To our customers,

---

**Old Company Name in Catalogs and Other Documents**

On April 1\(^{\text{st}}\), 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 1\(^{\text{st}}\), 2010
Renesas Electronics Corporation

---

Issued by: Renesas Electronics Corporation (http://www.renesas.com)
Send any inquiries to http://www.renesas.com/inquiry.

---

Note that the following URLs in this document are not available:
http://www.necel.com/
http://www2.renesas.com/

Please refer to the following instead:
Development Tools | http://www.renesas.com/tools
Download | http://www.renesas.com/tool_download

For any inquiries or feedback, please contact your region.
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.
User’s Manual

SM+
System Simulator
User Open Interface

Target Tool
  SM+ for V850 Ver. 2.00
  SM+ for 78K0 Ver. 1.01
  SM+ for 78K0S Ver. 1.01
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 November, 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).
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]

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 Francaise
  Velizy-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
  Taebry, 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

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 NT4.0/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+ 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+:

→ Read this manual in the order of the CONTENTS.
Related Documents: When using this manual, also refer to the following documents. Some related documents may be preliminary versions. Note, however, that whether a related document is preliminary is not indicated in this document.

Documents related to 78K Series development tools (User's Manual)

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Document No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC78K0 Ver. 3.70 C Compiler</td>
<td>U17201E</td>
</tr>
<tr>
<td>CC78K0S C Compiler</td>
<td>U17200E</td>
</tr>
<tr>
<td>RA78K0 Ver. 3.80 Assembler Package</td>
<td>U17199E</td>
</tr>
<tr>
<td>RA78K0S Assembler Package</td>
<td>U17198E</td>
</tr>
<tr>
<td>RA78K0S Assembler Package</td>
<td>U17197E</td>
</tr>
<tr>
<td>SM+ System Simulator</td>
<td>U17246E</td>
</tr>
<tr>
<td>SM78K Series Ver. 2.52 System Simulator</td>
<td>U16768E</td>
</tr>
<tr>
<td>ID78K0-NS Ver. 2.52 Integrated Debugger</td>
<td>U16488E</td>
</tr>
<tr>
<td>ID78K0S-NS Ver. 2.52 Integrated Debugger</td>
<td>U16584E</td>
</tr>
<tr>
<td>ID78K0-QB Ver. 2.81 Integrated Debugger</td>
<td>U16996E</td>
</tr>
<tr>
<td>ID78K0S-QB Ver. 2.81 Integrated Debugger</td>
<td>U17287E</td>
</tr>
<tr>
<td>PM+ Ver. 6.00 Project Manager</td>
<td>U17178E</td>
</tr>
</tbody>
</table>
### Documents related to V850 Series development tools (User's Manual)

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Document No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA850 Ver. 3.00 C Compiler Package</td>
<td></td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td>U17293E</td>
</tr>
<tr>
<td><strong>C Language</strong></td>
<td>U17291E</td>
</tr>
<tr>
<td><strong>Assembly Language</strong></td>
<td>U17292E</td>
</tr>
<tr>
<td><strong>Link Directives</strong></td>
<td>U17294E</td>
</tr>
<tr>
<td>ID850 Ver. 2.50 Integrated Debugger</td>
<td></td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td>U16217E</td>
</tr>
<tr>
<td>ID850NW Ver. 2.51 Integrated Debugger</td>
<td></td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td>U16454E</td>
</tr>
<tr>
<td>ID850NWC Ver. 2.51 Integrated Debugger</td>
<td></td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td>U16525E</td>
</tr>
<tr>
<td>ID850QB Ver. 2.80 Integrated Debugger</td>
<td></td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td>U16973E</td>
</tr>
<tr>
<td>SM+ System Simulator</td>
<td></td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>User Open Interface Specifications</strong></td>
<td>U17246E</td>
</tr>
<tr>
<td>SM850 Ver. 2.50 System Simulator</td>
<td></td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td>U16218E</td>
</tr>
<tr>
<td>SM850 Ver. 2.00 or later System Simulator</td>
<td></td>
</tr>
<tr>
<td><strong>External Part User Open Interface Specifications</strong></td>
<td>U14873E</td>
</tr>
<tr>
<td>RX850 Ver. 3.13 or later Real-Time OS</td>
<td></td>
</tr>
<tr>
<td><strong>Basics</strong></td>
<td>U13430E</td>
</tr>
<tr>
<td><strong>Installation</strong></td>
<td>U13410E</td>
</tr>
<tr>
<td><strong>Technical</strong></td>
<td>U13431E</td>
</tr>
<tr>
<td>RX850 Pro Ver. 3.13 Real-Time OS</td>
<td></td>
</tr>
<tr>
<td><strong>Basics</strong></td>
<td>U13773E</td>
</tr>
<tr>
<td><strong>Installation</strong></td>
<td>U13774E</td>
</tr>
<tr>
<td><strong>Technical</strong></td>
<td>U13772E</td>
</tr>
<tr>
<td>RD850 Ver. 3.01 Task Debugger</td>
<td></td>
</tr>
<tr>
<td><strong>Basics</strong></td>
<td>U13737E</td>
</tr>
<tr>
<td>RD850 Pro Ver. 3.01 Task Debugger</td>
<td></td>
</tr>
<tr>
<td><strong>Basics</strong></td>
<td>U13916E</td>
</tr>
<tr>
<td>AZ850 Ver. 3.20 System Performance Analyzer</td>
<td></td>
</tr>
<tr>
<td><strong>Basics</strong></td>
<td>U14410E</td>
</tr>
<tr>
<td>PG-FP4 Flash Memory Programmer</td>
<td></td>
</tr>
<tr>
<td><strong>Basics</strong></td>
<td>U15260E</td>
</tr>
<tr>
<td>TW850 Ver. 2.00 Performance Analysis Tuning Tool</td>
<td></td>
</tr>
<tr>
<td><strong>Basics</strong></td>
<td>U17421E</td>
</tr>
<tr>
<td>PM+ Ver. 6.00 Project Manager</td>
<td></td>
</tr>
<tr>
<td><strong>Basics</strong></td>
<td>U17178E</td>
</tr>
</tbody>
</table>
CONTENTS

