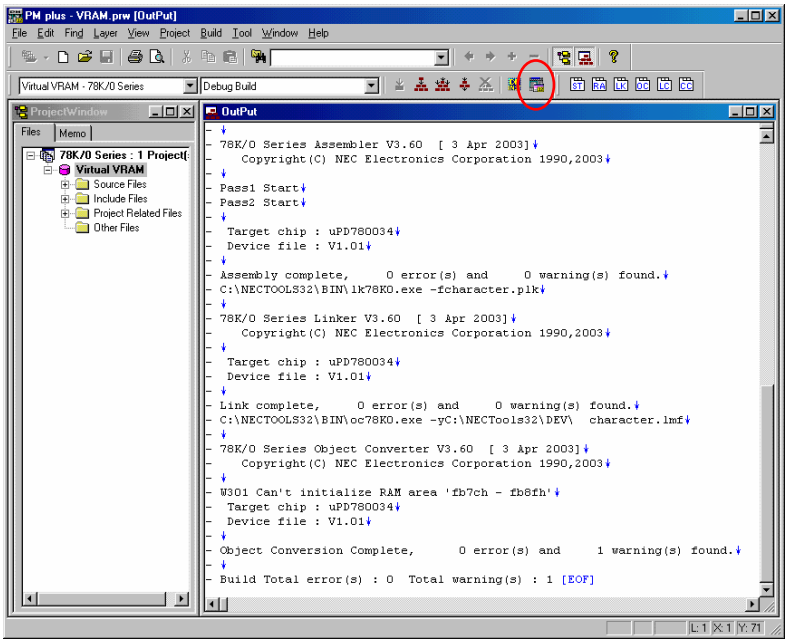
A System Simulator is a Windows-based software tool running on a host computer that simulates the operation of the target system, allowing you to run and debug your application program on the simulator. You can debug at the C source level or the assembly code level. With a System Simulator, you can separate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

Running the System Simulator (SM78Kxx)

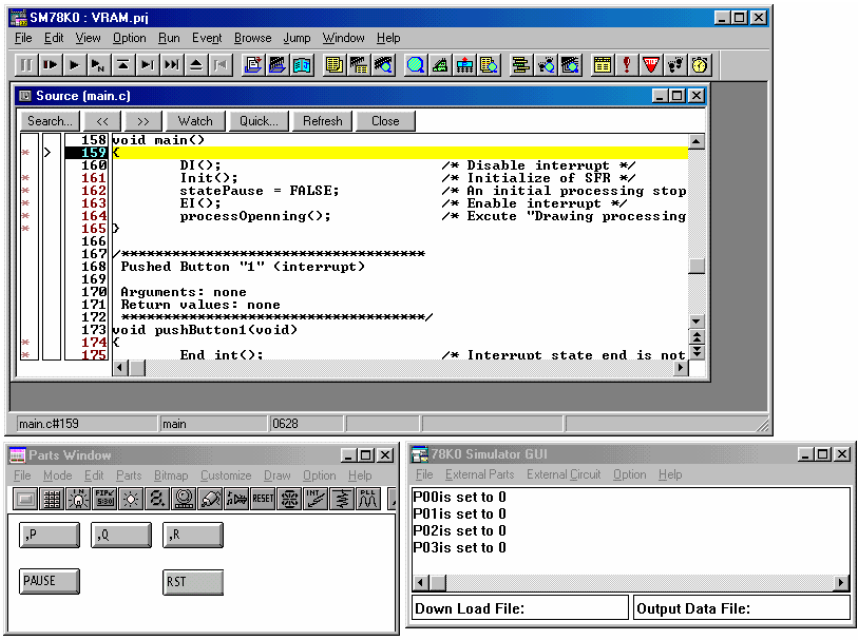
Next, you will run the System Simulator (SM78Kxx).

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

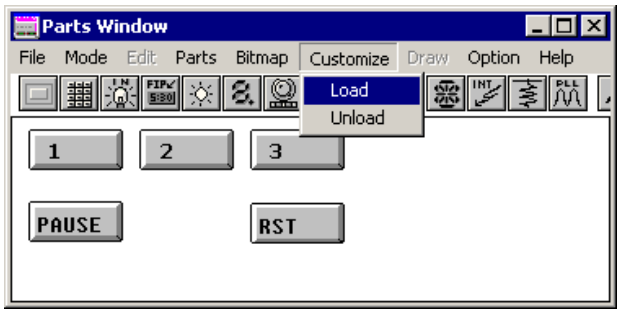
If the Debug button is not displayed, select Tool->Debugger selection->SM78Kxx System Simulator. For details on option settings, refer to Chapter 3 - System Simulator Basics.



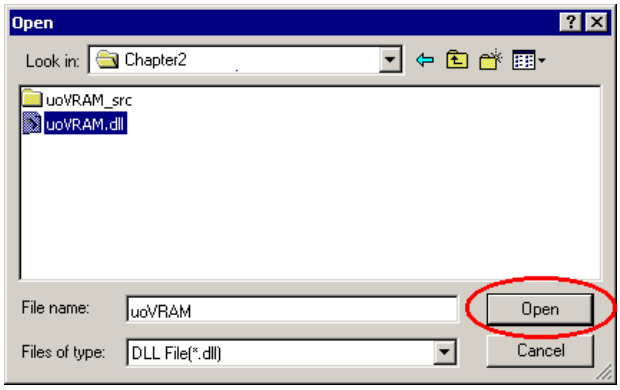
The SM78Kxx starts.



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

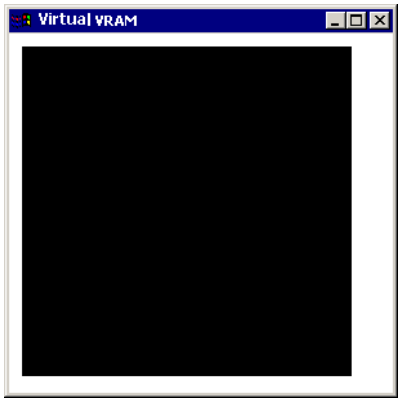


Select "uoVRAM.dll" and click Open.



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).

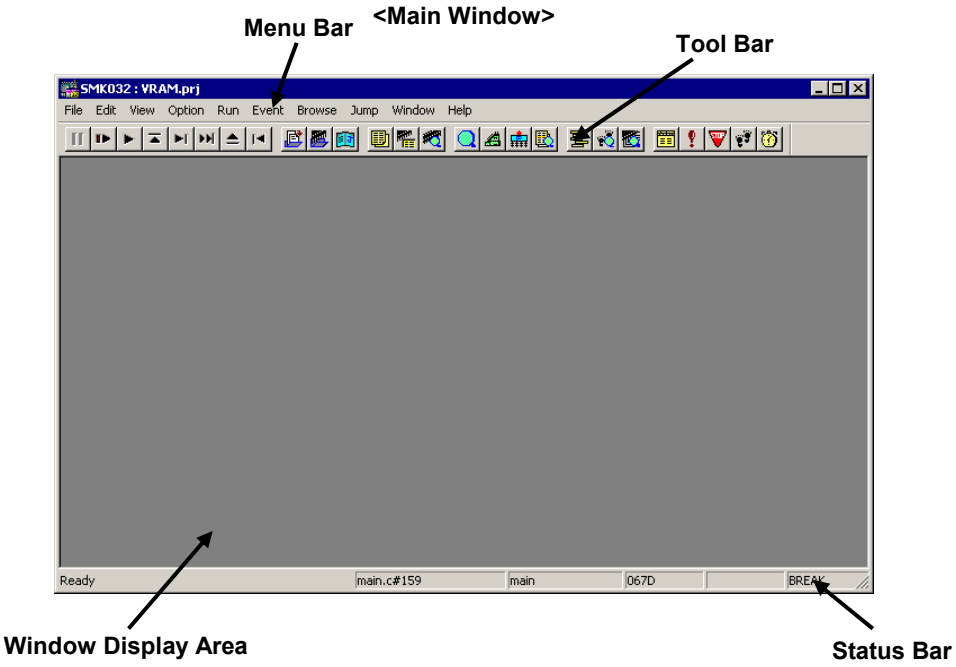
Introduction to the System Simulator (SM78Kxx)

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

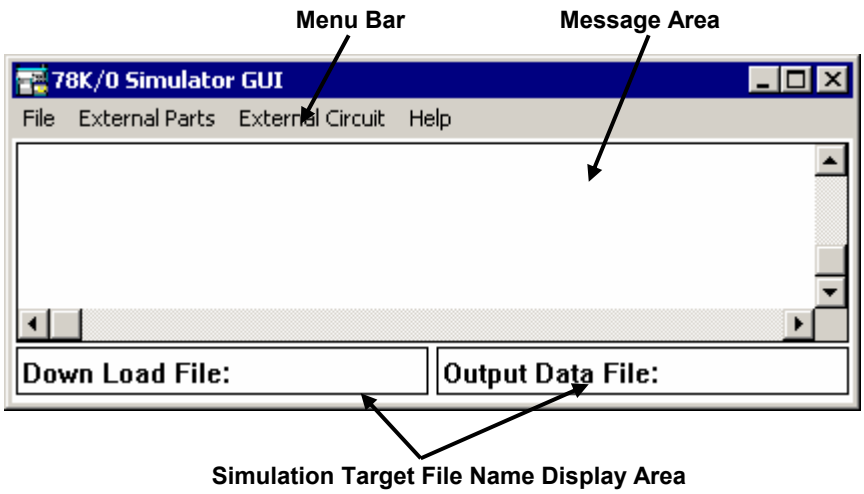
Main window:	Displays the status of the CPU core and controls the simulator execution.
Simulator GUI window:	Controls external parts

The initial screen of the SM78Kxx is as follows.

<Main Window>



<Simulator GUI Window>



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)**.

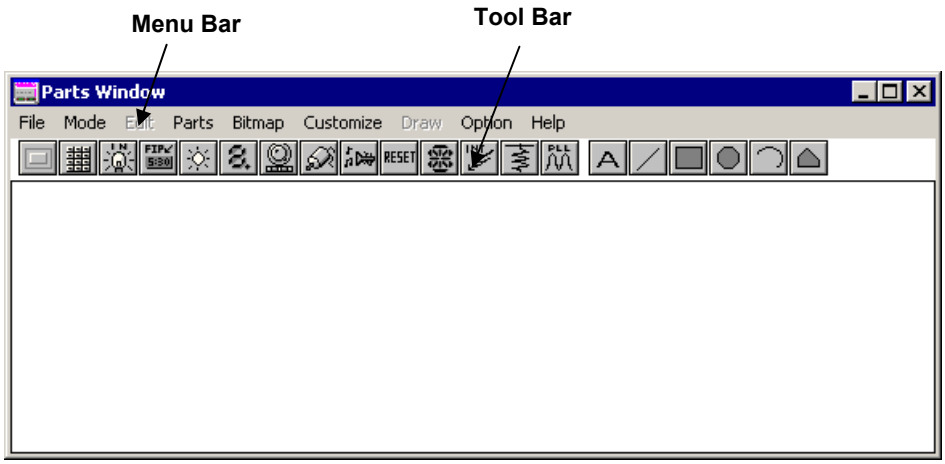
Introduction to the Input/Output Panel Window

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)**.



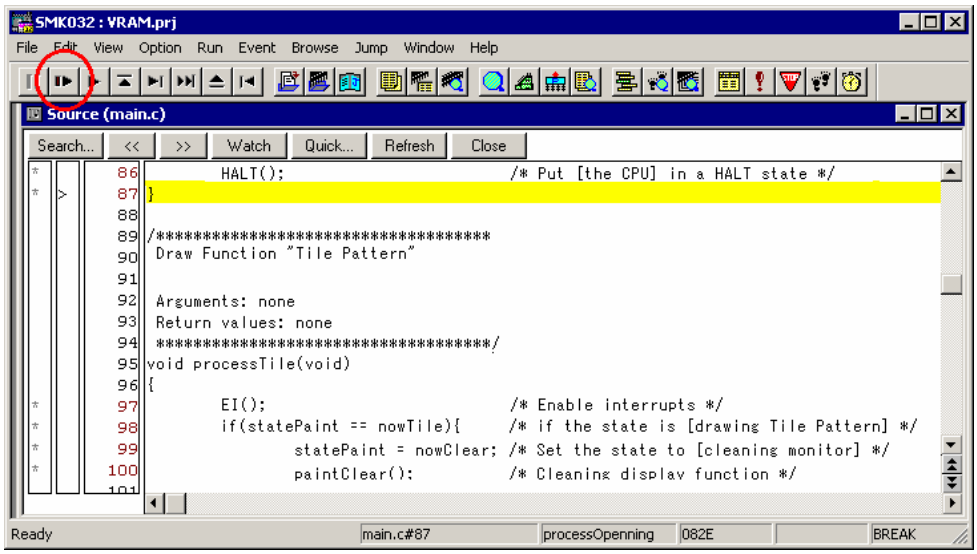
You can create additional parts as required by using Microsoft Visual C++™, and the same procedure as that used for creating the uoVRAM.dll file.

- 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)**.

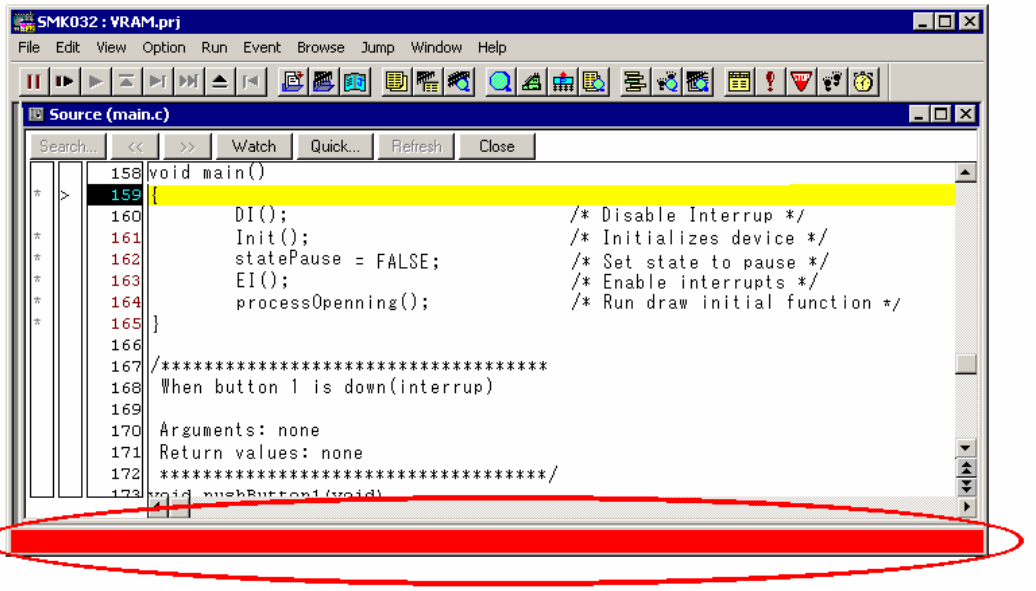
Executing the Program

Next, you will execute the program.

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

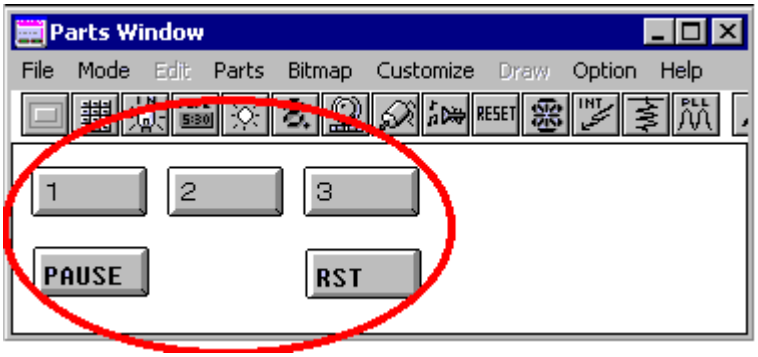


The program executes.



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.



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

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

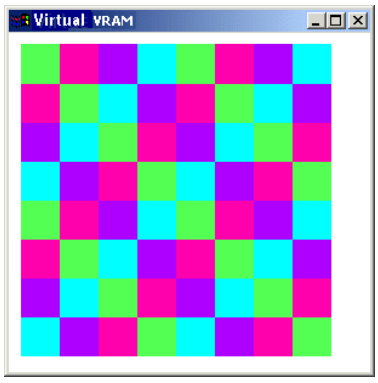
3 Clicking the [3] button switches to a program that draws the pattern shown on <Screen 3>.

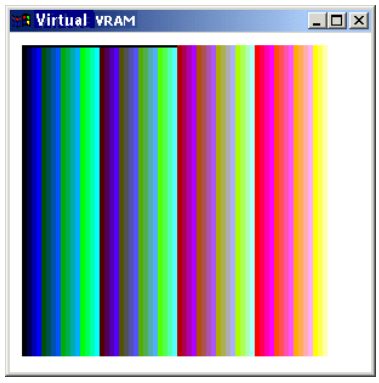
RST Clicking the [RST] button resets the target CPU under simulation.

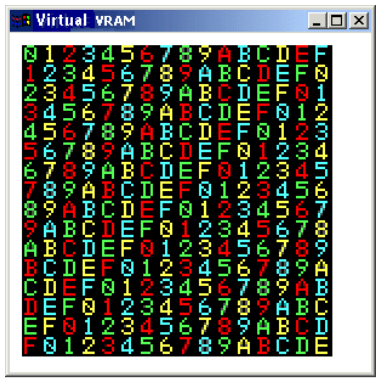
Clicking the PAUSE button while a pattern is being drawn in the Display window pauses the drawing program.

Clicking the PAUSE 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 ?

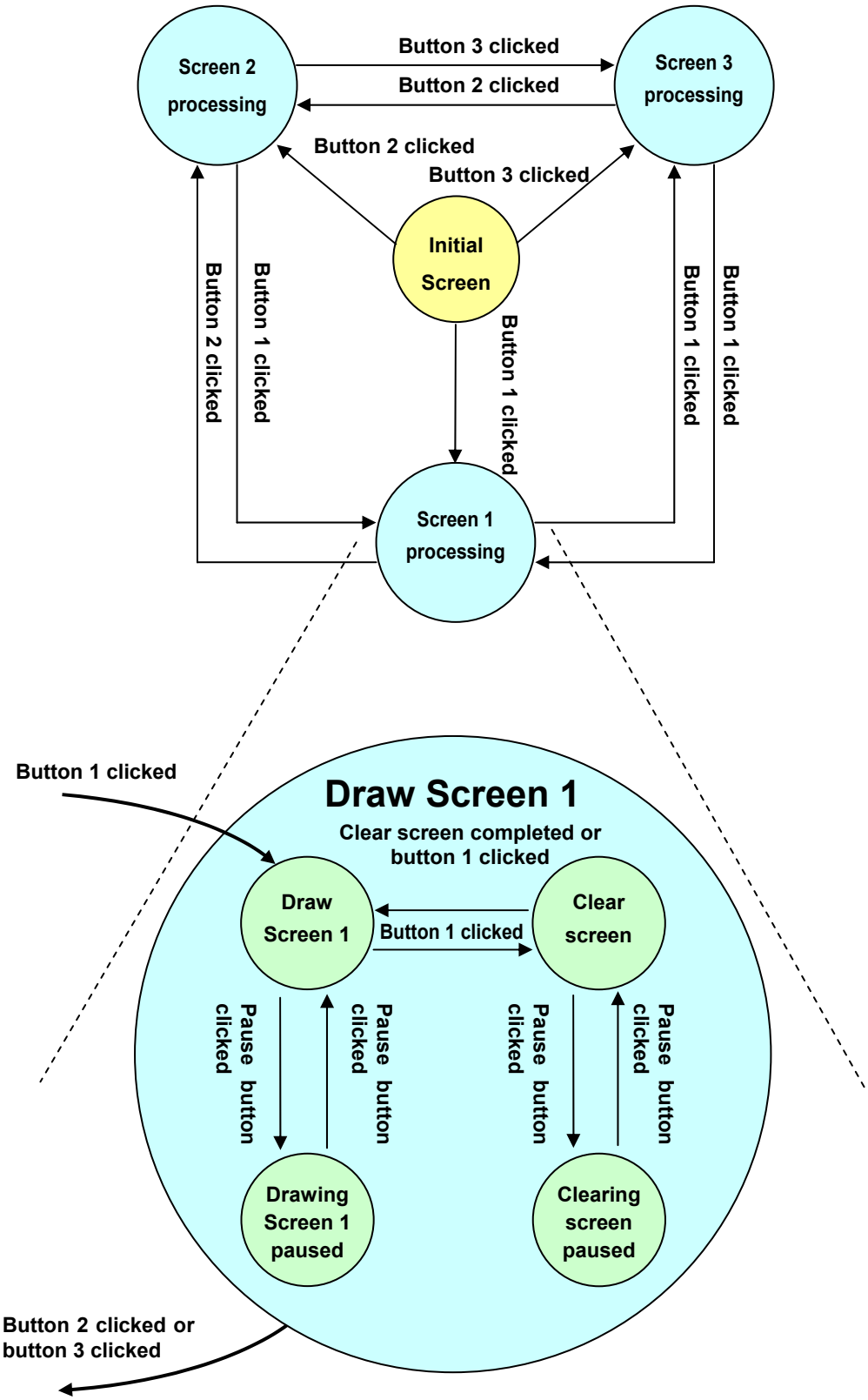
The target CPU mentioned here is a virtual μPD780034, which the SM78K0 simulates. For the 78K4, the target CPU is the μPD784035.

Resetting the target CPU means the SM78K0 simulates the application of a logical low signal to the $\overline{\text{RESET}}$ terminal of the virtual μPD780034.

As a result, the VRAM program running on the virtual μPD780034 returns to its initial state.

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

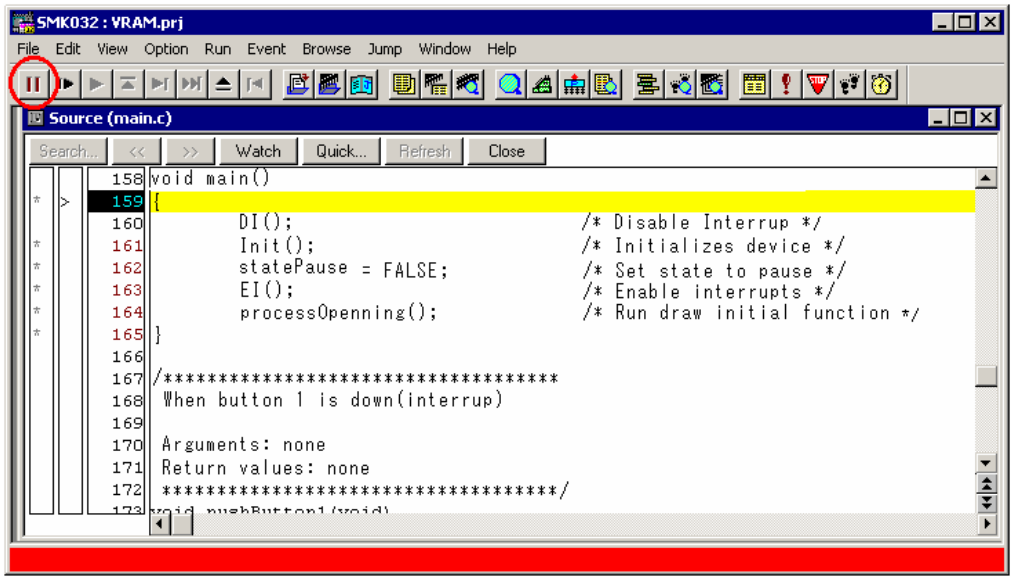
The following diagram shows the screen processing flow of VRAM program.



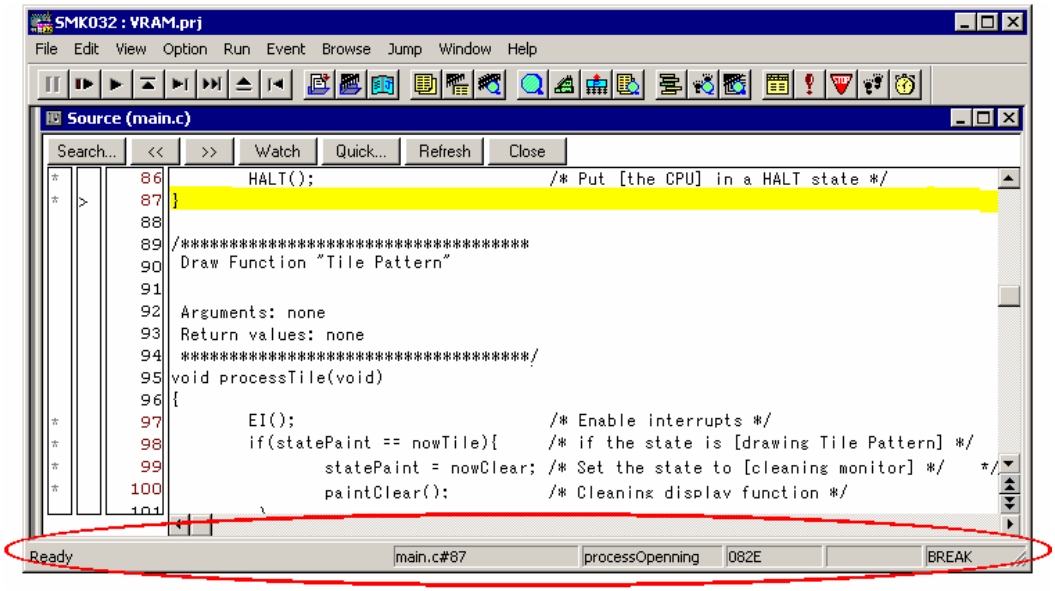
Stopping the Program

Next you will stop the execution of the program.

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



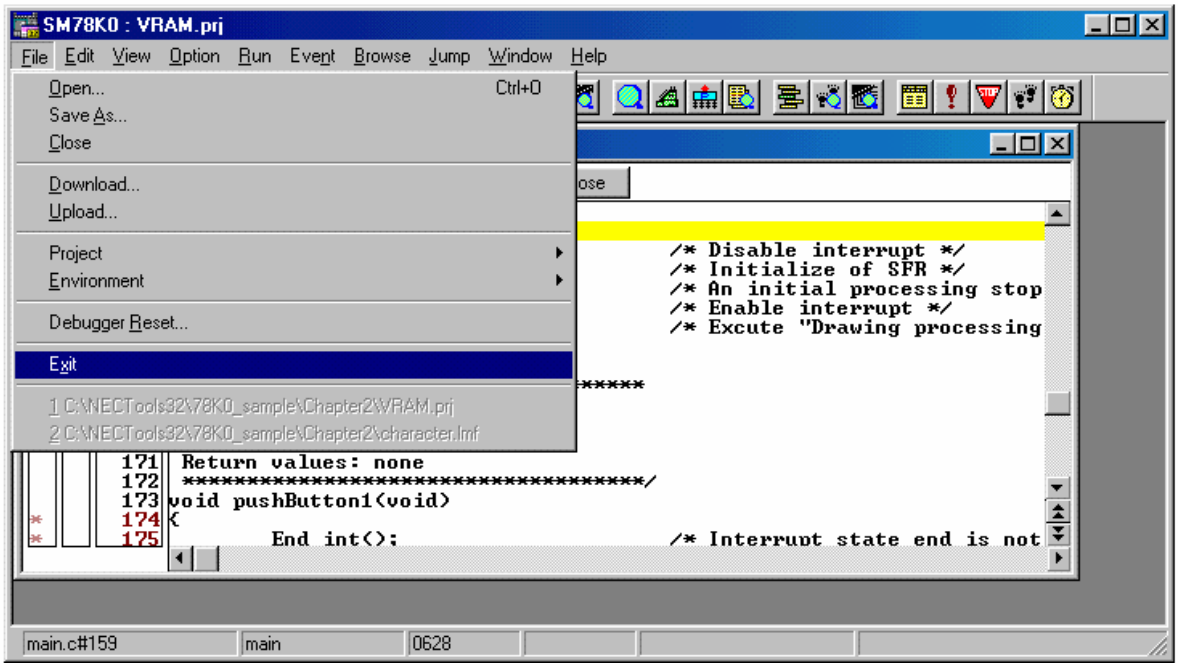
Program execution stops.



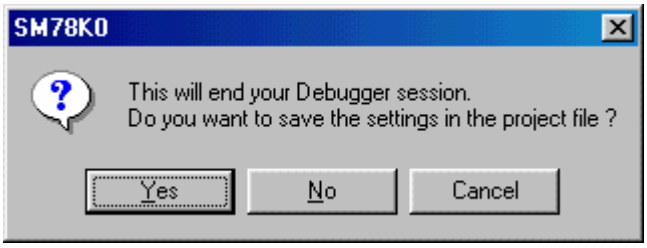
The status bar color returns to its original color when you stop the program.

Exiting the System Simulator (SM78Kxx)

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



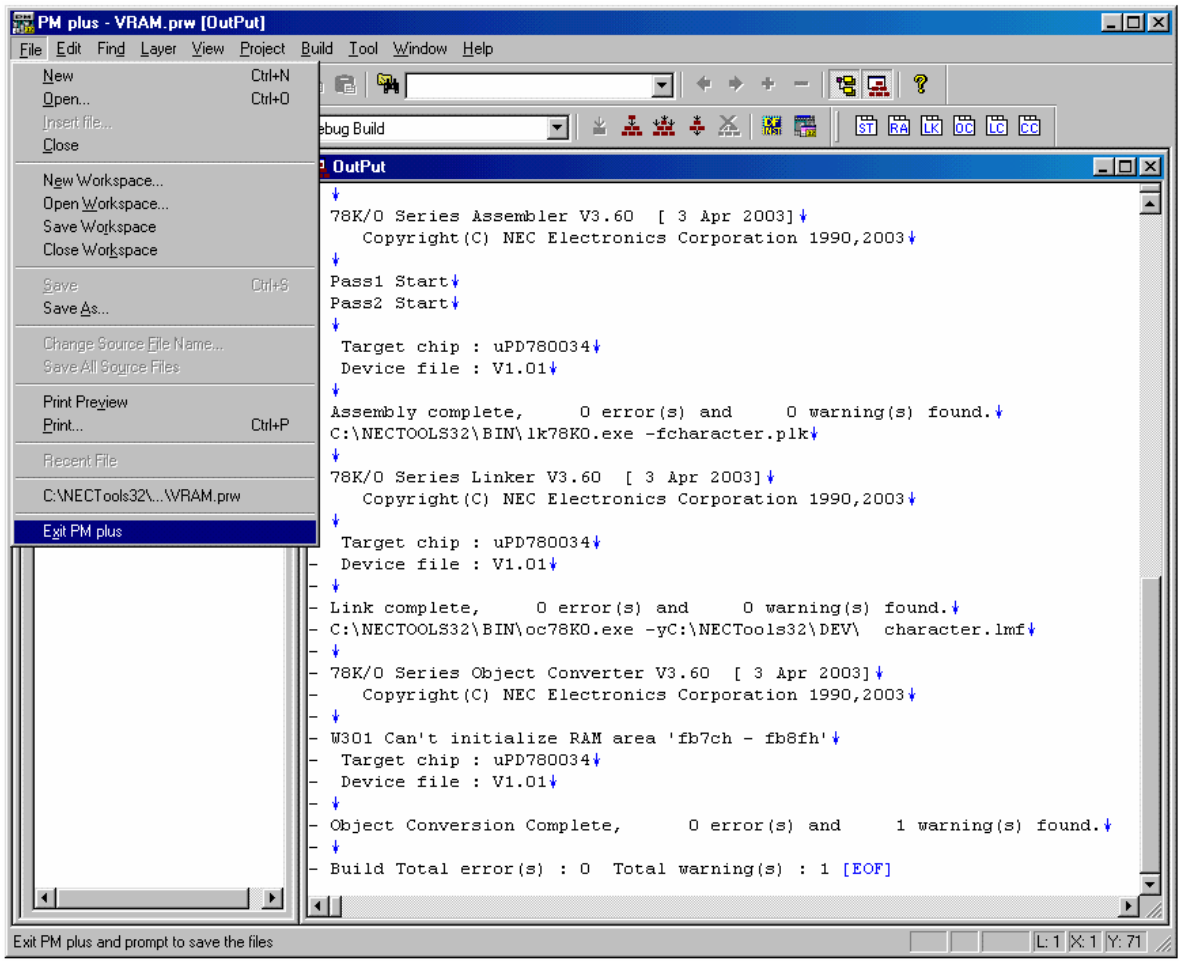
A dialog is displayed asking you if you want to exit.



