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

User's Manual

SM plus Ver. 1.00

System Simulator

User Open Interface

Target Device
V850 Series

Document No. U16907EJ1V0UM00 (1st edition)
Date Published August 2004 CP(K)

Printed in Japan

[MEMO]

Windows and WindowsNT are either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

- **The information in this document is current as of August, 2004. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".

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

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

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

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

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

(Note)

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

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

M8E 02. 11-1

Regional Information

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

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

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

[GLOBAL SUPPORT]
http://www.necel.com/en/support/support.html

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

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65030

- **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

[MEMO]

INTRODUCTION

Target Readers

The contents described in this manual use the Windows™ 98/WindowsMe/Windows NT™ 4.0/Windows2000/WindowsXP 32-bit application program format and this manual is therefore intended for users who have experience creating Windows 98/WindowsMe/Windows NT 4.0/Windows2000/WindowsXP 32-bit application programs.

Purpose

This manual explains the simulation environmental construction means of the target system using the user open interface among the simulation environmental construction means which the system simulator SM plus is preparing.

Organization

This manual is broadly divided into the following sections.

- OVERVIEW
- CREATING USER MODEL
- EMBEDDING USER MODEL
- FUNCTION REFERENCE
- SAMPLE PROGRAM

How to Use This Manual

It is assumed that readers of this manual have general knowledge of microcomputers and the C programming language. Readers will need to have a basic knowledge of how to create Windows 98/WindowsMe/Windows NT 4.0/Windows2000/WindowsXP 32-bit application programs.

For information on the system as the user open interface (supplied functions) and the functions created by the user (user-defined functions).:

→ Read **CHAPTER 4 FUNCTION REFERENCE**.

For information on the user open interface of SM plus:

→ Read this manual in the order of the **CONTENTS**.

Related Documents

Refer to the documents listed below when using this manual.
The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Documents related to development tools (User's Manuals) (1/2)

Document Name	Document No.
IE-703002-MC (In-circuit emulator for V853, V850/SA1, V850/SB1, V850/SB2, V850/SC1, V850/SC2, V850/SC3, V850/SF1, V850/SV1)	U11595E
IE-V850E-MC (In-circuit emulator for V850E/IA1, V850E/IA2), IE-V850E-MC-A (In-circuit emulator for V850E/MA1, V850E/MA2, V850E/SV2)	U14487E
IE-V850E1-CD-NW (PCMCIA card type on-chip debug emulator)	U16647E
IE-703003-MC-EM1 (In-circuit emulator option board for V853)	U11596E
IE-703017-MC-EM1 (In-circuit emulator option board for V850/SA1)	U12898E
IE-703037-MC-EM1 (In-circuit emulator option board for V850/SB1, V850/SB2)	U14151E
IE-703040-MC-EM1 (In-circuit emulator option board for V850/SV1)	U14337E
IE-703079-MC-EM1 (In-circuit emulator option board for V850/SF1)	U15447E

Document Name		Document No.
IE-703089-MC-EM1 (In-circuit emulator option board for V850/SC1, V850/SC2, V850/SC3)		U15776E
IE-703102-MC (In-circuit emulator for V850E/MS1)		U13875E
IE-703102-MC-EM1, IE-703102-MC-EM1-A (In-circuit emulator option board for V850E/MS1)		U13876E
IE-703107-MC-EM1 (In-circuit emulator option board for V850E/MA1)		U14481E
IE-703114-MC-EM1 (In-circuit emulator option board for V850E/IA2)		U16533E
IE-703116-MC-EM1 (In-circuit emulator option board for V850E/IA1)		U14700E
IE-V850ES-G1 (In-circuit emulator for V850ES)		U16313E
IE-703204-G1-EM1 (In-circuit emulator option board for V850ES/SA2, V850ES/SA3)		U16622E
IE-703217-G1-EM1 (In-circuit emulator option board for V850ES/KF1, V850ES/KG1, V850ES/KJ1)		U16594E
IE-703228-G1-EM1 (In-circuit emulator option board for V850ES/PM1)		U16879E
CA850 Ver. 2.70 C Compiler Package	Operation	U16932E
	C Language	U16930E
	Assembly Language	U16931E
	Link Directives	U16933E
PM plus Ver. 5.20		U16934E
ID850 Ver. 2.50 Integrated Debugger	Operation	U16217E
ID850NW Ver. 2.51 Integrated Debugger	Operation	U16454E
ID850NWC Ver. 2.51 Integrated Debugger	Operation	U16525E
ID850QB Ver. 2.80 Integrated Debugger	Operation	U16973E
SM plus Ver. 1.00 System Simulator	Operation	U16906E
SM plus Ver. 1.00 System Simulator	User Open Interface	This manual
SM850 Ver. 2.50 System Simulator	Operation	U16218E
SM850 Ver. 2.00 or Later System Simulator	External Part User Open Interface Specifications	U14873E
RX850 Ver. 3.13 or Later Real-Time OS	Basics	U13430E
	Installation	U13410E
	Technical	U13431E
RX850 Pro Ver. 3.15 Real-Time OS	Fundamental	U13773E
	Installation	U13774E
	Technical	U13772E
RD850 Ver. 3.01 Task Debugger		U13737E
RD850 Pro Ver. 3.01 Task Debugger		U13916E
AZ850 Ver. 3.10 System Performance Analyzer		U14410E
PG-FP4 Flash Memory Programmer		U15260E
TW850 Ver. 1.20 Performance Analysis Tuning Tool		U17037E

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 OVERVIEW

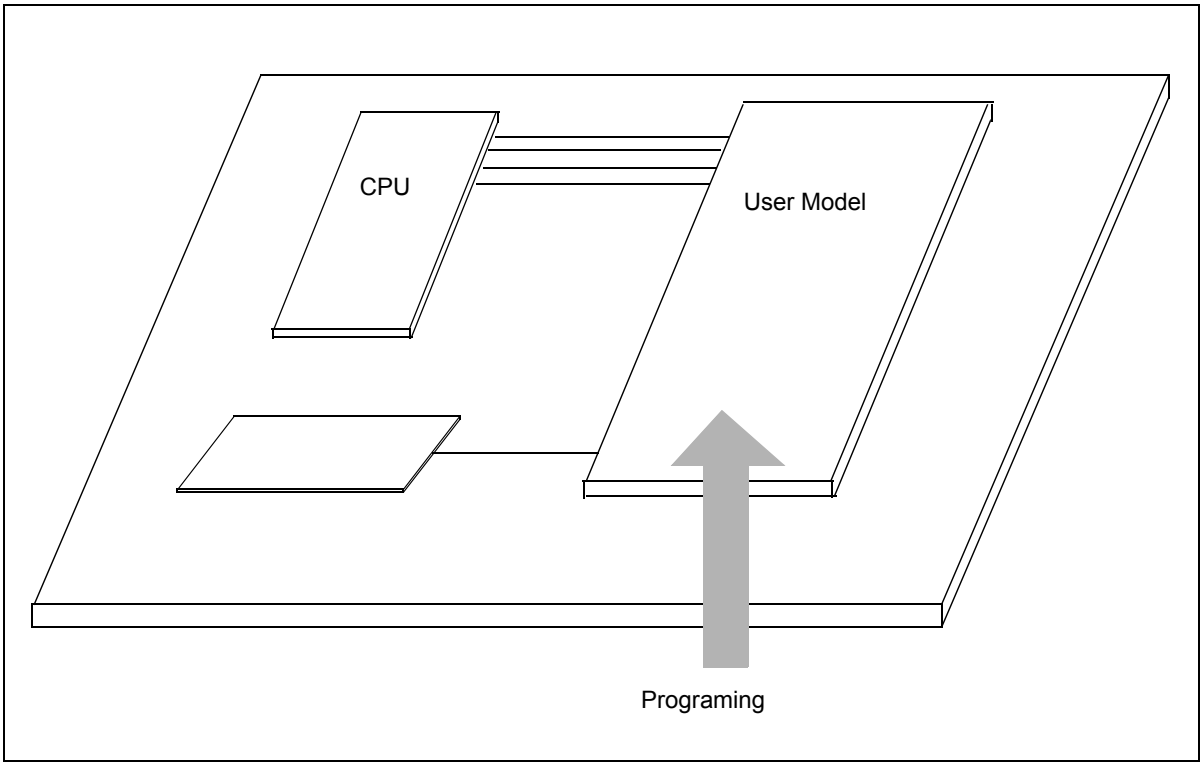
SM plus provides two ways of creating an environment where a target system, as well as a CPU (CPU core + internal peripherals), can be simulated.

One is the Parts window[Note], via which a user-friendly simulation environment can be organized through GUI manipulation, by supplying standard components for connection and their manipulation environment.

The other is to create the simulation environment of the target system that uses the user open interface to be explained in this manual. In this environment, functions that cannot be realized on the Parts window can be used if the user programs an external user model.

Note Refer to the SM plus Operation User's Manual (to be prepared).

Figure 1-1 Programming Image of User Model



1. 1 Types of Interface Functions

SM plus user open interface supplies the following types of interface functions (for details, refer to "CHAPTER 4 FUNCTION REFERENCE").

Table 1-1 Types of Functions Supplied by User Open Interface

Type	Description and Major Functions
Basic interface function	Basic function of simulation - Initialization notification - Reset notification, etc.
Time interface function	Cyclic timer function for time-series processing of model - Setting of timer - Clearing of timer - Notification of timer time, etc.
Pin interface function	Pin I/O function - Signal output to pin - Notification of signal input to pin
External bus interface function	Slave function of external bus - External bus read access notification - External bus write access notification, etc.
Serial interface function	Serial transmission/reception function - Transmission of serial data - Notification of reception of serial data, etc.
Signal output unit interface function	Function to output signals in accordance with signal data file - Signal output in accordance with signal data file, etc.

1. 2 Interface Methods

SM plus user open interface has the following interface methods.

1. 2. 1 C-language interface

The SM plus user open interface consists of a C-language API function[Note] set.

Therefore, program the user model in C language to create the environment in which to simulate the target system.

Note Application Program Interface

1. 2. 2 Callback function method

The callback function method is used to report a pointer to a function created on the user program side to the system in advance so that the system calls the function created on the user program side by using the pointer to that function.

The SM plus user open interface uses this callback function method as a means to call a user program from the system.

While the provided API functions call the system from the user program, the callback function method is used to call the user program from the system, such as when inputting a signal to a pin.

1. 2. 3 Event-driven method

The SM plus user open interface uses an event-driven method in which processing is described in accordance with an event.

