

I/O DLL Kit

User's Manual

Simulator Debugger Customizing Kit

User's Manual

Rev.1.00
Nov. 01, 2006

Renesas Technology
www.renesas.com

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, 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 products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (<http://www.renesas.com>)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
 - (1) artificial life support devices or systems
 - (2) surgical implantations
 - (3) healthcare intervention (e.g., excision, administration of medication, etc.)
 - (4) any other purposes that pose a direct threat to human lifeRenesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas 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 applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.

Overview

The I/O DLL kit is provided for the purpose of extending the functions of the Simulator Debuggers. Microsoft Visual C++, a Windows application development environment available on the market, is needed to create I/O DLLs.

This user's manual shows the basic information necessary to use the I/O DLL kit. For details about the language specifications of and the method for using Visual C++, refer to the user's manual included with your product or online help.

Supported Simulator Debuggers

The I/O DLL kit cannot be used in all of the Simulator Debuggers. For the Simulator Debuggers and their versions which can be run in conjunction with the I/O DLL kit, refer to the release notes for the I/O DLL kit in which they are detailed.

Rights to use

The rights to use the I/O DLL kit come under the provisions of the Software License Agreement for the Simulator Debuggers used. Please also be aware that the I/O DLL kit can only be used in developing your product, and cannot be used for any other purpose.

Technical support

Please note that technical support for the I/O DLL kit can be obtained by visiting our homepage (URL: <http://www.renesas.com/en/tools>) at which latest information is available.

Active X, Microsoft, MS-DOS, Visual Basic, Visual C++, Windows and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.

IBM and AT are registered trademarks of International Business Machines Corporation.

Intel and Pentium are registered trademarks of Intel Corporation.

Adobe and Acrobat are registered trademarks of Adobe Systems Incorporated.

All other brand and product names are trademarks, registered trademarks or service marks of their respective holders.

[MEMO]

Contents

1. ABSTRACT	1
2. CONFIGURATION	1
3. METHOD FOR CREATING THE I/O DLL	2
3.1. METHOD FOR USING THE IODLLTEMPLATE PROJECT.....	2
3.2. METHOD FOR CREATING THE NEW I/O DLL.....	3
4. METHOD FOR USING THE I/O DLL.....	5
5. METHOD FOR DEBUGGING THE I/O DLL.....	6
6. FUNCTIONS THAT SEND INFORMATION TO THE I/O DLL	7
7. FUNCTIONS THAT GET INFORMATION FROM SIMXX.EXE	11
8. NOTES.....	17

[MEMO]

1. Abstract

I/O DLL means a Dynamic Link Library (DLL) that operates in conjunction with the simulator engine of the Simulator Debugger.

The Simulator Debugger when properly set up can load an I/O DLL, and can operate it synchronously with the timing at which the program executes one instruction, read/writes to memory or generates an interrupt.

This allows to debug the target program by simulating the operation of the microcomputer's input/output ports or internal peripheral functions. This also makes data input/output and other linked operations with external tools possible.

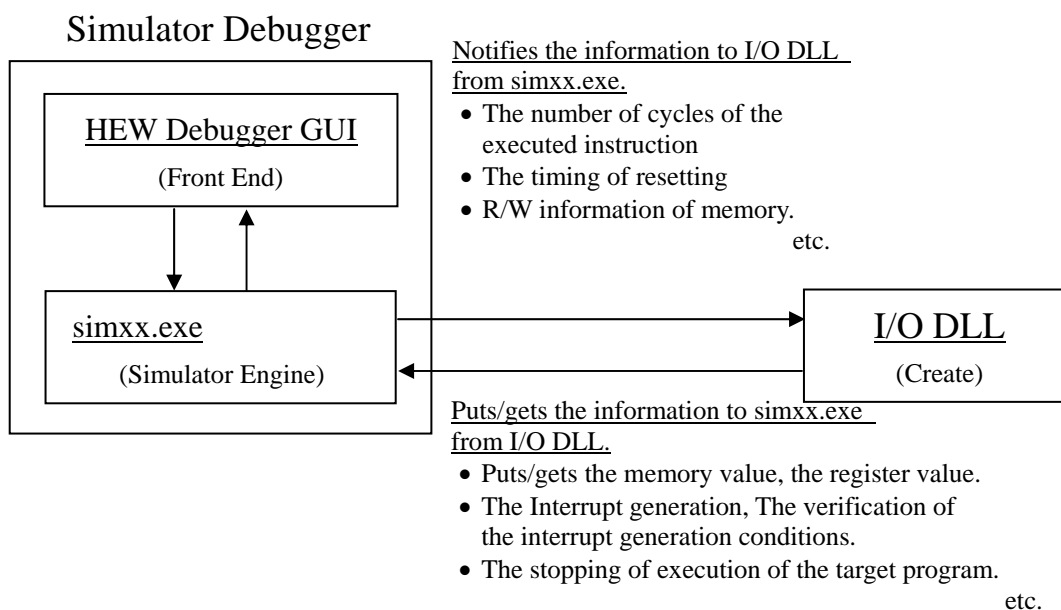
Any desired I/O DLL can be created using C/C++ languages. Microsoft Visual C++, a Windows application development environment available on the market, is needed to create I/O DLLs.