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.



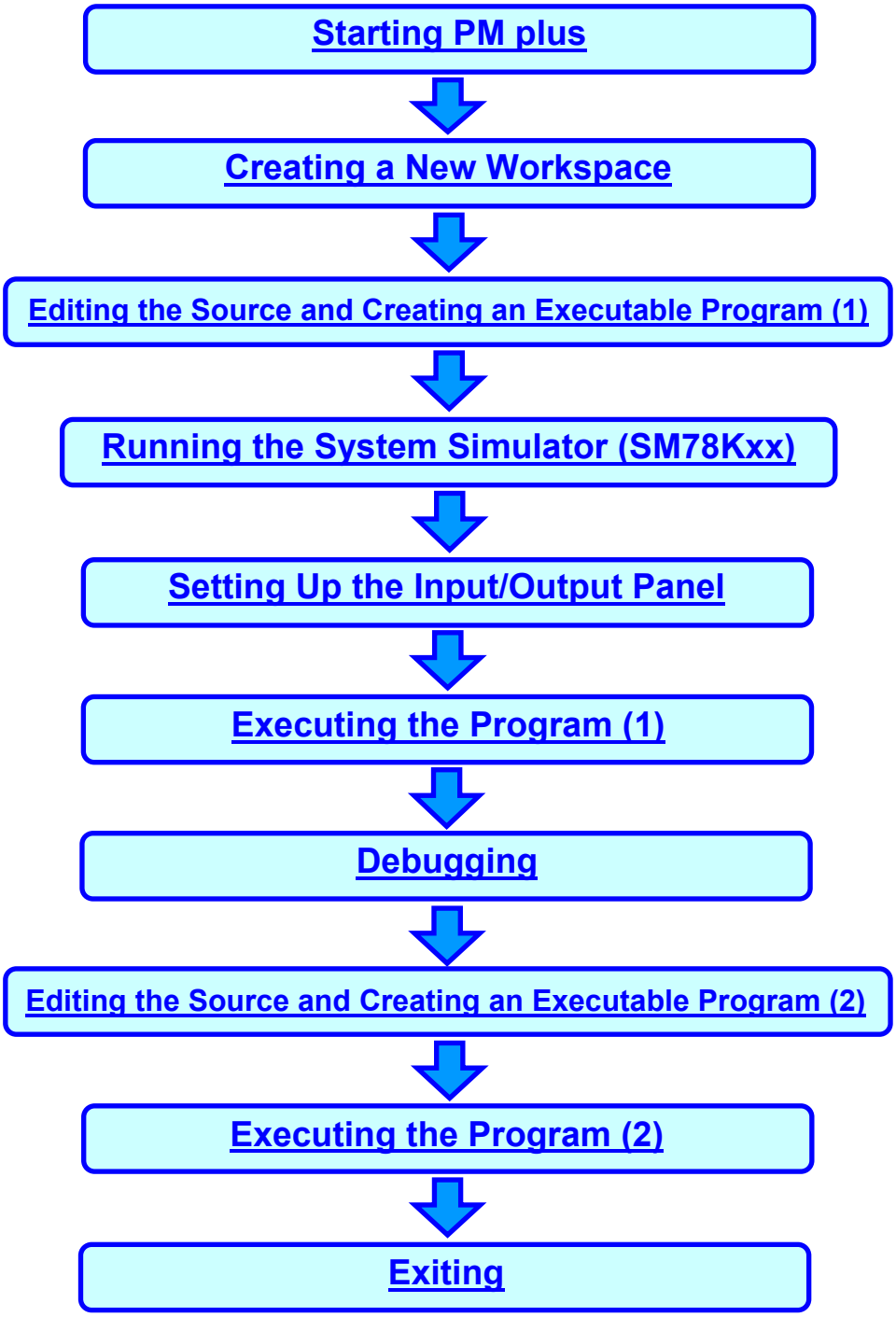
You will exit PM plus program.

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.

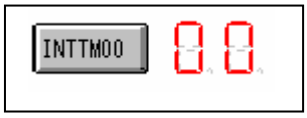


Counter Program Specifications

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. (INTWT 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 INTTM00 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 INTTM00 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

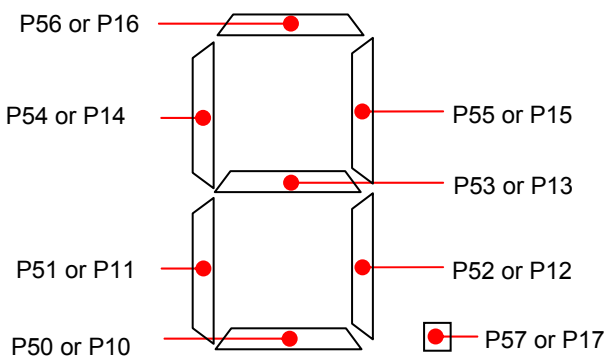
Variable	Contents
int count1	Stores the value of the first digit of the counter (1's)
int count10	Stores the value of the second digit of the counter (10's)

- 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.

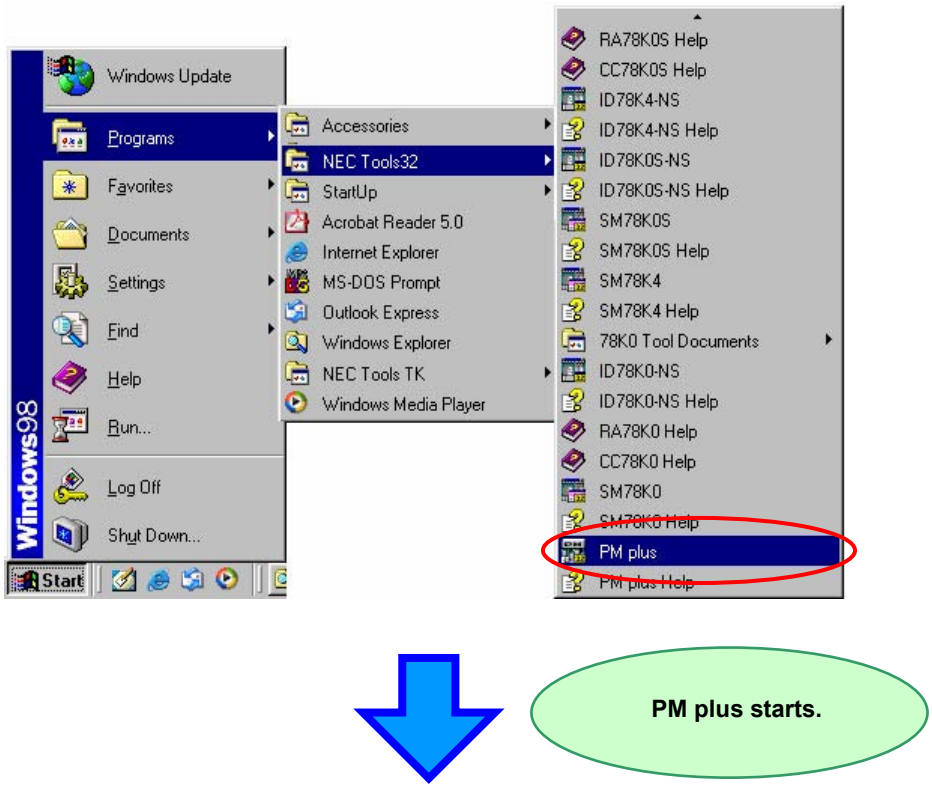
Port	Bit position	Port Address	Usage
P4 or P0	0	P40 or P00	When the state changes 0 -> 1, specifies that the contents of P1 are to be displayed on the first digit
	1	P41 or P01	When the state changes 0 -> 1, specifies that the contents of P1 are to be displayed on the second digit
P5 or P1	0	P50 or P10	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)

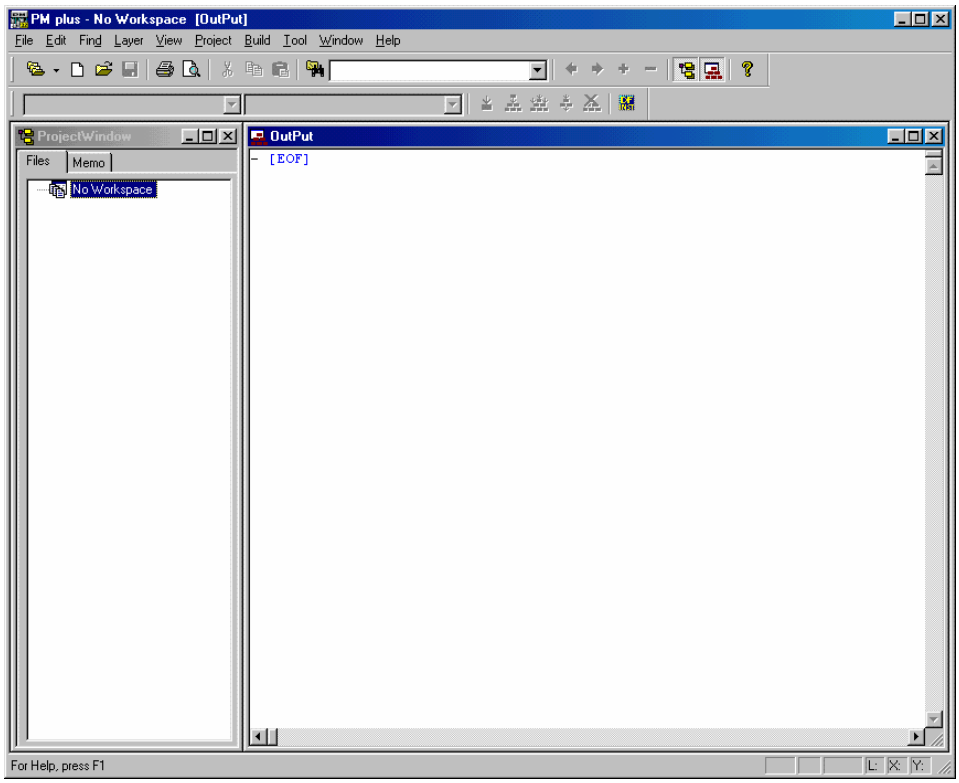


Starting PM plus

First, start PM plus.

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

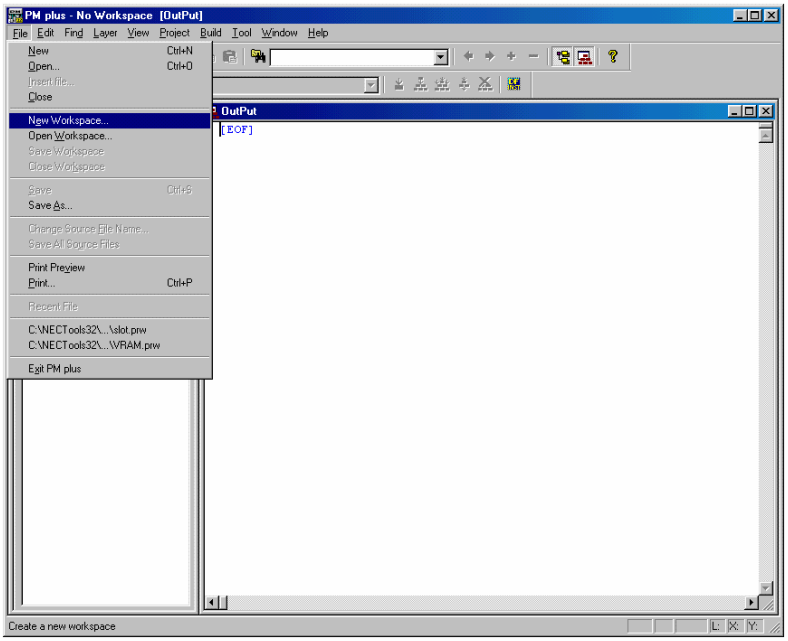




Creating a New Workspace

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.

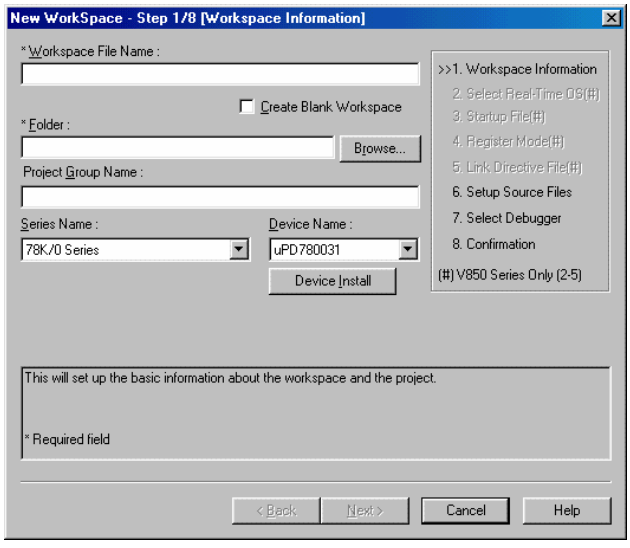
Workspace File Name:
- The name assigned to the file that stores the workspace information.

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.

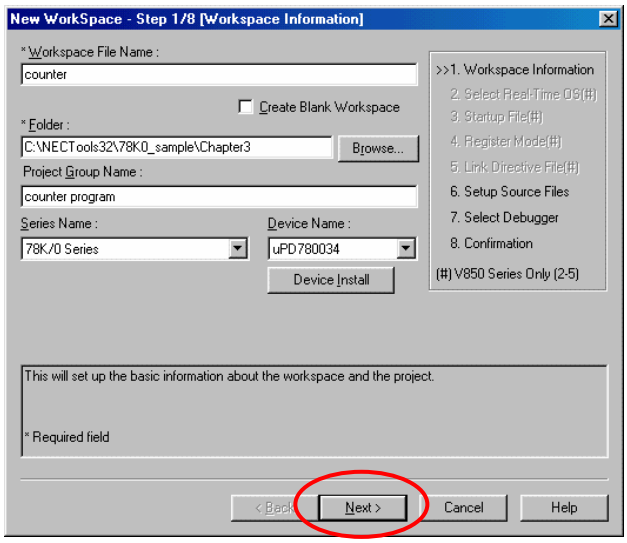
Device Name:
- Specifies the device name of the device file used.



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

Enter the following workspace information Into the dialog box:

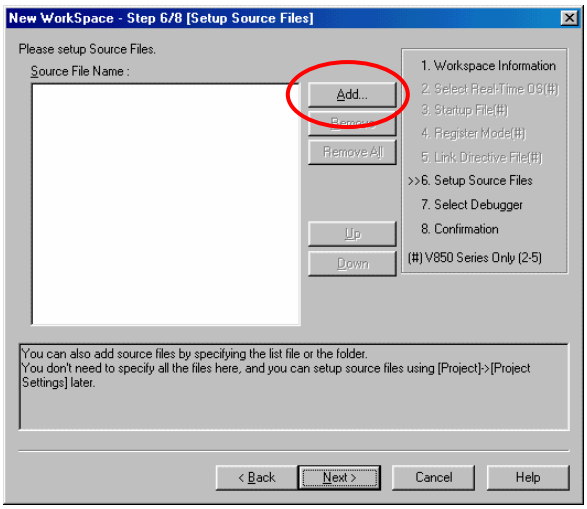
Workspace File Name:
counter
Folder:
For the 78K0:
\78k0_sample\Chapter3
(Click Browse... button and select the sample directory.)
For the 78K0S:
\78k0S_sample\Chapter3
For the 78K4:
\78k4_sample\Chapter3
Project Group Name:
counter program
Series Name:
78K/0 Series (for the 78K0)
78K/0S Series (for the 78K0S)
78K/4 Series (for the 78K4)
Device Name:
μPD780034 (for the 78K0)
μPD789046 (for the 78K0S)
μPD784035 (for the 78K4)



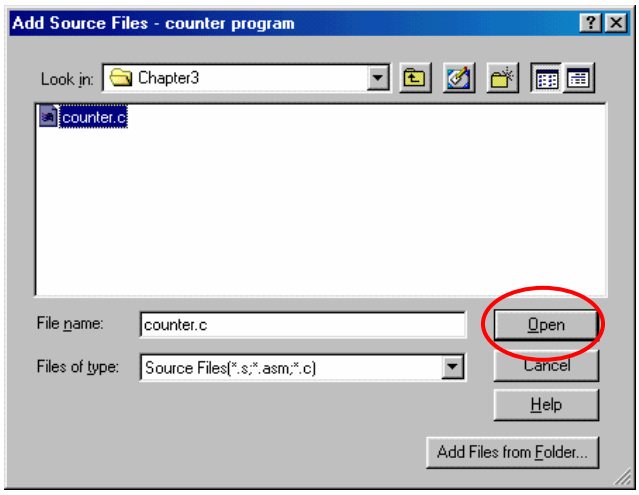
After completing the settings, click the Next button.

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

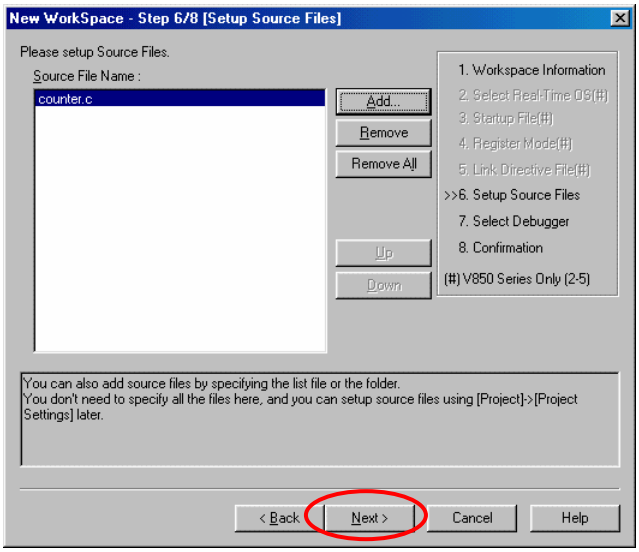
Click Add... button.



Select "counter.c" and click Open.

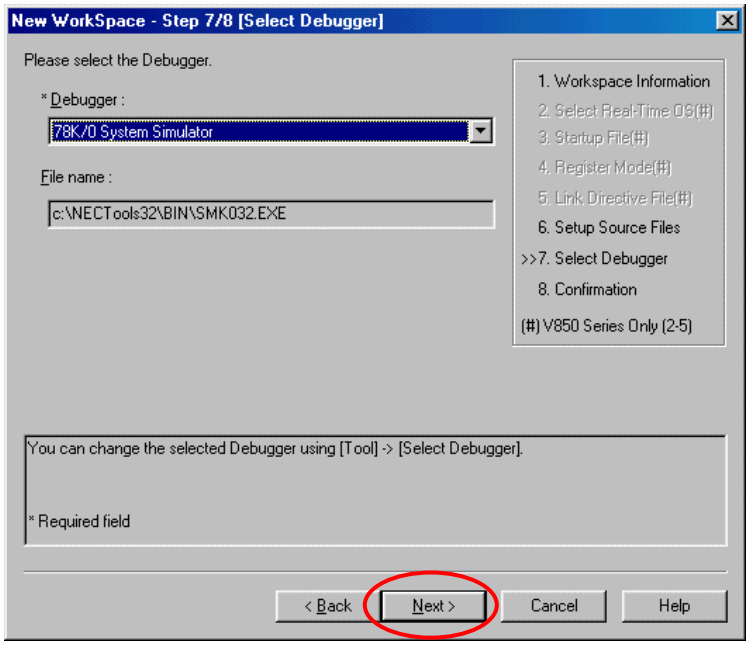


If the source file name does not appear in the list, this means that the folder position is not correctly set in the workspace information setting.
Perform the setting again, selecting the correct directory.



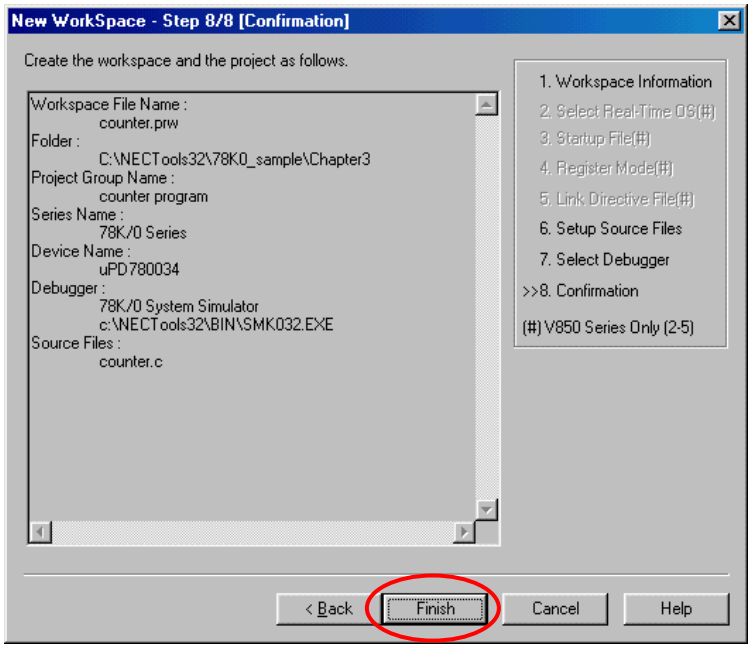
The source file "counter.c" is added to the project.

Specify the debugger to be used.

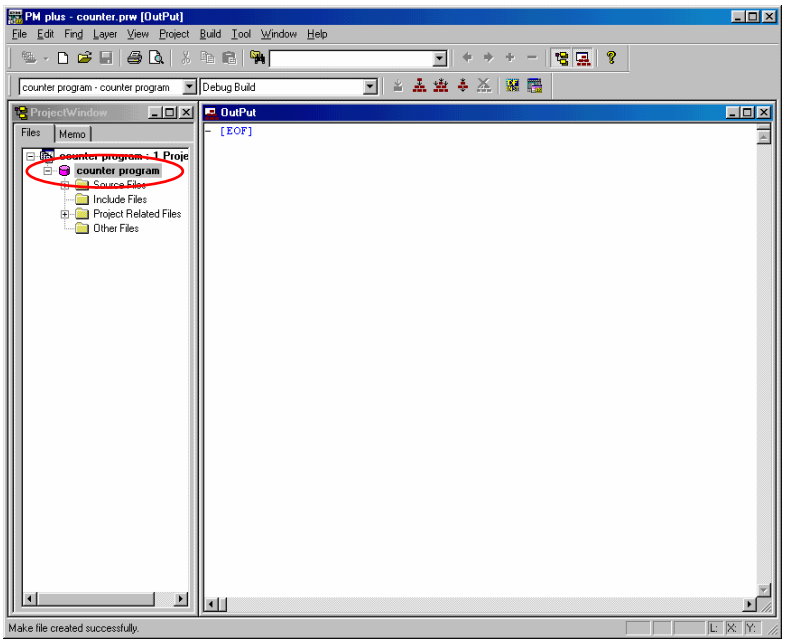


Click the Next button.

Check the set contents.



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



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

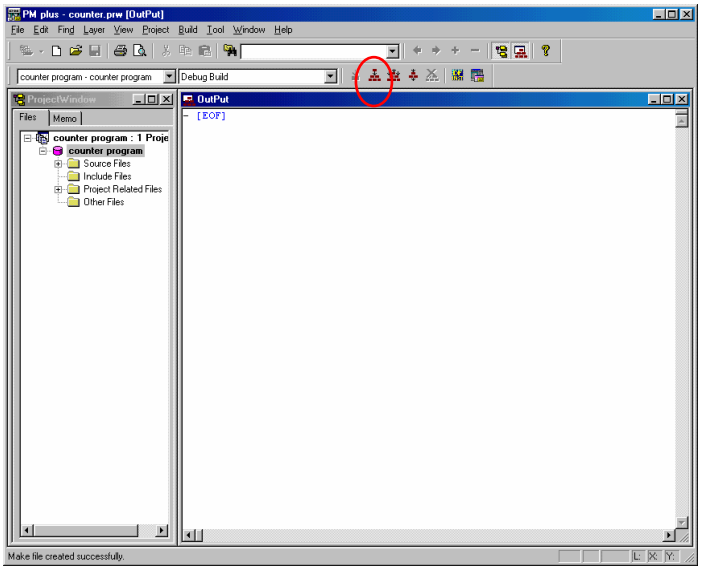
This completes the creation of the workspace.

You can add other source files at any time.

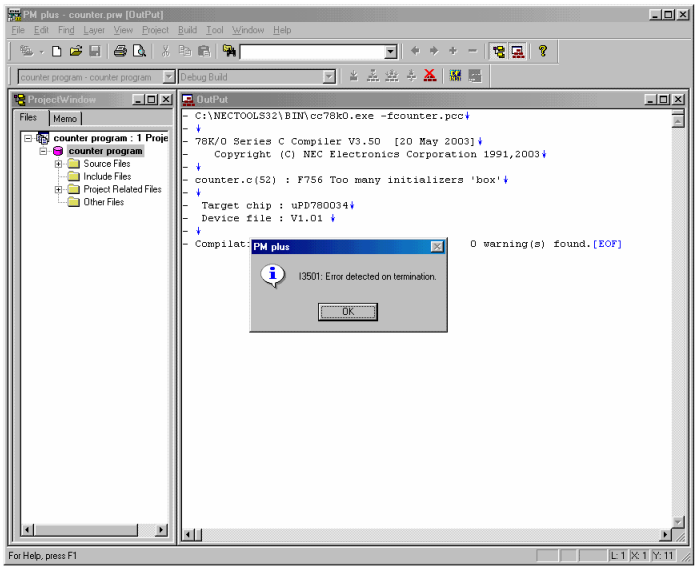
Editing the Source and Creating an Executable Program (1)

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.

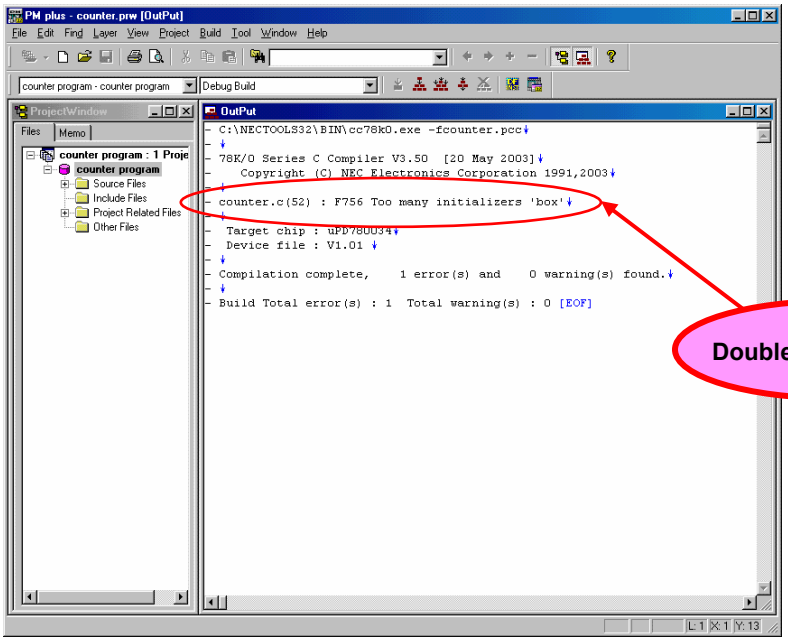
A detailed error message is displayed in the Output window.

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

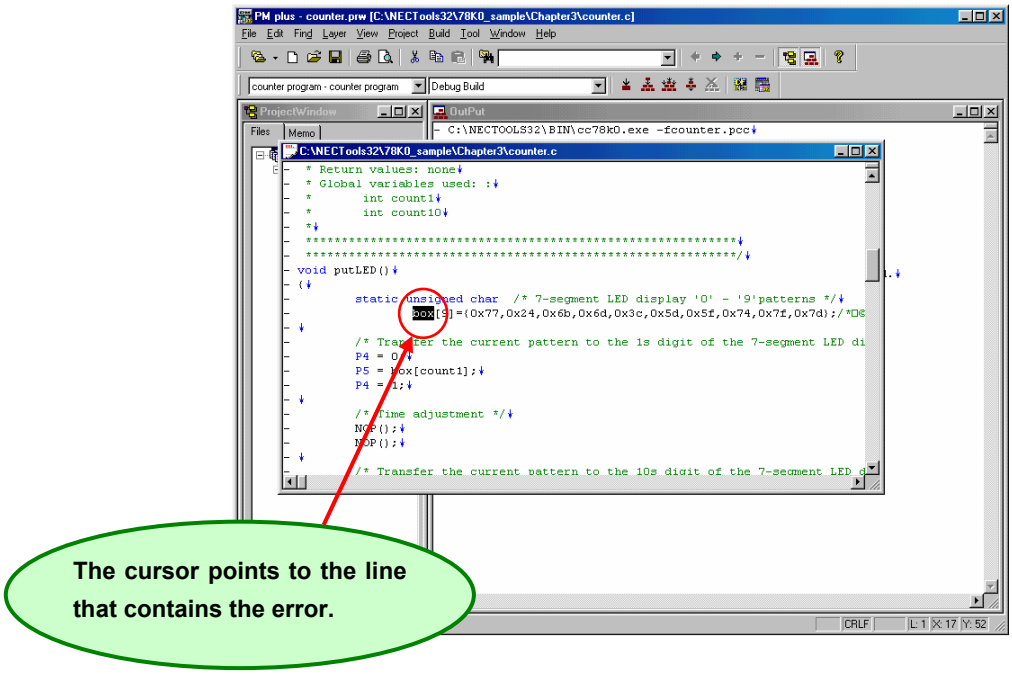
How to read the error message

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

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

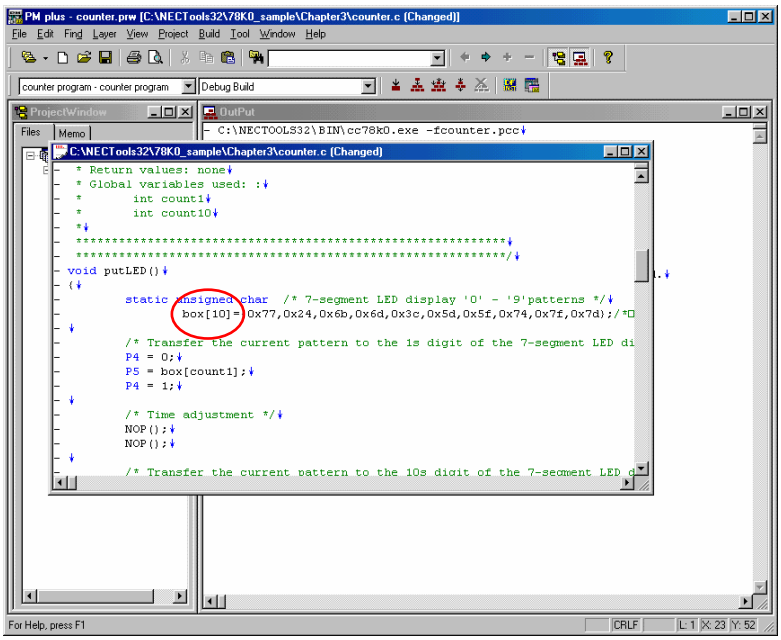


Editor opens.



The cursor points to the line that contains the error.