For example, a callback function prepared on the user model side is called if an event such as initialization of simulation, resetting the CPU, signal output to a pin, or access to the external bus occurs on the SM plus main body side. In addition, a time interface (= timer function) provided to perform time-series processing of a user model also calls a callback function prepared on the user model side when the specified time has elapsed.

1. 3 Development Environment

Use the following development tools to perform programming with the SM plus user open interface and create a DLL file.

- Microsoft Visual C++ V6.00 or later

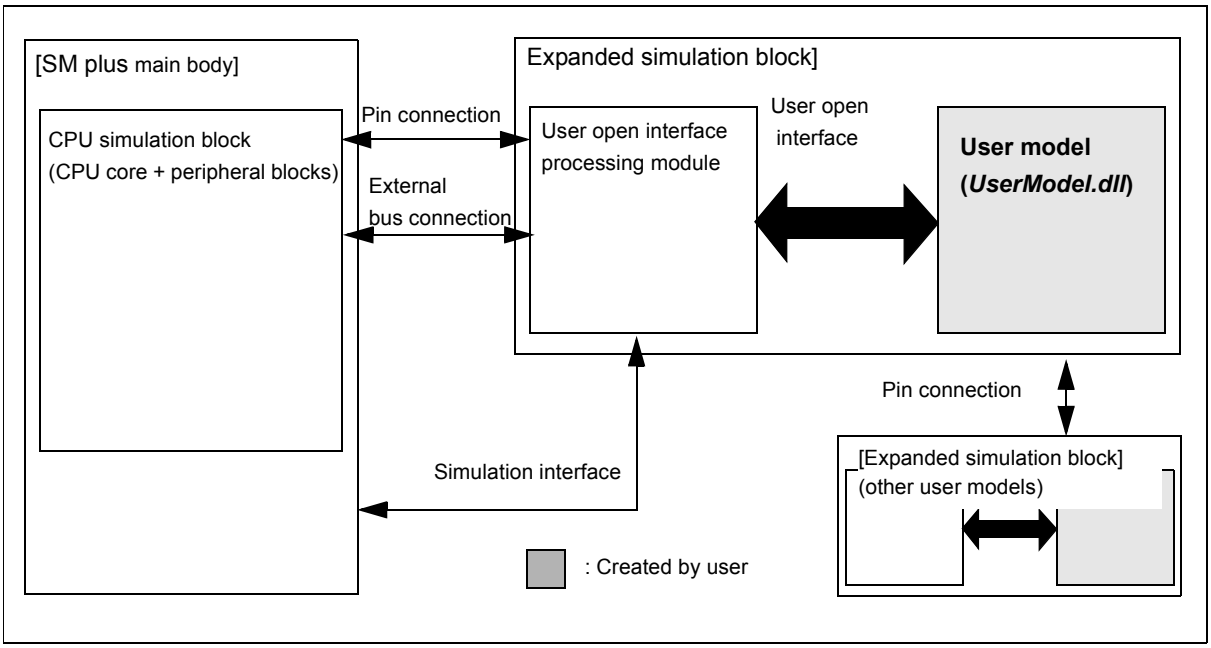
CHAPTER 2 CREATING USER MODEL

This chapter explains how to create a user model.

2. 1 Program Configuration

The following figure shows the program configuration when the SM plus user open interface is used to expand a system.

Figure 2-1 Program Configuration



To expand a system, a user model must be created first.

Because the user model operates in conjunction with the simulation system, it interfaces with the user open interface processing module. This interface is the user open interface.

The user model generates resources such as pins and external bus slaves via the user open interface during configuration (processing to configure the simulator that is performed when the SM plus is started). By connecting the pins and external bus slaves to the pins and external bus masters of the CPU simulation block, signals can be input to or output from the pins of the CPU simulation block and the external bus can be accessed from the CPU simulation block.

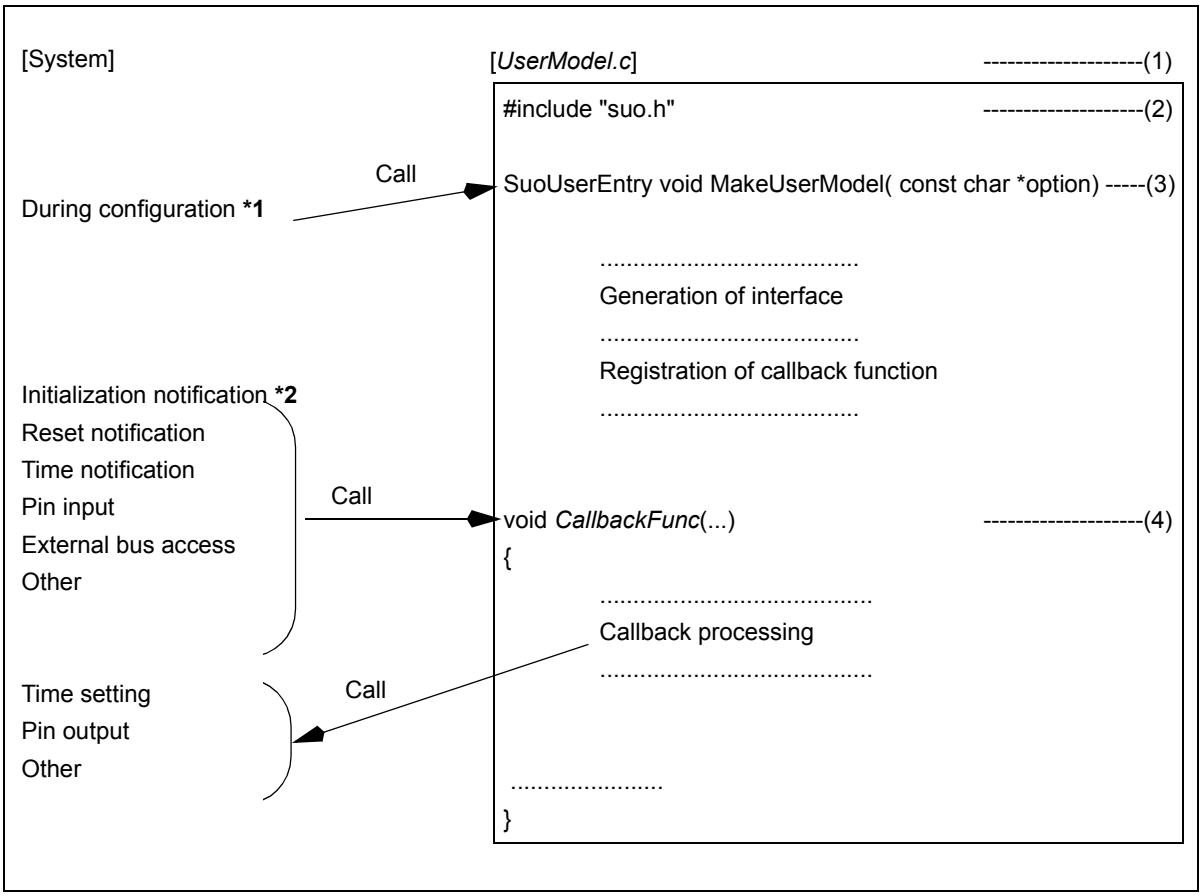
The generated pins and external bus slaves can also be connected to the expansion simulation block (other user models), as well as to the CPU simulation block.

2. 2 Outline of Programming

The user model is programmed in the dynamic link library (DLL) format of WIN32.

The template of a program file is shown below.

Figure 2-2 Template of Program File



***1** "Configuration" means simulator configuration processing that is executed when SM plus is started.

***2** An initialization notification is reported only once, immediately after SM plus is started when simulator configuration processing has been completed.

2. 3 Programming Details

Items (1) to (4) in Figure 2-2 are explained in detail below.

2. 3. 1 File name

(1) is the filename.

The suffix for a C-language file is "*.c". The file name can be determined freely.

2. 3. 2 Include file

(2) is an include file.

To use the user open interface, the header file "suo.h" must be included.

2. 3. 3 MakeUserModel function

(3) is the MakeUserModel function that is called from the system during configuration of SM plus.

The name of this function must be "MakeUserModel". (Refer to "4. 2 User-Defined Functions".)

Specification format

```
SuoUserEntry void MakeUserModel( const char *option);
```

The following two types of processing are described in this function.

(1) Interface generation

Because SM plus connects pins and buses during configuration processing when it is started, resources such as pins and buses that are to be connected during configuration must be generated.

To do this, call a function that generates an interface in the MakeUserModel function and generate an interface. The necessary resources will be also generated.

(2) Registering callback function

Callback functions can be registered as necessary.

Caution When describing a callback function for initialization, be sure to register it at this time; otherwise callback will not function. This is because initialization notification is reported immediately after the MakeUserModel function is called.

2. 3. 4 Callback function

(4) is a callback function.

A function that is called from the system is called a callback function.

Two or more callback functions, such as those for initialization notification, reset notification, time notification, pin input, and external bus access, can be created. (Refer to "4. 2 User-Defined Functions".)

A callback function that has been created must be registered in advance so that it can be called from the system.

The name of a callback function can be determined freely, and the format of the function differs depending on the type of callback. Describe processing in accordance with the callback contents in the callback function.