2. Configuration

The I/O DLL accesses the simulator engine `simxx.exe` (xx denotes the model name such as 308 or 30) to obtain information on memory or register values, etc.

In turn, `simxx.exe` notifies the I/O DLL of various information including memory read/write information and the number of cycles in which the program executed each instruction.

The simulator debugger and the I/O DLL are configured in the manner shown below.



3. Method for Creating the I/O DLL

This chapter describes how to create I/O DLLs using Microsoft Visual C++ 6.0 (hereafter abbreviated VC++). For details on how to use VC++, see the VC++ user's manual or help.

An I/O DLL may be created by making use of the I/O DLL sample project "IodllTemplate" which is included with the I/O DLL kit or newly from scratch. The IodllTemplate project is a VC++ template project provided for creating I/O DLLs.

3.1. Method for Using the IodllTemplate project

The following explains how to create I/O DLLs from the IodllTemplate project by using an I/O DLL for Simulator Debugger for M32C Series as an example. To create I/O DLLs for other simulator models, change the model name "308" described here with any desired model name (e.g., "100" or "30").

1. Choose "Open Workspace..." from the File menu of VC++. Open the project file "IodllTemplate.dsp" for the IodllTemplate project which is stored in a directory under the directory in which the I/O DLL package is installed (hereafter referred to as C:\¥Renesas¥Iodll).

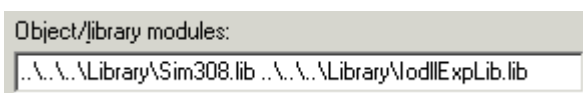
"Samples¥PDxxSIM¥IodllTemplate"

2. Choose "Setting..." from the Project menu of VC++. Check path settings for the I/O DLL library files sim308.lib and IodllExpLib(d).lib which are specified in the "Object/library modules:" column of the category "General" on the Link tab. (The file locations must be specified using absolute or relative paths.) These library files are stored in the "Library" directory of C:\¥Renesas¥Iodll.

- If settings are made for Win32 Debug, specify IodllExpLibd.lib.



- If settings are made for Win32 Release, specify IodllExpLib.lib.



3. Choose "Setting..." from the Project menu of VC++. Change the I/O DLL filename "IodllTemplate.dll" which is specified in the "Output file name:" column of the category "General" on the Link tab to another filename you want (extension ".dll") so that the I/O DLL output destination is the directory in which you installed Simulator Debugger.

Note that the Simulator Debugger is stored in the directory shown below:

HEW install directory

¥Tools¥Renesas¥DebugComp¥Platform¥PDTarget¥PD308SIM

Example: Output file name:

C:\¥Program Files¥Renesas¥HEW¥Tools¥Renesas¥DebugComp¥Platform¥

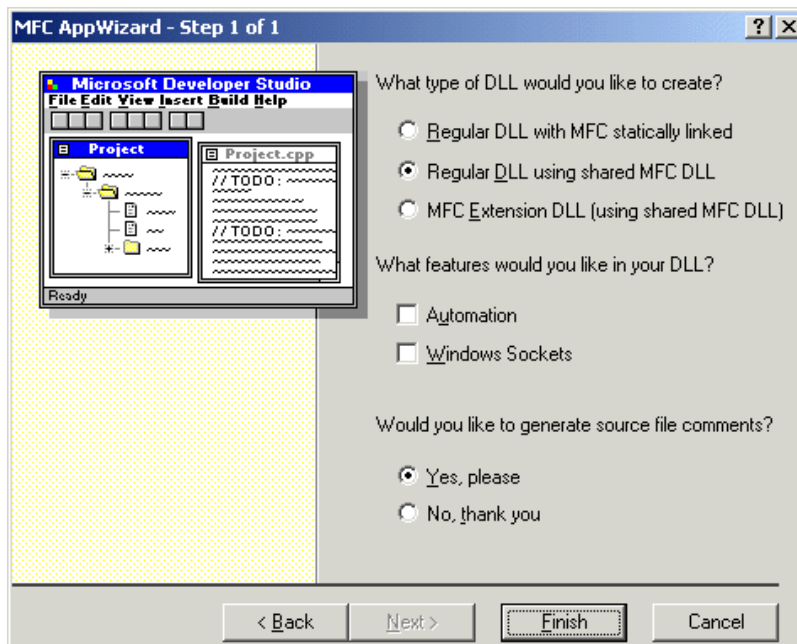
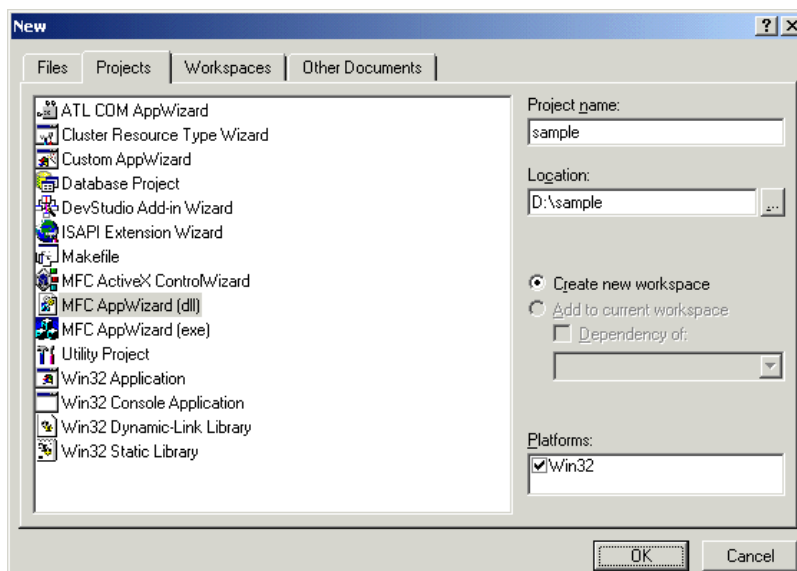
PDTarget¥PD308SIM¥sample.dll

4. Open `infunc\iofunc.cpp`, then write the code for simulation. The `infunc\iofunc.cpp` has implemented in it the functions called from `sim308.exe`. Use a function that accesses `sim308.exe` in the desired function as you write the code for simulation.
5. When you build, an I/O DLL (`sample.dll`) is created.

3.2. Method for Creating the new I/O DLL

To create a new I/O DLL without using the `IodllTemplate` project, follow the procedure described below.

1. Choose “New...” from the File menu of VC++. Then create a new project with settings shown below.
 - Select MFC AppWizard (dll) on the Projects tab.
 - Set any project name and location. (Example: “sample” project)
 - Choose “Create new workspace”.
 - Check Win32 in the “Platforms:”.
 - Click the [OK]->[Close].



2. Choose “Setting...” from the Project menu of VC++ and enter the settings described below.
 - Select the C/C++ tab and add a definition “IODLL_EXPORTS” to the “Preprocessor definitions:” column of the category “General.”

Preprocessor definitions:
_WINDLL,_AFXDLL,_MBCS,_USRDLL,IODLL_EXPORTS

- Select the Link tab and specify the I/O DLL library files sim308.lib and IodllExpLib(d).lib using absolute or relative paths in the “Object/library modules:” column of the category “General.” These library files are stored in the “Library” directory of C:\MTOOL\Iodll.
 - If settings are made for Win32 Debug, specify IodllExpLibd.lib.

Object/library modules:
..\..\Library\Sim308.lib ..\..\Library\IodllExpLibd.lib

- If settings are made for Win32 Release, specify IodllExpLib.lib.