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

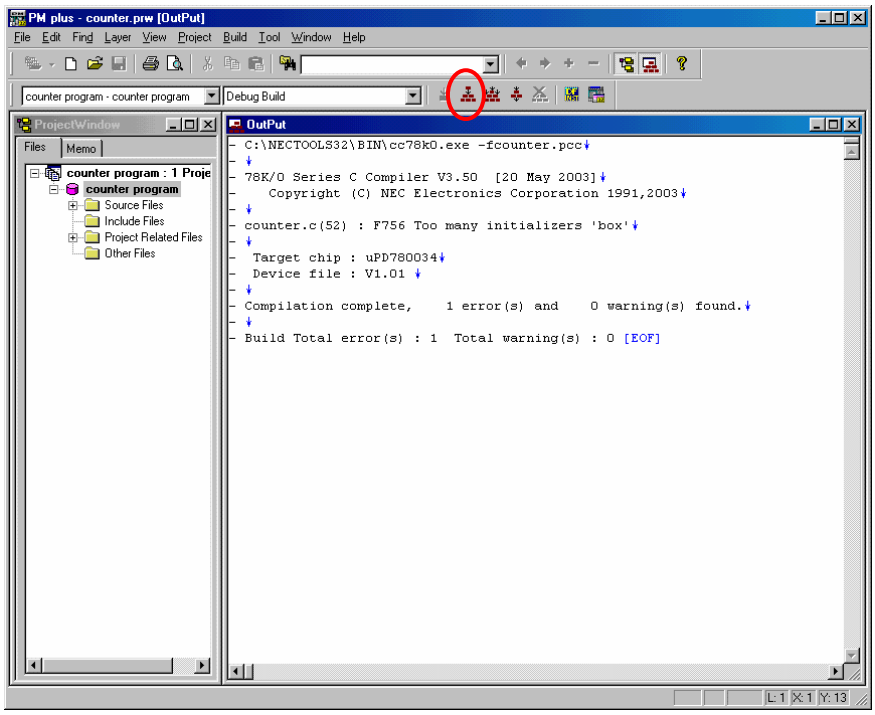


This completes the source editing.

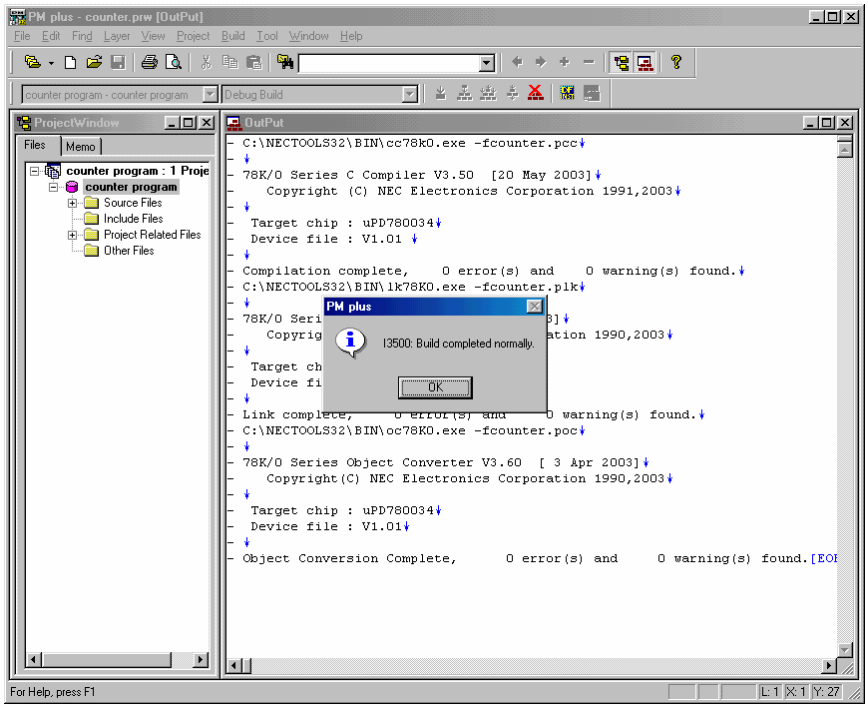
You corrected an error, so you must now rebuild the project.

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.



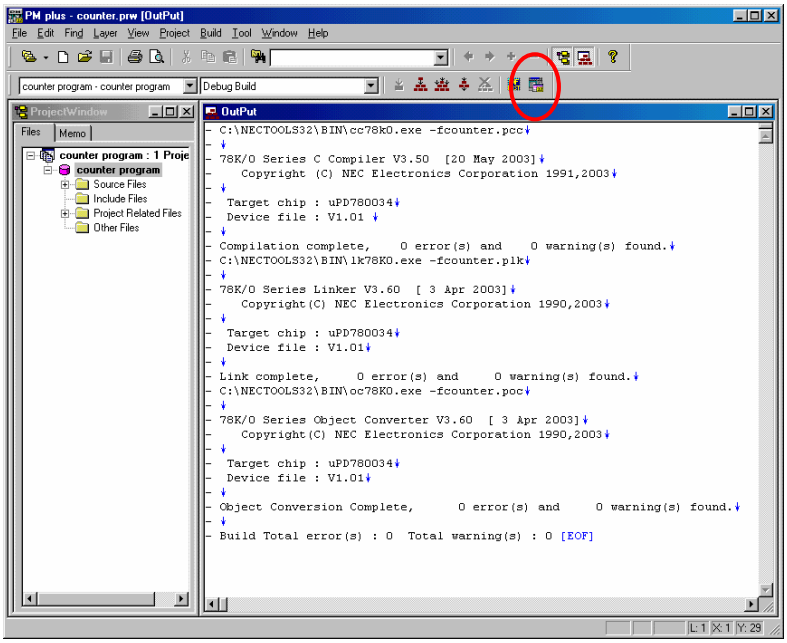
The build completes normally, and an executable program is created.

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

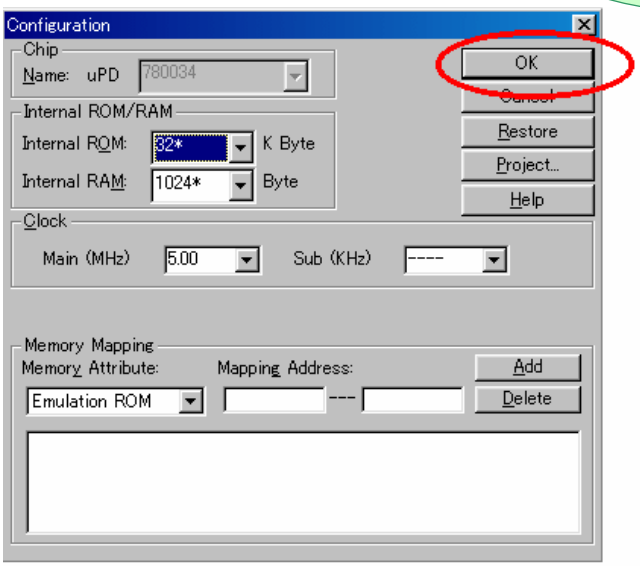
Running the System Simulator (SM78Kxx)

Start up the SM78K0.

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



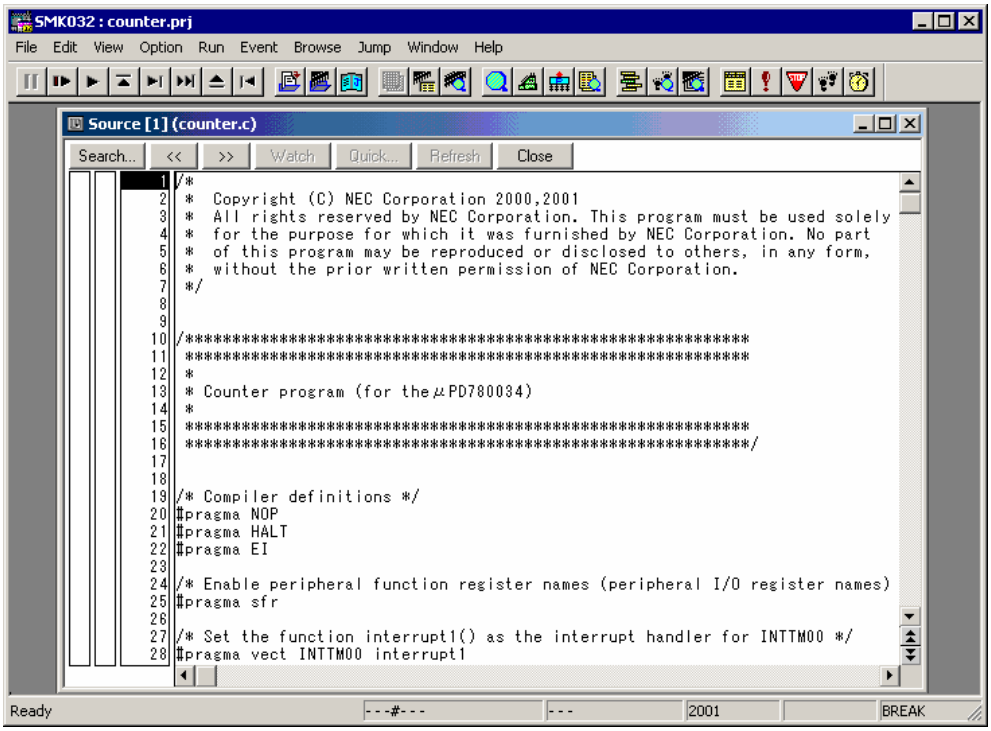
The Configuration dialog box opens.



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 μPD78xxxx, 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

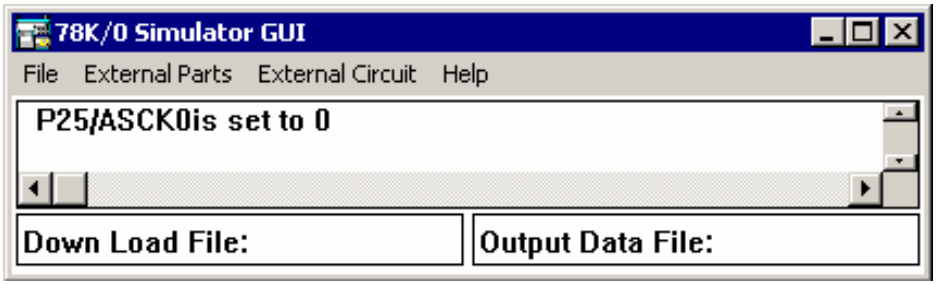
Before running the counter program, you must set up the button and 7-segment LED display used by the program.

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

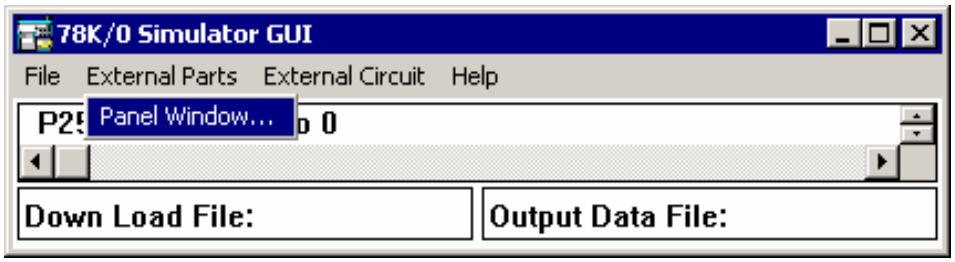
You can open the Input/Output Panel window from the 78Kxx simulator GUI window.

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

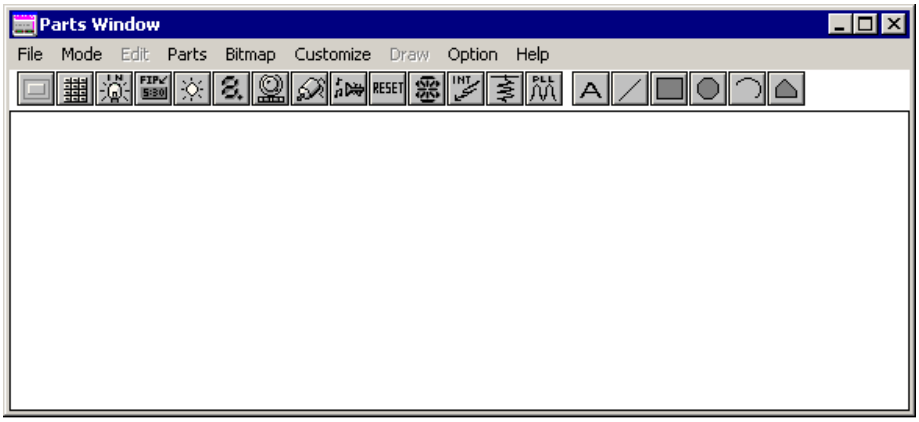
The 78Kxx simulator GUI window opens.



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



The Input/Output Panel window opens.

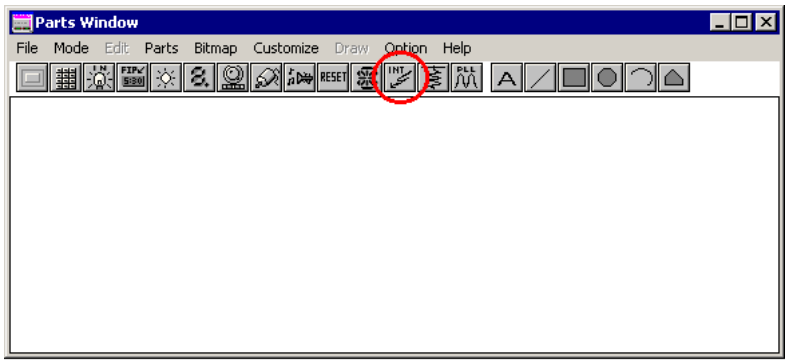


Next, you will set the internal interrupt button.

First, in the Input/Output Panel, click the Internal Interrupt button 

or, from the menus, select Connection ->Internal Interrupt Button...

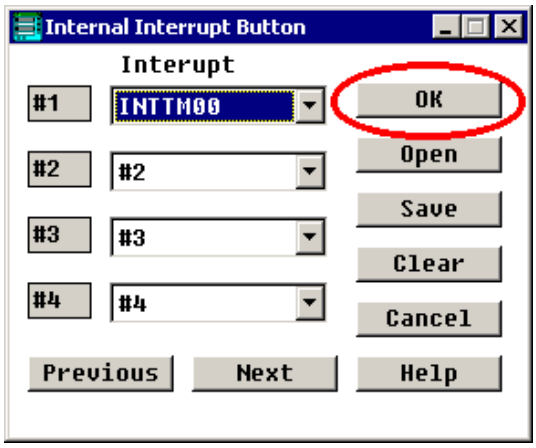
The Internal Interrupt Button Settings dialog box will open. Change the name of the first interrupt from "#1" to "INTTM00" for the 78K0, "INWT" for the 78K0S and "INTC00" for the 78K4.

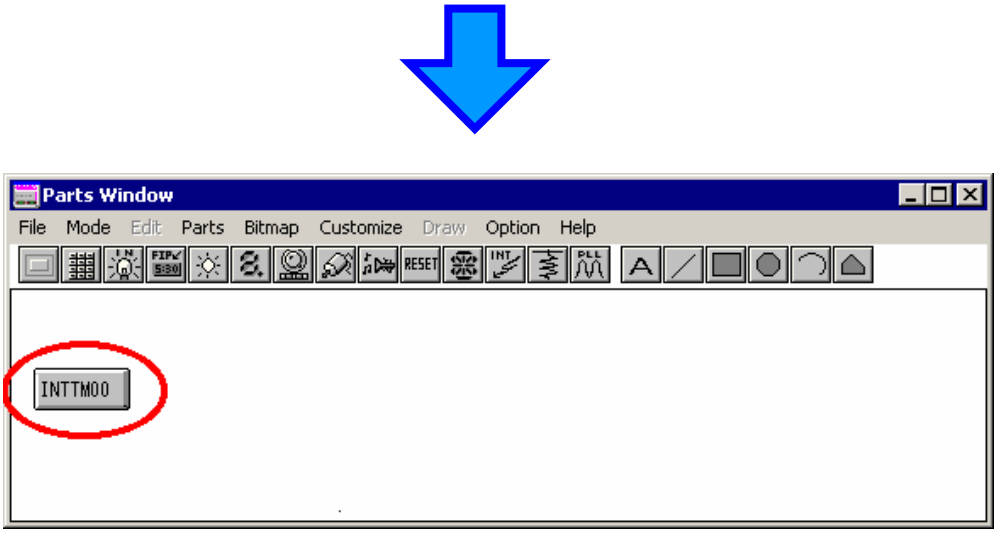


The Internal Interrupt Button Settings dialog box opens.

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.

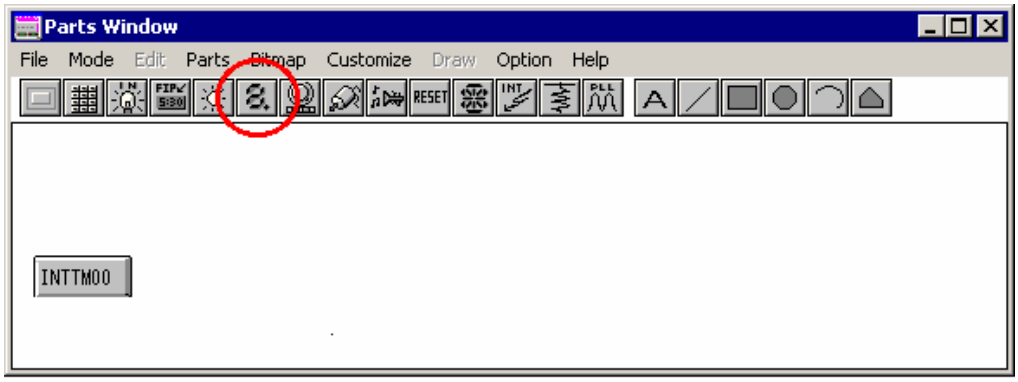
It is sometimes difficult to create the conditions required to generate an internal interrupt during debugging. You can, however, easily generate interrupts using this function to test the interrupt handling routines.

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.

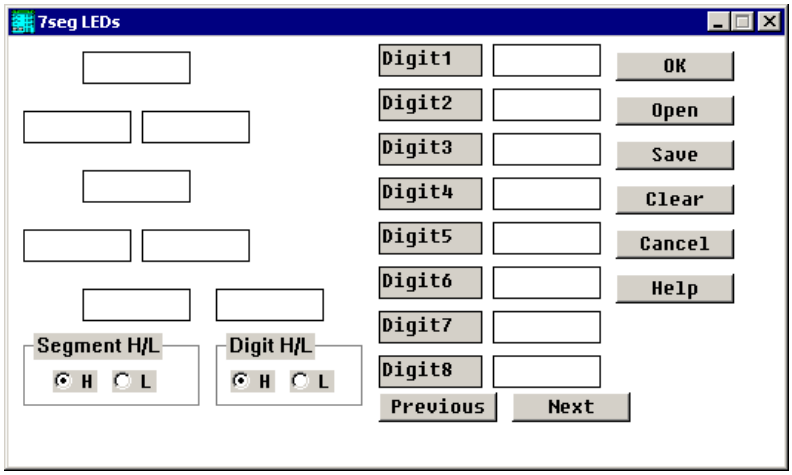
In the Input/Output Panel window, click the 7-segment LED Display Terminal Setting button  or, from the menus, select Connection->7-segment LED...

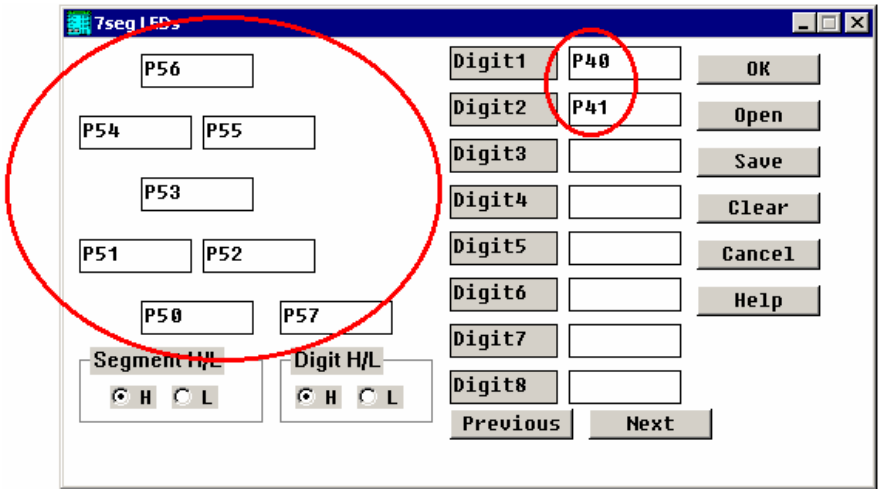
When the 7-segment LED Terminal Setting dialog opens, enter the connection information according to the chart below.



The7-segment LED Terminal Setting dialog opens.

Enter the connection information as follows.

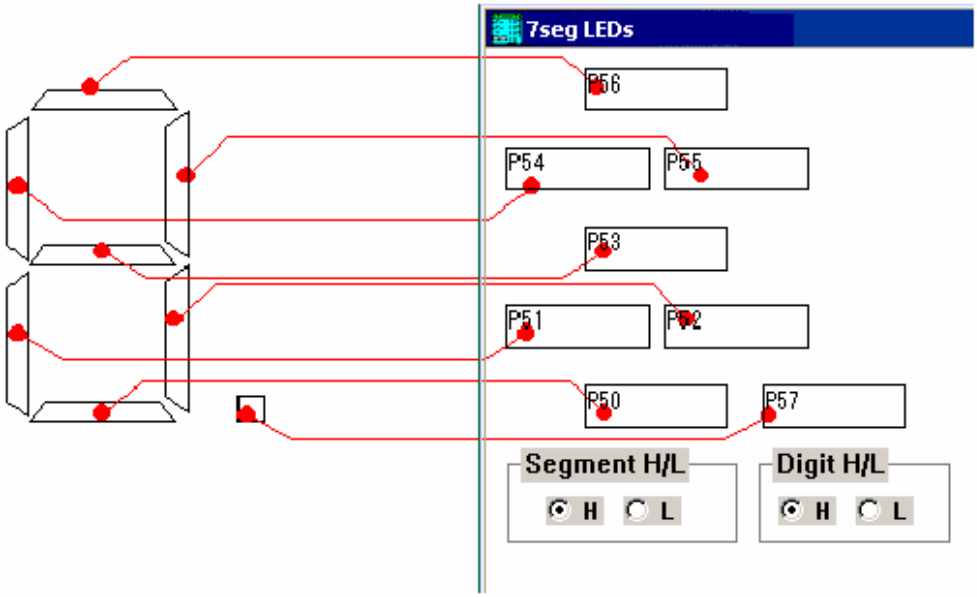




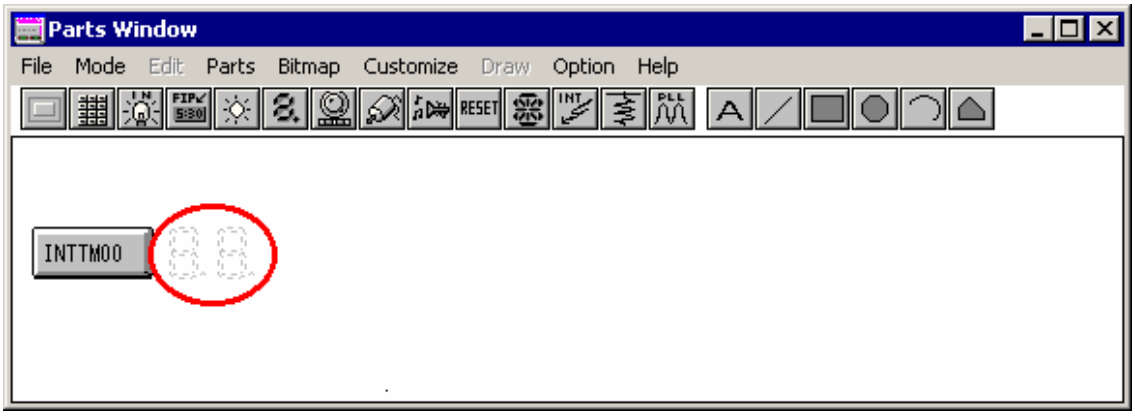
When you complete the settings, click OK button.

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.

<Virtual LED Display> **<7-segment LED terminal settings>**



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.

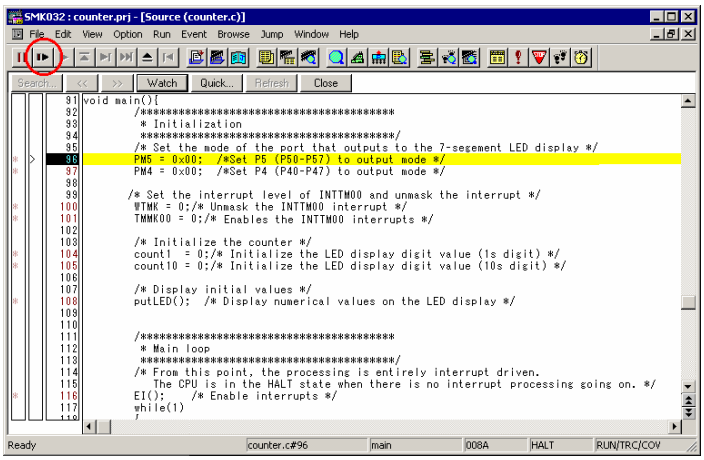
For further details, refer to the SM78K Series System Simulator Ver.2.52 Operation User's Manual (U16768E).

Executing the Program (1)

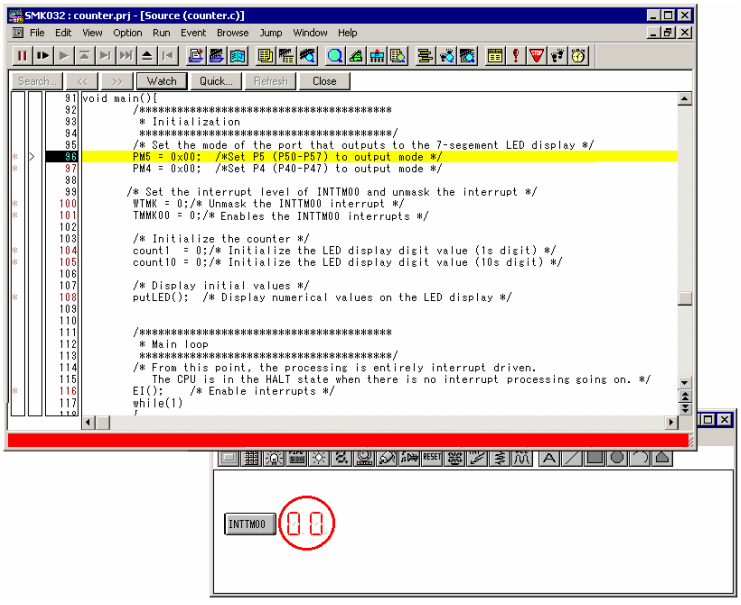
Now, you will run the counter program.

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.



Program execution starts.



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:

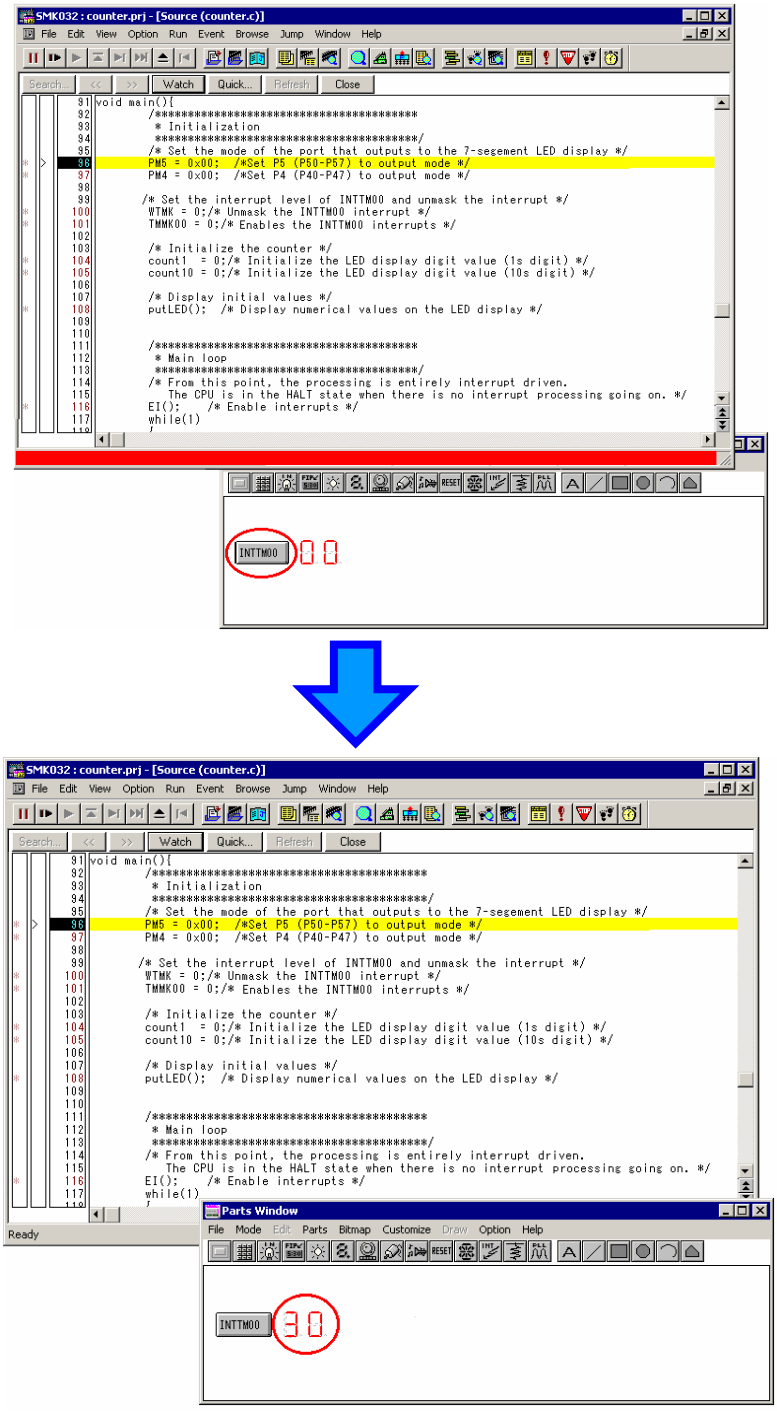
- Perform a restart operation again.

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.

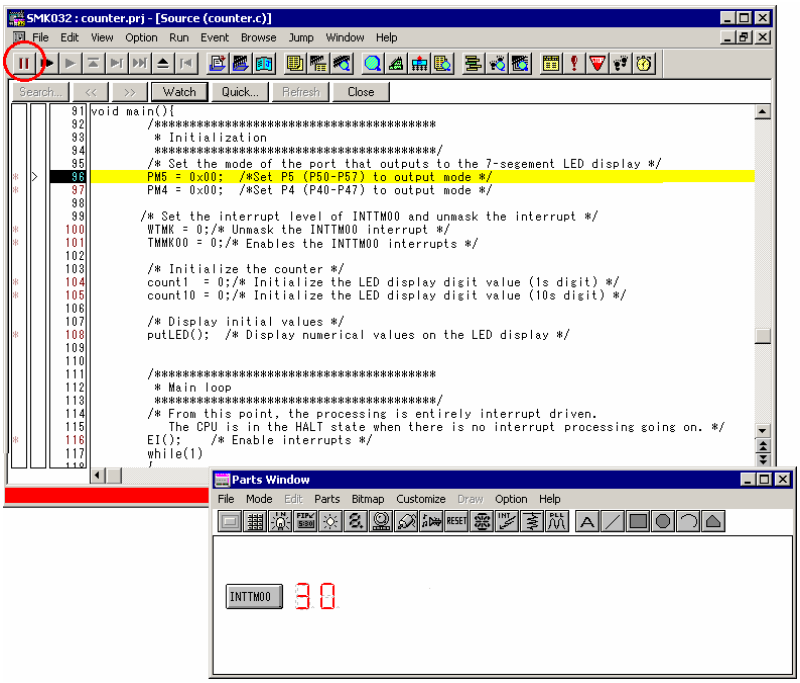
2) If the LED display dos not behave as described above (counting in increments of 10),

- setup the 7-segment LED terminals again.