2. 4 Example of Program File

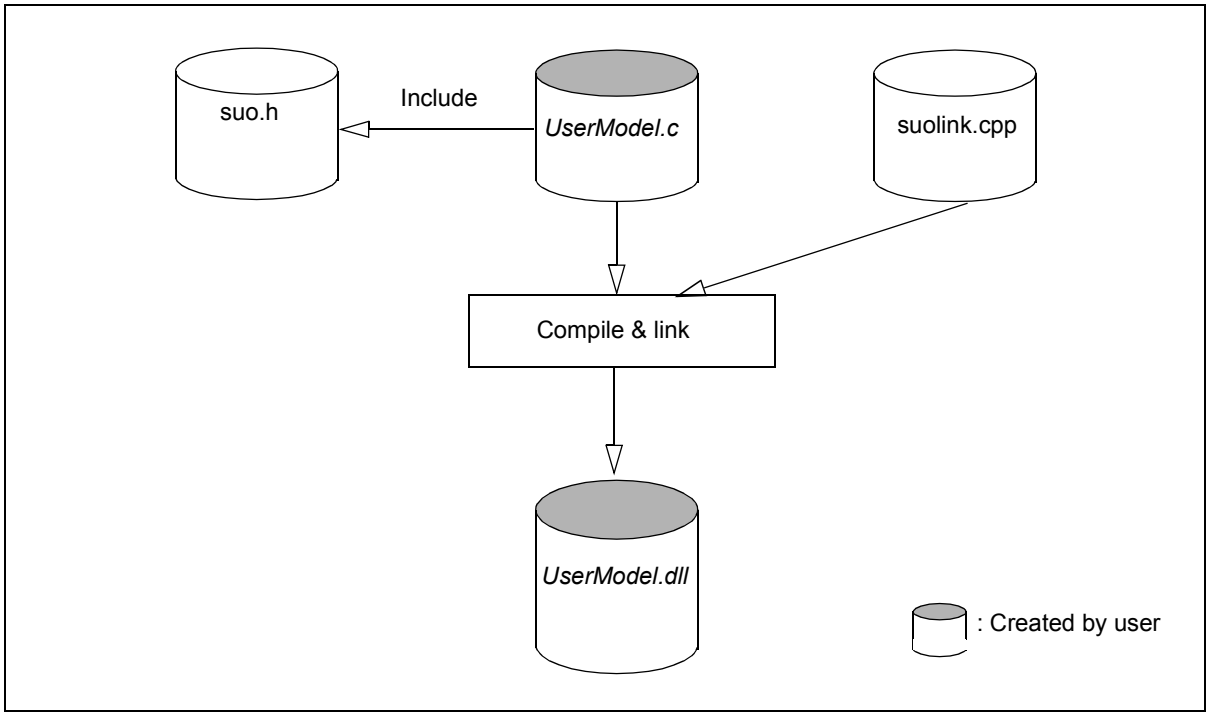
Figure 2-3 Example of Program File

```
#include "suo.h"
#include <memory.h>

void Init(void);
void InputP00(SuoHandle handle, int pinValue);
void ReadBUS1(SuoHandle handle, unsigned long addr, int accessSize, unsigned char data[]);
void WriteBUS1(SuoHandle handle, unsigned long addr, int accessSize, const unsigned char data[]);

SuoHandle p00;
SuoHandle p01;
SuoHandle bus1;
unsigned char mem[0x100];

/* MakeUserModel */
SuoUserEntry void MakeUserModel( const char *option)
{
        SuoCreatePin("P00", &p00);
        SuoCreatePin("P01", &p01);
        SuoCreateExtbus("BUS1", 0x200000, 0x100, &bus1);

        SuoSetInitCallback(Init);
        SuoSetInputDigitalPinCallback(p00, InputP00);
        SuoSetReadExtbusCallback(bus1, ReadBUS1);
        SuoSetWriteExtbusCallback(bus1, WriteBUS1);
}

/* callbacks */
void Init(void)
{
        memset(mem, 0, 0x100);
}

void InputP00(SuoHandle handle, int pinValue)
{
        SuoOutputDigitalPin(p01, pinValue);
}

void ReadBUS1(SuoHandle handle, unsigned long addr, int accessSize, unsigned char data[])
{
        memcpy(data, &mem[addr-0x200000], accessSize);
}

void WriteBUS1(SuoHandle handle, unsigned long addr, int accessSize, const unsigned char data[])
{
        memcpy(&mem[addr-0x200000], data, accessSize);
}
```

2. 5 Compilation and Linking

The following figure shows the flow of compilation and linking.

Figure 2-4 Flow of Compilation and Linking



By compiling and linking the above files, *UserModel.dll* is created from *UserModel.c* and suolink.cpp.

(1) suo.h

suo.h is a system header file for the user open interface. This file is included by the user program (*UserModel.c*) but is not compiled.

(2) suolink.cpp

suolink.cpp is a file that performs dynamic link processing with the user open interface processing module of the system.

(3) *UserModel.c*

UserModel.c is the source file of the user model to be created. The file name can be determined freely.

(4) *UserModel.dll*

UserModel.dll is a binary file of the user model (DLL file). The file name can be determined freely.

Caution To execute a DLL file in an environment in which Microsoft Visual C++ is not installed, the DLL file must be created using the release version.

CHAPTER 3 EMBEDDING USER MODEL

This chapter explains how to embed the created user model (*UserModel.dll*) in SM plus, which is done using the configuration file.

3. 1 Configuration File

The configuration file is used by a user to customize (add a user model to) SM plus.

The configuration file name is C:\NECTools32\bin\smplus.cfg[Note].

Note If the installation folder of SM plus is C:\NECTools32

3. 2 Description in Configuration File

Describe the user model generation processing and processing to connect pins and an external buses in the configuration file.

3. 2. 1 Creation

UserModel1 = Device("USEROPEN", "*UserModel1.dll UserOption1*");

The Device function is used to create a user model.

"USEROPEN" is a user open interface processing module (system module).

UserModel1.dll is the binary file (DLL format) of the user model to be created. The name can be determined freely by the user.

Specify a relative path or absolute path from the folder where the configuration file exists as the file path.

UserOption1 is an option character string for *UserModel1.dll*. This character string is passed to the "option" parameter of the MakeUserModel function as is.

UserModel1 is a variable that indicates the generated user model. The name can be determined freely by the user.

3. 2. 2 Pin connection

wire1 = Wire(1);	--- (1)
wire1 += cpu.Port("*PinName1*");	--- (2)
wire1 += *UserModel1*.Port("*UserPinName1*");	--- (3)

Pins are connected in the following sequence.

(1) Generate a wire (= line that connects pins) by using the Wire function.

Be sure to specify 1 for the argument of the Wire function.

wire1 is a variable that indicates the generated wire. The name can be determined freely.

(2) Connect one end of the wire to a pin of the CPU.

Specify the external CPU pin to be connected as "*PinName1*". Enclose the pin name between double quotation marks (" ").

(3) Connect the other end of the wire to a pin of the user model.

UserModel1 is a variable that indicates the generated user model.

Specify the name of the user model pin to be connected as "*UserPinName1*" (pin name generated in the MakeUserModel function). Enclose the pin name between double quotation marks (" ").

To connect two or more user model pins to the same wire, add this line.

3. 2. 3 External bus connection

extbus1 = BUS(*n*);	--- (1)
extbus1 += cpu.BusMasterIF("EXTBUS");	--- (2)
extbus1 += *UserModel1*.BusSlaveIF("*UserExtbusName1*");	--- (3)

An external bus is connected in the following sequence.

(1) Generate a bus by using the BUS function.

Argument *n* of the bus function is the data bus bit width. This may be 8, 16, or 32.

extbus1 is a variable that indicates the generated bus. The name can be determined freely.

(2) Connect one end of the bus to the external bus master of the CPU.

Specify the external bus master "EXTBUS".

(3) Connect the other end of the bus to the external bus of the user model.

UserModel1 is a variable that indicates the generated user model.

Specify the name of the external bus of the user model to be connected as "*UserExtbusName1*" (the external bus name generated in the MakeUserModel function). Enclose the external bus name between double quotation marks (" ").

Add this line to connect two or more user model external buses.

3. 2. 4 Other items

In addition to the above, connection in a specific format is necessary for operating the user open interface.

For the specific connection method, refer to the sample program supplied with the product.

3. 3 Example of Configuration File Description

Figure 3-1 shows an example of the configuration file description.

In this example, the following connection processing is performed.

Table 3-1 Connection in Sample Configuration File

Type of Connection	CPU		User Model (SampleModel.DLL)	
Pin	"P00/INTP0"	(P00 pin)	"P00"	(Pin manipulating P00)
	"P30/TXD1"	(Serial output pin)	"RXD"	(Serial input pin)
	"P31/RXD1"	(Serial input pin)	"TXD"	(Serial output pin)
External bus	"EXTBUS"	(External bus master)	"EXTBUS1"	(External bus slave 1)
	"EXTBUS"	(External bus master)	"EXTBUS2"	(External bus slave 2)

Figure 3-1 Example of Description in Configuration File

```
cpu = CPU('a');
# ----------------------------------------------------------------
#   SampleModel description
# ----------------------------------------------------------------

# Generate SampleModel.dll
model = Device("USEROPEN", "SampleModel.dll -a -b");

# Connect PIN (CPU.P00-MODEL.P00)
wire_P00 = Wire(1);
wire_P00 += cpu.Port("P00/INTP0");
wire_P00 += model.Port("P00");

# Connect PIN (CPU.TXD1-MODEL.RXD)
wire_RXD = Wire(1);
wire_RXD += cpu.Port("P30/TXD1");
wire_RXD += model.Port("RXD");

# Connect PIN (CPU.RXD1-MODEL.TXD)
wire_TXD = Wire(1);
wire_TXD += cpu.Port("P31/RXD1");
wire_TXD += model.Port("TXD");

# Connect BUS (CPU.EXTBUS-MODEL.EXTBUS1)
extbus = BUS(32);
extbus += cpu.BusMasterIF("EXTBUS");
extbus += model.BusSlaveIF("EXTBUS1");
extbus += model.BusSlaveIF("EXTBUS2");
```

CHAPTER 4 FUNCTION REFERENCE

This chapter shows how to reference the functions supplied by the system as the user open interface (supplied functions) and the functions created by the user (user-defined functions).

4. 1 List of Supplied Functions

The supplied functions are listed below.