CHAPTER 1  OVERVIEW ... 13
  1. 1 Types of Interface Functions ... 14
  1. 2 Interface Methods ... 15
    1. 2. 1 C-language interface ... 15
    1. 2. 2 Callback function method ... 15
    1. 2. 3 Event-driven method ... 15
  1. 3 Development Environment ... 15

CHAPTER 2  CREATING USER MODEL ... 16
  2. 1 Program Configuration ... 16
  2. 2 Outline of Programming ... 17
  2. 3 Programming Details ... 18
    2. 3. 1 File name ... 18
    2. 3. 2 Include file ... 18
    2. 3. 3 MakeUserModel function ... 18
    2. 3. 4 Callback function ... 18
  2. 4 Example of Program File ... 19
  2. 5 Compilation and Linking ... 20

CHAPTER 3  EMBEDDING USER MODEL ... 21
  3. 1 Configuration File ... 21
  3. 2 Description in Configuration File ... 22
    3. 2. 1 Creation ... 22
    3. 2. 2 Pin connection ... 22
    3. 2. 3 External bus connection ... 23
    3. 2. 4 Other items ... 23
  3. 3 Example of Configuration File Description ... 24

CHAPTER 4  FUNCTION REFERENCE ... 25
  4. 1 List of Supplied Functions ... 25
    4. 1. 1 Details of supplied functions ... 26
      SuoSetInitCallback ... 27
      SuoSetResetCallback ... 28
      SuoGetMainClock ... 29
      SuoCreateTimer ... 30
      SuoGetTimerHandle ... 31
      SuoSetTimer ... 32
      SuoKillTimer ... 33
      SuoSetNotifyTimerCallback ... 34
      SuoCreatePin ... 35
      SuoGetPinHandle ... 36
      SuoOutputDigitalPin ... 37
      SuoOutputAnalogPin ... 38
      SuoOutputHighImpedance ... 39
      SuoSetInputDigitalPinCallback ... 40
      SuoSetInputAnalogPinCallback ... 41
      SuoSetInputHighImpedanceCallback ... 42
      SuoCreateExtbus ... 43
      SuoGetExtbusHandle ... 44
      SuoSetReadExtbusCallback ... 45
      SuoSetWriteExtbusCallback ... 46
      SuoCreateSerialUART ... 47
      SuoCreateSerialCSI ... 48
      SuoGetSerialHandle ... 49
      SuoSetSerialParameterUART ... 50
      SuoSetSerialParameterCSI ... 52
      SuoGetSerialParameterUART ... 55
      SuoGetSerialParameterCSI ... 56
      SuoSendSerialData ... 57
SuoSendSerialDataList ... 58
SuoSendSerialFile ... 59
SuoSetNotifySentSerialCallback ... 60
SuoSetReceiveSerialCallback ... 61
SuoCreateWave ... 62
SuoGetWaveHandle ... 63
SuoSendWaveFile ... 64
SuoSetNotifySentWaveCallback ... 65

4. 2 User-Defined Functions ... 66
   4. 2. 1 Details of user-defined functions ... 66
      MakeUserModel ... 67
      InitFunc ... 68
      ResetFunc ... 69
      NotifyTimerFunc ... 70
      InputDigitalPinFunc ... 71
      InputAnalogPinFunc ... 72
      InputHighImpedanceFunc ... 73
      ReadExtbusFunc ... 74
      WriteExtbusFunc ... 75
      NotifySentSerialFunc ... 76
      ReceiveSerialFunc ... 77
      NotifySentWaveFunc ... 78

4. 3 Error Numbers ... 79

CHAPTER 5 SAMPLE PROGRAM ... 81
   5. 1 Timer ... 82
      5. 1. 1 Overview ... 82
      5. 1. 2 Configuration ... 82
      5. 1. 3 Operation ... 82
      5. 1. 4 Project file ... 82
      5. 1. 5 Details of file ... 83