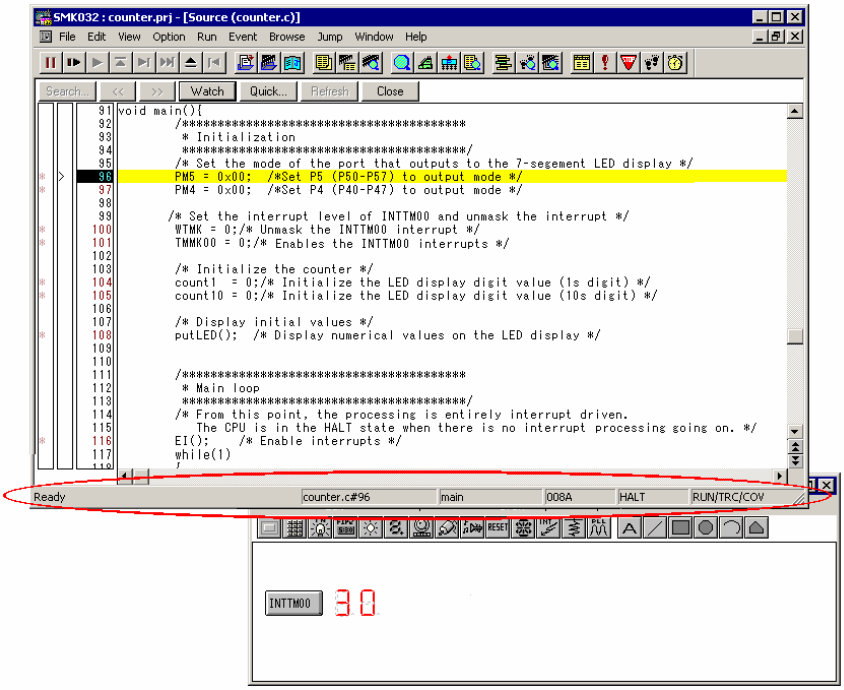
Since the counter is not counting up correctly, let's debug the program.

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

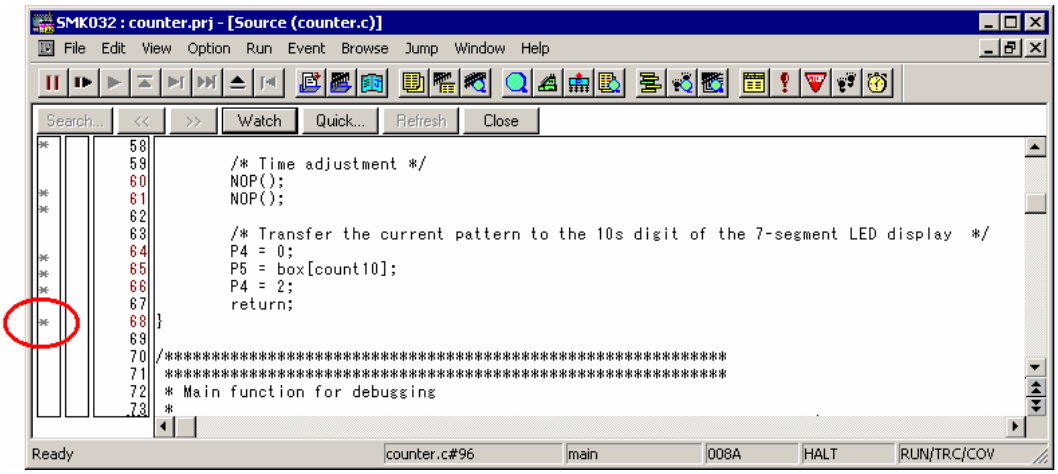
The variable count1 is associated with the 1s digit and count10 is associated with the 10s digit of the LED display.

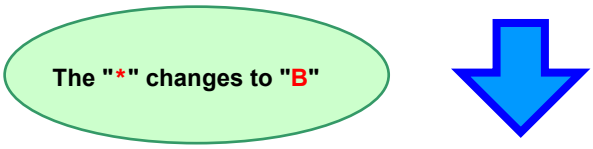
For details, refer to the section "Counter Program Specifications".

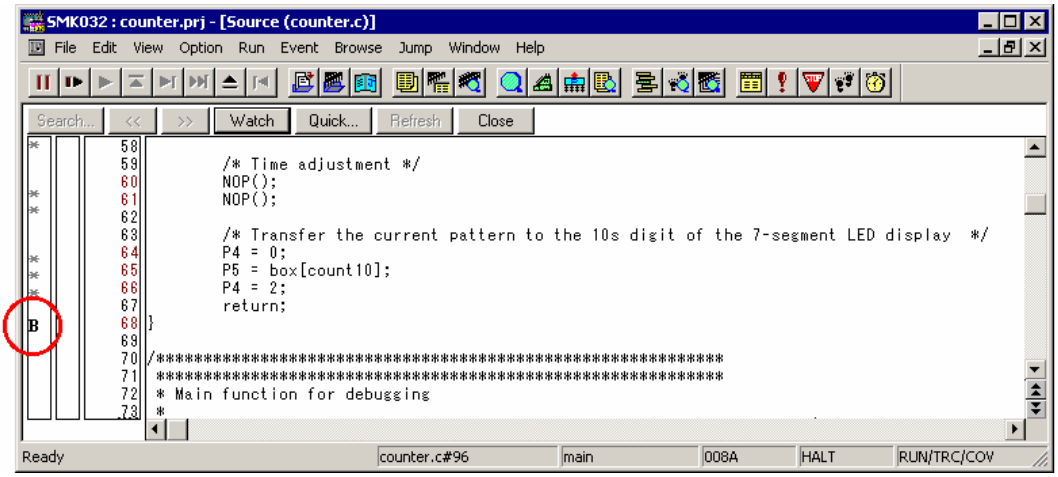
First, you will investigate what values the variables count1 and count10 take on when the LED display routine (putLED() function) is executed.

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 breakpoint 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.



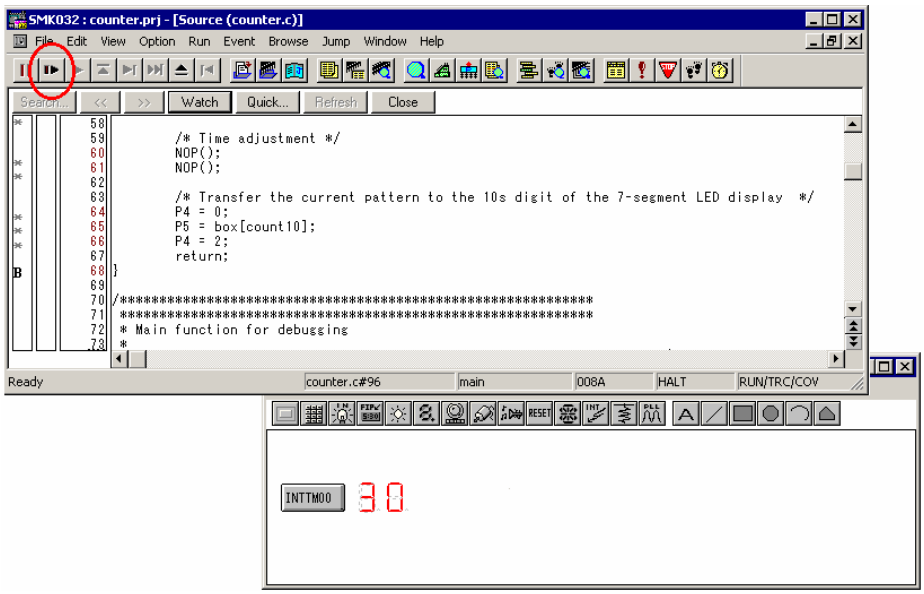




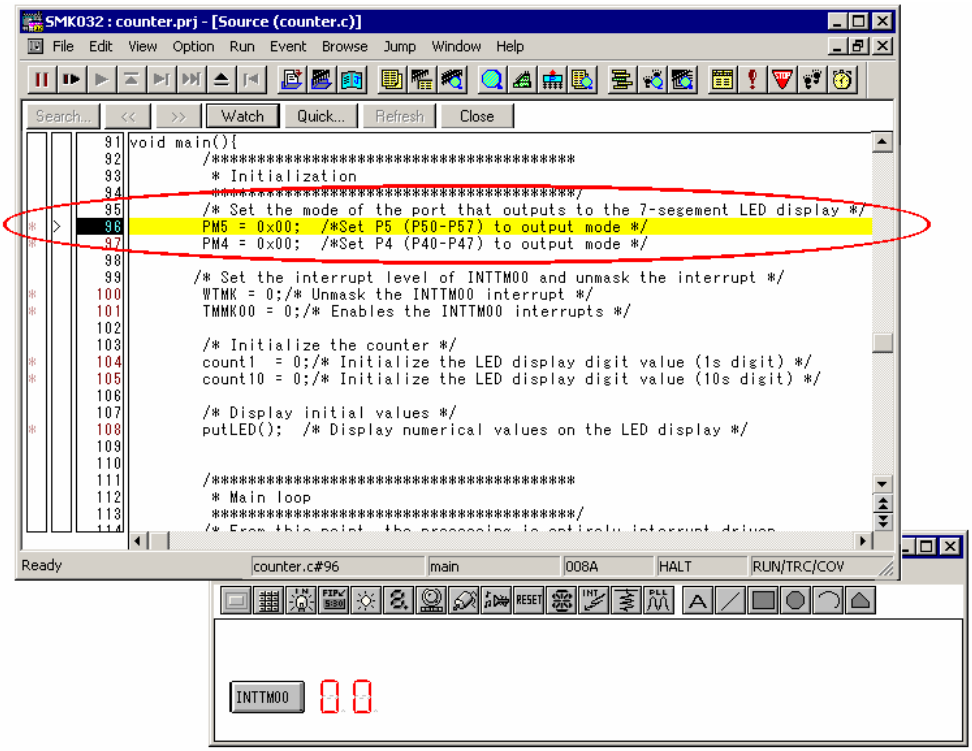
A break point is now set at line 68 (for the 78K0 and 78K4) or line 69 (for the 78K0S).

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.

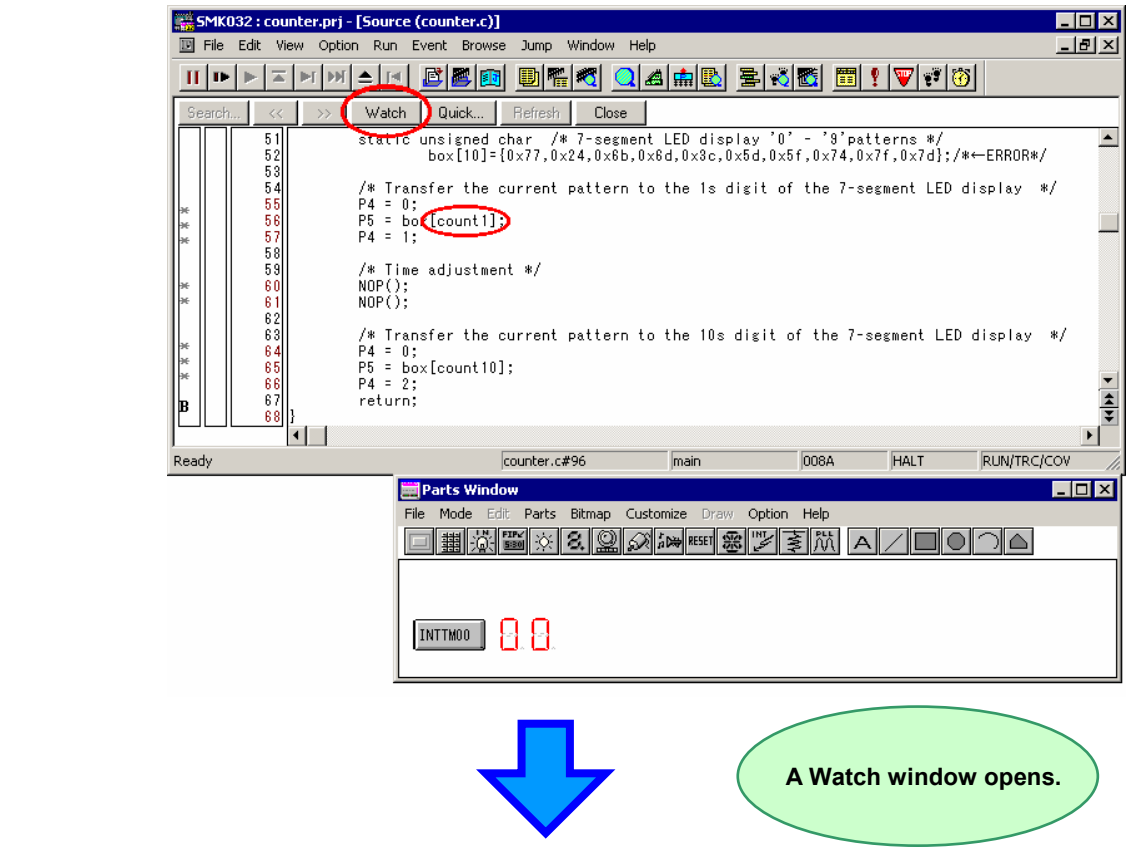


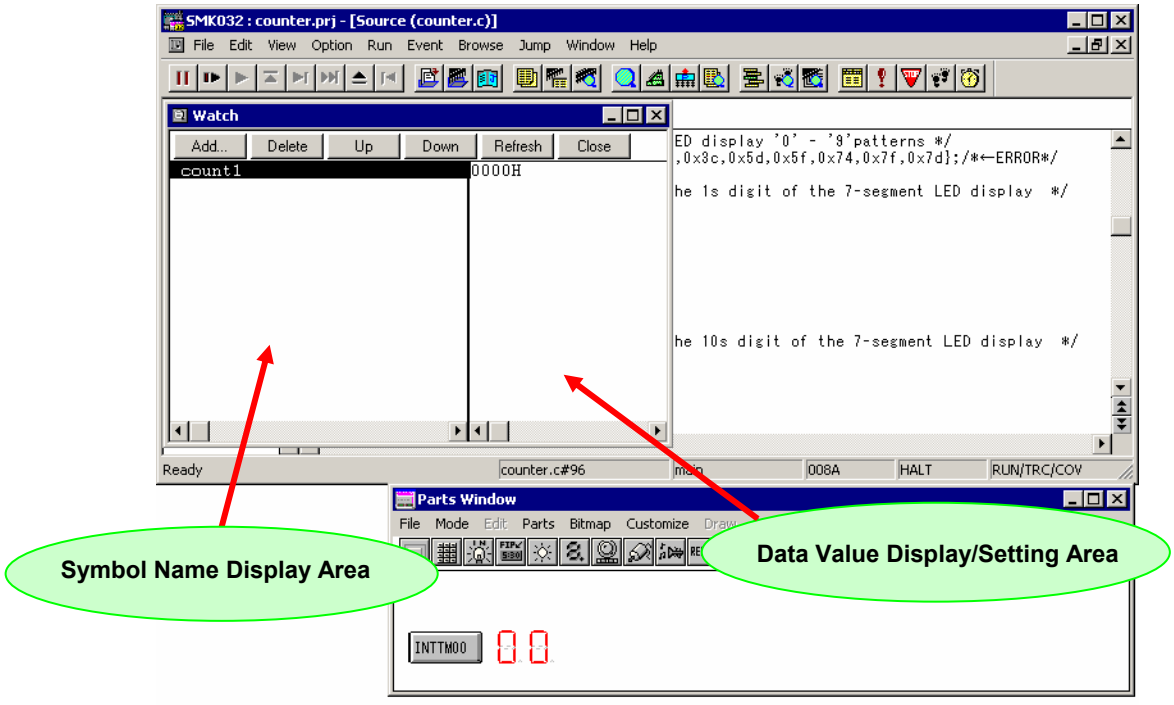
When program execution starts, the program stops almost immediately at the break point. The color of the line where the program stopped changes to yellow and a ">" symbol is displayed in the second column.

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.



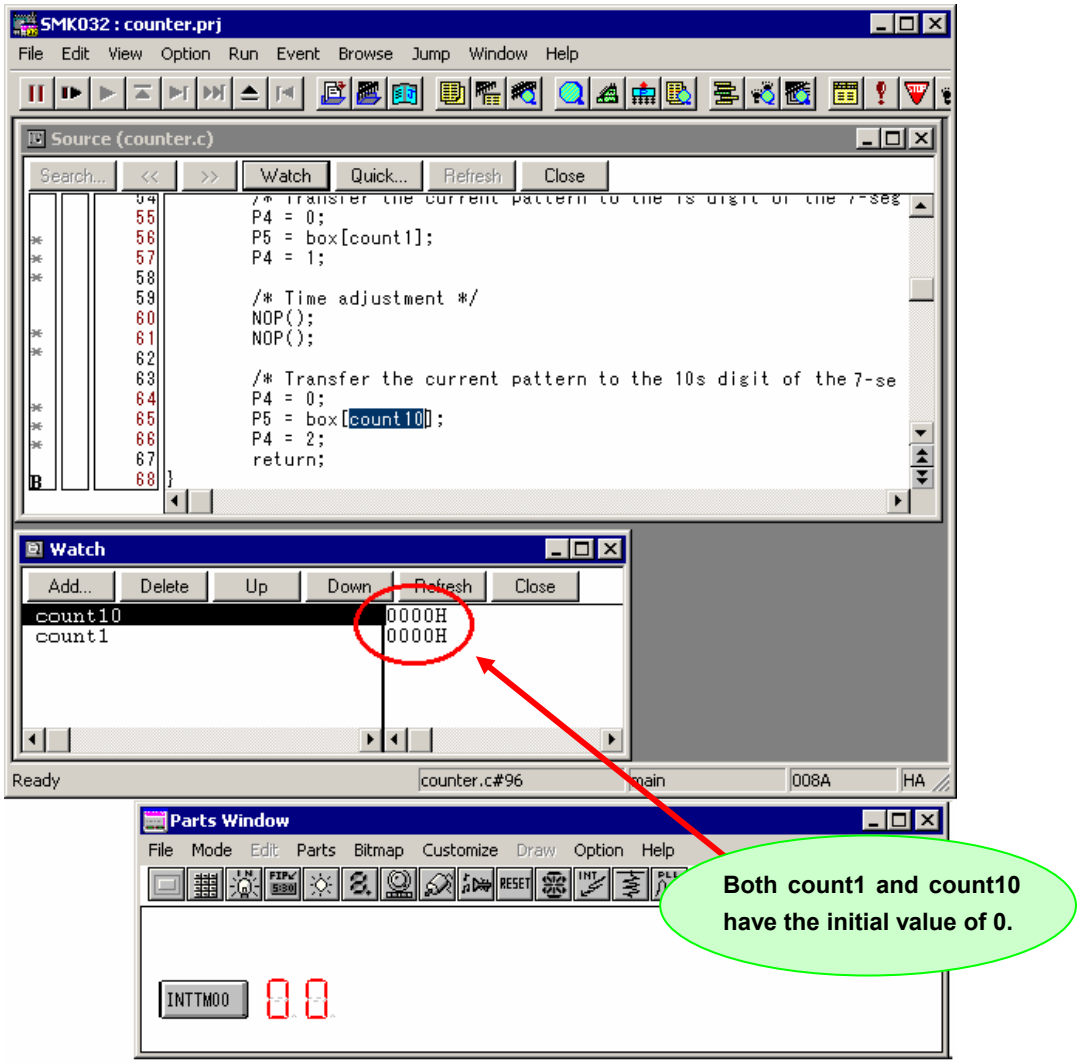


The Watch window contains a Symbol Name Display Area and a Data Value Display/Setting Area.

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.



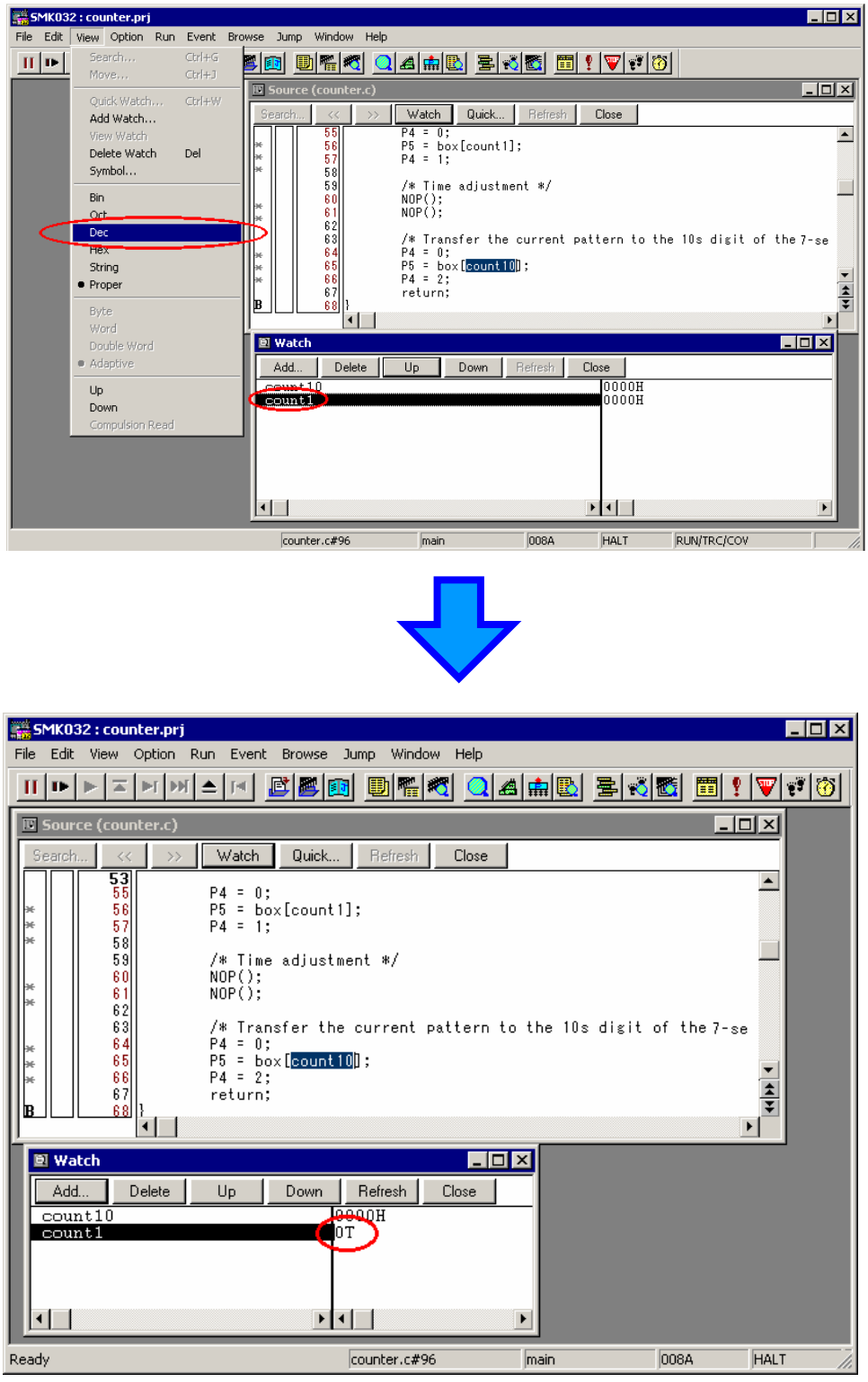
Looking at the Watch window you can confirm that both count1 and count10 have an initial value of 0.

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.

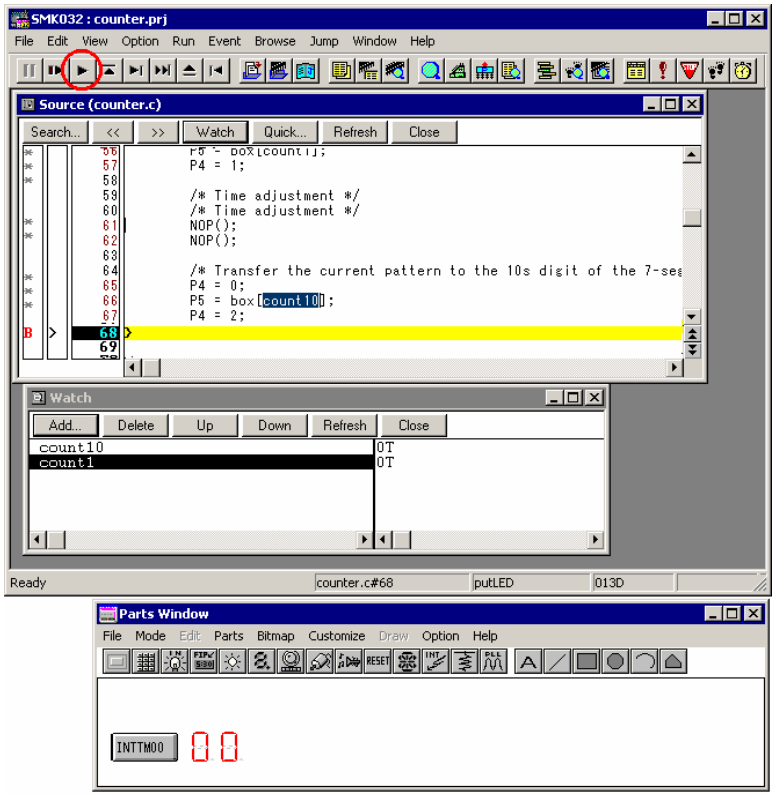


Similarly, change the format for cout10 to decimal.

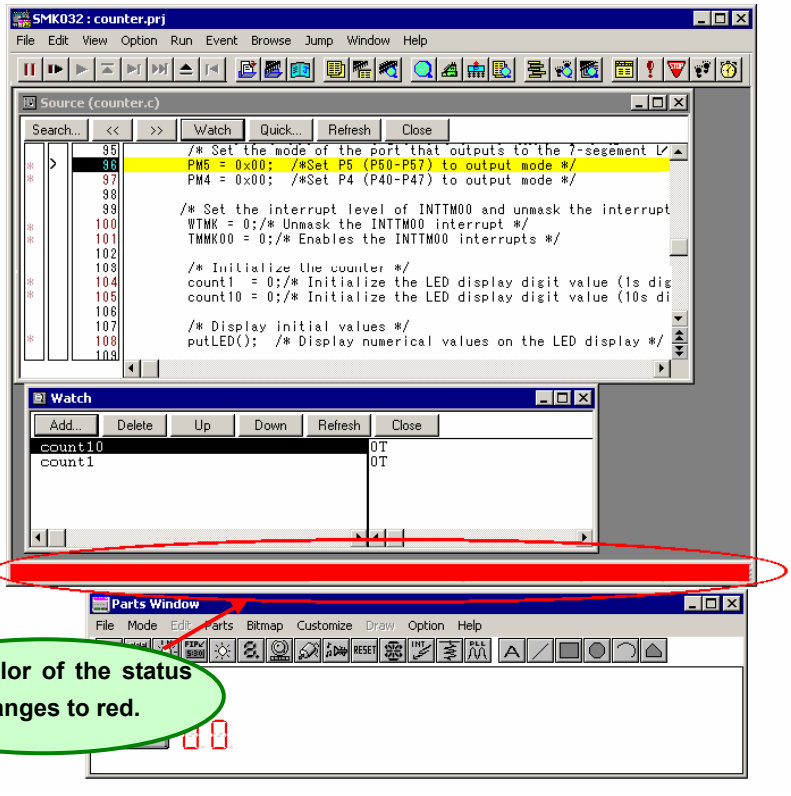
Next, you will determine if information is being correctly sent to the LED display when you click the INTTM00 button.

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.

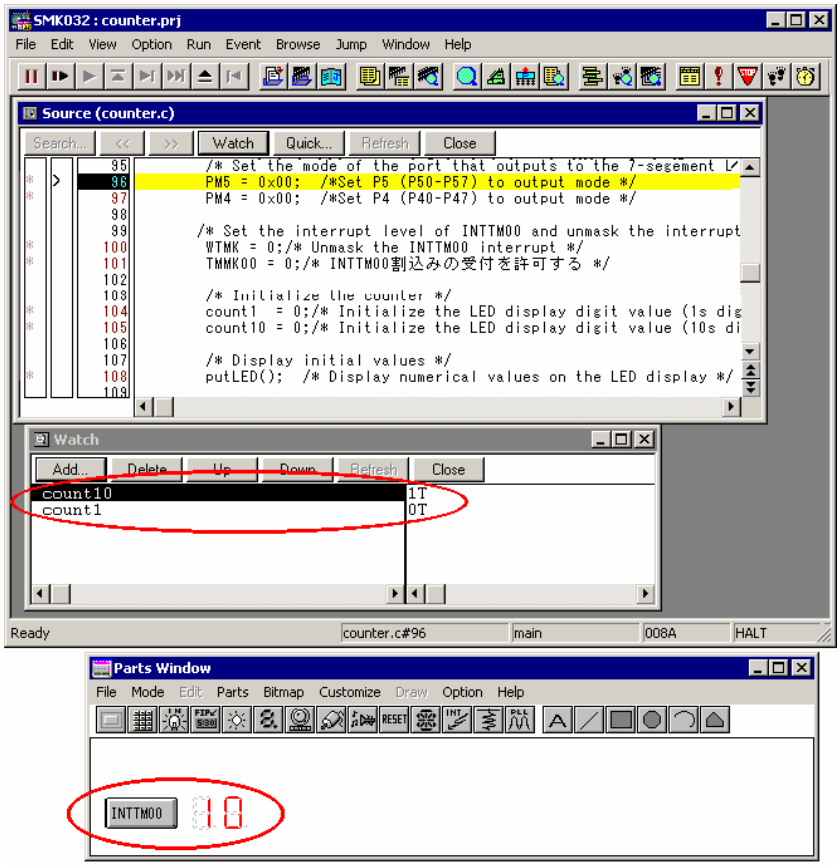


Program execution starts.



The color of the status bar changes to red.

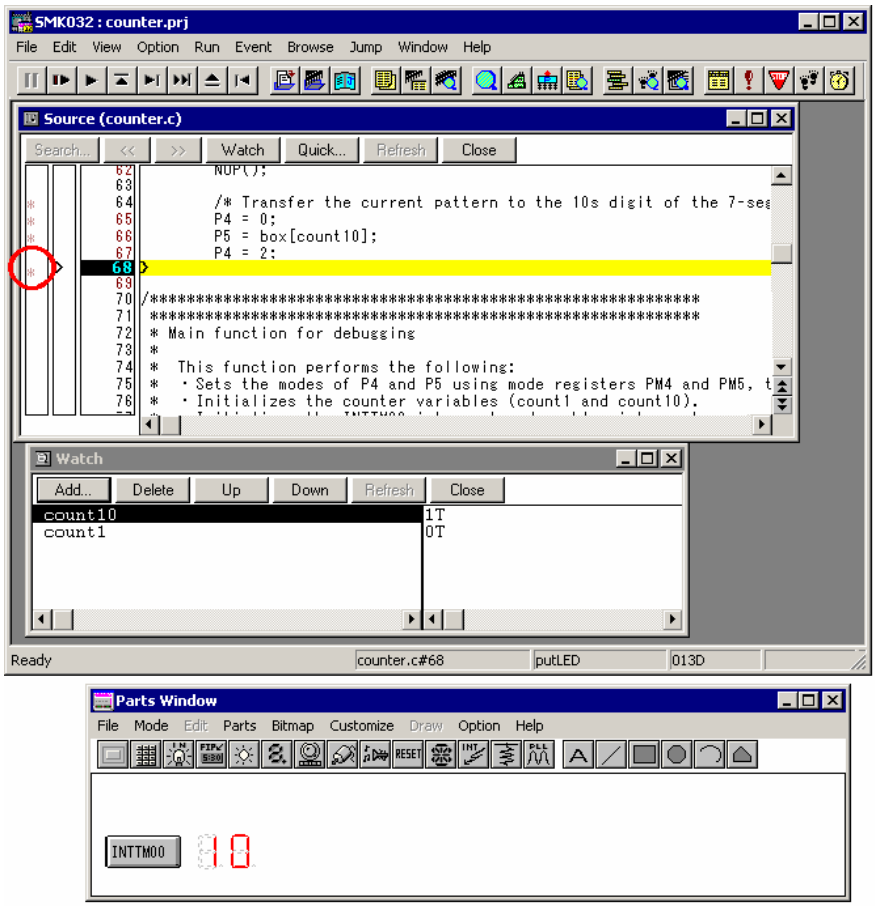
Let's look at the execution of the LED display routine (putLED() function) when the INTTM00 button is clicked. Click the INTTM00 button and observe the contents of the Watch window.