Object/library modules:
..\..\Library\Sim308.lib ..\..\Library\IodllExpLib.lib

3. Choose “Setting...” from the Project menu of VC++. Set the “Output file name:” column of the category “General” on the Link tab so that the I/O DLL output destination is the directory in which you installed Simulator Debugger.

Example: Output file name:

C:\Program Files\Renesas\HEW\Tools\Renesas\DebugComp\Platform\PDTarget\PD308SIM\sample.dll

4. Copy iofunc.cpp and iofunc.h from the “iofunc” directory of the IodllTemplate project into any directory of the new project you’ve created, and add them to the project following the procedure described below.
 - Choose “Add To Project” – “Files...” from the Project menu of VC++, select iofunc.cpp from the ensuing list, and click OK.
5. Open iofunc.h and change the main header file specification for the application included in it to the new main header filename you’ve created.

Example: #include ".\IodllTemplate.h" -> “#include “.sample.h”
6. Open iofunc.cpp, then write the code for simulation.
7. When you build, an I/O DLL (sample.dll) is created.

4. Method for Using the I/O DLL

This chapter describes how to use an I/O DLL in the Simulator Debugger after loading it.

Before you can use an I/O DLL, you must first register it to simxx.exe.

If simxx.exe has an I/O DLL registered in it when you start, it loads the registered I/O DLL before starting simulation.

The following explains how to use an I/O DLL after registering it to the Simulator Debugger by using Simulator Debugger for M32C Series as an example. To use I/O DLLs for other simulator models, change the model name "308" described here with any desired model name (e.g., "100" or "30").

1. Copy the I/O DLL file (.dll file) you want to use into the directory in which you installed Simulator Debugger.

Note that the Simulator Debugger is stored in the directory shown below:

HEW install directory

¥Tools¥Renesas¥DebugComp¥Platform¥PDTarget¥PD308SIM

2. Register the I/O DLL to simxx.exe. To do this, write the I/O DLL filename in sim308.exe's environment setup file "sim308.ini."

The sim308.ini file exists in the directory in which you installed Simulator Debugger. However, this file is nonexistent if you have never started Simulator Debugger since you installed it. In that case, this file needs to be newly created using an editor.

3. In the sim308.ini file, create a [DLLNAME] section and write an I/O DLL filename after "IODLL=," with the extension ".dll" removed as shown below.

Example: When an I/O DLL file name is "Sample.dll".

[DLLNAME]

IODLL=Sample

4. When you start Simulator Debugger, the I/O DLL is loaded.

If you do not use the I/O DLL, delete the description of the [DLLNAME] section that you created in the sim308.ini file before starting Simulator Debugger.

[DLLNAME] <- Delete

IODLL=Sample <- Delete

5. Method for Debugging the I/O DLL

This chapter describes how to debug the I/O DLL you've created.

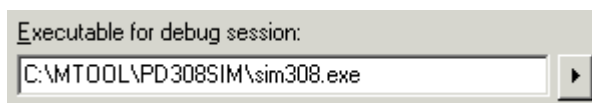
To debug the I/O DLL in VC++, you need to change settings of the I/O DLL project. The following shows the contents of changes you need to make by using the case of debugging an I/O DLL for Simulator Debugger for M32C Series as an example. To debug I/O DLLs for other simulator models, change the model name "308" described here with any desired model name (e.g., "100" or "30").

1. Following the procedure described in the preceding section, register the I/O DLL to make it usable.
2. Choose "Setting..." from the Project menu of VC++. Set the "Output file name:" column of the category "General" on the Link tab so that the I/O DLL output destination is the directory in which you installed Simulator Debugger.

Example: Output file name:

```
C:\Program Files\Renesas\HEW\Tools\Renesas\DebugComp\Platform\
PDTarget\PD308SIM\sample.dll
```

3. Choose "Setting..." from the Project menu of VC++. Set up the "Executable for debug session:" column on the Debug tab so that the simulator engine executable file "sim308.exe" stored in the directory in which you installed Simulator Debugger becomes the debug-time executable file.