APPENDIX A INDEX ... 88
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure No</th>
<th>Title, Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Programming Image of User Model ... 13</td>
</tr>
<tr>
<td>2-1</td>
<td>Program Configuration ... 16</td>
</tr>
<tr>
<td>2-2</td>
<td>Template of Program File ... 17</td>
</tr>
<tr>
<td>2-3</td>
<td>Example of Program File ... 19</td>
</tr>
<tr>
<td>2-4</td>
<td>Flow of Compilation and Linking ... 20</td>
</tr>
<tr>
<td>3-1</td>
<td>Example of Description in Configuration File ... 24</td>
</tr>
<tr>
<td>5-1</td>
<td>Timer Model Configuration ... 82</td>
</tr>
<tr>
<td>5-2</td>
<td>Timer Model Operation ... 82</td>
</tr>
</tbody>
</table>
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table No</th>
<th>Title, Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Types of Functions Supplied by User Open Interface ... 14</td>
</tr>
<tr>
<td>3-1</td>
<td>Connection in Sample Configuration File ... 24</td>
</tr>
<tr>
<td>4-1</td>
<td>Supplied Functions ... 25</td>
</tr>
<tr>
<td>4-2</td>
<td>CSI Phase Types (SuosetSerialParameterCSI Function) ... 54</td>
</tr>
<tr>
<td>4-3</td>
<td>User-Defined Functions ... 66</td>
</tr>
<tr>
<td>4-4</td>
<td>Error Numbers ... 79</td>
</tr>
<tr>
<td>5-1</td>
<td>Sample Program ... 81</td>
</tr>
<tr>
<td>5-2</td>
<td>Setting Information of Timer Model ... 82</td>
</tr>
</tbody>
</table>
CHAPTER 1 OVERVIEW

SM+ 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\(^\text{Note}\), 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.

\textbf{Note} Refer to the SM+ Operation User's Manual (to be prepared).

![Programming Image of User Model](image)

Figure 1-1 Programming Image of User Model
CHAPTER 1 OVERVIEW

1. 1 Types of Interface Functions

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

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

SM+ user open interface has the following interface methods.

1.2.1 C-language interface

The SM+ user open interface consists of a C-language API function\textsuperscript{Note} set.

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

\textbf{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+ 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 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+ 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+ 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+ 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+ 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+ 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+ is started.

*2 An initialization notification is reported only once, immediately after SM+ 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 system header file "suoh.h" must be included.

2.3.3 MakeUserModel function
(3) is the MakeUserModel function that is called from the system during configuration of SM+.
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+ 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 "4.1 List of Supplied 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

```c
#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+, 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+.

The configuration file name is C:\NECTools32\bin\smplus.cfgNote.

Note If the installation folder of SM+ 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

\[
\text{UserModel1} = \text{Device}("\text{USEROPEN}", "\text{UserModel1.dll UserOption1}");
\]

The Device function is used to create a user model. "USEROPEN" is a user open interface processing module (system module). 
\text{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 from the folder where the configuration file exists or absolute path as the file path. 
\text{UserOption1} is an option character string for \text{UserModel1.dll}. This character string is passed to the "option" parameter of the \text{MakeUserModel} function as is. 
\text{UserModel1} is a variable that indicates the generated user model. The name can be determined freely by the user.

3. 2. 2 Pin connection

\[
\begin{align*}
\text{wire1} &= \text{Wire}(1); & \text{--- (1)} \\
\text{wire1} &= \text{cpu.Port("PinName1")}; & \text{--- (2)} \\
\text{wire1} &= \text{UserModel1.Port("UserPinName1")}; & \text{--- (3)}
\end{align*}
\]

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. 
\text{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.
\text{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 \text{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 \( \text{MakeUserModel} \) 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>
<thead>
<tr>
<th>Type of Connection</th>
<th>CPU</th>
<th>User Model (SampleModel.DLL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td>&quot;P00/INTP0&quot; (P00 pin)</td>
<td>&quot;P00&quot; (Pin manipulating P00)</td>
</tr>
<tr>
<td>&quot;P30/TXD1&quot; (Serial output pin)</td>
<td>&quot;RXD&quot; (Serial input pin)</td>
<td></td>
</tr>
<tr>
<td>&quot;P31/RXD1&quot; (Serial input pin)</td>
<td>&quot;TXD&quot; (Serial output pin)</td>
<td></td>
</tr>
<tr>
<td>External bus</td>
<td>&quot;EXTBUS&quot; (External bus master)</td>
<td>&quot;EXTBUS1&quot; (External bus slave 1)</td>
</tr>
<tr>
<td>&quot;EXTBUS&quot; (External bus master)</td>
<td>&quot;EXTBUS2&quot; (External bus slave 2)</td>
<td></td>
</tr>
</tbody>
</table>

```
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 the functions supplied by the system as the user open interface (refer to “Table 4-1 Supplied Functions”) and the functions created by the user (refer to “Table 4-3 User-Defined Functions”).

4. 1 List of Supplied Functions

The supplied functions are listed below.