Table 4-1 Supplied Functions

Function Name	Description	Remark
Basic interface functions		
SuoSetInitCallback	Registers initialization callback	Note 2
SuoSetResetCallback	Registers reset callback	-
Time interface functions		
SuoCreateTimer	Generates timer	Note 1
SuoGetTimerHandle	Acquires timer handle	-
SuoSetTimer	Sets timer time	Note 3
SuoKillTimer	Cancels timer time	Note 3
SuoSetNotifyTimerCallback	Registers timer time notification callback	-
Pin interface function		
SuoCreatePin	Generates pin	Note 1
SuoGetPinHandle	Acquires pin handle	-
SuoOutputDigitalPin	Outputs digital pin value	Note 3
SuoOutputAnalogPin	Outputs analog pin value	Note 3
SuoSetInputDigitalPinCallback	Registers digital pin value input callback	-
SuoSetInputAnalogPinCallback	Registers analog pin value input callback	-
External bus interface functions		
SuoCreateExtbus	Generates external bus	Note 1
SuoGetExtbusHandle	Acquires external bus handle	-
SuoSetReadExtbusCallback	Registers external bus read access callback	-
SuoSetWriteExtbusCallback	Registers external bus write access callback	-
Serial interface functions		
SuoCreateSerialUART	Generates serial interface (UART type)	Note 1
SuoCreateSerialCSI	Generates serial interface (CSI type)	Note 1
SuoGetSerialHandle	Acquires serial interface handle	-

Table 4-1 Supplied Functions

Function Name	Description	Remark
SuoSetSerialParameterUART	Sets serial interface parameter (UART type)	Note 3
SuoSetSerialParameterCSI	Sets serial interface parameter (CSI type)	Note 3
SuoGetSerialParameterUART	Acquires serial interface parameter (UART type)	Note 3
SuoGetSerialParameterCSI	Acquires serial interface parameter (CSI type)	Note 3
SuoSendSerialData	Performs serial transmission (1 data)	Note 3
SuoSendSerialDataList	Performs serial transmission (more than one data)	Note 3
SuoSendSerialFile	Performs serial transmission (serial file)	Note 3
SuoSetNotifySentSerialCallback	Registers serial interface transmission end notification callback	-
SuoSetReceiveSerialCallback	Registers serial interface reception callback	-
Signal output unit interface functions		
SuoCreateWave	Generates signal output unit	Note 1
SuoGetWaveHandle	Acquires signal output unit handle	-
SuoSendWaveFile	Performs transmission via signal output unit (signal data file)	Note 3
SuoSetNotifySentWaveCallback	Registers signal output unit transmission end notification callback	-

Note 1 This function can only be called in the MakeUserModel function. It cannot be called in a callback function.

Note 2 A callback function is not executed unless it is called at the timing of the MakeUserModel function.

Note 3 This function cannot be called in the MakeUserModel function. It can only be called in a callback function.

4. 1. 1 Details of supplied functions

This section shows how to reference the supplied functions.

SuoSetInitCallback

Initialization callback registration

```
void  SuoSetInitCallback(SuoInitCallback func);
```

Parameters

func

Return value

None

Explanation

This function registers the user-defined function that performs initialization processing.

The function registered by this function is called only once, when SM plus is started.

If NULL is specified for *func*, registration is canceled.

Example

```
void InitFunc(void);

/* MakeUserModel */
SuoUserEntry void MakeUserModel(const char *option)
{
        .........
        SuoSetInitCallback(InitFunc);        /* Set initialize function */
}

/* Initialize function */
void InitFunc(void){
        .........
}
```

SuoSetResetCallback

Reset callback registration

```
void  SuoSetResetCallback(SuoResetCallback func);
```

Parameters

func	Specifies a pointer to the user-defined function that performs reset processing. (Refer to " ResetFunc".)

Return value

None

Explanation

This function registers the user-defined function that performs reset processing.

The registered function is called when the CPU is reset.

If NULL is specified for *func*, registration is canceled.

Example

```
void ResetFunc(void);

func1()
{
        .........
        SuoSetResetCallback(ResetFunc);          /* Set reset function */
}

/* Reset function */
void ResetFunc(void){
        .........
}
```

SuoCreateTimer

Timer generation

```
int  SuoCreateTimer(const char* timerName, SuoHandle* handle);
```

Parameters

timerName	Specifies the name of the timer interface.
handle	Specifies the location where the handle of the timer interface is to be stored.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoCreateTimer function generates a timer interface.

The generated timer interface is associated with the name specified for *timerName*.

If the function is successful, the handle of the generated timer interface can be obtained.

The timer interface can then be controlled by specifying this handle.

The handle can also be obtained by using the SuoGetTimerHandle function.

The SuoCreateTimer function can only be called in the MakeUserModel function. An error occurs if it is called at any other timing.

Example

```
SuoHandle hTim1;

SuoUserEntry void MakeUserModel(const char *option)
{
        .........
        SuoCreateTimer("TIM1", &hTim1);          /* Create "TIM1" */
}
```

SuoGetTimerHandle

Timer handle acquisition

```
SuoHandle  SuoGetTimerHandle(const char* timerName);
```

Parameters

timerName	Specifies the name of the timer interface.

Return value

If the function is successful, the handle of the specified timer interface is returned.

If the function fails, NULL is returned.

Explanation

The SuoGetTimerHandle function is used to obtain the handle of the specified timer interface.

If the function is successful, the handle of the specified timer interface is returned.

Specify the name specified by the SuoCreateTimer function as *timerName*.

If a different name is specified, NULL is returned.

Example

```
SuoHandle hTim1;

func1()
{
        .........
        hTim1 = SuoGetTimerHandle("TIM1");          /* Get handle of "TIM1" */
}
```

SuoSetTimer

Timer time setting

```
int  SuoSetTimer(SuoHandle handle, int timeUnit, unsigned long timeValue);
```

Parameters

handle — Specifies the handle of the timer interface.

timeUnit — Specifies the time unit (specify any of the following).

Value	Meaning
SUO_MAINCLK	Main clock cycle units
SUO_USEC	ms units

timeValue — Specifies the timer cycle time. The unit is the same as *timeUnit*.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoSetTimer function sets a cyclic timer for the specified timer interface.

The cycle time is specified by the value of *timeValue* in units of *timeUnit*. Zero must not be specified for timeValue.

The timer starts operating immediately after this function is called.

If a timer notification function has been registered by the SuoSetNotifyTimer function, the timer notification function is called in each cycle.

The timer continues operating until it is stopped by the SuoKillTimer function.

If the SuoSetTimer function is called for the timer that is currently operating, the timer is reset and starts operating with the specified cycle time.

Example

```
SuoHandle hTim1;

func1()
{
        .....
        SuoSetTimer(hTim1, SUO_USEC, 20);        /* Invoke 20us cyclic timer */
}
```

SuoKillTimer

Timer time cancellation

```
int  SuoKillTimer(SuoHandle handle);
```

Parameters

handle	Specifies the handle of the timer interface.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoKillTimer function stops the cyclic timer of the specified timer interface.

If the timer is operating, the timer is stopped. If the timer is stopped, nothing is done (in this case, an error does not occur).

Example

```
SuoHandle hTim1;

func1()
{
        .........
        SuoKilllTimer(hTim1);                /* Stop timer */
}
```

SuoSetNotifyTimerCallback

Timer time notification callback registration

```
int  SuoSetNotifyTimerCallback(SuoHandle handle, SuoNotifyTimerCallback func);
```

Parameters

handle	Specifies the handle of the timer interface.
func	Specifies a pointer to the user-defined function that reports the time of the timer. (Refer to " NotifyTimerFunc".)

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

This function registers a user-defined function that performs processing when the time of the timer is reported.

The registered function is called in every timer cycle of the specified timer interface.

If NULL is specified for *func*, registration is canceled.

Example

```
void NotifyTimerFunc(SuoHandle handle);
SuoHandle hTim1;

func1()
{
        .........
        SuoSetNotifyTimerCallback(hTim1, NotifyTimerFunc);        /* Set notify-timer function */
}

/* Notify-timer function */
void NotifyTimerFunc(SuoHandle handle)
{
        .........
}
```

SuoCreatePin

Pin generation

```
int  SuoCreatePin(const char* pinName, SuoHandle* handle);
```

Parameters

pinName	Specifies the name of the pin.
handle	Specifies the location where the handle of a pin interface is to be stored.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoCreatePin function generates a pin interface.

The generated pin interface is associated with the name specified for *pinName*.

Also, the pin specified as *pinName* is generated.

If the function is successful, the handle of the generated pin interface can be obtained.

The pin interface can then be controlled by specifying this handle. The handle can also be obtained by using the SuoGetPinHandle function.

The SuoCreatePin function can only be called in the MakeUserModel function. An error occurs if it is called at any other timing.

Example

```
SuoHandle hPinP00;
SuoHandle hPinABC;

SuoUserEntry void MakeUserModel(const char *option)
{
        .........
        SuoCreatePin("P00", &hPinP00);          /* Create "P00" */
        SuoCreatePin("ABC", &hPinABC);          /* Create "ABC" */
}
```

SuoGetPinHandle

Pin handle acquisition

```
SuoHandle  SuoGetPinHandle(const char* pinName);
```

Parameters

pinName	Specifies the name of the pin.

Return value

If the function is successful, the handle of the specified pin interface is returned.

If the function fails, NULL is returned.

Explanation

The SuoGetPinHandle function is used to obtain the handle of the specified pin interface.

If the function is successful, the handle of the specified pin interface is returned.

Specify the name of a function specified by the SuoCreatePin function as *pinName*.

If a different name is specified, NULL is returned.

Example

```
SuoHandle hPinP00;

func1()
{
        .........
        hPinP00 = SuoGetPinHandle("P00");          /* Get handle of "P00" */
}
```

SuoOutputDigitalPin

Digital pin value output

```
int  SuoOutputDigitalPin(SuoHandle handle, int pinValue);
```

Parameters

handle Specifies the handle of the pin interface.

pinValue Specifies the value to be output to a pin (specify any of the following).

Value	Meaning
SUO_HIGH (=1)	HIGH value
SUO_LOW (=0)	LOW value

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoOutputDigitalPin function outputs a digital value signal to the specified pin interface.

To output an analog value signal, use the SuoOutputAnalogPin function.

Example

```
SuoHandle hPinP00;

func1()
{
        .........
        SuoOutputDigitalPin(hPinP00, SUO_HIGH);          /* Output HIGH */
}
```

SuoOutputAnalogPin

Analog pin value output

```
int  SuoOutputAnalogPin(SuoHandle handle, double pinValue);
```

Parameters

handle	Specifies the handle of the pin interface.
pinValue	Specifies the value (analog value) to be output to a pin (unit: V (volts)).

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoOutputAnalogPin function outputs an analog value signal to the specified pin interface.

Specify an analog value in V (volts), as floating-point data.

To output a digital value signal, use the SuoOutputDigitalPin function.

Example

```
SuoHandle hPinP00;

func1()
{
        .........
        SuoOutputAnalogPin(hPinP00, 3.5);        /* Output 3.5V */
}
```

SuoSetInputDigitalPinCallback

Digital pin value input callback registration

```
int  SuoSetInputDigitalPinCallback(SuoHandle handle, SuoInputDigitalPinCallback func);
```

Parameters

handle	Specifies the handle of the pin interface.
func	Specifies the pointer to a user-defined function that performs digital pin input processing. (Refer to " InputDigitalPinFunc".)

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

This function is used to register a user-defined function that performs digital pin input processing.

The registered function is called when a signal is input to the specified pin.

If NULL is specified for *func*, registration is canceled.