Example: Executable for debug session:

```
C:\Program Files\Renesas\HEW\Tools\Renesas\DebugComp\Platform\
PDTarget\PD308SIM\sim308.exe
```

4. After you finished setting up, choose "Start Debug" -- "Go" from the Build menu of VC++ to start debugging. The message shown below will appear.

"sim308.exe does not contain debugging information. Press OK to continue."

Because the I/O DLL can be debugged, click OK to continue.

5. To debug the I/O DLL, start Simulator Debugger after the simulator engine sim308.exe has started up. (After sim308.exe started up, it is registered (displayed) in the Windows system tray. So you can see that sim308.exe is up and running before you start Simulator Debugger.)

For details the debugging function of VC++, see the VC++ user's manual or help.

6. Functions that Send Information to the I/O DLL

This chapter describes specifications of the functions that send information on the simulator engine side to the I/O DLL.

These functions are implemented in the I/O DLL project's source file "iofunc\iofunc.cpp." The simulator engine simxx.exe calls these functions as it sends information to the I/O DLL. Make sure codes for simulation are written in these functions.

Function name	Abstract
NotifyStepCycle	Notifies the number of execution cycles each time an instruction is executed.
NotifyPreExecutionPC	Notifies the current PC value immediately before executing an instruction.
NotifyReset	Notifies that the target program was reset.
NotifyStart	Notifies that the simulator engine was started.
NotifyEnd	Notifies that the simulator engine was terminated.
NotifyInterrupt	Notifies the vector number (vector address) for an interrupt when that interrupt was generated.
NotifyPreReadMemory	Notifies the read address and data length immediately before reading from memory when a memory read from the target program occurred.
NotifyPostWriteMemory	Notifies the write address, data length and data value after data was written from the target program to memory.

Below shows a specification of the function.

- **Notifies the number of execution cycles an instruction is executed**

Function name: void NotifyStepCycle(int cycle)

Parameter: int cycle The number of the executed cycles.

Return value: None

Functions: Notifies the number of execution cycles each time an instruction is executed.

- **Notifies the current PC value before executing an instruction**

Function name: void NotifyPreExecutionPC(unsigned long address)

Parameter: unsigned long address PC value before executing

Return value: None

Functions: Notifies the current PC value immediately before executing an instruction.

- **Notifies the reset**

Function name: void NotifyReset(void)

Parameter: None

Return value: None

Functions: Notifies that the target program was reset.

- **Notifies the start of simulator engine**

Function name: void NotifyStart(void)

Parameter: None

Return value: None

Functions: Notifies that the simulator engine was started.

- **Notifies the terminate of simulator engine**

Function name: void NotifyEnd(void)

Parameter: None

Return value: None

Functions: Notifies that the simulator engine was terminated.

- **Notifies the generate of interrupt**

Function name: void NotifyInterrupt(unsigned long vec)

Parameter: unsigned long vec Vector number.

Return value: None

Functions: Notifies the vector number for an interrupt when that interrupt was generated.

- **Notifies immediately before reading from memory**

Function name: void NotifyPreReadMemory(unsigned long address, int length)

Parameter:	unsigned long address	Memory address.
	int length	Data length of memory data.
	1	1 byte
	2	2 byte
	3	3 byte
	4	4 byte

Return value: None

Functions: Notifies the read address and data length immediately before reading from memory when a memory read from the target program occurred.

- **Notifies after data was written from memory**

Function name: void NotifyPostWriteMemory(unsigned long address, int length, unsigned long data)

Parameter: unsigned long address Memory address.

 int length Data length of memory data.

 1 1 byte

 2 2 byte

 3 3 byte

 4 4 byte

 unsigned long data Memory data value.

Return value: None

Functions: Notifies the write address, data length and data value after data was written from the target program to memory.