Table 4-1  Supplied Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic interface functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SuoSetInitCallback</td>
<td>Registers initialization callback</td>
<td>Note 2</td>
</tr>
<tr>
<td>SuoSetResetCallback</td>
<td>Registers reset callback</td>
<td>-</td>
</tr>
<tr>
<td>SuoGetMainClock</td>
<td>Acquires the cycle of the main clock for simulation</td>
<td>Note 3</td>
</tr>
<tr>
<td><strong>Time interface functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SuoCreateTimer</td>
<td>Generates timer</td>
<td>Note 1</td>
</tr>
<tr>
<td>SuoGetTimerHandle</td>
<td>Acquires timer handle</td>
<td>-</td>
</tr>
<tr>
<td>SuoSetTimer</td>
<td>Sets timer time</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoKillTimer</td>
<td>Cancels timer time</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoSetNotifyTimerCallback</td>
<td>Registers timer time notification callback</td>
<td>-</td>
</tr>
<tr>
<td><strong>Pin interface function</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SuoCreatePin</td>
<td>Generates pin</td>
<td>Note 1</td>
</tr>
<tr>
<td>SuoGetPinHandle</td>
<td>Acquires pin handle</td>
<td>-</td>
</tr>
<tr>
<td>SuoOutputDigitalPin</td>
<td>Outputs digital pin value</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoOutputAnalogPin</td>
<td>Outputs analog pin value</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoOutputHighImpedance</td>
<td>Outputs high-impedance for the pin</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoSetInputDigitalPinCallback</td>
<td>Registers digital pin value input callback</td>
<td>-</td>
</tr>
<tr>
<td>SuoSetInputAnalogPinCallback</td>
<td>Registers analog pin value input callback</td>
<td>-</td>
</tr>
<tr>
<td>SuoSetInputHighImpedanceCallback</td>
<td>Registers pin high-impedance state report callback</td>
<td>-</td>
</tr>
<tr>
<td><strong>External bus interface functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SuoCreateExtbus</td>
<td>Generates external bus</td>
<td>Note 1</td>
</tr>
<tr>
<td>SuoGetExtbusHandle</td>
<td>Acquires external bus handle</td>
<td>-</td>
</tr>
<tr>
<td>SuoSetReadExtbusCallback</td>
<td>Registers external bus read access callback</td>
<td>-</td>
</tr>
<tr>
<td>SuoSetWriteExtbusCallback</td>
<td>Registers external bus write access callback</td>
<td>-</td>
</tr>
</tbody>
</table>
### Serial interface functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuoCreateSerialUART</td>
<td>Generates serial interface (UART type)</td>
<td>Note 1</td>
</tr>
<tr>
<td>SuoCreateSerialCSI</td>
<td>Generates serial interface (CSI type)</td>
<td>Note 1</td>
</tr>
<tr>
<td>SuoGetSerialHandle</td>
<td>Acquires serial interface handle</td>
<td>-</td>
</tr>
<tr>
<td>SuoSetSerialParameterUART</td>
<td>Sets serial interface parameter (UART type)</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoSetSerialParameterCSI</td>
<td>Sets serial interface parameter (CSI type)</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoGetSerialParameterUART</td>
<td>Acquires serial interface parameter (UART type)</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoGetSerialParameterCSI</td>
<td>Acquires serial interface parameter (CSI type)</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoSendSerialData</td>
<td>Performs serial transmission (1 data)</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoSendSerialDataList</td>
<td>Performs serial transmission (more than one data)</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoSendSerialFile</td>
<td>Performs serial transmission (serial file)</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoSetNotifySentSerialCallback</td>
<td>Registers serial interface transmission end notification callback</td>
<td>-</td>
</tr>
<tr>
<td>SuoSetReceiveSerialCallback</td>
<td>Registers serial interface reception callback</td>
<td>-</td>
</tr>
</tbody>
</table>

### Signal output unit interface functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuoCreateWave</td>
<td>Generates signal output unit</td>
<td>Note 1</td>
</tr>
<tr>
<td>SuoGetWaveHandle</td>
<td>Acquires signal output unit handle</td>
<td>-</td>
</tr>
<tr>
<td>SuoSendWaveFile</td>
<td>Performs transmission via signal output unit (signal data file)</td>
<td>Note 3</td>
</tr>
<tr>
<td>SuoSetNotifySentWaveCallback</td>
<td>Registers signal output unit transmission end notification callback</td>
<td>-</td>
</tr>
</tbody>
</table>

**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

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

**Parameters**

- `func`: Specifies a pointer to the user-defined function that performs initialization processing. (Refer to "InitFunc".)

**Return value**

None

**Explanation**

This function is used to register the user-defined function that performs initialization processing. The function registered by this function is called only once, when SM+ is started.