Example

```
void InputDigitalPinFunc(SuoHandle handle, int pinValue);
SuoHandle hPinP00;

func1()
{
        .........
        SuoSetInputDigitalPinCallback(hPinP00, InputDigitalPinFunc);  /* Set input-digital-pin function */
}

/* Input-digital-pin function */
void InputDigitalPinFunc(SuoHandle handle, int pinValue)
{
        .........
}
```

SuoSetInputAnalogPinCallback

Analog pin value input callback registration

```
int  SuoSetInputAnalogPinCallback(SuoHandle handle, SuoInputAnalogPinCallback func);
```

Parameters

handle	Specifies the handle of the pin interface.
func	Specifies the pointer to a user-defined function that performs analog pin input processing. (Refer to " InputAnalogPinFunc".)

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

This function is used to register a user-defined function that performs analog pin input processing.

The registered function is called when a signal is input to the specified pin.

If NULL is specified for *func*, registration is canceled.

Example

```
void InputAnalogPinFunc(SuoHandle handle, double pinValue);
SuoHandle hPinP00;

func1()
{
        .........
        SuoSetInputAnalogPinCallback(hPinP00, InputAnalogPinFunc);      /* Set input-analog-pin function */
}

/* Input-analog-pin function */
void InputAnalogPinFunc(SuoHandle handle, double pinValue)
{
        .........
}
```

SuoCreateExtbus

External bus generation

```
int  SuoCreateExtbus(const char* extbusName, unsigned long addr, unsigned long size, SuoHandle* handle);
```

Parameters

extbusName	Specifies the name of the external bus.
addr	Specifies the first address of the external memory area.
size	Specifies the size of the external memory area.
handle	Specifies the location where the handle of the external bus interface is to be stored.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoCreateExtbus function is used to generate an external bus interface.

The generated pin interface is associated with the name specified for *extbusName*.

If the function is successful, the handle of the generated external bus interface can be obtained.

The external bus interface can then be controlled by specifying this handle. The handle can also be obtained by using the SuoGetExtbusHandle function.

The SuoCreateExtbus function can only be called in the MakeUserModel function. An error occurs if it is called at any other timing.

Example

```
SuoHandle hExtbus1;

SuoUserEntry void MakeUserModel(const char *option)
{
        .....
        SuoCreateExtbus("EXTBUS1", 0x200000, 0x1000, &hExtbus1);        /* Create "EXTBUS1" */
}
```

SuoGetExtbusHandle

External bus handle acquisition

```
SuoHandle  SuoGetExtbusHandle(const char* extbusName);
```

Parameters

extbusName	Specifies the name of the external bus.

Return value

If the function is successful, the handle of the specified external bus interface is returned.

If the function fails, NULL is returned.

Explanation

The SuoGetExtbusHandle function is used to obtain the handle of the specified external bus interface.

If the function is successful, the handle of the specified external bus interface is returned.

Specify the name specified by the SuoCreateExtbus function as *extbusName*.

If a different name is specified, NULL is returned.

Example

```
SuoHandle hExtbus1;

func1()
{
        .....
        hExtbus1 = SuoGetExtbusHandle("EXTBUS1");        /* Get handle of "EXTBUS1" */
}
```

SuoSetReadExtbusCallback

External bus read access callback registration

```
int  SuoSetReadExtbusCallback(SuoHandle handle, SuoReadExtbusCallback func);
```

Parameters

handle	Specifies the handle of the external bus interface.
func	Specifies the pointer to a user-defined function that performs read access processing of an external bus. (Refer to " ReadExtbusFunc".)

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

This function is used to register a user-defined function that performs read access processing of an external bus.

The registered function is called if a read request is issued to the specified external bus (in the registered address range).

If NULL is specified for *func*, registration is canceled.

Example

```
void ReadExtbusFunc(SuoHandle handle, unsigned long addr, int accessSize, unsigned char data[]);
SuoHandle hExtbus1;

func1()
{
        .........
        SuoSetReadExtbusCallback(hExtbus1, ReadExtbusFunc);       /* Set read-external-bus function */
}

/* Read-external-bus function */
void ReadExtbusFunc(SuoHandle handle, unsigned long addr, int accessSize, unsigned char data[])
{
        .........
}
```

SuoSetWriteExtbusCallback

External bus write access callback registration

```
int  SuoSetWriteExtbusCallback(SuoHandle handle, SuoWriteExtbusCallback func);
```

Parameters

handle	Specifies the handle of the external bus interface.
func	Specifies the pointer to a user-defined function that performs write access processing of an external bus. (Refer to " WriteExtbusFunc".)

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

This function is used to register a user-defined function that performs write access processing of an external bus.

The registered function is called when a write request is issued to the specified external bus (in the registered address range).

If NULL is specified for *func*, registration is canceled.

Example

```
void WriteExtbusFunc(SuoHandle handle, unsigned long addr, int accessSize, const unsigned char data[]);
SuoHandle hExtbus1;

func1()
{
        .........
        SuoSetWriteExtbusCallback(hExtbus1, WriteExtbusFunc);       /* Set write-external-bus function */
}

/* Write-external-bus function */
void WriteExtbusFunc(SuoHandle handle, unsigned long addr, int accessSize, const unsigned char data[])
{
        .........
}
```

SuoCreateSerialUART

Serial interface generation (UART type)

```
int  SuoCreateSerialUART(const char* serialName, const char* pinNameTXD, const char* pinNameRXD,
SuoHandle* handle);
```

Parameters

serialName	Specifies the name of the serial interface.
pinNameTXD	Specifies the name of the transmit data pin used by the serial interface.
pinNameRXD	Specifies the name of the receive data pin used by the serial interface.
handle	Specifies the location where the handle of the serial interface is to be stored.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoCreateSerialUART function is used to generate a serial interface (UART type).

The generated pin interface is associated with the name specified for *serialName*.

In addition, pins specified as *pinNameTXD* and *pinNameRXD* are also generated.

If the function is successful, the handle of the generated serial interface can be obtained.

The serial interface can then be controlled by specifying this handle. The handle can also be obtained by using the SuoGetSerialHandle function.

The SuoCreateSerialUART function can only be called in the MakeUserModel function. An error occurs if it is called at any other timing.

Example

```
SuoHandle hUart1;

SuoUserEntry void MakeUserModel(const char *option)
{
        .....
        SuoCreateSerialUART("UART1", "TXD1", "RXD1", &hUart1);          /* Create "UART1" */
}
```

SuoCreateSerialCSI

Serial interface generation (CSI type)

```
int  SuoCreateSerialCSI(const char* serialName, const char* pinNameSO, const char* pinNameSI, const char*
pinNameSCK, SuoHandle* handle);
```

Parameters

serialName	Specifies the name of the serial interface.
pinNameSO	Specifies the name of the transmit data pin used by the serial interface.
pinNameSI	Specifies the name of the receive data pin used by the serial interface.
pinNameSCK	Specifies the name of the clock pin used by the serial interface.
handle	Specifies the location where the handle of the serial interface is to be stored.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoCreateSerialCSI function is used to generate a serial interface (CSI type).

The generated pin interface is associated with the name specified for *serialName*.

In addition, the pins specified as *pinNameSO*, *pinNameSI*, and *pinNameSCK* are also generated.

If the function is successful, the handle of the generated serial interface can be obtained.

The serial interface can then be controlled by specifying this handle. The handle can also be obtained by using the SuoGetSerialHandle function.

The SuoCreateSerialCSI function can only be called in the MakeUserModel function. An error occurs if it is called at any other timing.

Example

```
SuoHandle hCsi1;

SuoUserEntry void MakeUserModel(const char *option)
{
        .....
        SuoCreateSerialCSI("CSI1", "SO1", "SI1", "SCK1", &hCsi1);     /* Create "CSI1" */
}
```

SuoGetSerialHandle

Serial interface handle acquisition

```
SuoHandle  SuoGetSerialHandle(const char* serialName);
```

Parameters

serialName	Specifies the name of the serial interface.

Return value

If the function is successful, the handle of the specified pin interface is returned.

If the function fails, NULL is returned.

Explanation

The SuoGetSerialHandle function is used to obtain the handle of the specified pin interface.

If the function is successful, the handle of the specified pin interface is returned.

Specify the name specified by the SuoCreateSerialXXX function as *serialName*.

If a different name is specified, NULL is returned.

Example

```
SuoHandle hSerial1;

func1()
{
        .........
        hSerial1 = SuoGetSerialHandle("SERIAL1");          /* Get handle of "SERIAL1" */
}
```

SuoSetSerialParameterUART

Serial interface parameter setting (UART type)

```
int  SuoSetSerialParameterUART(SuoHandle handle, const SuoSerialParameterUART* param);
```

Parameters

handle	Specifies the handle of the serial interface.
param	Specifies the location where the parameters of the serial interface (UART type) are to be stored. Specify a pointer to SuoSerialParameterUART structure.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoSetSerialParameterUART function is used to set parameters (UART type) related to the serial operation of the specified serial interface.

The default values of the parameters are as follows.

Baud rate	:	9600bps
Transfer direction	:	LSB first
Data bit length	:	7 bits
Stop bit length	:	1 bit
Parity	:	None

Example

```
SuoHandle hUart1;

func1()
{
        SuoSerialParameterUART param;
        .....
        param.baudrate = 19200;                         /* 19200bps     */
        param.direction = SUO_LSBFIRST;                 /* LSB First    */
        param.dataLength = 8;                           /* databit 8bit */
        param.stopLength = 1;                           /* stopbit 1bit */
        param.parity = SUO_EVENPARITY;                  /* even parity  */
        SuoSetSerialParameterUART(hUart1, &param);      /* Set parameter of UART1 */
}
```

Structure

SuoSerialParameterUART structure

typedef struct {

unsigned long baudrate; Baud rate value (in bps)

int direction; Transfer direction (specify any of the following)

Value	Meaning
SUO_MSBFIRST	MSB first
SUO_LSBFIRST	LSB first

int dataLength; Data bit length (specify 1 to 32)

int stopLength; Stop bit length (specify 1 or 2)

int parity; Parity (specify any of the following)

Value	Meaning
SUO_NONEPARITY	No parity
SUO_ZEROPARITY	0 parity (0 parity during transmission, no parity check during reception)
SUO_ODDPARITY	Odd parity
SUO_EVENPARITY	Even parity

} SuoSerialParameterUART;

SuoSetSerialParameterCSI

Serial interface parameter setting (CSI type)

```
int  SuoSetSerialParameterCSI(SuoHandle handle, const SuoSerialParameterCSI* param);
```

Parameters

handle	Specifies the handle of the serial interface.
param	Specifies the location where the parameters of the serial interface (CSI type) are to be stored. Specify a pointer to SuoSerialParameterCSI structure.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoSetSerialParameterCSI function is used to set parameters (CSI type) related to the serial operation of the specified serial interface.