The following shows an example of how to write the above functions. The shaded sections in this example are the template part of the above functions that are implemented in the `iofunc.cpp` file.

```
void NotifyStepCycle(int cycle)
{
    unsigned long tabsrData, ta0Data, ta0icData;

    if (sCountFlag == FALSE) { // Check on count beginning flag.
        return;
    }
    RequestGetMemory(TABSR, 1, &tabsrData);
    if ((tabsrData & 0x01) == 0x01) { // Check on count beginning flag.
        sCountCycle += cycle;
        RequestGetMemory(TA0, 2, &ta0Data);
        if (sCountCycle >= ta0Data + 1) { // Count down of counter.
            RequestGetMemory(TA0IC, 1, &ta0icData);
            RequestInterrupt(TA0INT, ta0icData & 0x7); // Generation of timer A0 interrupt.
            sCountCycle = 0;
        }
    }
    return;
}

void NotifyPreExecutionPC(unsigned long address)
{
    return;
}

:
:
:

void NotifyPostWriteMemory(unsigned long address, int length)
{
    if (address == TABSR) {
        if ((data & 0x01) == 0x01) { // Check on count beginning flag.
            sCountFlag = TRUE;
        }
    }
    return;
}
```

7. Functions that Get Information from simxx.exe

This chapter describes specifications of the functions that get information on the simulator engine side from the I/O DLL.

These functions are implemented on the simxx.exe side. The I/O DLL calls these functions to get information from the simulator engine. Use these functions to write codes for simulation within the functions implemented in the I/O DLL project's source file "iofunc\iofunc.cpp."

Function name	Abstract
RequestGetMemory	Gets memory data from the specified address.
RequestPutMemory	Sets memory data at the specified address.
RequestGetRegister	Gets a value from the specified register.
RequestPutRegister	Sets a value in the specified register.
RequestInterrupt	Generates a specified interrupt.
RequestInterruptStatus	Gets the status of generated interrupts.
RequestTotalCycle	Inspects a total number of current execution cycles.
RequestInstructionNum	Inspects a total number of currently executed instructions.
RequestStop	Causes the target program to stop running.
RequestErrorNum	Gets an error number when an error occurred in the immediately preceding function that was executed.

Below shows a specification of the function.

- **Gets memory data**

Function name: int RequestGetMemory(unsigned long address, int length, unsigned long * data)

Parameter: unsigned long address Memory address.
 int length Data length of memory data.

1	1 byte
2	2 byte
3	3 byte
4	4 byte

unsigned long * data Memory data storage area.

Return value: int status

TRUE Succeeded.
 FALSE Error.

Functions: Gets memory data from the specified address.

The read access information performed by this function is not reflected in the virtual port input and I/O script facilities.

- **Gets register value**

Function name: `int RequestGetRegister(int regNo, unsigned long * regValue)`

Parameter: `int regNo` Register number.

For register numbers, see the header file “`iofunc\iofunc.h`” that is included in the I/O DLL sample program. Register numbers are defined in this file.

Example: Definition for PD308SIM.

regNo	Register
REG_Rx_F	Bank0 Rx registers. (x = 0 to 3)
REG_RxH_F	High-order 8 bits of the Bank0 Rx register. (x = 0 to 1)
REG_RxL_F	Low-order 8 bits of the Bank0 Rx register. (x = 0 to 1)
REG_Ax_F	Bank0 Ax registers. (x = 0 to 1)
REG_FB_F	Bank0 FB register.
REG_SB_F	Bank0 SB register.
REG_Rx_R	Bank1 Rx registers. (x = 0 to 3)
REG_RxH_R	High-order 8 bits of the Bank1 Rx register. (x = 0 to 1)
REG_RxL_R	Low-order 8 bits of the Bank1 Rx register. (x = 0 to 1)
REG_Ax_R	Bank1 Ax register. (x = 0 to 1)
REG_FB_R	Bank1 FB register.
REG_SB_R	Bank1 SB register.
REG_Rx	Rx register indicated by the B flag. (x = 0 to 3)
REG_RxH	High-order 8 bits of the Rx register indicated by the B flag. (x = 0 to 1)
REG_RxL	Low-order 8 bits of the Rx register indicated by the B flag. (x = 0 to 1)
REG_Ax	Ax register indicated by the B flag. (x = 0 to 1)
REG_FB	FB register indicated by the B flag.
REG_SB	SB register indicated by the B flag.
REG_USP	USP register.
REG_ISP	ISP register.
REG_FLG	FLG register.
REG_PC	Program counter.
REG_INTB	INTB register.
REG_SVF	SVF register.
REG_SVP	SVP register.
REG_VCT	VCT register.
REG_DMDx	DMDx registers. (x = 0 to 1)
REG_DCTx	DCTx registers. (x = 0 to 1)
REG_DRCx	DRCx registers. (x = 0 to 1)
REG_DMAx	DMAx registers. (x = 0 to 1)
REG_DSAx	DSAx registers. (x = 0 to 1)
REG_DRAx	DRAx registers. (x = 0 to 1)

Description example: To get the R0 register value in bank 0

```
RequestGetRegister( REG_R0_F, &regValue );
```

`unsigned long * regValue` Register value storage area.