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

**Example**

```c
void InitFunc(void);

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

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

Reset callback registration

```c
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 is used to register 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**

```c
void ResetFunc(void);

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

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

**Acquisition of main clock cycle for simulation**

```c
int SuoGetMainClock(unsigned long* time);
```

### Parameters

- `time` Specifies the location where the main clock cycle value (unit: pS) is to be stored.

### Return value

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

### Explanation

This function is used to acquire the cycle of the main clock for the simulation environment currently being executed.

### Example

```c
unsigned long time;

func1()
{
    ........
    SuoGetMainClock(&time); /* Get main clock */
}
```
**SuoCreateTimer**

Timer generation

```c
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 this function is successful, SUO_NOERROR is returned.

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

### Explanation

This function is used to generate a timer interface.

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

If this 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.

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

### Example

```c
SuoHandle hTim1;

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

Timer handle acquisition

SuoHandle SuoGetTimerHandle(const char* timerName);

Parameters

\texttt{timerName} \hspace{1cm} \text{Specifies the name of the timer interface.}

Return value

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

Explanation

This function is used to obtain the handle of the specified timer interface.
If this function is successful, the handle of the specified timer interface is returned.
Specify the name specified by the SuoCreateTimer function as \texttt{timerName}.
If a different name is specified, NULL is returned.

Example

\begin{verbatim}
SuoHandle hTim1;
func1()
{
    ..........
    hTim1 = SuoGetTimerHandle("TIM1"); /* Get handle of "TIM1" */
}
\end{verbatim}
**SuoSetTimer**

Timer time setting

```c
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).

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUO_MAINCLK</td>
<td>Main clock cycle units</td>
</tr>
<tr>
<td>SUO_USEC</td>
<td>ms units</td>
</tr>
</tbody>
</table>

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

### Return value

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

### Explanation

This function is used to set 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 `SuoSetNotifyTimerCallback` function, the timer notification function is called in each cycle.

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

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

### Example

```c
Suohandle hTim1;

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

Timer time cancellation

```c
int SuokillTimer(SuoHandle handle);
```

### Parameters

- **handle**: Specifies the handle of the timer interface.

### Return value

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

### Explanation

This function is used to stop 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

```c
SuokHandle hTim1;

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

Timer time notification callback registration

```c
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 this function is successful, SUO_NOERROR is returned.

If this 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 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**

```c
void NotifyTimerFunc(SuoHandle handle)
{
    ........
    SuoSetNotifyTimerCallback(hTim1, NotifyTimerFunc); /* Set notify-timer function */
}

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

### Pin generation

```c
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 this function is successful, SUO_NOERROR is returned.
- If this function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers").

### Explanation

This function is used to generate a pin interface. The generated pin interface is associated with the name specified for `pinName`. Also, the pin specified as `pinName` is generated.

If this 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.

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

### Example

```c
SuoHandle hPinP00;
SuoHandle hPinABC;

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

Pin handle acquisition

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

**Parameters**

- `pinName`  Specifies the name of the pin.

**Return value**

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

**Explanation**

This function is used to obtain the handle of the specified pin interface. If this 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**

```c
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).

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUO_HIGH</td>
<td>HIGH value</td>
</tr>
<tr>
<td>SUO_LOW</td>
<td>LOW value</td>
</tr>
</tbody>
</table>

### Return value

If this function is successful, SUO_NOERROR is returned.

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

### Explanation

This function is used to output a digital value signal to the specified pin interface.

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

### Example

```c
SuoHandle hPinP00;

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

**Analog pin value output**

```c
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 this function is successful, SUO_NOERROR is returned.

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

**Explanation**

This function is used to output 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**

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

High-impedance pin output

```c
int SuoOutputHighImpedance(SuoHandle handle);
```

**Parameters**

`handle` Specifies the handle of the pin interface.

**Return value**

If this function is successful, SUO_NOERROR is returned.

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

**Explanation**

This function is used to output high-impedance for the specified digital/analog pin interface.

**Example**

```c
SuoHandle hPinP00;

func1()
{
    ........
    SuoOutputHighImpedance(hPinP00); /* Output High-impedance */
}
```
**SuoSetInputDigitalPinCallback**

Digital pin value input callback registration

```c
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 this function is successful, SUO_NOERROR is returned.

If this 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**

```c
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 this function is successful, SUO_NOERROR is returned.
- If this 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)
{
    ..........  
}
```
**SuoSetInputHighImpedanceCallback**

Pin high-impedance state report callback registration

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int SuoSetInputHighImpedanceCallback(SuoHandle handle, SuoInputHighImpedanceCallback func);</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

- **handle**: Specifies the handle of the pin interface.
- **func**: Specifies the pointer to a user-defined function that performs processing when all the connected pins enter the high-impedance state. (Refer to "InputHighImpedanceFunc".)

### Return value

If this function is successful, SUO_NOERROR is returned.

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

### Explanation

This function is used to register the user-defined function that performs processing when all the pins connected to digital/analog pins enter the high-impedance state.

If NULL is specified for func, registration is canceled.

### Example

```c
void InputHighImpedanceFunc(SuoHandle handle);
SuoHandle hPinP00;

func1()
{
    ........
    SuoSetInputHighImpedanceCallback(hPinP00,InputHighImpedanceFunc);
    /* Set input-high-impedance function */
}

/* Input-high-impedance function */
void InputHighImpedanceFunc(SuoHandle handle)
{
    ........
}
SuoCreateExtbus

External bus generation

```c
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 this function is successful, SUO_NOERROR is returned.
- If this function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

**Explanation**

This function is used to generate an external bus interface. The generated external bus interface is associated with the name specified for `extbusName`. If this 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.