When you click the INTTM00 button, the program stops at the break point.
Here, if the incremention 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 dos 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.



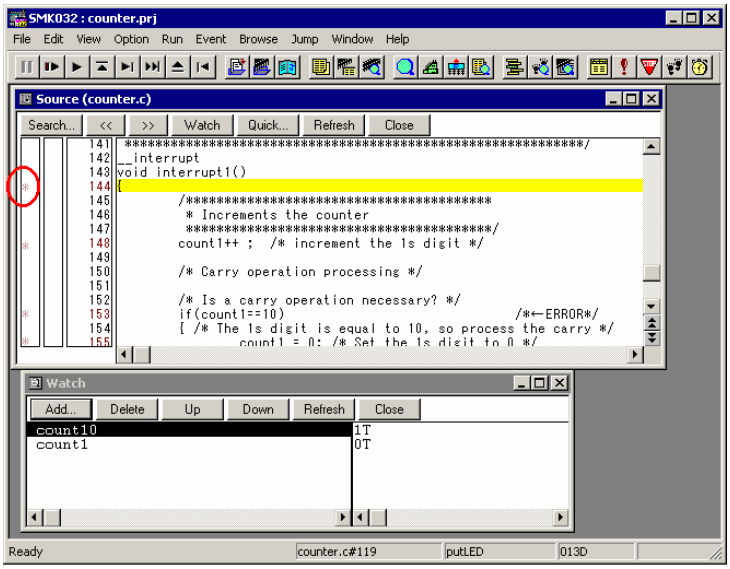
Click the "B" in the leftmost column. The "B" changes to "*" and the break point is removed.

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

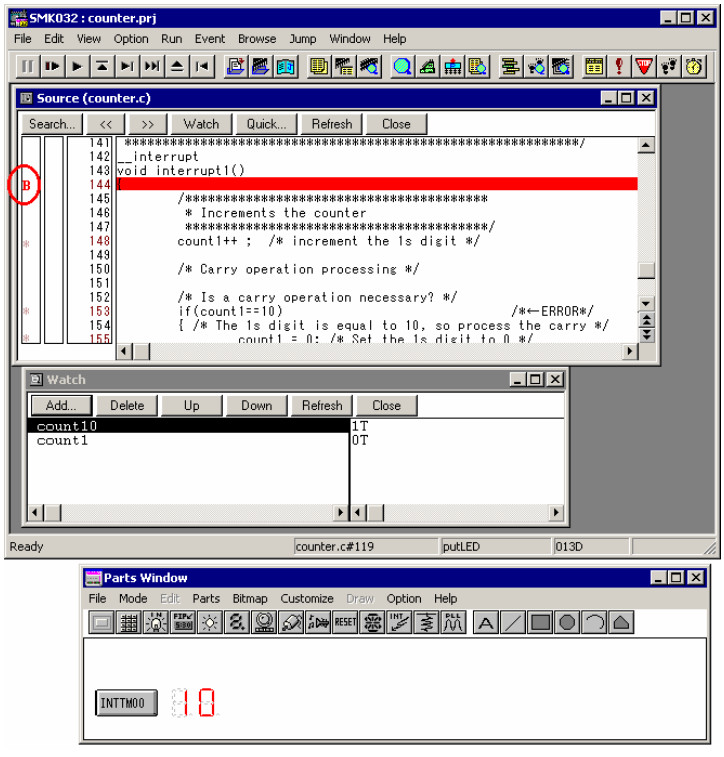
Let's investigate how the values of count1 and count10 change inside the internal interrupt handling routine (interrupt1() function) that executes when an internal interrupt occurs.

Set a break point at line 144.

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



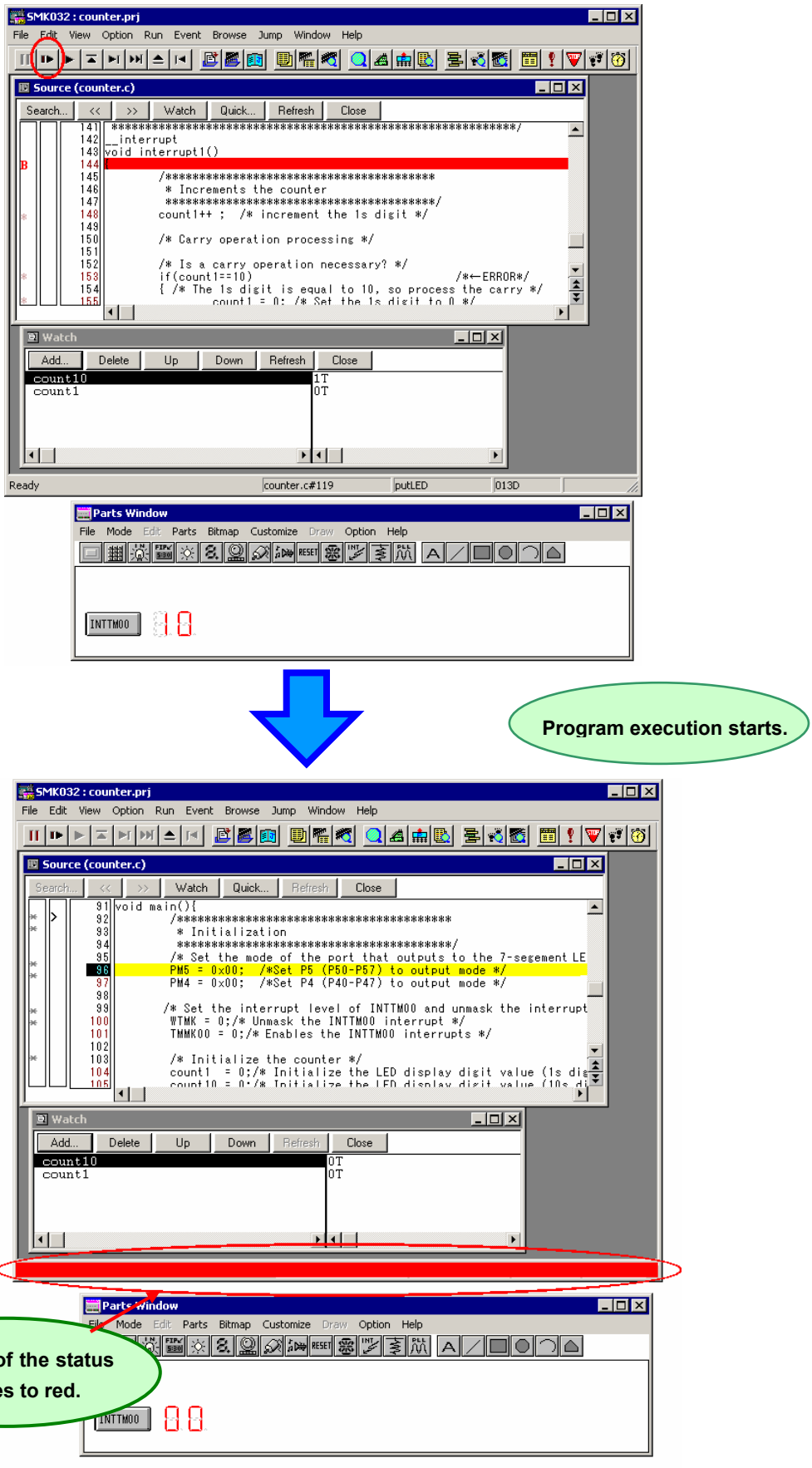
The "*" changes to "B".



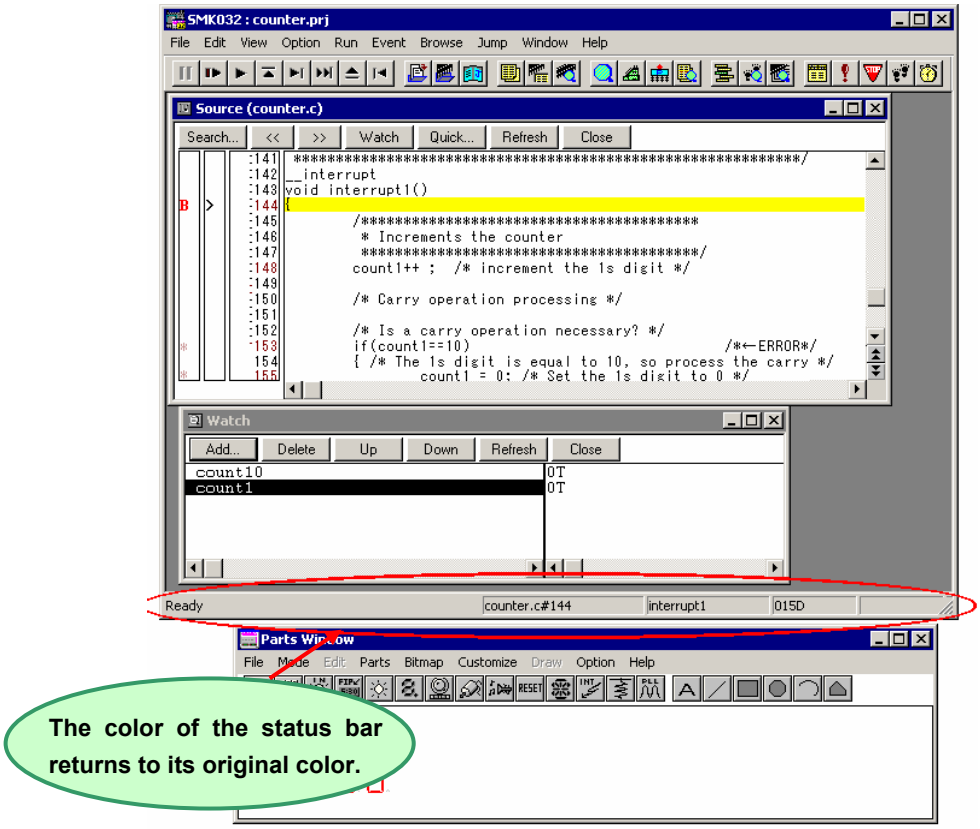
A break point is now set at line 144.

Next, perform a restart.

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



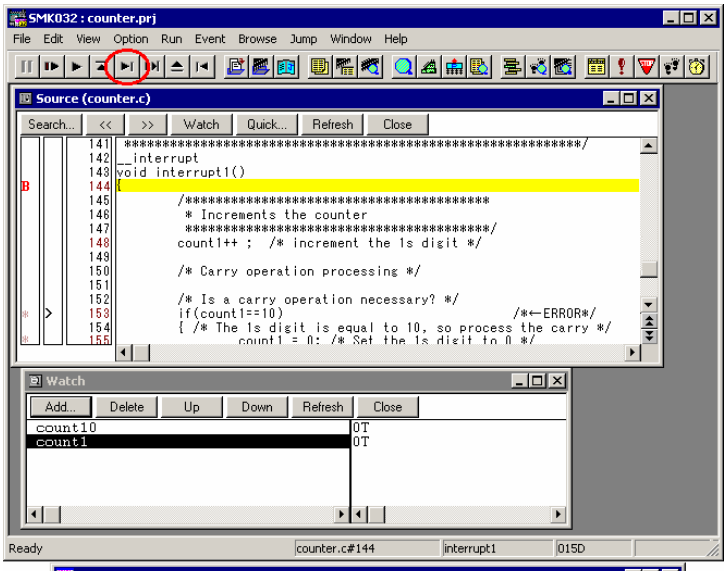
Click the INTTM00 button.

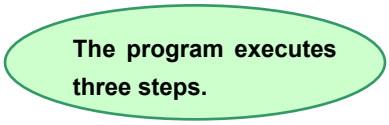


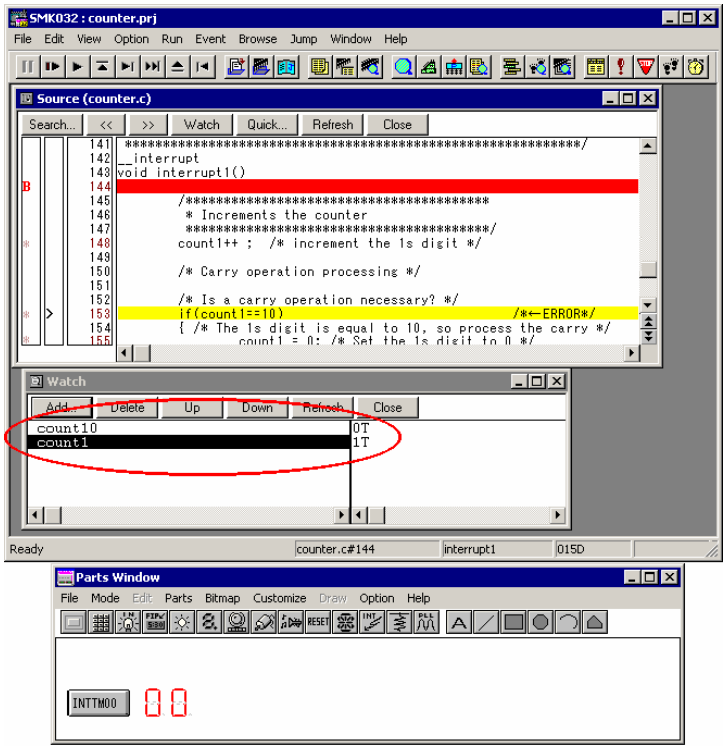
When you click the INTTM00 button, the program stops at the break point on line 144.

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.





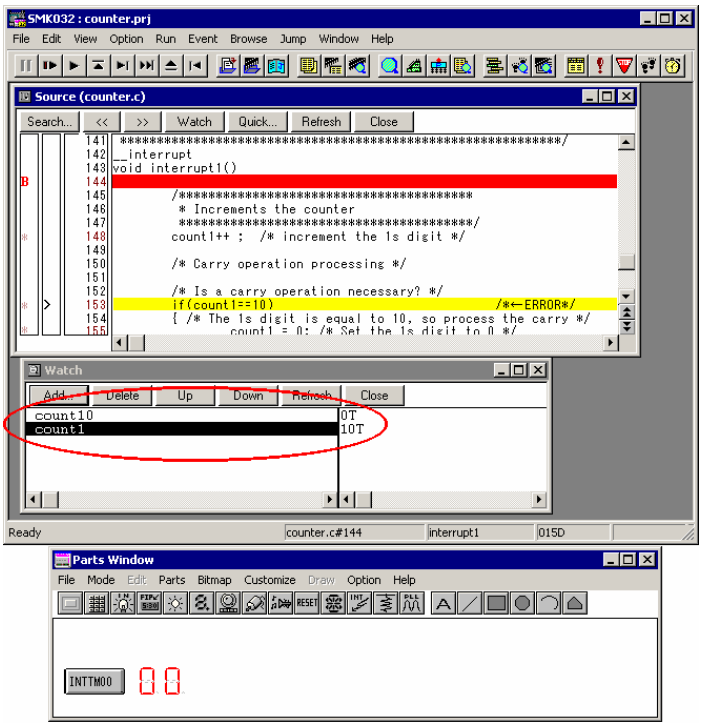


Observe the value of count1 in the Watch window after step execution at line148 completes.

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.



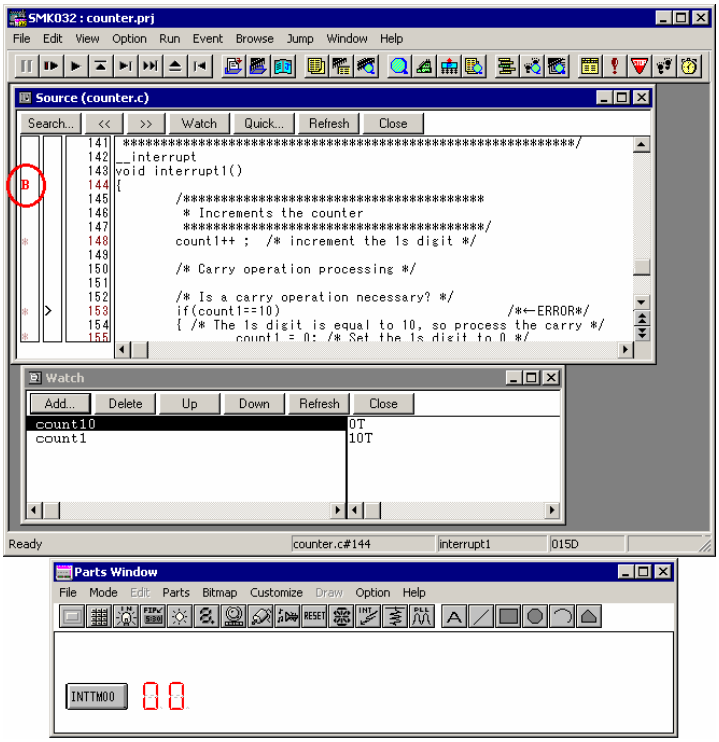
After step execution stops at line153, the value of count1 changes from 1 to 10.

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.

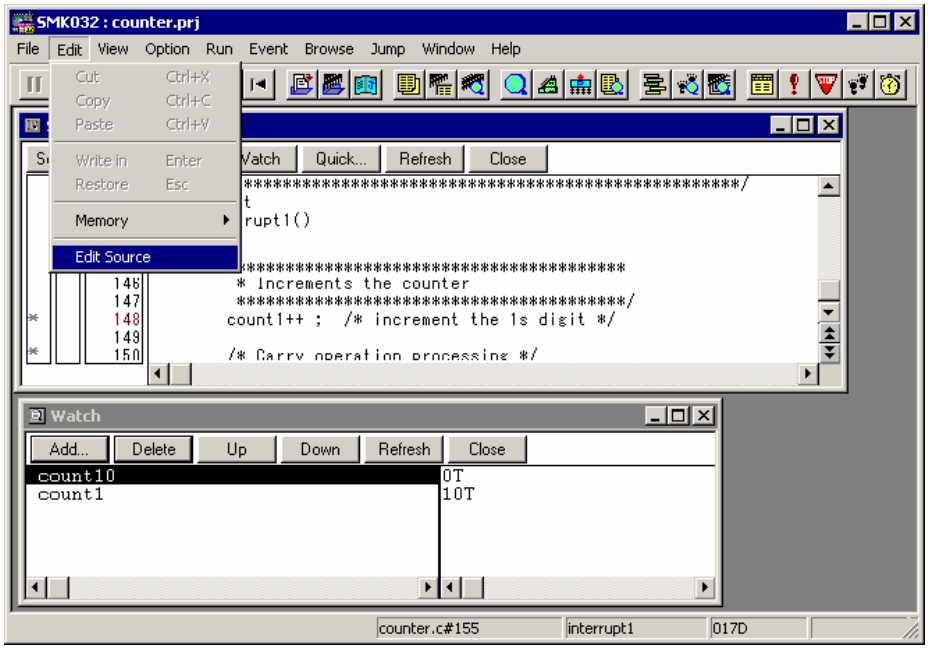
Click the "B" in the leftmost column. The "B" changes to "*" and the break point is removed.



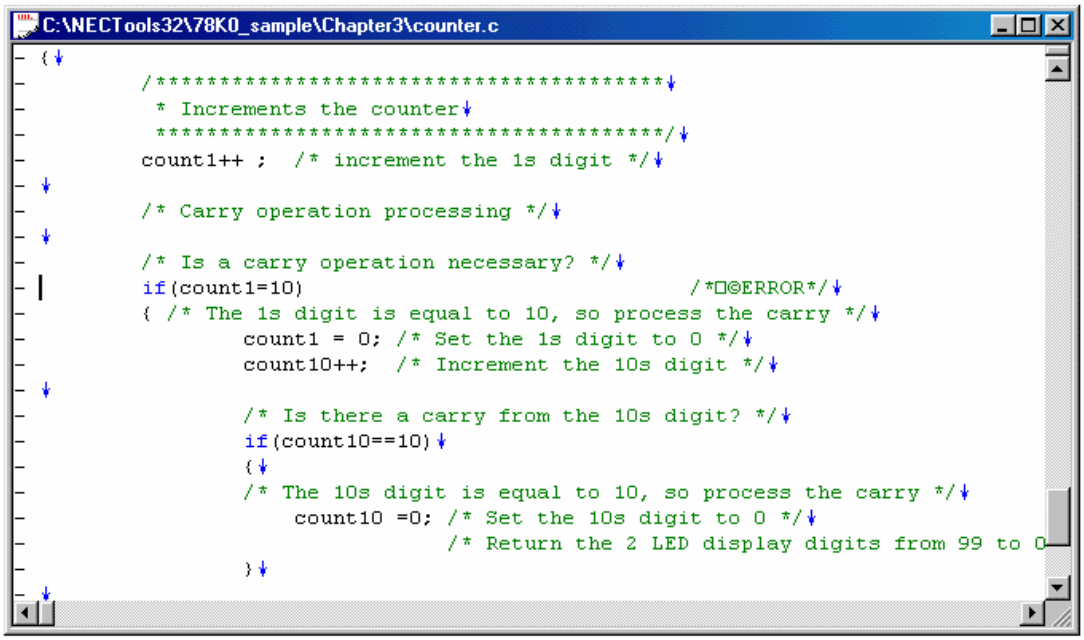
Editing the Source and Creating an Executable Program (2)

Next you will correct the error in the counter program.

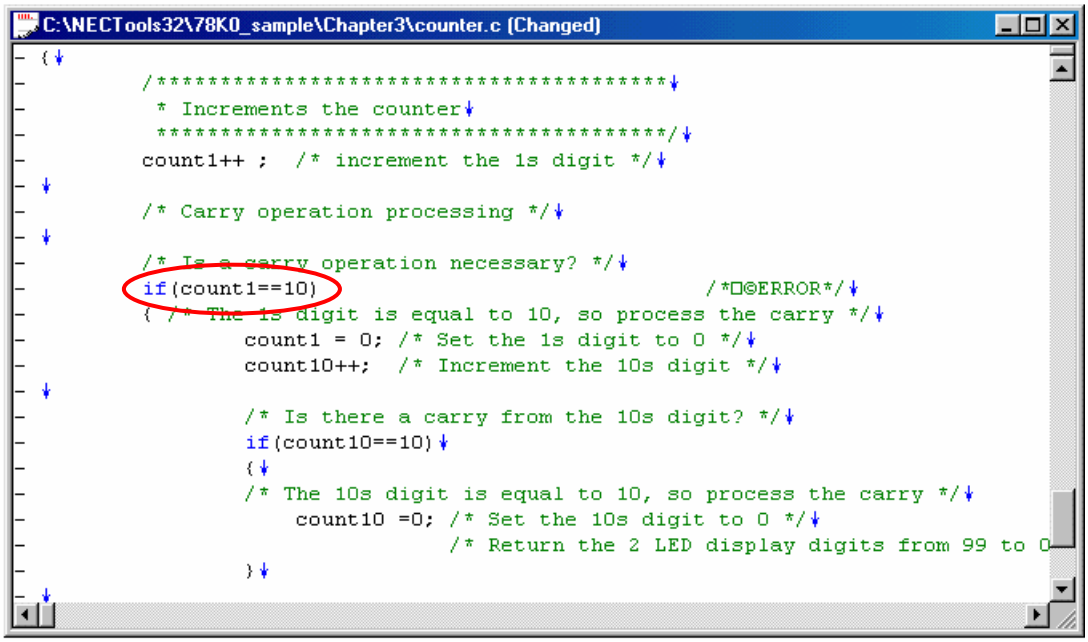
In the SM78Kxx Main window, from the menus select Edit->Edit source.



Editor opens.

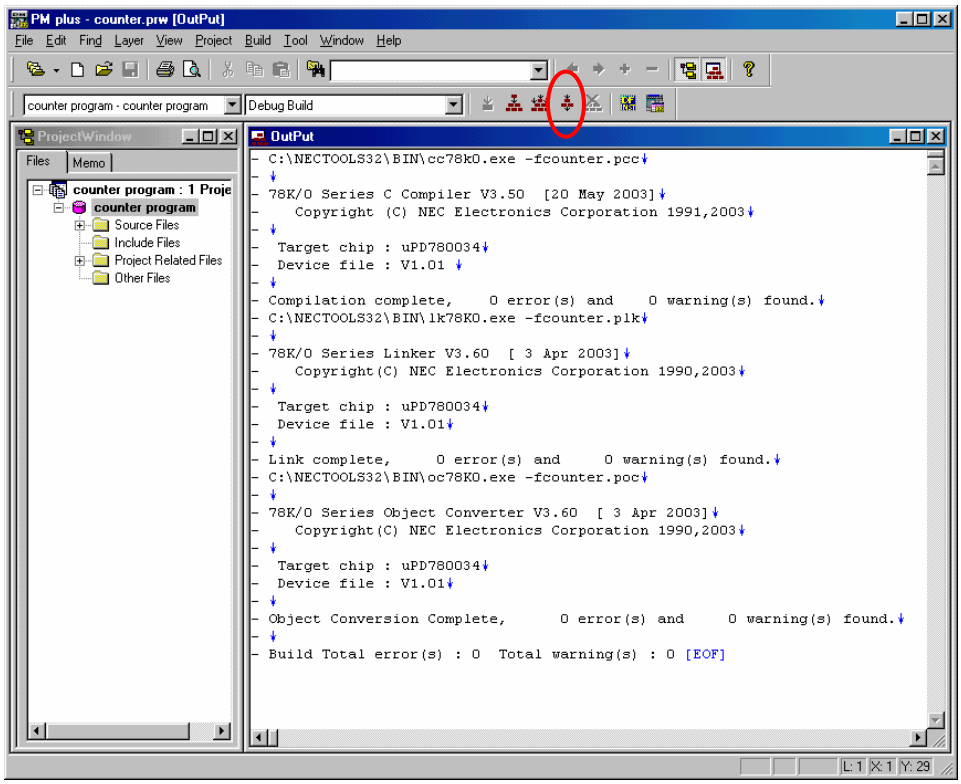


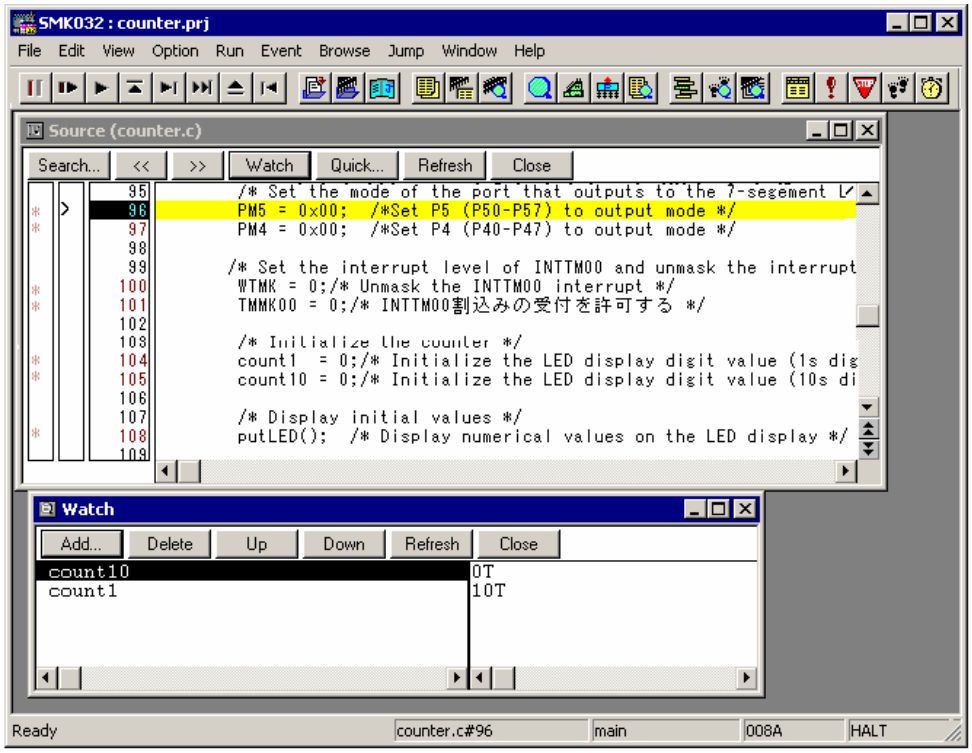
Correct line 153 by changing "count1=10" to "count1==10" in the if statement.



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.



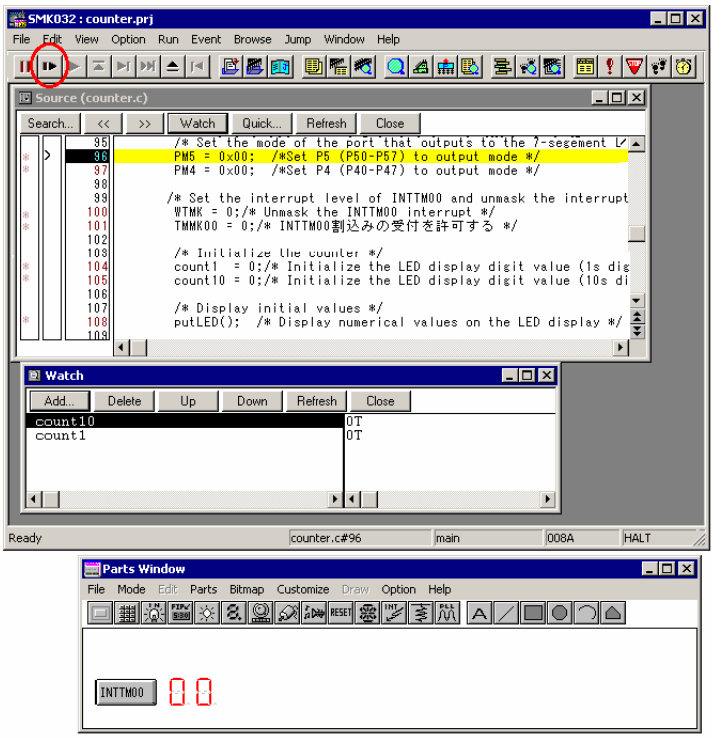


When build is completed, the SM78Kxx automatically downloads an executable program file.

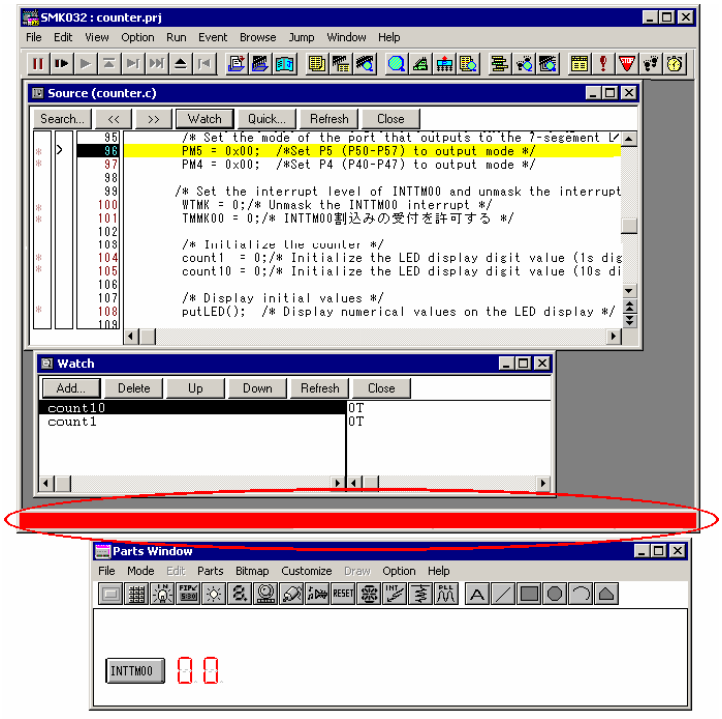
Executing the Program (2)

Now, perform a restart.