Return value: int status

TRUE Succeeded.

FALSE Error.

Functions: Gets a value from the specified register.

● **Sets a register value**

Function name: int RequestPutRegister(int regNo, unsigned long regValue)

Parameter: int regNo Register number.

unsigned long regValue Register value.

Return value: int status

TRUE Succeeded.

FALSE Error.

Functions: Sets a value in the specified register.

● **Generates a interrupt**

Function name: int RequestInterrupt(unsigned long vec, int ipl)

Parameter: unsigned long vec Vector number.

int ipl Priority.

Return value: int status

TRUE Succeeded.

FALSE Error.

Functions: Generates a specified interrupt.

● **Gets the status of generated interrupts**

Function name: int RequestInterruptStatus(unsigned long * vec)

Parameter: unsigned long * vec Vector number.

Return value: int happen Generation status of interrupt.

TRUE Generated. Vector number are stored in vec.

FALSE Un-generated. Vec is indeterminate.

Functions: Gets the status of generated interrupts.

- **Inspects a total number of execution cycles**

Function name: void RequestTotalCycle(unsigned long * cycle)

Parameter: unsigned long * cycle A total number of execution cycles.

Return value: None

Functions: Inspects a total number of current execution cycles.

- **Inspects a total number of executed instructions.**

Function name: void RequestInstructionNum(unsigned long * inst)

Parameter: unsigned long * inst A total number of currently executed instructions.

Return value: None

Functions: Inspects a total number of currently executed instructions.

- **Causes the target program to stop running**

Function name: void RequestStop(void)

Parameter: None

Return value: None

Functions: Causes the target program to stop running.

● Gets error information

Function name: int RequestErrorNum(void)

Parameter: None

Return value: int errNum Error number.

errNom	Contents
000	No error.
001	Address value is out of range.
002	Can't read/write, because there are no memory at that area.
003	Can't get enough memory.
004	Data size is out of range.
005	Can't access a specified address.
100	Description of register is illegal.
101	Data value is illegal.
200	Specified vector out of range.
201	Specified level of priority out of range.
202	Can't get enough memory.

Functions: Gets an error number when an error occurred in the immediately preceding function that was executed.

This function allows to get information on what error occurred when one of the following functions was called and its returned value was false.

- RequestGetMemory function
- RequestPutMemory function
- RequestGetRegister function
- RequestPutRegister function
- RequestInterrupt function

Use example:

```
unsigned long data;
char str[5];
if ( RequestGetMemory( 0x800, 4, data ) == FALSE ) {
    sprintf( str, "%d", RequestErrorNum() );
    MessageBox( NULL, str, "Error number", MB_OK );
    // Display error number to message box.
}
```

The following shows an example of how to write the above functions. The shaded sections in this example are the template part of the above functions that are implemented in the iofunc.cpp file.

```
void NotifyPostWriteMemory(unsigned long address, int length, unsigned long data)
{
    if (address == 0x3e0) {
        RequestGetRegister(REG_PC, &val);           // Get PC Value.
        RequestPutMemory(0x800, 4, val);           // Store PC value in 0x800 address.
    } else if (address == 0x3e1) {
        RequestPutRegister(REG_R0_F, data);        // Store value in bank0 R0 register.
        RequestInterrupt(21, 7);                   // Generation of timer A0 interrupt.
    }
    return;
}
```

8. Notes

1. Changes of values input/output to or from memory using the I/O DLL cannot be referenced using the GUI output, virtual port input/output or I/O script facilities of Simulator Debugger.
2. Only one I/O DLL can be specified in Simulator Debugger. You cannot specify multiple I/O DLLs.

[MEMO]

I/O DLL Kit
User's Manual

Publication Date: Nov. 01, 2006 Rev.1.00

Published by: Sales Strategic Planning Div.
Renesas Technology Corp.

Edited by: Microcomputer Tool Development Department
Renesas Solutions Corp.

I/O DLL Kit
User's Manual