The default values of the parameters are as follows.

Mode	:	Slave
Transfer clock	:	0
Phase	:	Normal phase
Transfer direction	:	MSB first
Data bit length	:	8 bits

Example

```
SuoHandle hCsi1;

func1()
{
        SuoSerialParameterCSI param;
        .....
        param.mode = SUO_SLAVE;                 /* slave       */
        param.frequency = 1000000;              /* 1MHz        */
        param.phase = 0;                        /* normal      */
        param.direction = SUO_LSBFIRST;         /* LSB First   */
        param.dataLength = 8;                   /* databit 8bit */
        SuoSetSerialParameterCSI(hCsi1, &param);  /* Set parameter of CSI1 */
}
```

Structure

SuoSerialParameterCSI structure

typedef struct {

int mode;

Operation mode (specify any of the following)

Value	Meaning
SUO_MASTER	Master operation
SUO_SLAVE	Slave operation

unsigned long frequency;

Frequency of transfer clock (in Hz)
Zero must not be specified if master operation is specified.

int phase;

Phase (specify any of the following)
For details, refer to Table 4-2 .

Value	Meaning
0	Normal phase
SUO_PRECEDEDATA	Data output first
SUO_REVERSECLOCK	Clock reversal
SUO_PRECEDEDATA | SUO_REVERSECLOCK	Specifies both data output first and clock reversal

int direction;

Transfer direction (specify any of the following)

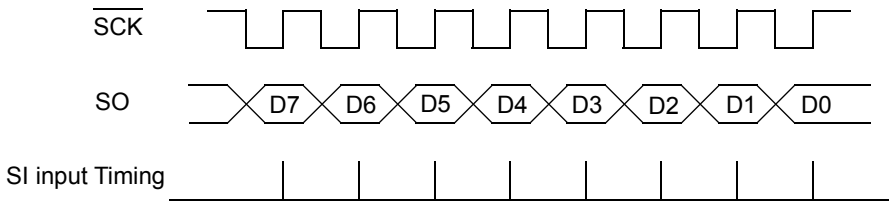
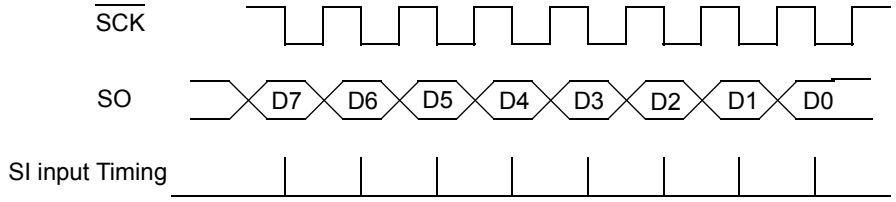
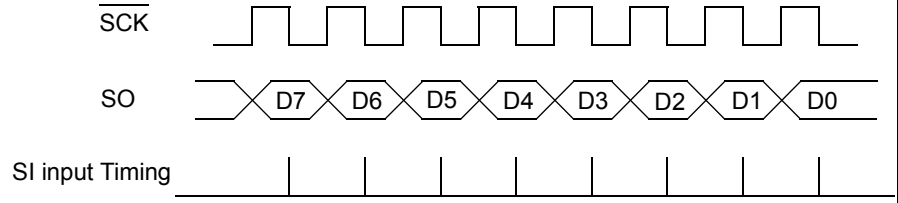
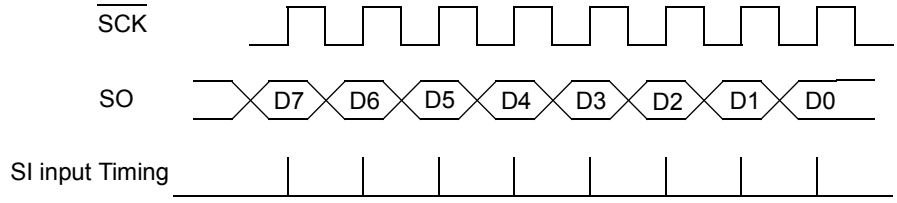
Value	Meaning
SUO_MSBFIRST	MSB first
SUO_LSBFIRST	LSB first

int datalength;

Data bit length (1 to 32)

} SuoSerialParameterCSI;

Table 4-2 CSI Phase Types (SuoSetSerialParameterCSI Function)

Value of Phase	Phase
0	
SUO_PRECEDEDATA	
SUO_REVERSECLOCK	
SUO_PRECEDEDATA| SUO_REVERSECLOCK	

SuoGetSerialParameterUART

Serial interface parameter acquisition (UART type)

```
int  SuoGetSerialParameterUART(SuoHandle handle, SuoSerialParameterUART* param);
```

Parameters

handle	Specifies the handle of the serial interface.
param	Specifies the location where the parameters of the serial interface (UART type) are to be stored. Specify a pointer to SuoSerialParameterUART structure.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoGetSerialParameterUART function is used to obtain the parameters (UART type) related to serial operation of the specified serial interface.

Example

```
SuoHandle hUart1;

func1()
{
        SuoSerialParameterUART param;
        .........
        SuoGetSerialParameterUART(hUart1, &param);        /* Get parameter of UART1 */
        .........
}
```

SuoGetSerialParameterCSI

Serial interface parameter acquisition (CSI type)

```
int  SuoGetSerialParameterCSI(SuoHandle handle, SuoSerialParameterCSI* param);
```

Parameters

handle	Specifies the handle of the serial interface.
param	Specifies the location where the parameters of the serial interface (CSI type) are to be stored. Specify a pointer to SuoSerialParameterCSI structure.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoGetSerialParameterCSI function is used to obtain the parameters (CSI type) related to serial operation of the specified serial interface.

Example

```
SuoHandle hCsi1;

func1()
{
        SuoSerialParameterCSI param;
        .........
        SuoGetSerialParameterCSI(hCsi1, &param);        /* Get parameter of CSI1 */
        .........
}
```

SuoSendSerialData

Serial data transmission (1 data)

```
int  SuoSendSerialData(SuoHandle handle, unsigned long data);
```

Parameters

handle	Specifies the handle of the serial interface.
data	Specifies the transmit data (1 data).

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoSendSerialData function is used to start transmitting one serial data.

It takes time to complete transmitting the serial data. To report the time of completion of transmission, set the transmission end notification function by using the SuoSetNotifySentSerialCallback function.

If the SuoSendSerialData function is called for a serial interface that is currently transmitting data, an error occurs.

Example

```
SuoHandle hSerial1;

func1()
{
        .....
        SuoSendSerialData(hSerial1, 0x80);          /* Send 0x80 */
}
```

SuoSendSerialDataList

Serial data transmission (two or more data)

```
int  SuoSendSerialDataList(SuoHandle handle, long count, unsigned long dataList[]);
```

Parameters

handle	Specifies the handle of the serial interface.
count	Specifies the number of data to be transmitted (1 to 32767).
dataList[]	Specifies the transmit data (two or more data). Specify an array consisting of the number of data to be transmitted.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoSendSerialDataList function is used to start transmitting two or more serial data.

It takes time to complete transmission of the serial data. To report the time of completion of transmission, set the transmission end notification function by using the SuoSetNotifySentSerialCallback function.

If the SuoSendSerialDataList function is called for a serial interface that is currently transmitting data, an error occurs.

Example

```
SuoHandle hSerial1;

func1()
{
        unsigned long dataList[6] = {0x73, 0x65, 0x72, 0x69, 0x61, 0x6c};
        .........
        SuoSendSerialDataList(hSerial1, 6, dataList);          /* Send dataList */
}
```

SuoSendSerialFile

Serial data transmission (serial file)

```
int  SuoSendSerialFile(SuoHandle handle, const char* serialFile);
```

Parameters

handle	Specifies the handle of the serial interface.
serialFile	Specifies the serial file. A serial file is a file that has been saved after being edited on the serial window of SM plus.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoSendSerialFile function is used to start transmitting serial data described in a serial file.

If *serialFile* is specified by a relative path, it is treated as relative to the path of the user model (*UserModel.dll*).

It takes time to complete transmission of the serial data. To report the time of completion of transmission, set the transmission end notification function by using the SuoSetNotifySentSerialCallback function.

If the SuoSendSerialFile function is called for a serial interface that is currently transmitting data, an error occurs.

Example

```
SuoHandle hSerial1;

func1()
{
        .....
        SuoSendSerialFile(hSerial1, "foo.ser");        /* Send serial data on "foo.ser" */
}
```

SuoSetNotifySentSerialCallback

Serial interface transmission end notification callback registration

```
int  SuoSetNotifySentSerialCallback(SuoHandle handle, SuoNotifySentSerialCallback func);
```

Parameters

handle	Specifies the handle of the serial interface.
func	Specifies the pointer to a user-defined function that performs processing when serial transmission is completed. (Refer to " NotifySentSerialFunc".)

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

This function is used to register a user-defined function that performs processing when serial transmission is completed.

The registered function is called when one or more serial data specified to be transmitted have been completely transmitted.

If NULL is specified for *func*, registration is canceled.

Example

```
void NotifySentSerialFunc(SuoHandle handle);
SuoHandle hSerial1;

func1()
{
        .........
        SuoSetNotifySentSerialCallback(hSerial1, NotifySentSerialFunc);     /* Set notify-sent-serial function */
}

/* Notify-sent-serial function */
void NotifySentSerialFunc(SuoHandle handle)
{
        .........
}
```

SuoSetReceiveSerialCallback

Serial data reception callback registration

```
int  SuoSetReceiveSerialCallback(SuoHandle handle, SuoReceiveSerialCallback func);
```

Parameters

handle	Specifies the handle of the serial interface.
func	Specifies the pointer to a user-defined function that performs processing when serial data is received. (Refer to " ReceiveSerialFunc".)

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

This function is used to register a user-defined function that performs processing when serial data is received.

The registered function is called when one serial data has been received.

If NULL is specified for *func*, registration is canceled.