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

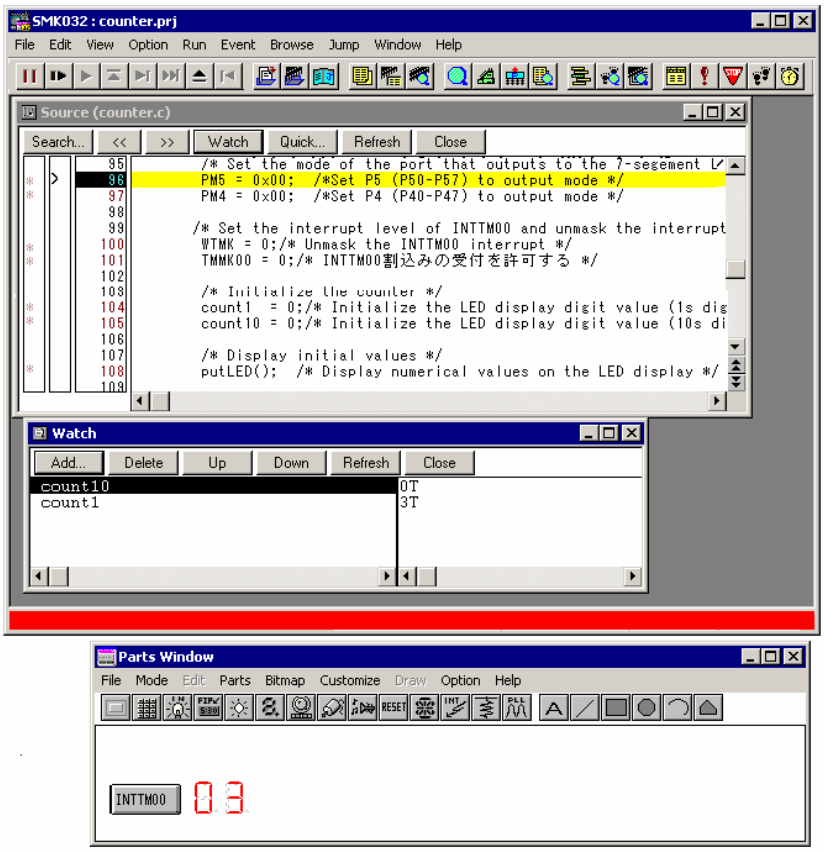


Program execution starts.



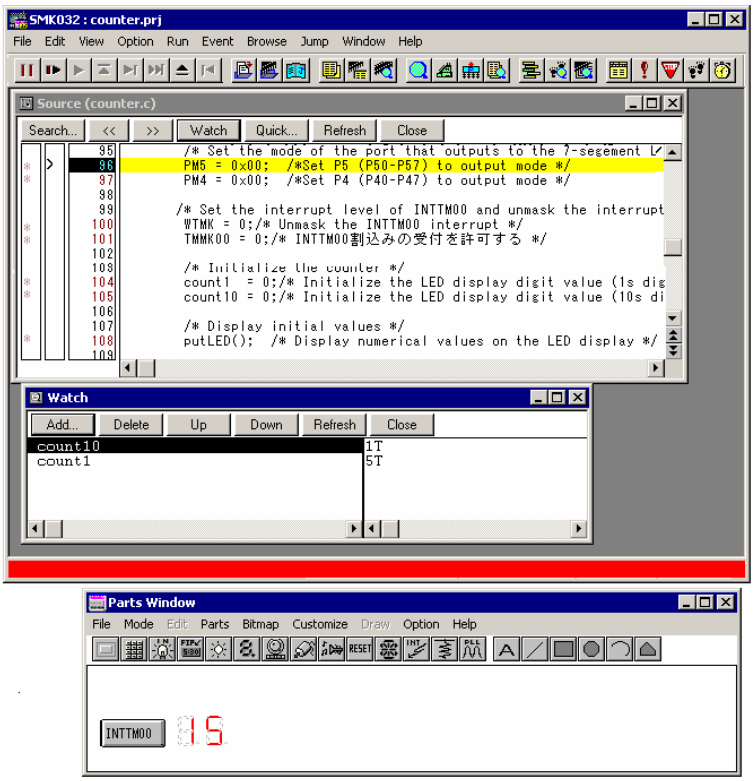
Confirm whether the incrementation works correctly.

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.



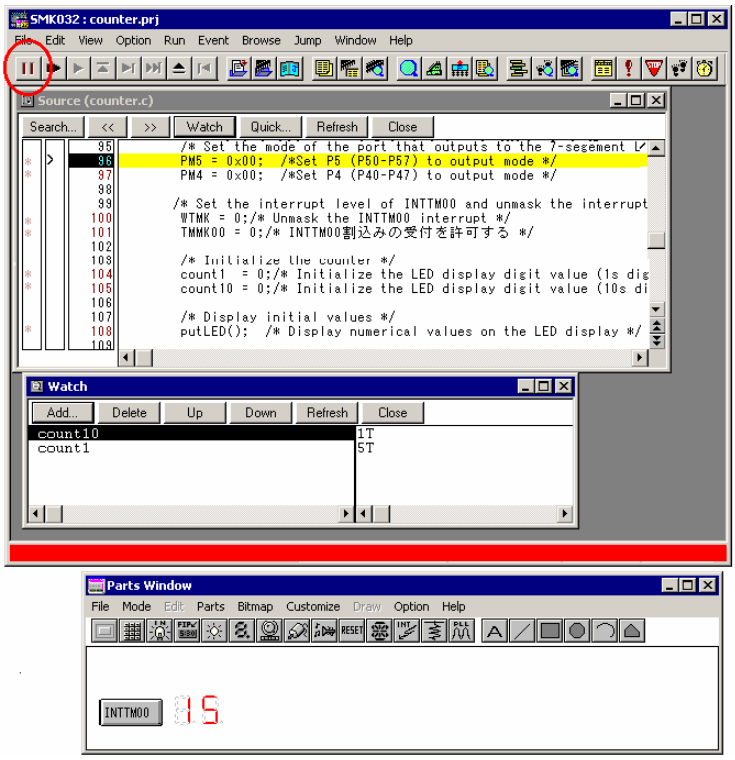
The 10s digit does, in fact, increment correctly.

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

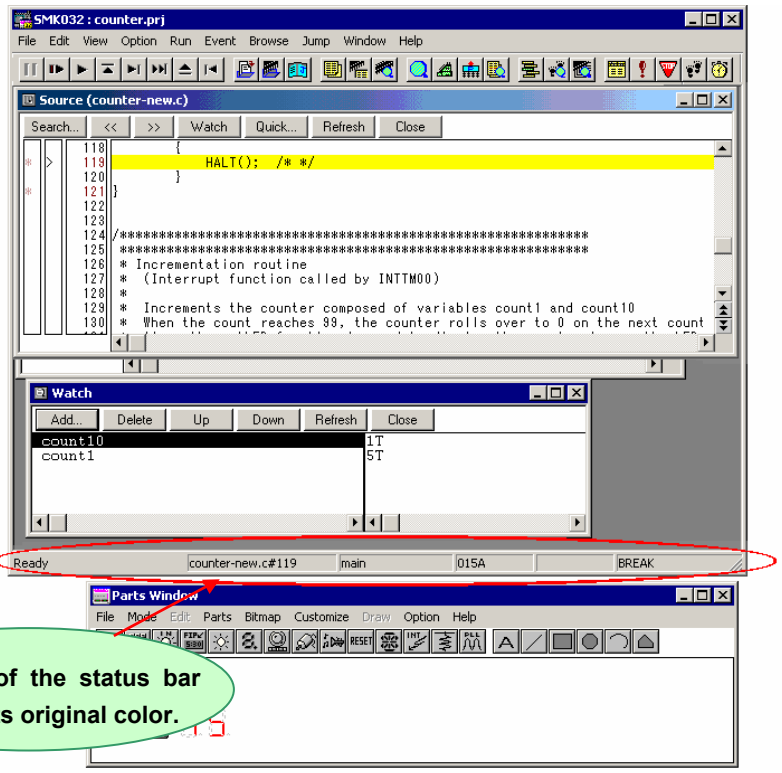
Lastly, you will confirm whether the counter overflow handling is working correctly.

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.

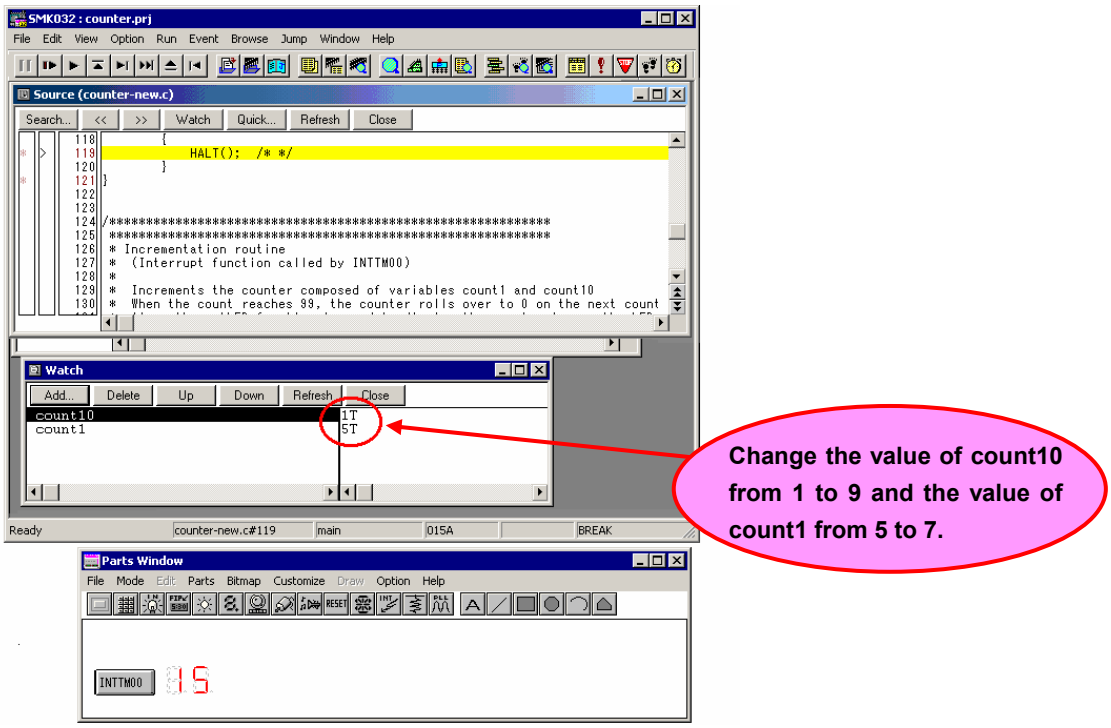


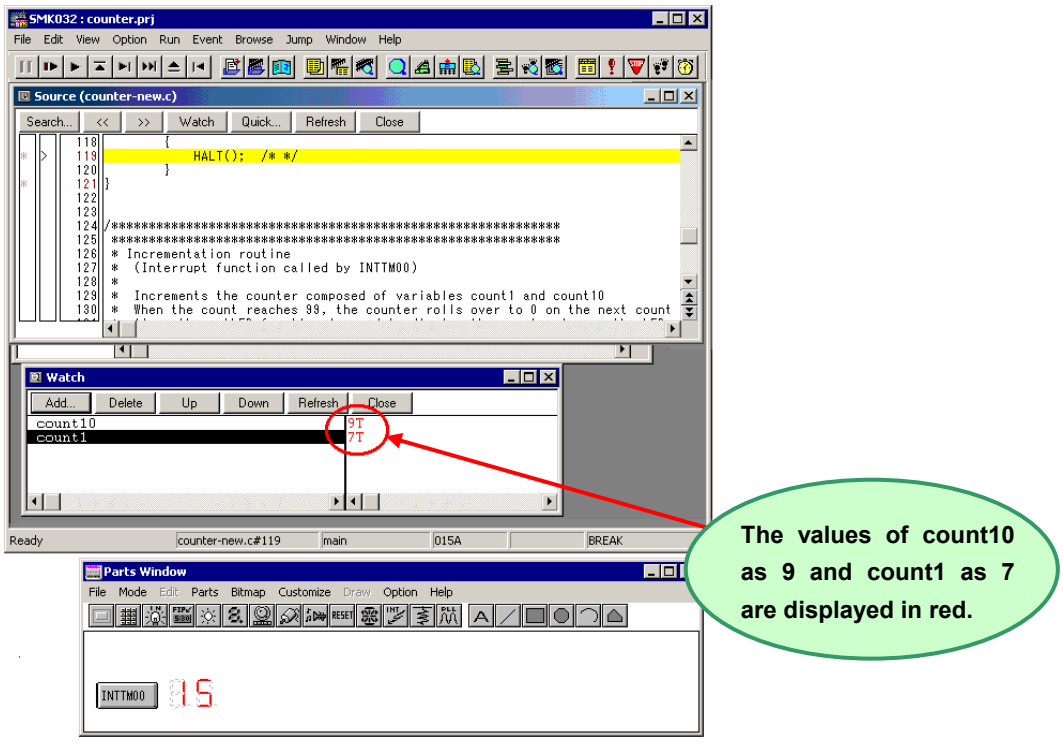
Program execution stops.



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

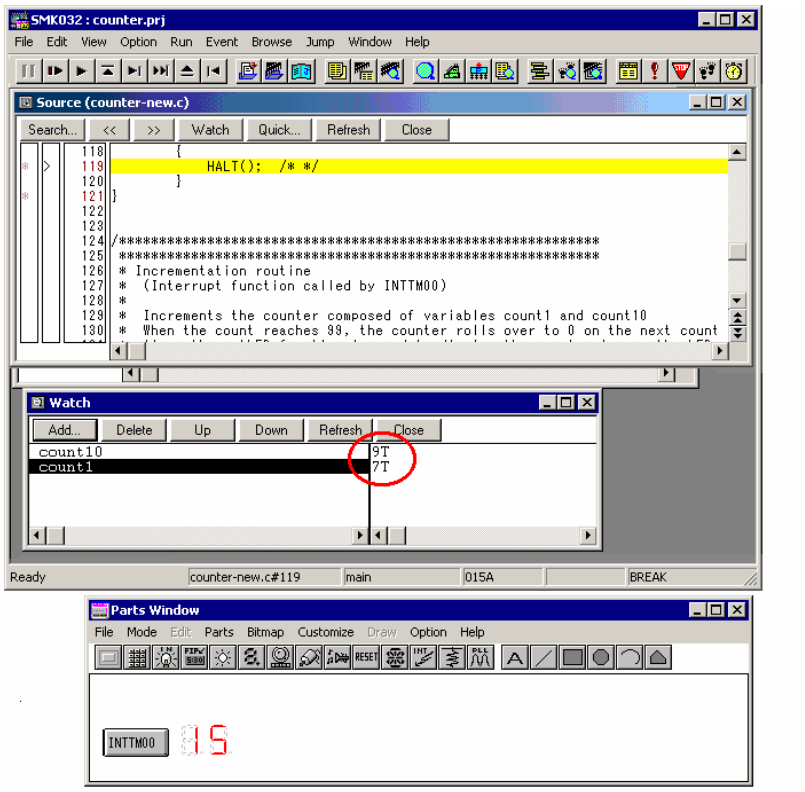
Next, in the Watch window, change the values in the Data Value Display/Setting area.
Move the cursor to the Data Value Display/Setting area in the Watch window and change the value of count10 to 9 and count1 to 7.





Now, with the values changed and displayed in red, press the Return key.

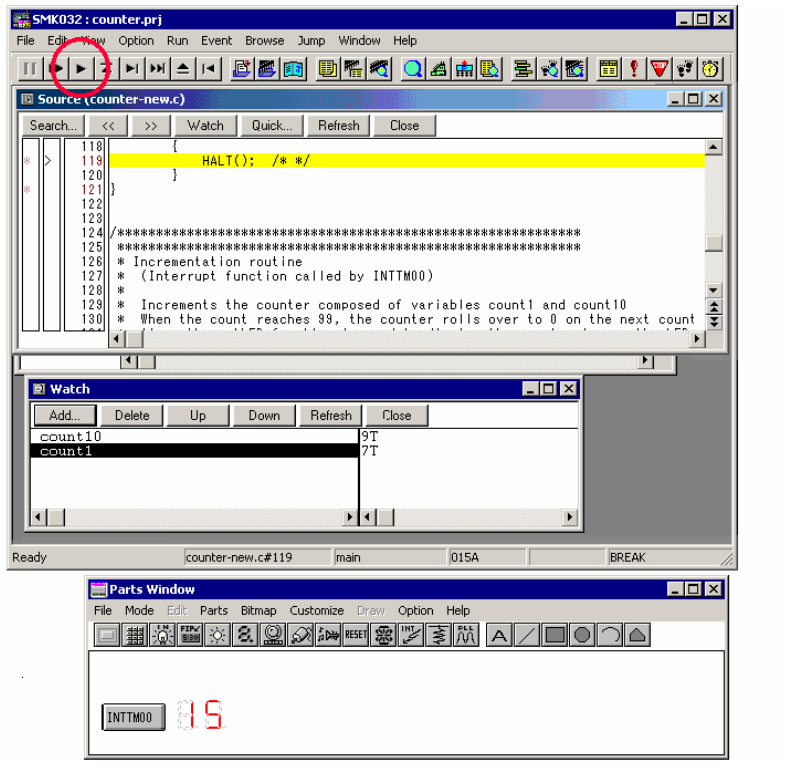
The values of count10 and count1 change from red to black.

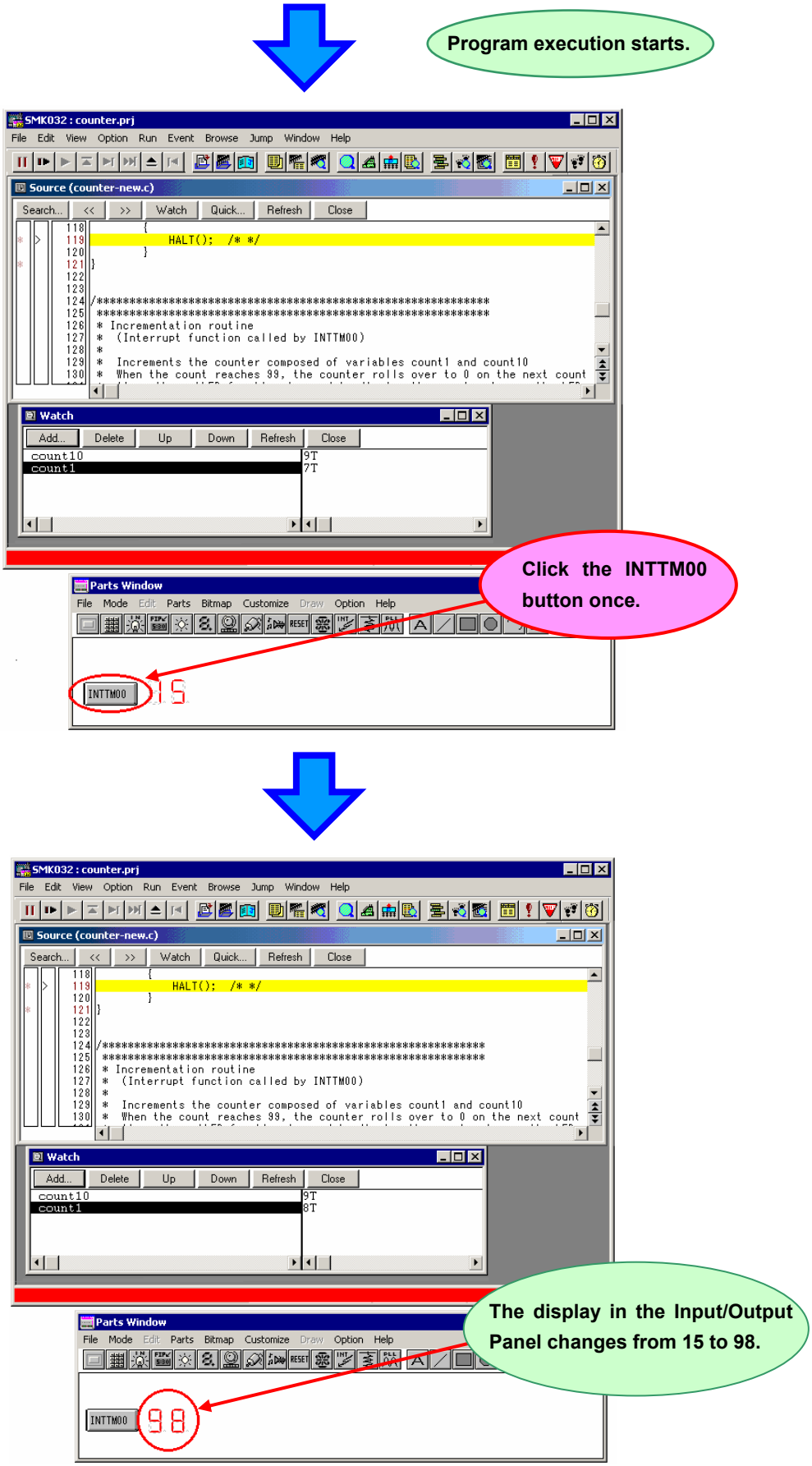


At this point, count10 has become 9 and count1 has become 7.

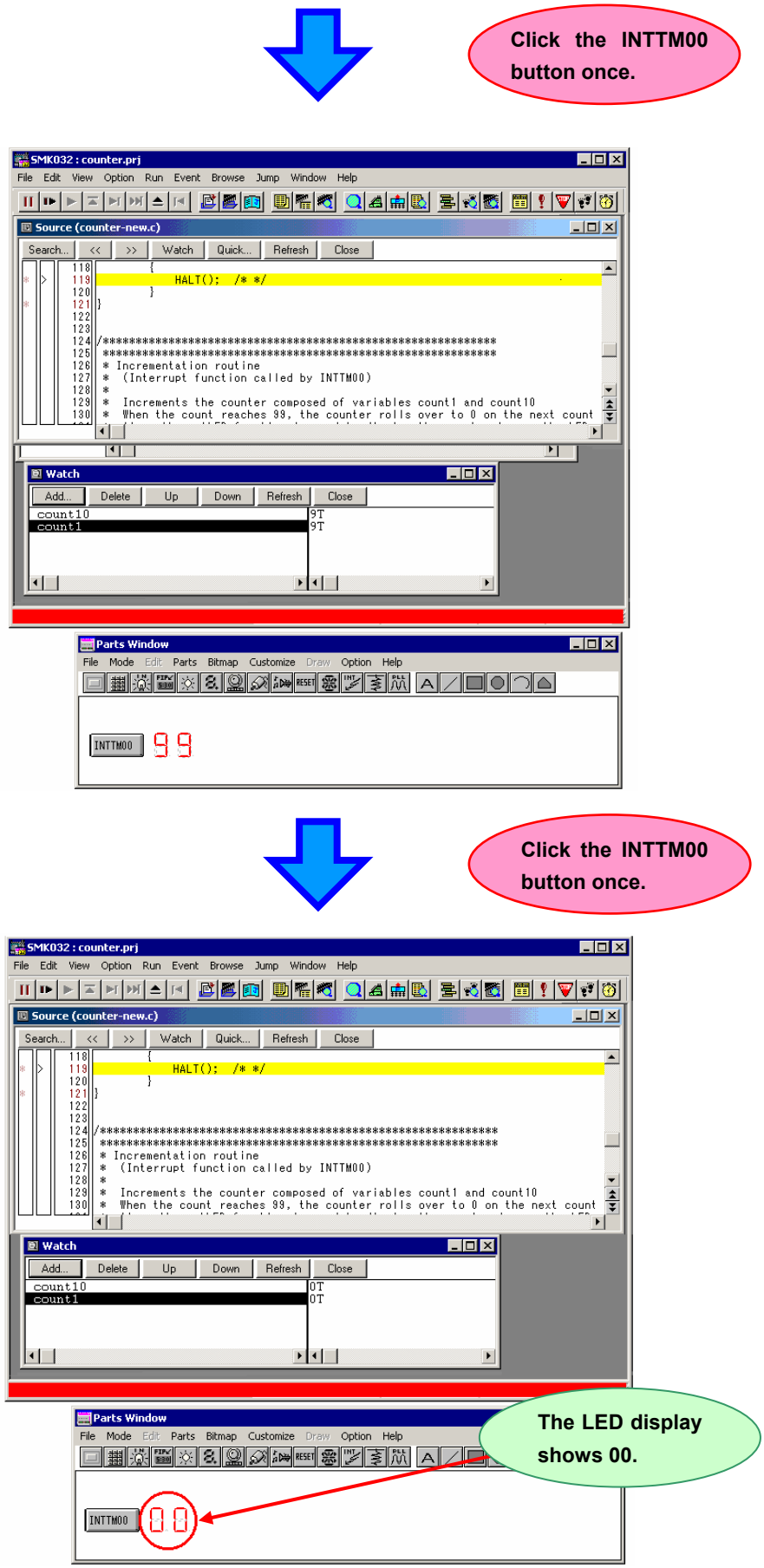
Now you can continue execution of the program.

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





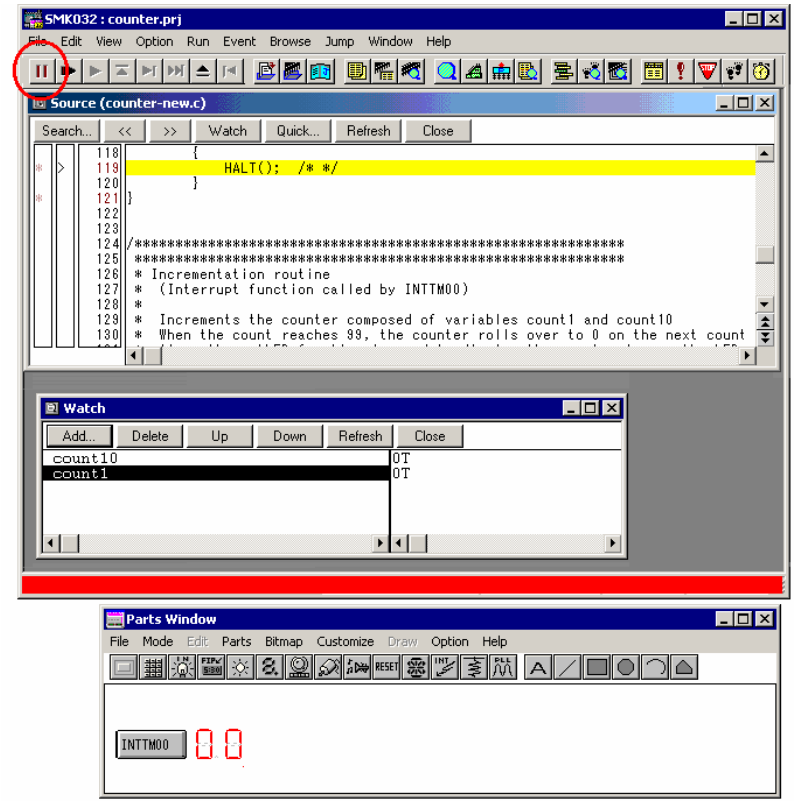
Now, the program is at the same point as if you had clicked the INTTM00 button 98 times.
Continuing on, click the INTTM00 button again.



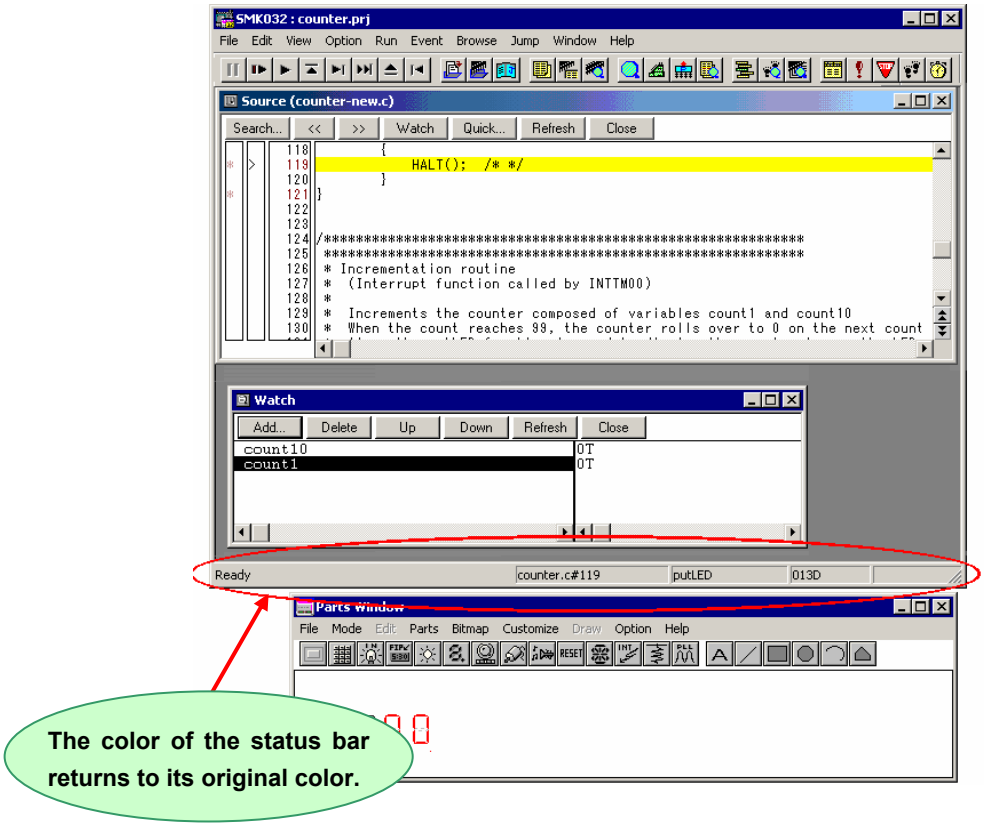
You have determined that the counter overflow handling operates correctly.

Stop the program.

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



Program execution stops.

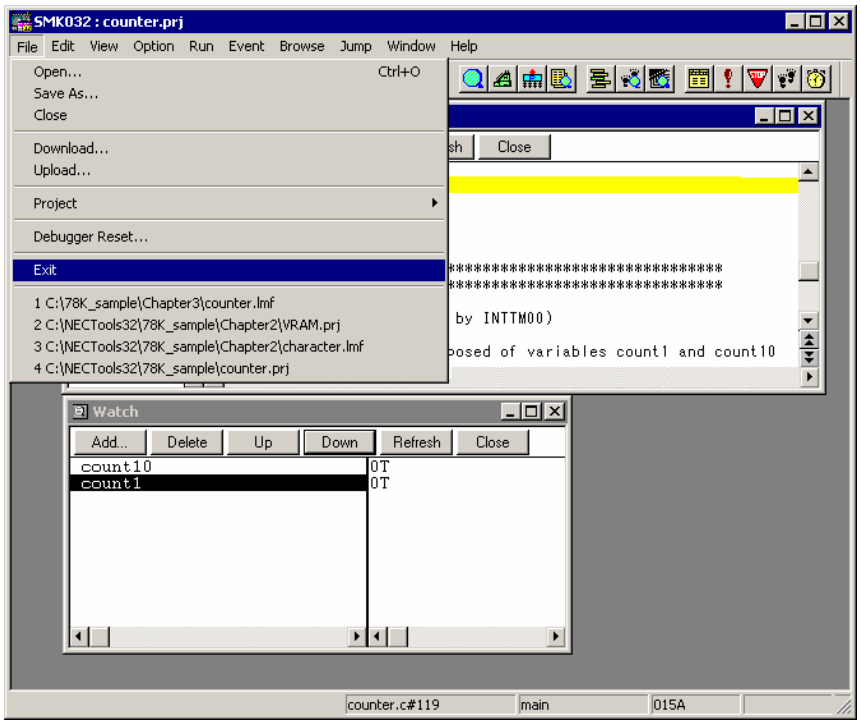


The color of the status bar returns to its original color.