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

**Example**

```c
SuoHandle hExtbus1;

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

**External bus handle acquisition**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuoGetExtbusHandle</td>
<td>This function is used to obtain the handle of the specified external bus interface.</td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>extbusName</code></td>
<td>Specifies the name of the external bus.</td>
</tr>
</tbody>
</table>

### Return value

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

If this function fails, NULL is returned.

### Explanation

This function is used to obtain the handle of the specified external bus interface. If this 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

```c
SuoHandle hExtbus1;

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

**External bus read access callback registration**

```c
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 this function is successful, SUO_NOERROR is returned.
- If this 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**

```c
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

```c
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 this function is successful, SUO_NOERROR is returned.
- If this 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

```c
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 */
vold WriteExtbusFunc(SuoHandle handle, unsigned long addr, int accessSize, const unsigned char data[])
{
    ........
} 
```
**SuoCreateSerialUART**

Serial interface generation (UART type)

```c
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 this function is successful, SUO_NOERROR is returned.
- If this function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

**Explanation**

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

The generated serial interface is associated with the name specified for `serialName`.

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

If this 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**

```c
SuoHandle hUart1;

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

Serial interface generation (CSI type)

```c
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 this function is successful, SUO_NOERROR is returned.
- If this function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers").

**Explanation**

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

The generated serial interface is associated with the name specified for `serialName`.

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

If this 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 `SuoGetSerialHandle` function.

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

**Example**

```c
SuoHandle hCsi1;

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

**Serial interface handle acquisition**

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

### Parameters

- `serialName` Specifies the name of the serial interface.

### Return value

- If this function is successful, the handle of the specified serial interface is returned.
- If this function fails, NULL is returned.

### Explanation

This function is used to obtain the handle of the specified serial interface.

- If this function is successful, the handle of the specified serial interface is returned.
- Specify the name specified by the `SuoCreateSerialUART` or `SuoCreateSerialCSI` function as `serialName`.
- If a different name is specified, NULL is returned.

### Example

```c
SuoHandle hSerial1;

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

Serial interface parameter setting (UART type)

```c
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 this function is successful, SUO_NOERROR is returned.
- If this function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

**Explanation**

This 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

```c
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) */
    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) */
} SuoSerialParameterUART;

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUO_MSBFIRST</td>
<td>MSB first</td>
</tr>
<tr>
<td>SUO_LSBFIRST</td>
<td>LSB first</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUO_NONEPARITY</td>
<td>No parity</td>
</tr>
<tr>
<td>SUO_ZEROPARITY</td>
<td>0 parity (0 parity during transmission, no parity check during reception)</td>
</tr>
<tr>
<td>SUO_ODDPARITY</td>
<td>Odd parity</td>
</tr>
<tr>
<td>SUO_EVENPARITY</td>
<td>Even parity</td>
</tr>
</tbody>
</table>
SuoSetSerialParameterCSI

Serial interface parameter setting (CSI type)

```c
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 this function is successful, SUO_NOERROR is returned.

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

**Explanation**

This 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

```c
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)

<table>
<thead>
<tr>
<th>Value of Phase</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SUO_PRECEDEDATA</td>
<td></td>
</tr>
<tr>
<td>SUO_REVERSECLOCK</td>
<td></td>
</tr>
<tr>
<td>SUO_PRECEDEDATA</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase</th>
<th>Value</th>
<th>Bit Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SI input Timing</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D6</th>
<th>D7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 this function is successful, SUO_NOERROR is returned.
- If this function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers".)

### Explanation

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

### Example

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

Serial interface parameter acquisition (CSI type)

```c
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 this function is successful, `SUO_NOERROR` is returned.
- If this function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers").

**Explanation**

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

**Example**

```c
SuoHandle hCsi1;

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

Serial data transmission (1 data)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>Specifies the handle of the serial interface.</td>
</tr>
<tr>
<td>data</td>
<td>Specifies the transmit data (1 data).</td>
</tr>
</tbody>
</table>

Return value

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

Explanation

This 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 this function is called for a serial interface that is currently transmitting data, an error occurs.

Example

```c
SuoHandle hSerial1;

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

Serial data transmission (two or more data)

```c
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 this function is successful, SUO_NOERROR is returned.
- If this function fails, an error number is returned. (Refer to "Table 4-4 Error Numbers").

### Explanation

This 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 this function is called for a serial interface that is currently transmitting data, an error occurs.

### Example

```c
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)

```c
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+.

**Return value**

If this function is successful, SUO_NOERROR is returned.

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

**Explanation**

This 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 this function is called for a serial interface that is currently transmitting data, an error occurs.

**Example**

```c
SuoHandle hSerial1;

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

Serial interface transmission end notification callback registration

```c
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 this function is successful, SUO_NOERROR is returned.
- If this 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

```c
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 this function is successful, SUO_NOERROR is returned.
If this 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**

```c
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 this function is successful, SUO_NOERROR is returned.