Example

```
void ReceiveSerialFunc(SuoHandle handle, unsigned long data, int status);
SuoHandle hSerial1;

func1()
{
        .....
        SuoSetReceiveSerialCallback(hSerial1, ReceiveSerialFunc);    /* Set receive-serial function */
}

/* Receive-serial function */
void ReceiveSerialFunc(SuoHandle handle, unsigned long data, int status)
{
        .....
}
```

SuoCreateWave

Signal output unit generation

```
int  SuoCreateWave(const char* waveName, int count, const char* pinNameList[], SuoHandle* handle);
```

Parameters

waveName	Specifies the name of the signal output unit.
count	Specifies the number of pins used by the signal output unit.
pinNameList	Specifies the names of the pins used by the signal output unit. Specify names in an array equivalent to the number of pins.
handle	Specifies the location where the handle of the signal output unit interface is to be stored.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoCreateWave function is used to generate a signal output unit interface.

The generated signal output unit interface is associated with the name specified for *waveName*.

In addition, the pins specified by *count*/*pinNameList* are also generated.

If the function is successful, the handle of the generated signal output unit interface can be obtained.

The signal output unit interface can then be controlled by specifying this handle. The handle can also be obtained by using the SuoGetWaveHandle function.

The SuoCreateWave function can only be called in the MakeUserModel function. An error occurs if it is called at any other timing.

Example

```
SuoHandle hWave1;

SuoUserEntry void MakeUserModel(const char *option)
{
        .....
        char* pinNameList[4] = {"P00", "P01", "P02", "P03"};
        SuoCreateWave("WAVE1", 4, pinNameList, &hWave1);        /* Create "WAVE1" */
}
```

SuoGetWaveHandle

Signal output unit handle acquisition

```
SuoHandle  SuoGetWaveHandle(const char* waveName);
```

Parameters

waveName	Specifies the name of the signal output unit.

Return value

If the function is successful, the handle of the specified signal output unit interface is returned.

If the function fails, NULL is returned.

Explanation

The SuoGetWaveHandle function is used to obtain the handle of the specified signal output unit interface.

If the function is successful, the handle of the specified pin interface is returned.

Specify the name specified by the SuoCreateWave function as *waveName*.

If a different name is specified, NULL is returned.

Example

```
SuoHandle hWave1;

func1()
{
        .........
        hWave1 = SuoGetWaveHandle("WAVE1");          /* Get handle of "WAVE1" */
}
```

SuoSendWaveFile

Transmission by signal output unit (signal data file)

```
int  SuoSendWaveFile(SuoHandle handle, const char* waveFile);
```

Parameters

handle	Specifies the handle of the signal output unit interface.
waveFile	Specifies the signal data file. A signal data file is a file that has been saved after being edited on the signal data editor window of SM plus.

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

The SuoSendWaveFile function is used to start transmitting a signal value whose timing is described in a signal data file.

If *waveFile* is specified by a relative path, it is treated as relative to the path of the user model (*UserModel.dll*).

It takes time to complete transmitting the signal data file. To report the time of completion of transmission, set the transmission end notification function by using the SuoSetNotifySentWaveCallback function.

If the SuoSendWaveFile function is called for a signal output unit interface that is currently transmitting data, the data being transmitted is canceled and the newly specified data is transmitted.

Example

```
SuoHandle hWave1;

func1()
{
        .....
        SuoSendWaveFile(hSerial1, "foo.wvi");        /* Send pin data on "foo.wvi" */
}
```

SuoSetNotifySentWaveCallback

Signal output unit transmission end notification callback registration

```
int  SuoSetNotifySentWaveCallback(SuoHandle handle, SuoNotifySentWaveCallback func);
```

Parameters

handle	Specifies the handle of the signal output unit interface.
func	Specifies the pointer to a user-defined function that performs processing when transmission by the signal output unit is completed. (Refer to "NotifySentWaveFunc".)

Return value

If the function is successful, SUO_NOERROR is returned.

If the function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

Explanation

This function is used to register a user function that performs processing when transmission by the signal output unit is completed.

The registered function is called when all signal data specified to be transmitted have been completely transmitted.

If NULL is specified for *func*, registration is canceled.

Example

```
void NotifySentWaveFunc(SuoHandle handle);
SuoHandle hWave1;

func1()
{
        .........
        SuoSetNotifySentWaveCallback(hWave1, NotifySentWaveFunc);      /* Set notify-sent-wave function */
}

/* Notify-sent-wave function */
void NotifySentWaveFunc(SuoHandle handle)
{
        .........
}
```

4. 2 User-Defined Functions

The following table lists the user-defined functions (MakeUserModel entry functions and callback functions).

Table 4-3 User-Defined Functions

Function Name	Description
MakeUserModel	MakeUserModel function
InitFunc	Initialization callback function
ResetFunc	Reset callback function
NotifyTimerFunc	Timer time notification callback function
InputDigitalPinFunc	Digital pin input value callback function
InputAnalogPinFunc	Analog pin input value callback function
ReadExtbusFunc	External bus read access callback function
WriteExtbusFunc	External bus write access callback function
NotifySentSerialFunc	Serial interface transmission end notification callback function
ReceiveSerialFunc	Serial interface reception callback function
NotifySentWaveFunc	Serial interface reception callback function

4. 2. 1 Details of user-defined functions

This section shows how to reference user-defined functions.

MakeUserModel

MakeUserModel function [user-defined function]

```
SuoUserEntry void  MakeUserModel(const char *option);
```

Caution Because MakeUserModel is a static entry function of the user model, this function name must be used.

Parameters

option	Option character string specified in the configuration file. If no option is specified in the configuration file, NULL character (" ") is assumed.

Return value

None

Explanation

The MakeUserModel function must be used to generate the resources to be used with the user model.

Any function other than the MakeUserModel function cannot generate the resources.

In addition, the MakeUserModel function must be used to register a callback function as necessary.

In particular, an initialization callback function must be registered by the MakeUserModel function (because the initialization timing has passed even if a function is registered by a function other than MakeUserModel).

Example

```
SuoHandle hTim1;
SuoHandle hPinP00;
SuoHandle hExtbus1;

void InitFunc(void);
void ResetFunc(void);

SuoUserEntry void MakeUserModel(const char *option)
{
        /* Create source */
        SuoCreateTimer("TIM1", &hTim1);                                  /* Create "TIM1" */
        SuoCreatePin("P00", &hPinP00);                                   /* Create "P00" */
        SuoCreateExtbus("EXTBUS1", 0x200000, 0x1000, &hExtbus1);         /* Create "EXTBUS1" */

        /* Set callbacks */
        SuoSetInitCallback(InitFunc);                                    /* Set initialize function */
        SuoSetResetCallback(ResetFunc);                                  /* Set reset function */
}
```

InitFunc

Initialization callback function [user-defined function]

```
void  InitFunc (void);
```

Caution InitFunc is a place holder for a user-defined function name, so this function name does not have to be used.

Parameters

None

Return value

None

Explanation

InitFunc describes initialization processing.

Use the SuoSetInitCallback function to register InitFunc as a callback function.

ResetFunc

Reset callback function [user-defined function]

```
void  ResetFunc (void);
```

Caution ResetFunc is a place holder for a user-defined function name, so this function name does not have to be used.

Parameters

None

Return value

None

Explanation

ResetFunc describes the reset processing.

Use the SuoSetResetCallback function to register ResetFunc as a callback function.

NotifyTimerFunc

Timer time notification callback function [user-defined function]

```
void  NotifyTimerFunc (SuoHandle handle);
```

Caution NotifyTimerFunc is a place holder for a user-defined function name, so this function name does not have to be used.

Parameters

handle	Timer interface handle.

Return value

None

Explanation

NotifyTimerFunc describes the processing when the timer time is reported.

Use the SuoSetNotifyTimerCallback function to register NotifyTimerFunc as a callback function.

InputDigitalPinFunc

Digital pin input value callback function [user-defined function]

```
void  InputDigitalPinFunc (SuoHandle handle, int pinValue);
```

Caution InputDigitalPinFunc is a place holder for a user-defined function name, so this function name does not have to be used.

Parameters

handle Pin interface handle.

pinValue Value (digital value) input to the pin (specify any of the following)

Value	Meaning
SUO_HIGH (=1)	HIGH value
SUO_LOW (=0)	LOW value

Return value

None

Explanation

InputDigitalPinFunc describes the digital pin input processing.

Use the SuoSetInputDigitalPinCallback function to register InputDigitalPinFunc as a callback function.

InputAnalogPinFunc

Analog pin input value callback function [user-defined function]

```
void  InputAnalogPinFunc (SuoHandle handle, double pinValue);
```

Caution InputAnalogPinFunc is a place holder for a user-defined function name, so this function name does not have to be used.

Parameters

handle	Pin interface handle.
pinValue	Value (analog value) input to the pin (unit: V (volts))

Return value

None

Explanation

InputAnalogPinFunc describes the analog pin input processing.

Use the SuoSetInputAnalogPinCallback function to register InputAnalogPinFunc as a callback function.

ReadExtbusFunc

External bus read access callback function [user-defined function]

```
void  ReadExtbusFunc (SuoHandle handle, unsigned long addr, int accessSize, unsigned char data[]);
```

Caution ReadExtbusFunc is a place holder for a user-defined function name, so this function name does not have to be used.

Parameters

handle	External bus interface handle
addr	Address
accessSize	Access size
data[]	Data storage area. As many data as the access size must be stored.

Return value

None

Explanation

ReadExtbusFunc describes the read access processing of an external bus.

Data must be stored in *data[]*.

Use the SuoSetReadExtbusCallback function to register ReadExtbusFunc as a callback function.

WriteExtbusFunc

External bus write access callback function [user-defined function]

```
void  WriteExtbusFunc (SuoHandle handle, unsigned long addr, int accessSize, const unsigned char data[]);
```

Caution WriteExtbusFunc is a place holder for a user-defined function name, so this function name does not have to be used.

Parameters

handle	External bus interface handle
addr	Address
accessSize	Access size
data[]	Data storage area. As many data as the access size must be stored.

Return value

None

Explanation

WriteExtbusFunc describes write access processing of an external bus.

Use the SuoSetWriteExtbusCallback function to register WriteExtbusFunc as a callback function.

NotifySentSerialFunc

Serial interface transmission end notification callback function [user-defined function]

```
void  NotifySentSerialFunc (SuoHandle handle);
```

Caution NotifySentSerialFunc is a place holder for a user-defined function name, so this function name does not have to be used.

Parameters

handle	Serial interface handle

Return value

None

Explanation

NotifySentSerialFunc describes the processing when transmission by a serial interface has been completed.

Use the SuoSetNotifySentSerialCallback function to register NotifySentSerialFunc as a callback function.