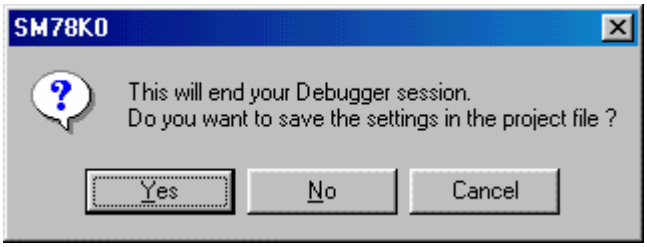
Exiting

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.



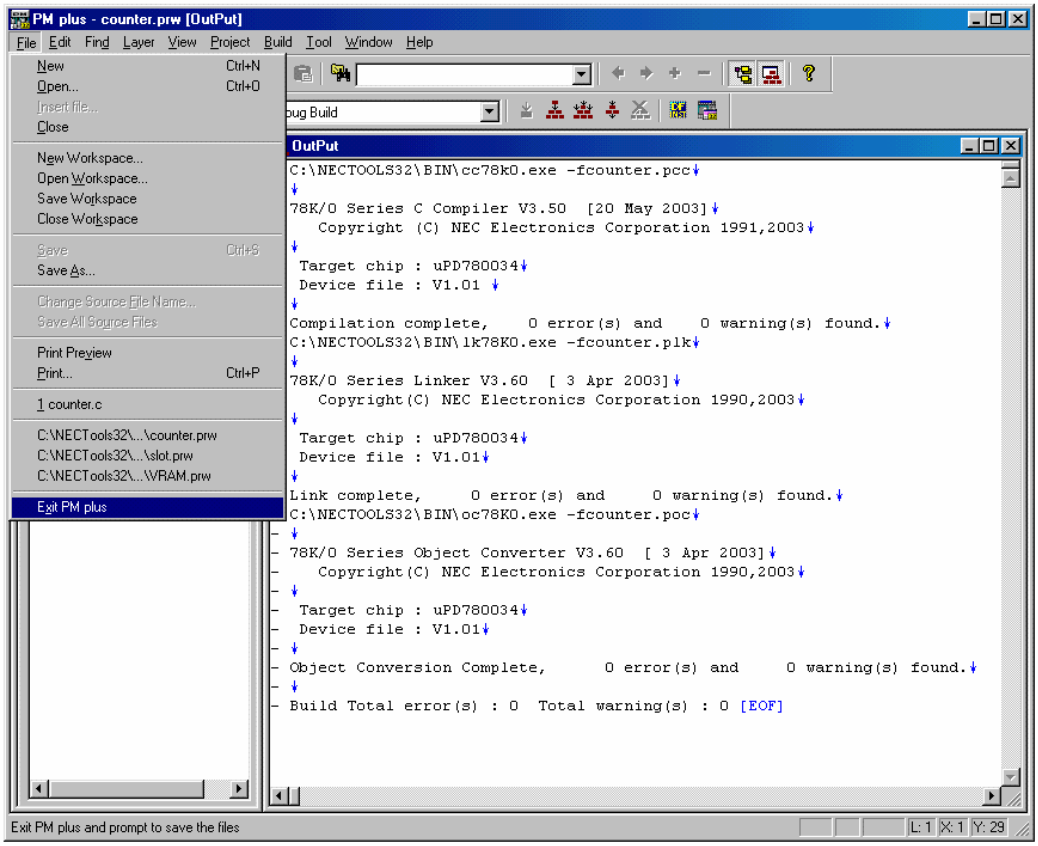
To save the settings performed in this chapter, such as the Input/Output Panel settings, click Yes button. To discard the settings, click Yes button. (To return to PM plus, click Cancel button.)

"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.

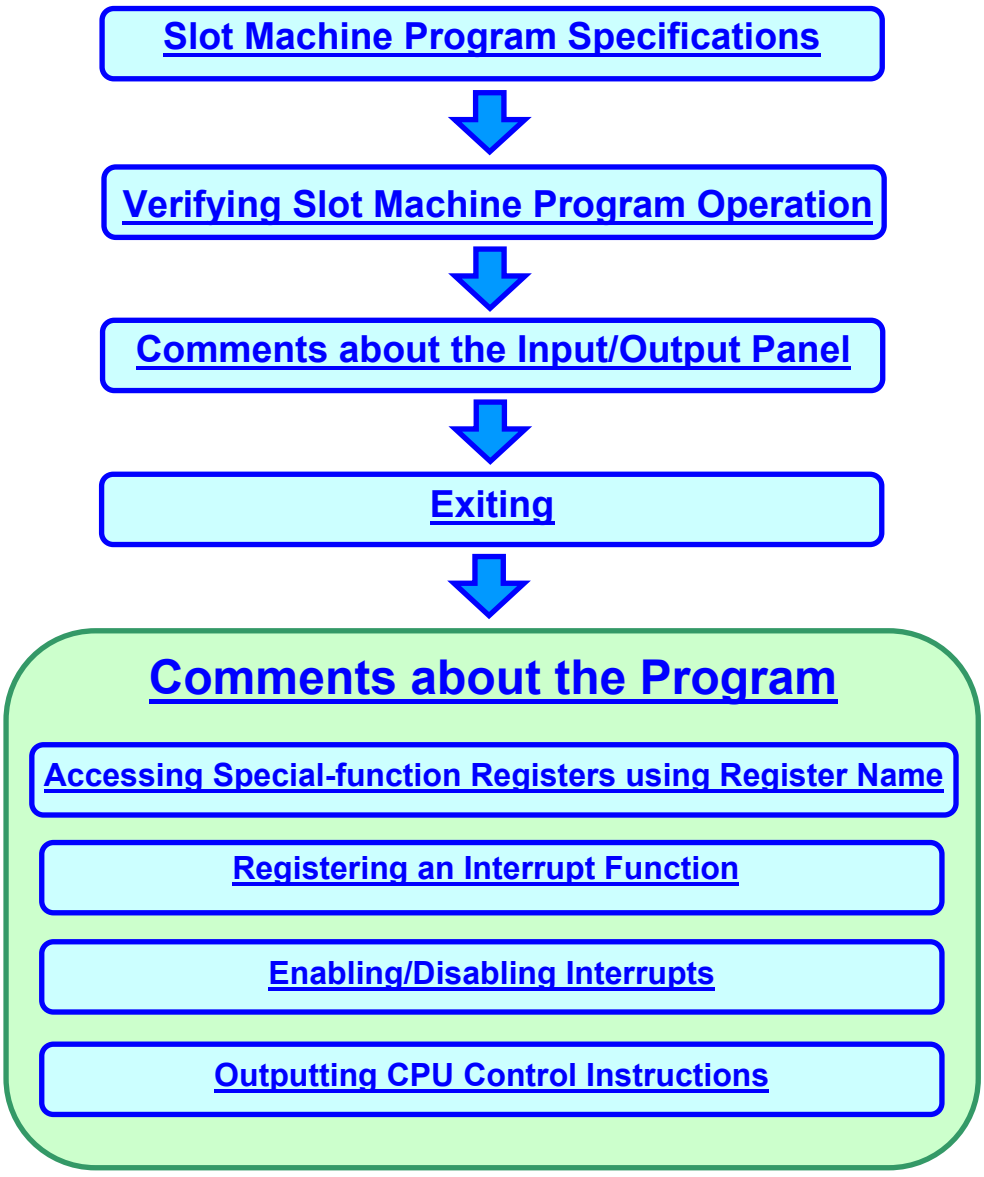
Chapter 4 Programming

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) interrupt_func.c **(function definition)**	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**
	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

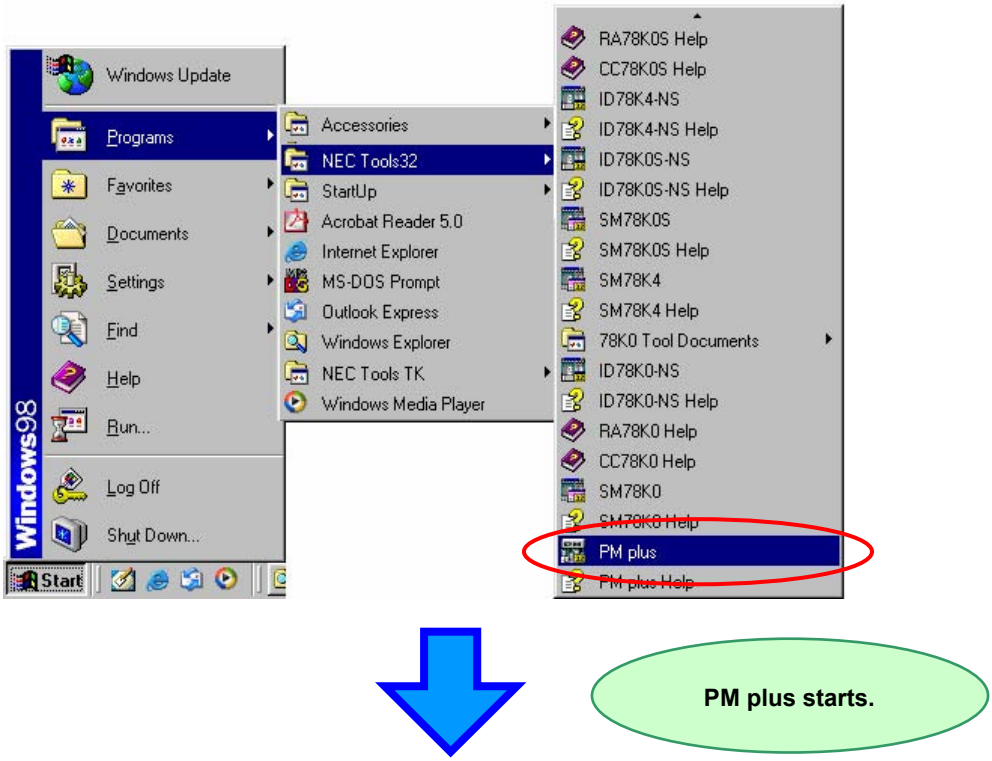
- The following I/O ports are used for LED display control and interrupt input:
  - For the 78K0: P0, P4 and P5
  - For the 78K0S: P1, P2 and P54
  - For the 78K4: P2, P4 and P5

	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

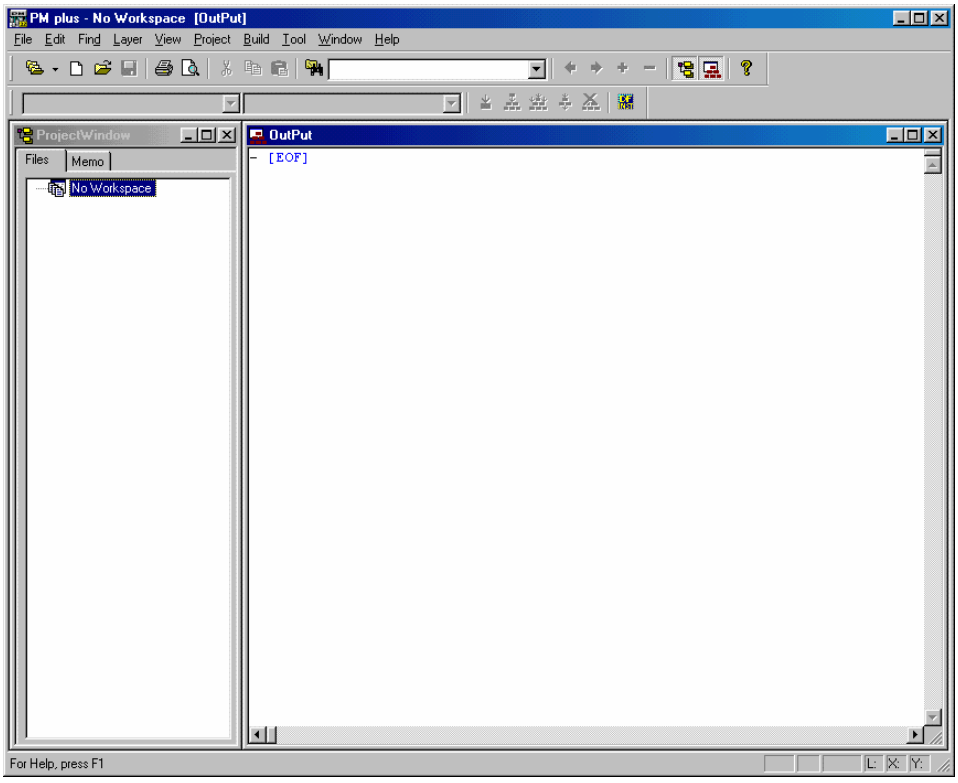
Verifying Slot Machine Program Operation

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.

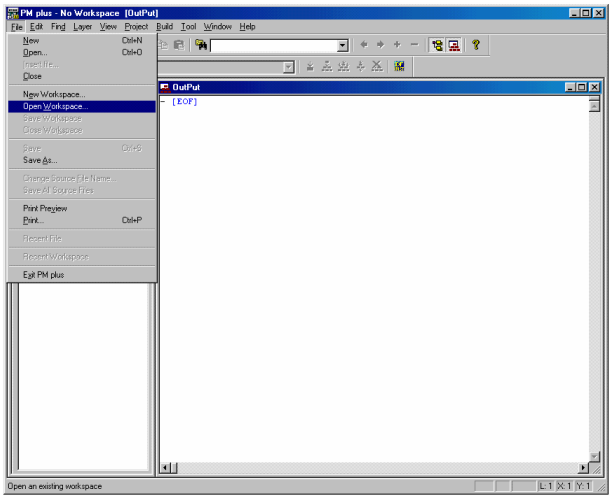


Reading the Workspace File

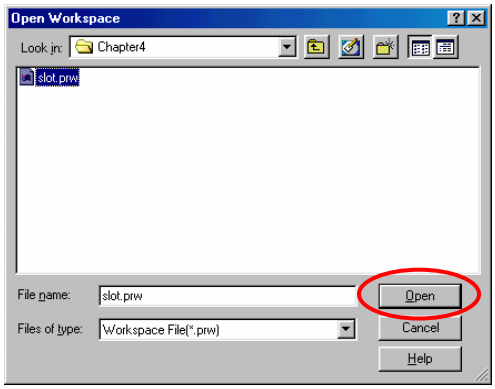
In this chapter, you will use a workspace file that has already been created.

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

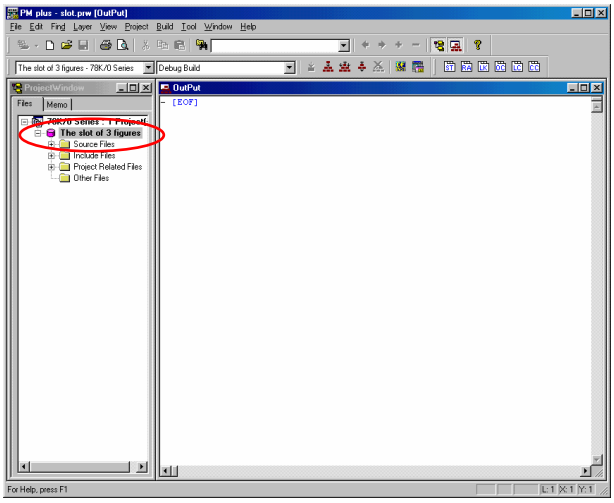
If you have not yet set up the Sample Environment, please refer to Chapter 1 - Tutorial Sample Environment.



Open the Chapter4 directory.



Select slot.prw and click Open.

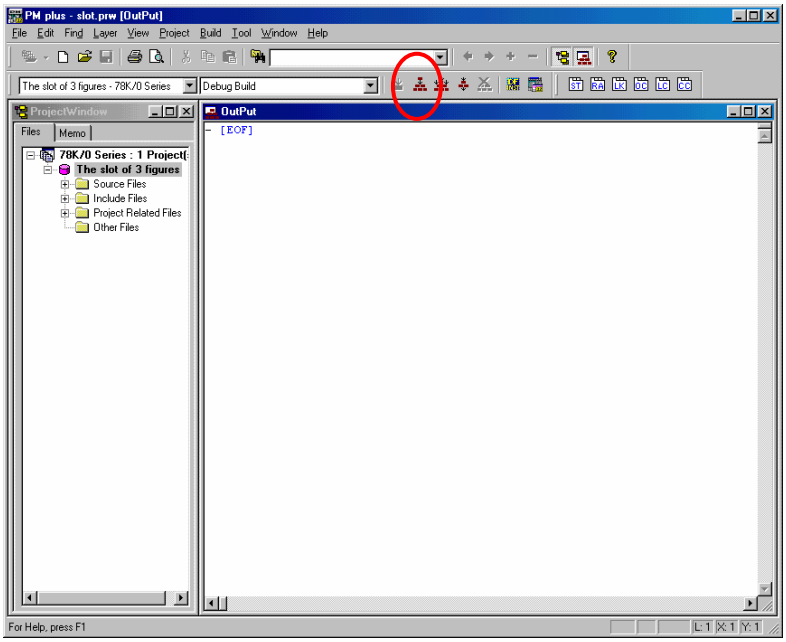


The slot.prw workspace file is read.

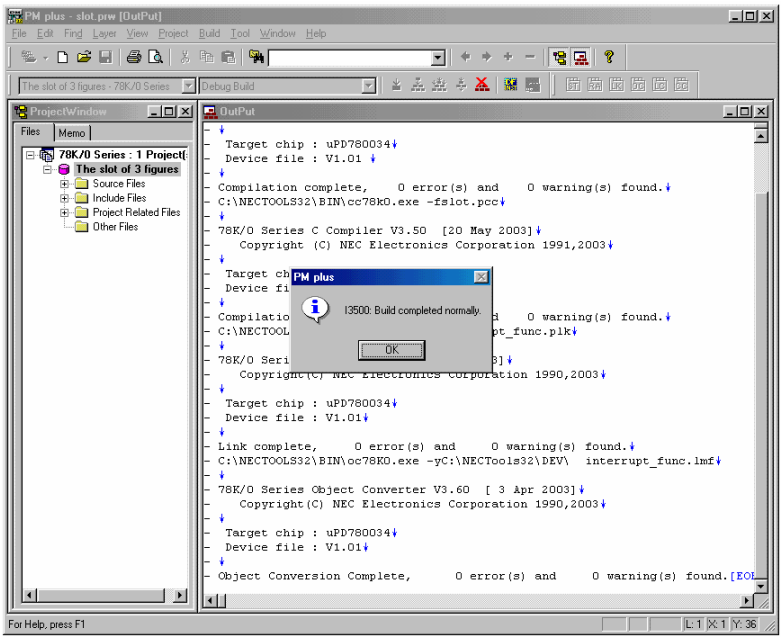
Creating an Executable Program

Next, you will create and executable program.

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



The build process starts.

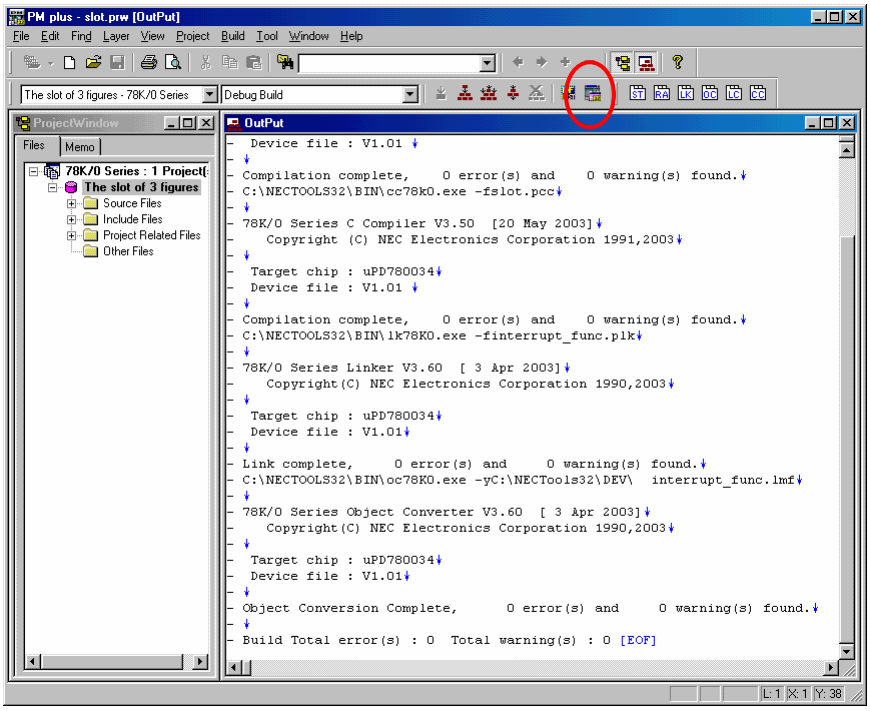


The build completes normally, and an executable program is created.

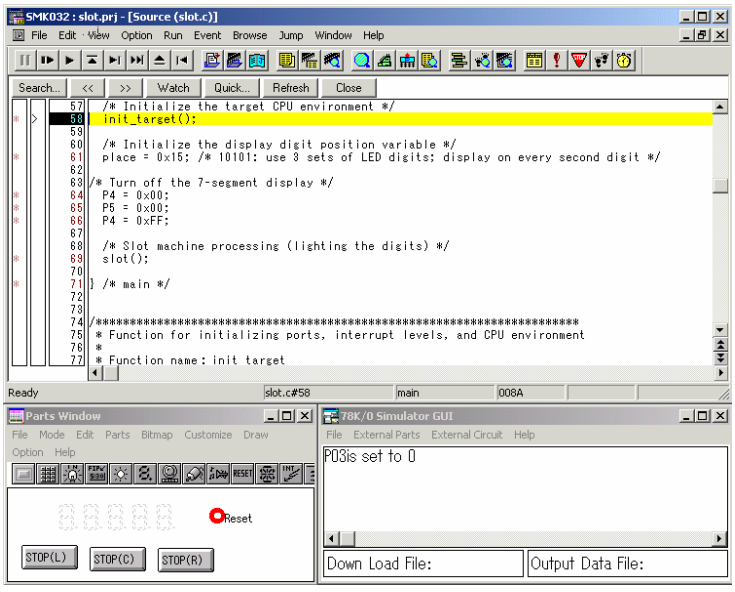
Running the System Simulator (SM78Kxx)

Next, you will run the SM78Kxx.

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



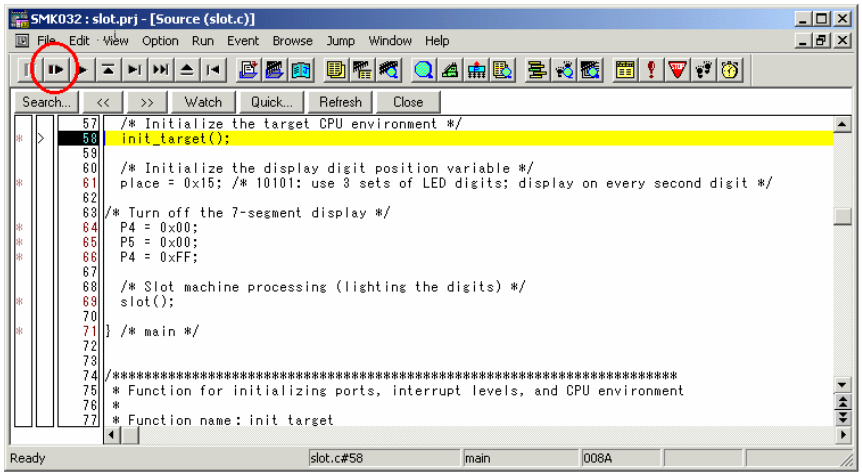
The SM78Kxx starts.



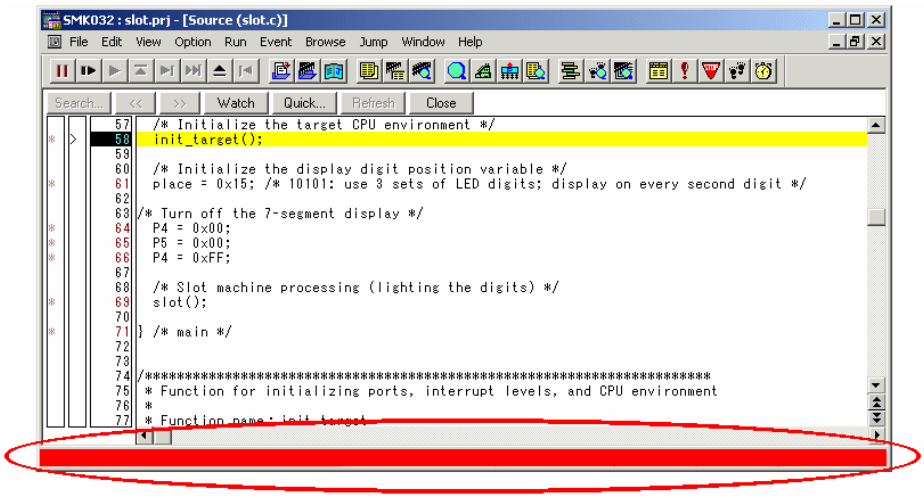
Running the Program

Now, you will run the slot machine program.

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

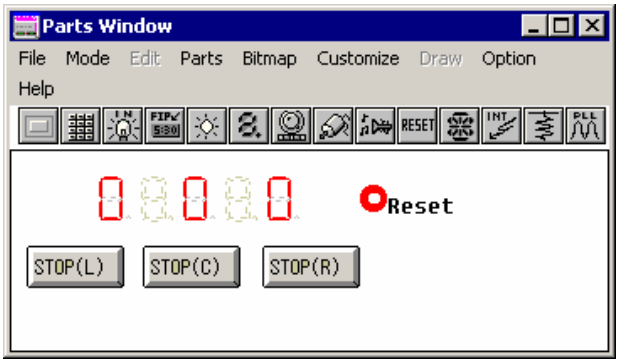


Program execution starts.



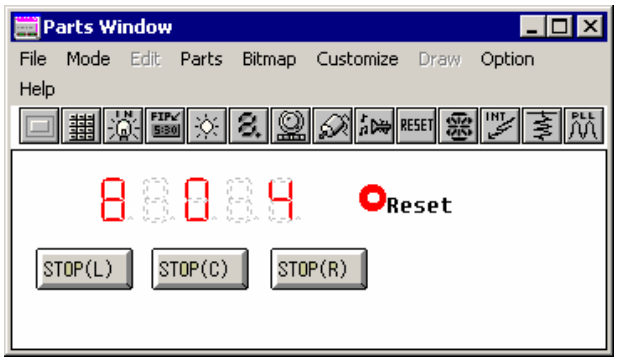
The color of the status bar changes to red during program execution.

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.



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

STOP(C) When the STOP(C) button is clicked, the center LED display digit stops.

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

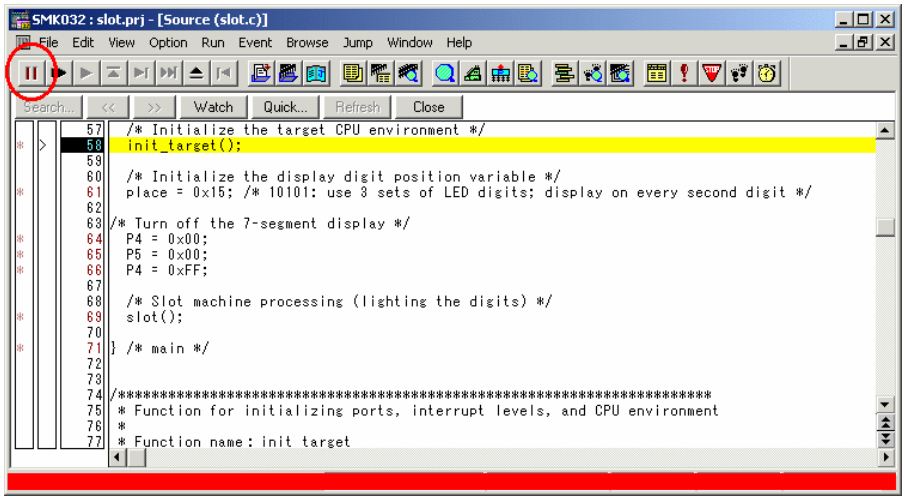
When the Reset button is clicked, the system returns to its initial state, where the digits resume cycling.

This completes the verification of the slot machine program operation.

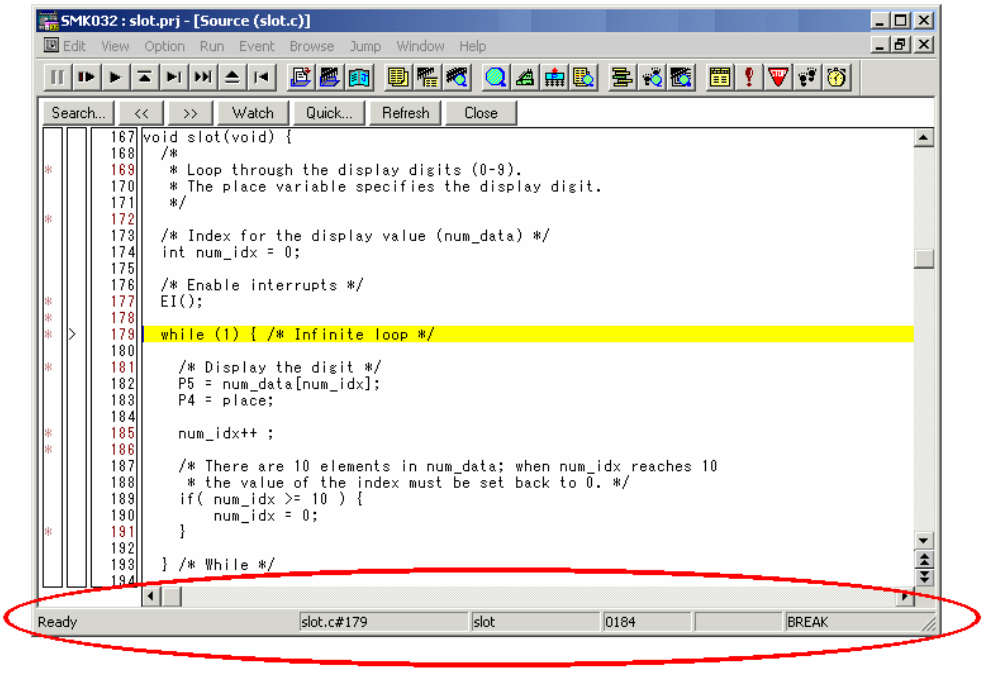
Stopping the Program

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.



Program execution stops.

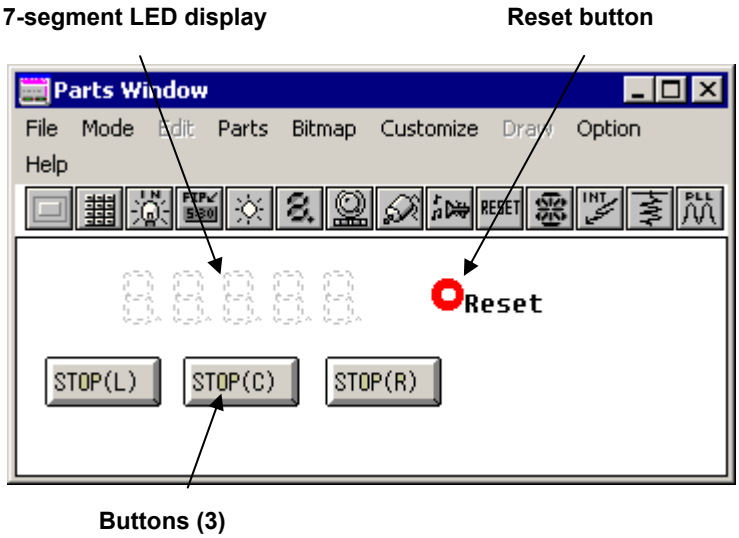


The status bar color returns to its original color when you stop the program.