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

**Explanation**

This 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 this 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.

This 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>waveName</td>
<td>Specifies the name of the signal output unit.</td>
</tr>
</tbody>
</table>

**Return value**

- If this function is successful, the handle of the specified signal output unit interface is returned.
- If this function fails, NULL is returned.

**Explanation**

This function is used to obtain the handle of the specified signal output unit interface. If this function is successful, the handle of the specified signal output unit interface is returned. Specify the name specified by the **SuoCreateWave** function as `waveName`. If a different name is specified, NULL is returned.

**Example**

```c
SuoHandle hWave1;

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

Transmission by signal output unit (signal data file)

```c
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+.

#### Return value

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

#### Explanation

This 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 this 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

```c
SuoHandle hWave1;

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

Signal output unit transmission end notification callback registration

```c
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 this function is successful, SUO_NOERROR is returned.
- If this 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**

```c
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>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakeUserModel</td>
<td>MakeUserModel function</td>
</tr>
<tr>
<td>InitFunc</td>
<td>Initialization callback function</td>
</tr>
<tr>
<td>ResetFunc</td>
<td>Reset callback function</td>
</tr>
<tr>
<td>NotifyTimerFunc</td>
<td>Timer time notification callback function</td>
</tr>
<tr>
<td>InputDigitalPinFunc</td>
<td>Digital pin input value callback function</td>
</tr>
<tr>
<td>InputAnalogPinFunc</td>
<td>Analog pin input value callback function</td>
</tr>
<tr>
<td>InputHighImpedanceFunc</td>
<td>Pin high-impedance state report callback function</td>
</tr>
<tr>
<td>ReadExtbusFunc</td>
<td>External bus read access callback function</td>
</tr>
<tr>
<td>WriteExtbusFunc</td>
<td>External bus write access callback function</td>
</tr>
<tr>
<td>NotifySentSerialFunc</td>
<td>Serial interface transmission end notification callback function</td>
</tr>
<tr>
<td>ReceiveSerialFunc</td>
<td>Serial interface reception callback function</td>
</tr>
<tr>
<td>NotifySentWaveFunc</td>
<td>Serial interface reception callback function</td>
</tr>
</tbody>
</table>

4.2.1 Details of user-defined functions

This section shows how to reference user-defined functions.
MakeUserModel

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

This function must be used to generate the resources to be used with the user model. Any function other than this function cannot generate the resources. In addition, this function must be used to register a callback function as necessary. In particular, an initialization callback function must be registered by this function (because the initialization timing has passed even if a function is registered by a function other than this function).

Example

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

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 */
}
```

SuoUserEntry void MakeUserModel(const char *option);

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

None

This function must be used to generate the resources to be used with the user model. Any function other than this function cannot generate the resources. In addition, this function must be used to register a callback function as necessary. In particular, an initialization callback function must be registered by this function (because the initialization timing has passed even if a function is registered by a function other than this function).

Example

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

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 is used to describe initialization processing.

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

Reset callback function [user-defined function]

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

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

**Parameters**

None

**Return value**

None

**Explanation**

ResetFunc is used to describe the reset processing.

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

Timer time notification callback function [user-defined function]

```c
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 is used to describe 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]

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>handle</code></td>
<td>Pin interface handle.</td>
</tr>
<tr>
<td><code>pinValue</code></td>
<td>Value (digital value) input to the pin (specify any of the following)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUO_HIGH (=1)</td>
<td>HIGH value</td>
</tr>
<tr>
<td>SUO_LOW (=0)</td>
<td>LOW value</td>
</tr>
</tbody>
</table>

**Return value**

None

**Explanation**

InputDigitalPinFunc is used to describe 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]

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>handle</code></td>
<td>Pin interface handle.</td>
</tr>
<tr>
<td><code>pinValue</code></td>
<td>Value (analog value) input to the pin (unit: V (volts))</td>
</tr>
</tbody>
</table>

**Return value**

None

**Explanation**

InputAnalogPinFunc is used to describe the analog pin input processing.

Use the `SuoSetInputAnalogPinCallback` function to register InputAnalogPinFunc as a callback function.
**InputHighImpedanceFunc**  
Pin high-impedance state report callback function [user-defined function]

```c
void InputHighImpedanceFunc (SuoHandle handle);
```

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

**Parameters**

- `handle`  
  Pin interface handle.

**Return value**

None

**Explanation**

InputHighImpedanceFunc is used to describe processing when all the pins connected to digital/analog pins enter the high-impedance state.

Use the `SuoSetInputHighImpedanceCallback` function to register InputHighImpedanceFunc as a callback function.
ReadExtbusFunc

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

```c
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 is used to describe 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]

```c
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 is used to describe 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]

```c
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 is used to describe 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]

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>handle</code></td>
<td>Serial interface handle</td>
</tr>
<tr>
<td><code>data</code></td>
<td>Received serial data</td>
</tr>
<tr>
<td><code>status</code></td>
<td>Receive status (specify any of the following)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal reception</td>
</tr>
<tr>
<td>SUO_PARITYERR</td>
<td>Parity error (if parity bit does not match)</td>
</tr>
<tr>
<td>SUO_FRAMINGERR</td>
<td>Framing error (if stop bit is not detected)</td>
</tr>
</tbody>
</table>

**Return value**

None

**Explanation**

ReceiveSerialFunc is used to describe 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 is used to describe 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 (`suoh.h`). This table lists the error numbers.