ReceiveSerialFunc

Serial interface reception callback function [user-defined function]

```
void  ReceiveSerialFunc (SuoHandle handle, unsigned long data, int status);
```

Caution ReceiveSerialFunc is a place holder for a user-defined function name, so this function name does not have to be used.

Parameters

handle Serial interface handle

data Received serial data

status Receive status (specify any of the following)

Value	Meaning
0	Normal reception
SUO_PARITYERR	Parity error (if parity bit does not match)
SUO_FRAMINGERR	Framing error (if stop bit is not detected)

Return value

None

Explanation

ReceiveSerialFunc describes the processing during reception by a serial interface.

Use the SuoSetReceiveSerialCallback function to register ReceiveSerialFunc as a callback function.

NotifySentWaveFunc

Signal output unit transmission end notification callback function [user-defined function]

```
void  NotifySentWaveFunc (SuoHandle handle);
```

Caution NotifySentWaveFunc is a place holder for a user-defined function name, so this function name does not have to be used.

Parameters

handle	Signal output unit interface handle

Return value

None

Explanation

NotifySentWaveFunc describes the processing to be performed when transmission by a signal output unit has been completed.

Use the SuoSetNotifySentWaveCallback function to register NotifySentWaveFunc as a callback function.

4. 3 Error Numbers

Many return values of the supplied function are error numbers. An error number is indicated by a macro name defined by the supplied header file (suo.h). This table lists the error numbers.

Table 4-4 Error Numbers

Error No. (Macro)	Meaning (Top: Outline, Bottom: Details)
SUO_NOERROR	No error has occurred (normal completion).
SUO_CANTALLOC	Memory cannot be allocated.
	Memory cannot be allocated.
SUO_ILLIFNAME	The interface name is not correct.
	NULL or "" is specified for the interface name. Or, an interface name that has not been generated is specified for a handle acquisition function.
SUO_ILLHANDLE	The handle is not correct.
	A handle other than that of the generated interface is specified.
SUO_ILLPARAM	The parameter is not correct.
	A value other than those that can be specified is specified as a parameter.
SUO_CANTCALL	The function cannot be called.
	A function that can be called only by the MakeUserModel function is called by another function. Or, a function that can be called by a function other than the MakeUserModel function is called by the MakeUserModel function. For the rule of calling, refer to Remark in Table 4-1 .
SUO_CONFLICTRES	The resources to be generated conflict.
	Two or more names that are the same as an interface name or pin name generated in the MakeUserModel function exist.
SUO_ILLFILENAME	The file name is not correct.
	NULL or a name including an invalid character is specified for a file name.
SUO_CANTOPENFILE	The file cannot be opened.
	The file does not exist, or is not permitted to be read.
SUO_ILLFILEFMT	The file format is not correct.
	A file of a different type is specified.
SUO_ILLFILECONT	The file contents are not correct.
	The contents of data described in the file include a contradiction, or no data exists in the file.
SUO_ILLPINNAME	The pin name is not correct.
	NULL or "" is specified for the pin name.
SUO_ILLADDRRANGE	The address range is not correct.
	The address range is not valid.

Table 4-4 Error Numbers

Error No. (Macro)	Meaning (Top: Outline, Bottom: Details)
SUO_UNDERSENDING	Already being transmitted.
	New transmission cannot be started because transmission is in progress.

CHAPTER 5 SAMPLE PROGRAM

This chapter explains a sample program of a user model created by using the SM plus user open interface.

This table shows the sample program.

Table 5-1 Sample Program

No.	Sample Name	Description
1	Timer	Sample using timer interface

5. 1 Timer

5. 1. 1 Overview

Timer model is a sample program using a timer interface.

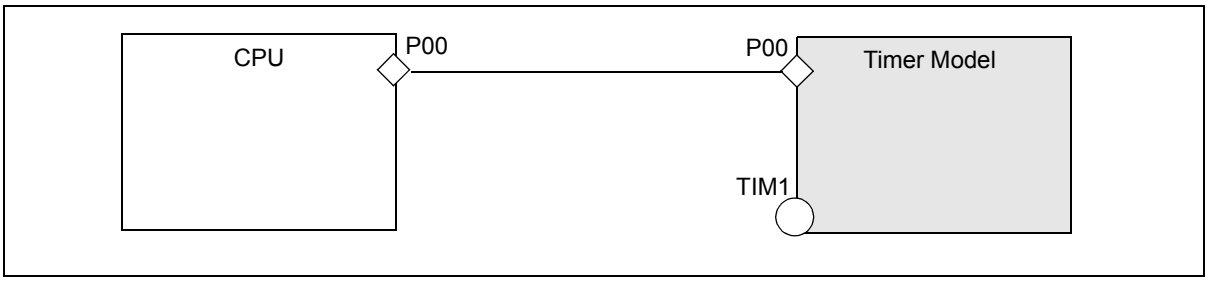
It outputs a value to a pin at fixed time intervals.

5. 1. 2 Configuration

The timer model generates the P00 pin and TIM1 timer.

The generated P00 pin is connected to the P00 pin of the CPU.

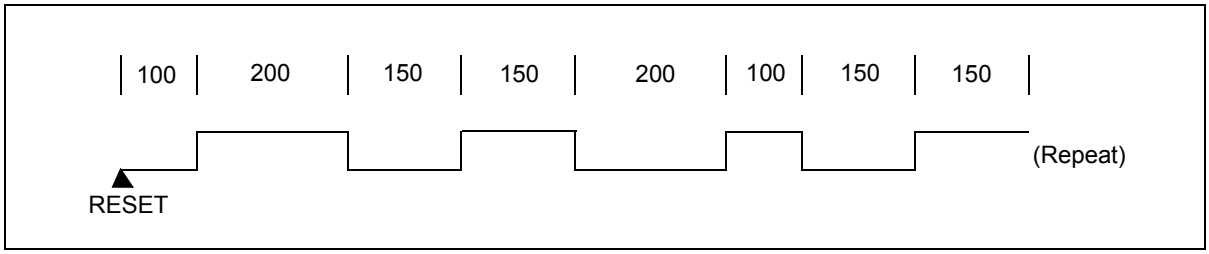
Figure 5-1 Timer Model Configuration



5. 1. 3 Operation

The timer model calculates a predetermined time by using the timer interface, and alternately outputs a low level and a high level to the P00 pin. The output value and output time are as shown below.

Figure 5-2 Timer Model Operation



5. 1. 4 Project file

The following table shows the setting information of the Visual C++ project file of the Timer model.

Table 5-2 Setting Information of Timer Model

Information	Description
Project type	Win32 Dynamic-Link Library
Source file	..\sys\suolink.c, uo_timer.c
Path of include file	..\sys (Specify the folder storing suo.h.)

5. 1. 5 Details of file

(1) Source file of model (uo_timer.c)

(1/3)

```
#include <windows.h>
#include "suo.h"

/* Handle */
SuoHandle p00;
SuoHandle tim1;

/* Wave-Table */
#define MAXWAVE 8
struct _WaveTable {
    unsigned long time;                 /* Wait Time [MainClk] */
    int pinValue;                       /* Pin Value (SUO_HIGH or SUO_LOW) */
} waveTable[MAXWAVE] = {
    100,    SUO_HIGH,
    200,    SUO_LOW,
    150,    SUO_HIGH,
    150,    SUO_LOW,
    200,    SUO_HIGH,
    100,    SUO_LOW,
    150,    SUO_HIGH,
    150,    SUO_LOW
};
int waveIndex;

/* Declare */
void Reset(void);
void NotifyTimer1(SuoHandle handle);
void puterr(int error);

/* MakeUserModel */
SuoUserEntry void MakeUserModel(void)
{
    int error;

    /* Create interface */
    if((error = SuoCreateTimer("TIM1", &tim1)) != SUO_NOERROR){
        puterr(error);
        return;
    }
    if((error = SuoCreatePin("P00", &p00)) != SUO_NOERROR){
        puterr(error);
        return;
    }
```

(2/3)

```
    /* Set callback */
    SuoSetResetCallback(Reset);
    SuoSetNotifyTimerCallback(tim1, NotifyTimer1);
}

/* Reset callback */
void Reset(void)
{
    int error;

    /* Initialize Wave-Tabel index */
    waveIndex = 0;

    /* Output LOW(initial value) to P00 */
    if((error = SuoOutputDigitalPin(p00, SUO_LOW)) != SUO_NOERROR){
        puterr(error);
        return;
    }

    /* Set wait time */
    if((error = SuoSetTimer(tim1, SUO_MAINCLK, waveTable[waveIndex].time)) != SUO_NOERROR){
        puterr(error);
        return;
    }
}
/* NotifyTimer callback */
void NotifyTimer1(SuoHandle handle)
{
    int error;

    /* Output value to P00 */
    if((error = SuoOutputDigitalPin(p00, waveTable[waveIndex].pinValue)) != SUO_NOERROR){
        puterr(error);
        return;
    }

    /* Set next Wave-Tabel index */
    waveIndex++;
    if(waveIndex >= MAXWAVE){
        waveIndex = 0;
    }
```

(3/3)

```
    /* Set wait time */
    if((error = SuoSetTimer(tim1, SUO_MAINCLK, waveTable[waveIndex].time)) != SUO_NOERROR){
        puterr(error);
        return;
    }
}

/* Report error */
void puterr(int error)
{
    char message[80];
    wsprintf(message, "The user open interface error (0x%04x) occurred.", error);
    MessageBox(NULL, message, "ERROR", MB_OK|MB_ICONERROR);
}
```

(2) Configuration file (smplus.cfg)

(1/1)

```
cpu = CPU('a');

# ------------------------------------------------------------
#   UO_TIMER description (CPU=uPD70F3289Y)
# ------------------------------------------------------------

# Generate uo_timer.dll
uo_timer = Device("USEROPEN", "-f=uo_timer.dll");

# Connect PIN (CPU.P00-UO_TIMER.P00)
wire_P00 = Wire(1);
wire_P00 += cpu.Port("P00/TIP61/TOP61");
wire_P00 += uo_timer.Port("P00");
```

(3) Source file of target program (lm_timer.c)

(1/1)

```
/* Target Program for UO_TIMER */

#pragma ioreg


void main()
{
   unsigned char value;

   PM0 = 0xff;              /* set input mode */
   PM1 = 0x00;              /* set output mode */

   while(1){
      value = P0.0;         /* input signal from "P00" */
      P1.0 = value;         /* output signal to "P10" */
   }
}
```

APPENDIX A INDEX