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.

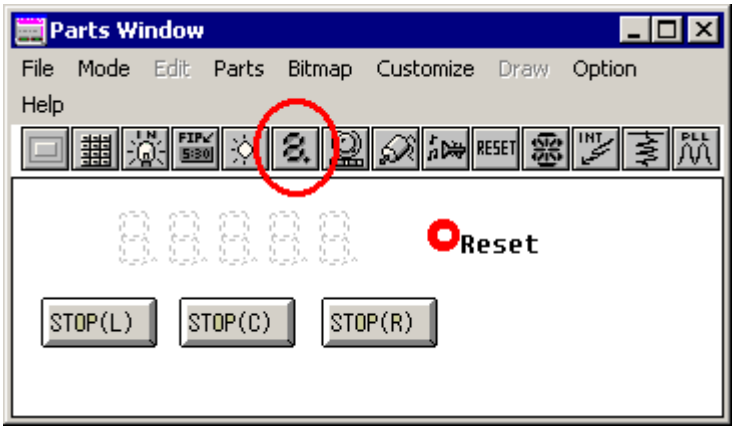
The settings for each device are described below.



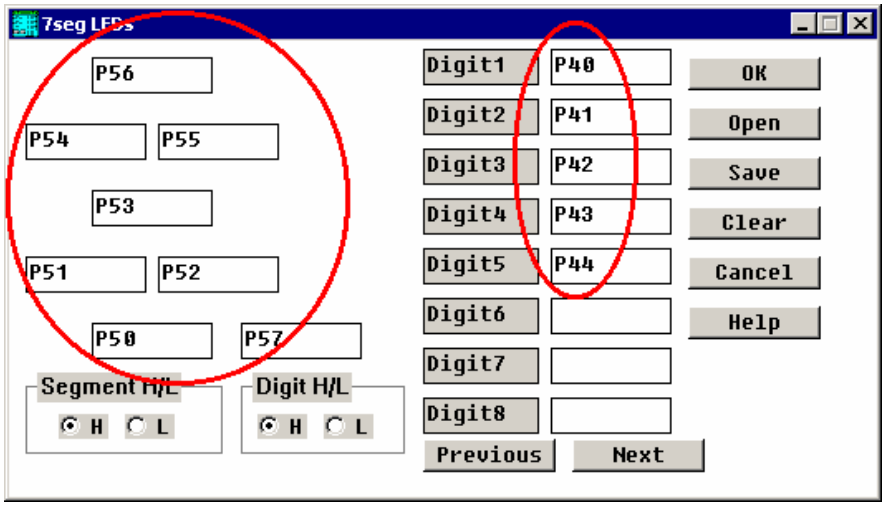
First, the 7-segment LED display terminal settings are described.

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.

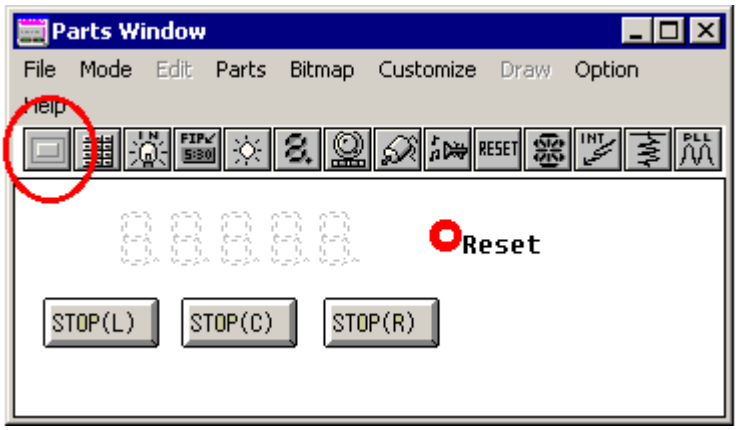
Since the slot machine program uses 5 digits, Digit Signal 1 through Digit Signal 5 are connected.

For details on 7-segment LED display terminal settings, refer to Chapter 3 - System Simulator Basics.

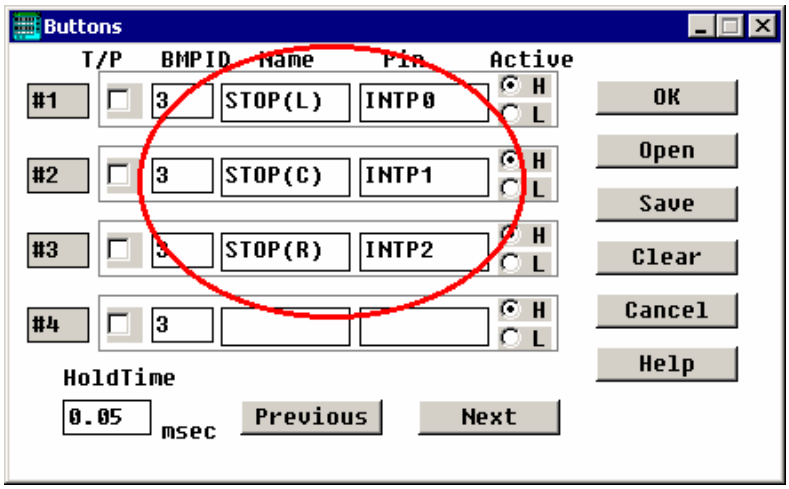
Next, the button settings are described.

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.

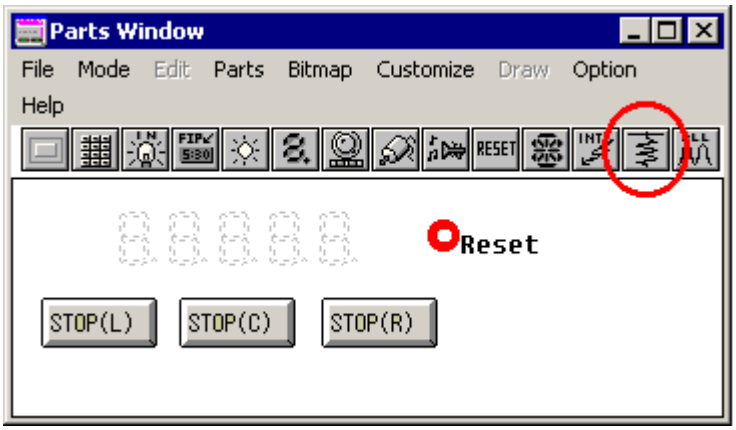


In the Button Terminal Setting dialog, you input the connection information for a button that is displayed in the Input/Output Panel window. A button can be connected to any terminal, and will apply the desired input value to the terminal when the button is clicked. For the program in this chapter, the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

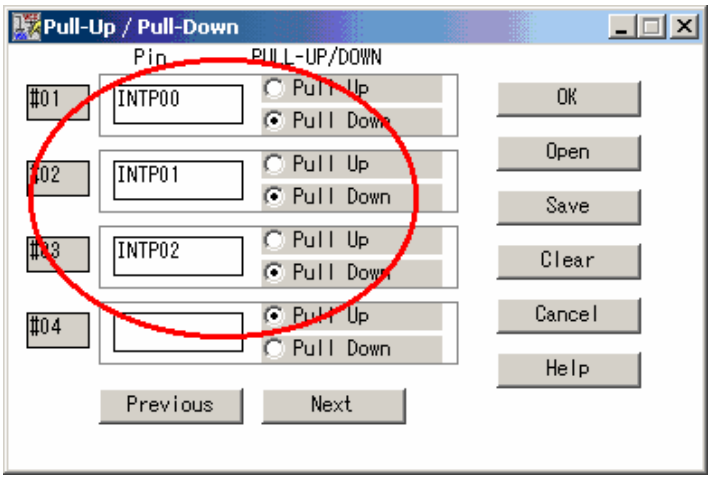
An internal interrupt button can be used to generate an internal interrupt when clicked, but in this case, the buttons are connected to the external interrupt terminals and a high state is detected on the terminal to trigger an interrupt.

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 ?

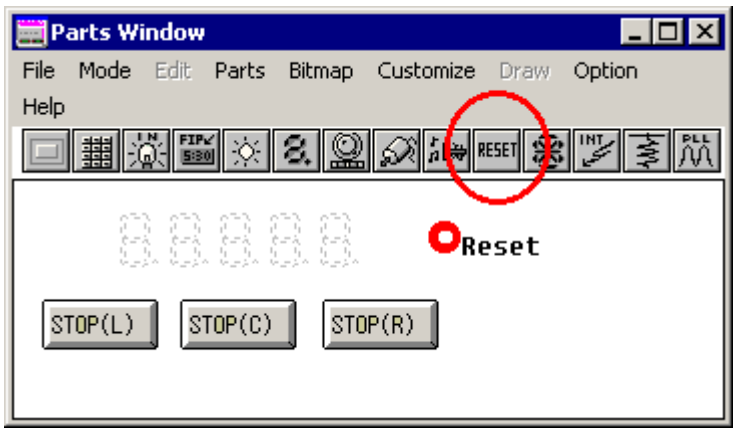
For some external parts of the SM78Kxx, the state is undefined when the part is not operating. A button is one of those parts. For such external parts, the pull-up/pull-down setting must be performed to define the state of the terminal when the button is not operating.
In addition, the pull-up/pull-down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

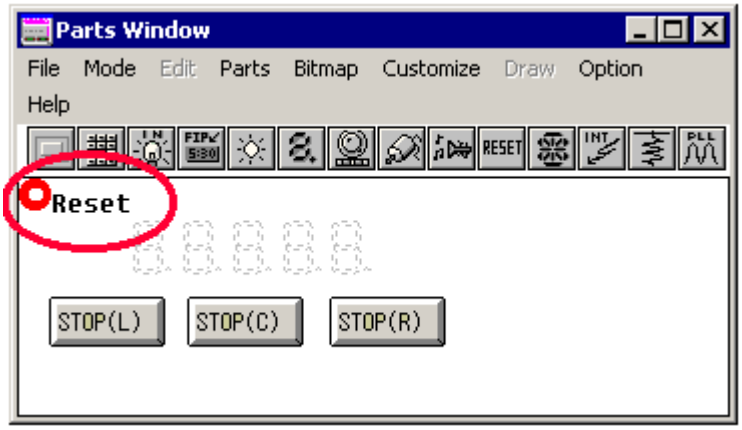
In the Input/Output Panel, click the Reset button 

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.

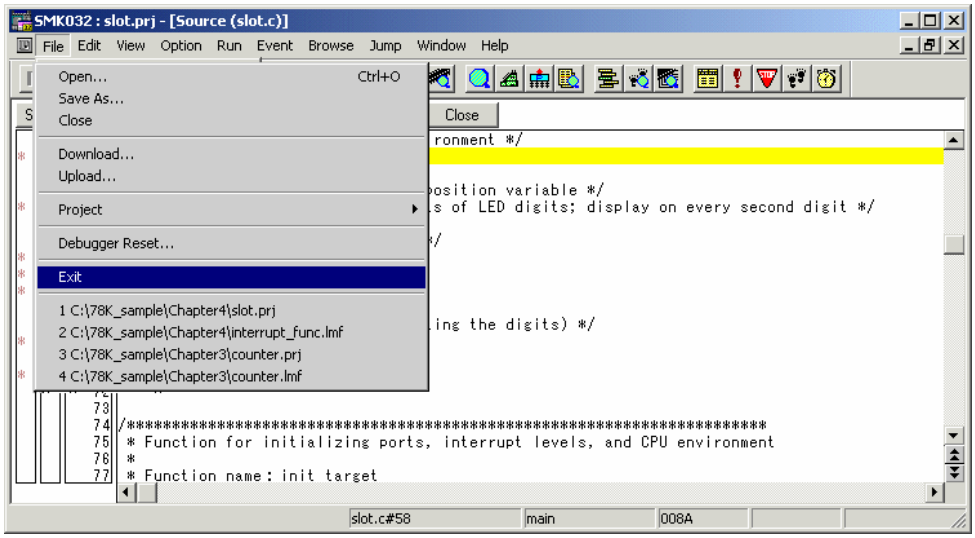
When you click the reset button during program execution, a reset signal is input to the simulator.

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

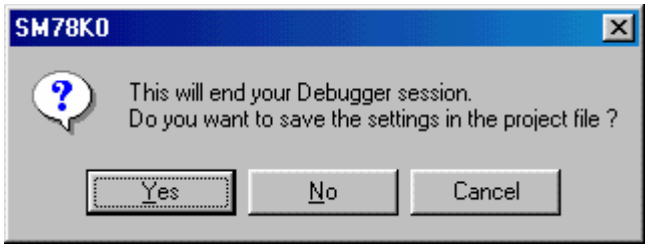
Exiting

Next, you will exit the SM78Kxx.

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



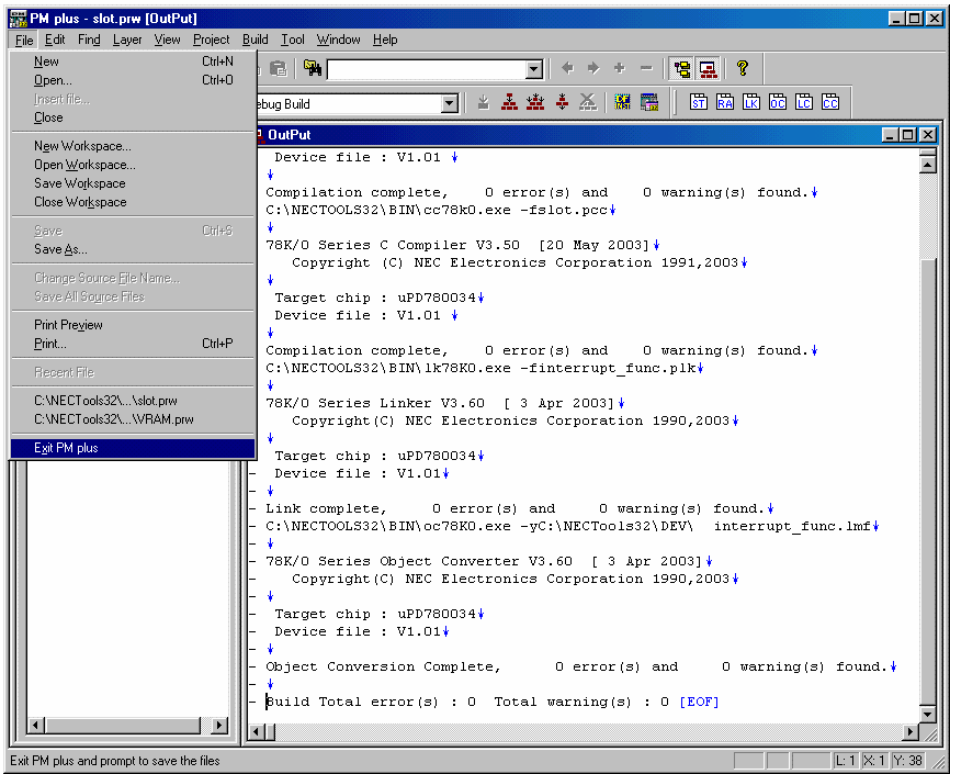
A dialog is displayed asking you if you want to exit.



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

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.

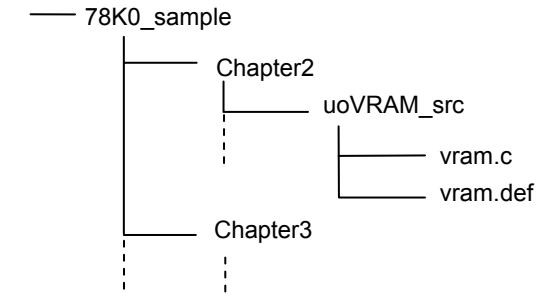
This appendix provides you with a reference on how you can create external parts that use the External Part User Open Interface.
For additional details on the External Part User Open Interface, refer to either the **SM78K Series System Simulator Ver.2.30 or later External Part User Open Interface Specification User's Manual (U15802E)**.

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

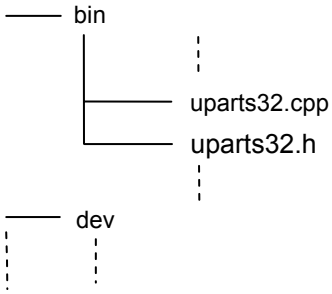
<Files used>

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, upart32.cpp and uparts32.h, which are installed together with the SM78K0.

- Virtual VRAM program source files
  Expanded sample program directories



- Files used by the External Part Open User Interface
  C:\NECTools32 (Directory in which the SM78K0 is installed)

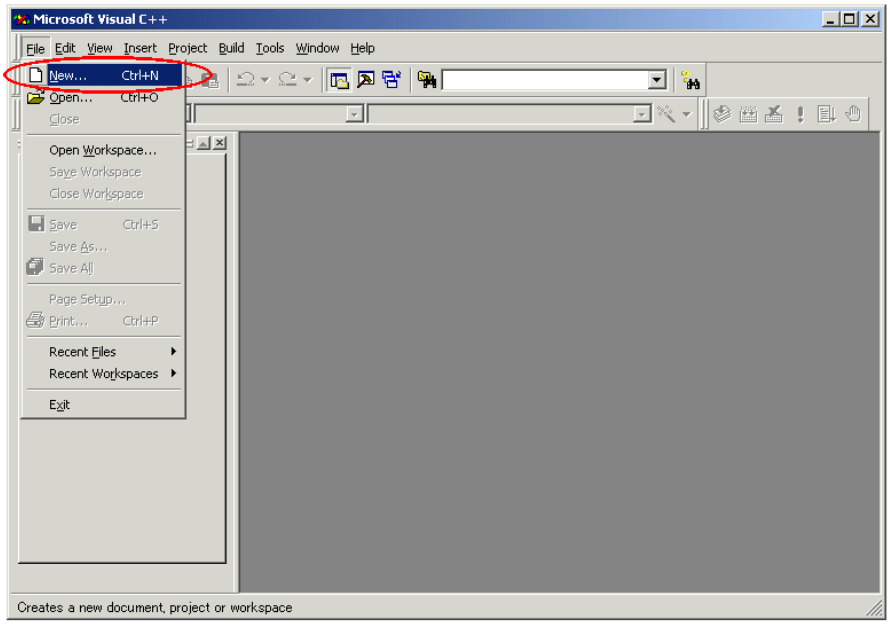


< Procedure for creating uoVRAM.dll >

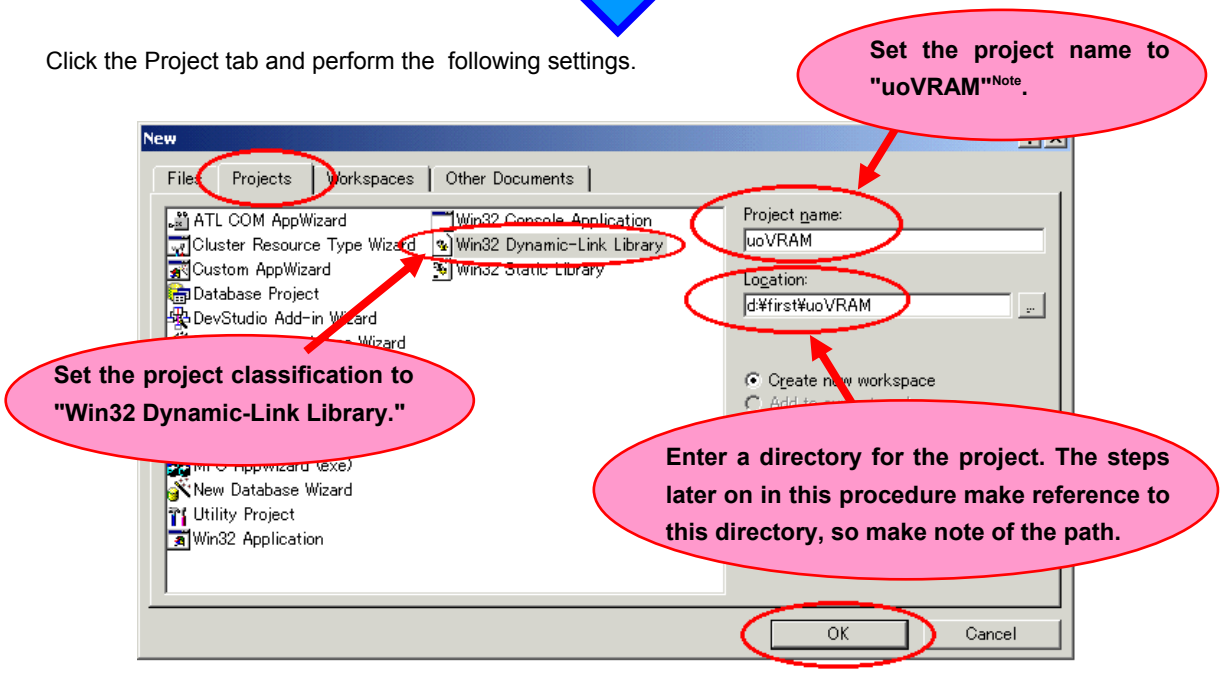
Use the following procedure to build uoVRAM.dll using VC++ Ver.6.

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.

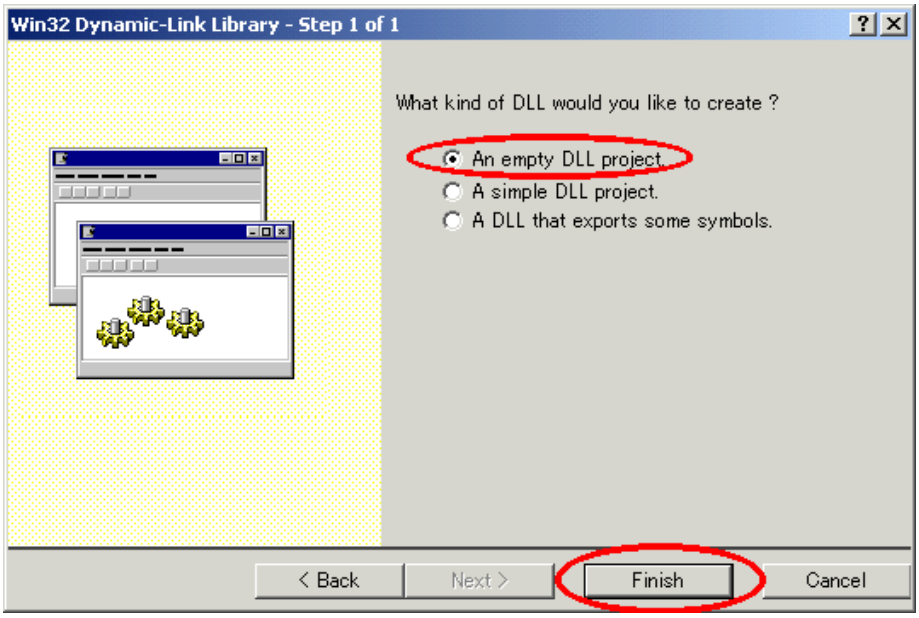


When you have completed the settings, click OK button.

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.

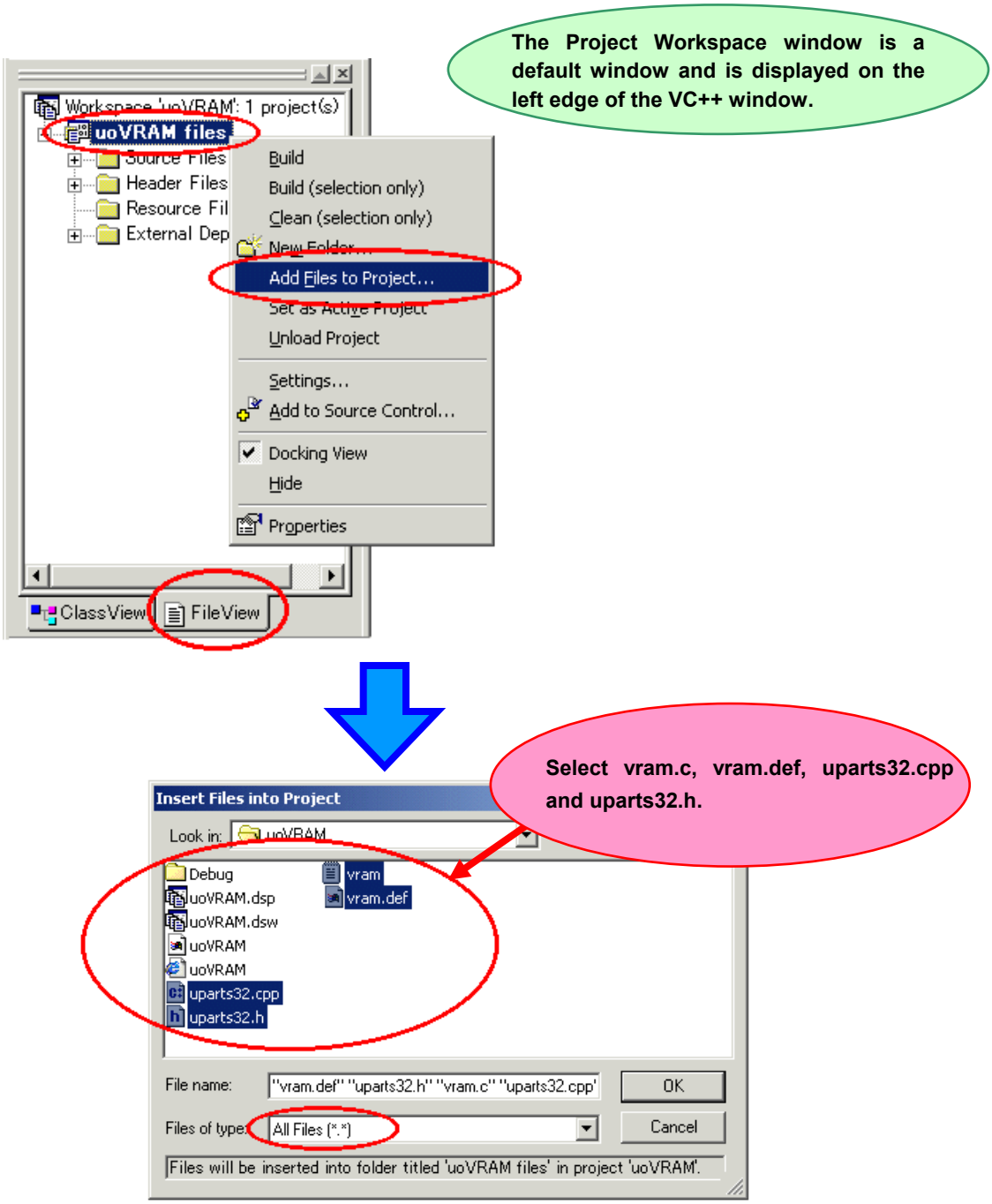
Select "Empty DLL project."



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).

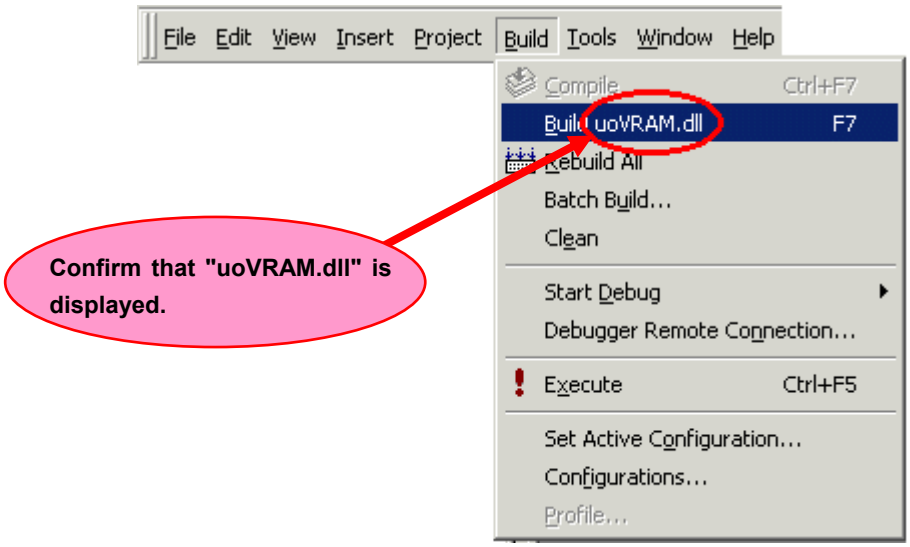
3. Next, you will register the files copied in step 2 (vram.c, vram.def, uparts32.cpp, uparts32.h) with the project you created in step 1. You will perform the operations in the VC++ Project Workspace window inside the File view tab.
   First, click on the File view tab.
   Then, right click on uoVRAM 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.
 *
 *  count1stores the 1s digit and is displayed on the first LED digit.
 *  count10stores the 10s digit and is displayed on the second LED digit.
```

(2/4)

```
 * 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
 ************************************************************
```

(3/4)

```
************************************************************/
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 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()
{
```

(4/4)

```
    /***************************************
     * 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;
}
```

78K0S Source Listing (1/4)

```
/*
*    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
 *
 *    count1stores the 1s digit and is displayed on the first LED digit.
 *    count10stores 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  */
```

(2/4)

```
        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 */
```

(3/4)

```
        /* 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 */
```

(4/4)

```
        /* 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;
}
```

78K4 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 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
 *
 *  count1stores the 1s digit and is displayed on the first LED digit.
 *  count10stores 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
 *
 ************************************************************
```

(2/4)

```
 ************************************************************/
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 */
```

(3/4)

```
        /* 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
 *
 ************************************************************
 ************************************************************/
```

(4/4)

```
__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);
```

(2/4)

```
/************************************************************************
 * 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) */
```

(3/4)

```
/* 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
              *                          INTP0
              *
              *                        INTP1
              *
              *                      INTP2
              *
              *                    INTP3
              */

EGN = 0x00;  /* 0x00 = X X X X 0 0 0 0
              *                          INTP0
              *
              *                        INTP1
              *
              *                      INTP2
              *
              *                    INTP3
              *
              *
              * EGP | EGN  |
              *-------------------------
              *  0  |  0   | Interrupt disabled
              *  0  |  1   | Negative edge
              *  1  |  0   | Positive edge
              *  1  |  1   | Positive and negative edge
              */
```

(4/4)

```
  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 */
```

78K0S 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 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
```

(2/4)

```
 * 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) */
```

(3/4)

```
  /*
   * The active edge for triggering an external input on the external input terminals
   * is set using external interrupt mode register INTM0.
   *
   * In this program, to make the external interrupt requests work with INTP0, INTP1 and
   * INTP2, external interrupt mode register 0 (INTM0) is used to set each interrupt to
   * positive edge triggered.
   */
  INTM0 = 0x54;  /* 0x54 = 010101XX
                  *                    ↑  ↑  ↑
                  *                    |  |  |
                  *                    |  |  INTP0
                  *                    |  |
                  *                    |  INTP1
                  *                    |  |
                  *                    |  |
                  *                    INTP2
                  *
                  * ------------------------
                  *  0 0   | Negative edge
                  *  0 1   | Positive edge
                  *  1 0   | RFU (for future use)
                  *  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.
   */
```

(4/4)

```
  /* 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 */
```

78K4 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 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
```

(2/4)

```
 * 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;
```

(3/4)

```
  /*
   * The active edge for triggering an external input on the external input terminals
   * is set using external interrupt mode registers INTMn (n= 0 - 1).
   *
   * In this program, to make the external interrupt requests work with INTP0,INTP1 and
INTP2,
   * external interrupt mode register 0 (INTM0) is used to set each interrupt to
   * positive edge triggered.
   */
  INTM0 = 0x54;  /* 0x54 = 010101XX
                  *                    ↑  ↑  ↑
                  *                    |  |  |
                  *                    |  |  INTP0
                  *                    |  |
                  *                    |  INTP1
                  *                    |
                  *                    INTP2
                  *
                  * -------------------------
                  *  0 0   | Negative edge
                  *  0 1   | Positive edge
                  *  1 0   | RFU (for future use)
                  *  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 handler
 *    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.
   */
```

(4/4)

```
  /* 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 */
}

*/
```