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

<table>
<thead>
<tr>
<th>Error No. (Macro)</th>
<th>Meaning (Top: Outline, Bottom: Details)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUO_UNDERSENDING</td>
<td>Already being transmitted.</td>
</tr>
<tr>
<td></td>
<td>New transmission cannot be started because transmission is in progress.</td>
</tr>
</tbody>
</table>
CHAPTER 5 SAMPLE PROGRAM

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

Table 5-1  Sample Program

<table>
<thead>
<tr>
<th>No.</th>
<th>Sample Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Timer</td>
<td>Sample using timer interface</td>
</tr>
</tbody>
</table>
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.

![Timer Model Configuration](image)

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.

![Timer Model Operation](image)

5. 1. 4 Project file
The following table shows the setting information of the Visual C++ project file of the Timer model.

<table>
<thead>
<tr>
<th>Information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project type</td>
<td>Win32 Dynamic-Link Library</td>
</tr>
<tr>
<td>Source file</td>
<td>..\sys\suolink.c, uo_timer.c</td>
</tr>
<tr>
<td>Path of include file</td>
<td>..\sys (Specify the folder storing suo.h.)</td>
</tr>
</tbody>
</table>
5. 1. 5 Details of file

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

(1/3)

```c
#include <windows.h>
#include "suoh.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;
    }
```
/* 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;
        }
    }
/* 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)

```plaintext
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");
```

User's Manual  U17247EJ2V0UM
(3) Source file of target program (lm_timer.c)

```c
/* 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

### B
- Basic interface functions ... 25

### C
- Callback function ... 18
- Callback function method ... 15
- Compilation and linking ... 20
- Configuration file ... 21

### D
- Development environment ... 15
- Dynamic link library (DLL) ... 17

### E
- Error numbers ... 79
- Event-driven method ... 15
- External bus interface functions ... 25

### I
- InitFunc ... 68
- InputAnalogPinFunc ... 72
- InputDigitalPinFunc ... 71
- InputHighImpedanceFunc ... 73
- Interface Functions ... 14
- Interface Methods ... 15

### L
- List of Supplied Functions ... 25

### M
- MakeUserModel ... 67
- MakeUserModel function ... 18

### N
- NotifySentSerialFunc ... 76
- NotifySentWaveFunc ... 78
- NotifyTimerFunc ... 70

### P
- Pin interface function ... 25
- Program configuration ... 16
- Programming ... 17

### R
- ReadExtbusFunc ... 74
- ReceiveSerialFunc ... 77
- ResetFunc ... 69

### S
- Serial interface functions ... 26
- Signal output unit interface functions ... 26
- suo.h ... 20
- SuoCreateExtbus ... 43
- SuoCreatePin ... 35
- SuoCreateSerialCSI ... 48
- SuoCreateSerialUART ... 47
- SuoCreateTimer ... 30
- SuoCreateWave ... 62
- SuoGetExtbusHandle ... 44
- SuoGetMainClock ... 29
- SuoGetPinHandle ... 36
- SuoGetSerialHandle ... 49
- SuoGetSerialParameterCSI ... 56
- SuoGetSerialParameterUART ... 55
- SuoGetTimerHandle ... 31
- SuoGetWaveHandle ... 63
- SuoKillTimer ... 33
- suolink.cpp ... 20
- SuoOutputAnalogPin ... 38
- SuoOutputDigitalPin ... 37
- SuoOutputHighImpedance ... 39
- SuoSdorfSerialData ... 57
- SuoSdorfSerialDataList ... 58
- SuoSdorfSerialFile ... 59
- SuoSdorfWaveFile ... 64
- SuoSduSetInitCallback ... 27
- SuoSduSetInputAnalogPinCallback ... 41
- SuoSduSetInputDigitalPinCallback ... 40
- SuoSduSetInputHighImpedanceCallback ... 42
- SuoSduSetNotifySentSerialCallback ... 60
- SuoSduSetNotifySentWaveCallback ... 65
- SuoSduSetNotifyTimerCallback ... 34
- SuoSduSetReadExtbusCallback ... 45
- SuoSduSetReceiveSerialCallback ... 61
- SuoSduSetResetCallback ... 28
- SuoSduSetSerialParameterCSI ... 52
- SuoSduSetSerialParameterUART ... 50
- SuoSduSetTimer ... 32
- SuoSduSetWriteExtbusCallback ... 46

### T
- Time interface functions ... 25

### U
- User model ... 16
- UserModel.c ... 20
- UserModel.dll ... 20

### W
- WriteExtbusFunc ... 75