User's Manual

PDxxSIM I/O DLL Kit

User's Manual

Renesas Electronics
www.renesas.com

Rev.1.00 2003.05

Keep safety first in your circuit designs!

Notes regarding these materials

For inquiries about the contents of this document or product, fill in the text file the installer generates in the following directory and email to your local distributor.

¥SUPPORT¥Product-name¥SUPPORT.TXT

Renesas Tools Homepage　http://www.renesas.com/en/tools

Preface

The PDxxSIM I/O DLL kit is provided for the purpose of extending the functions of the M3T-PDxxSIM 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 PDxxSIM 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 PDxxSIM I/O DLL kit cannot be used in all of the M3T-PDxxSIM simulator debuggers. For the simulator debuggers and their versions which can be run in conjunction with the PDxxSIM I/O DLL kit, refer to the release notes for the PDxxSIM I/O DLL kit in which they are detailed.

Rights to use

The rights to use the PDxxSIM I/O DLL kit come under the provisions of the Software License Agreement for the M3T-PDxxSIM simulator debuggers used. Please also be aware that the PDxxSIM 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 PDxxSIM I/O DLL kit can be obtained by visiting our homepage (URL: http://www.renesas.com/ en/tools /) at which latest information is available.

[MEMO]

Contents

[MEMO]

[MEMO]

1. Abstract

I/O DLL means a Dynamic Link Library (DLL) that operates in conjunction with the simulator engine of the Simulator Debugger PDxxSIM (xx denotes the model name such as 308 or 30).

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.

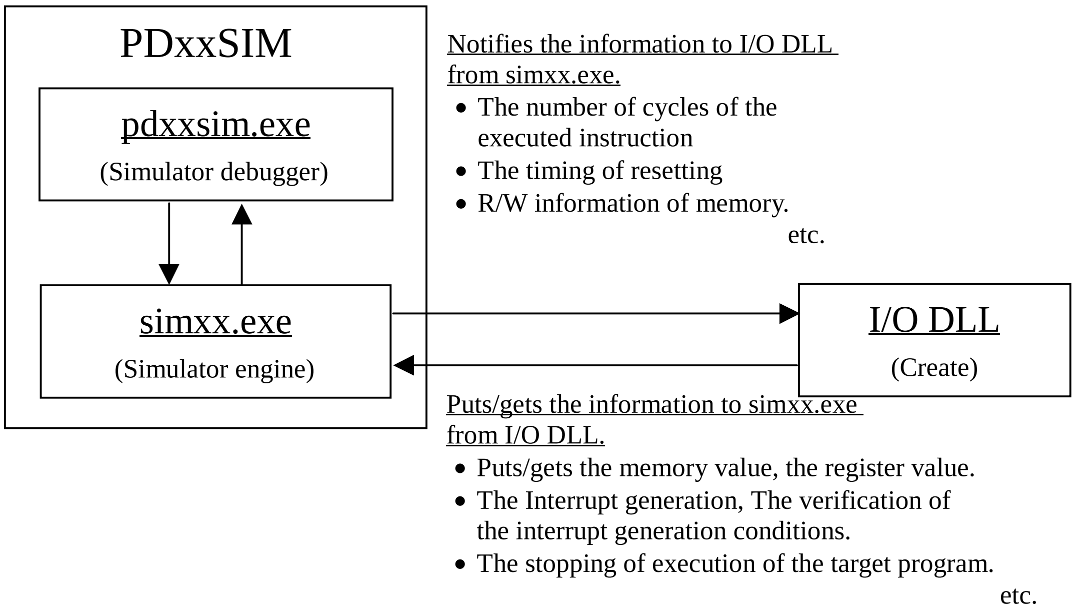
Any desired I/O DLL can be created using C/C++ languages. Microsoft Visual C++, a Windows application development environment available on the market, is needed to create I/O DLLs.

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.


PDxxSIM
pdxxsim.exe
(Simulator debugger)
simxx.exe
(Simulator engine)
I/O DLL
(Create)
Notifies the information to I/O DLL from simxx.exe.
• The number of cycles of the executed instruction
• The timing of resetting
• R/W information of memory.
etc.
Puts/gets the information to simxx.exe from I/O DLL.
• Puts/gets the memory value, the register value.
• The Interrupt generation, The verification of the interrupt generation conditions.
• The stopping of execution of the target program.
etc.

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

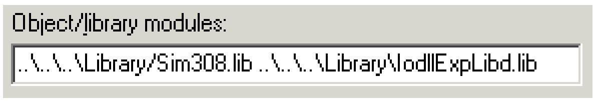
The following explains how to create I/O DLLs from the IodllTemplate project by using an I/O DLL for PD308SIM as an example. To create I/O DLLs for other simulator models, change the model name "308" described here with any desired model name (e.g., "30" or "32R").

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:¥MTOOL¥Iodll).

   "Samples¥Pdxxsim¥IodllTemplate"

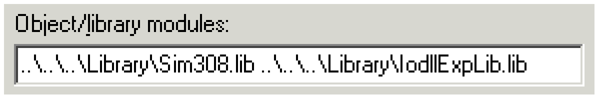
2. Choose "Setting…" from the Project menu of VC++. Check path settings for the I/O DLL library files sim308.lib and IodllExpLib(d).lib which are specified in the "Object/library modules:" column of the category "General" on the Link tab. (The file locations must be specified using absolute or relative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

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

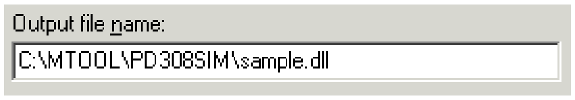
   Object/library modules:
   ..\..\..\Library/Sim308.lib ..\..\..\Library\IodllExpLibd.lib

   - 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++. 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 PD308SIM (hereafter referred to as C:¥MTOOL¥PD308SIM).

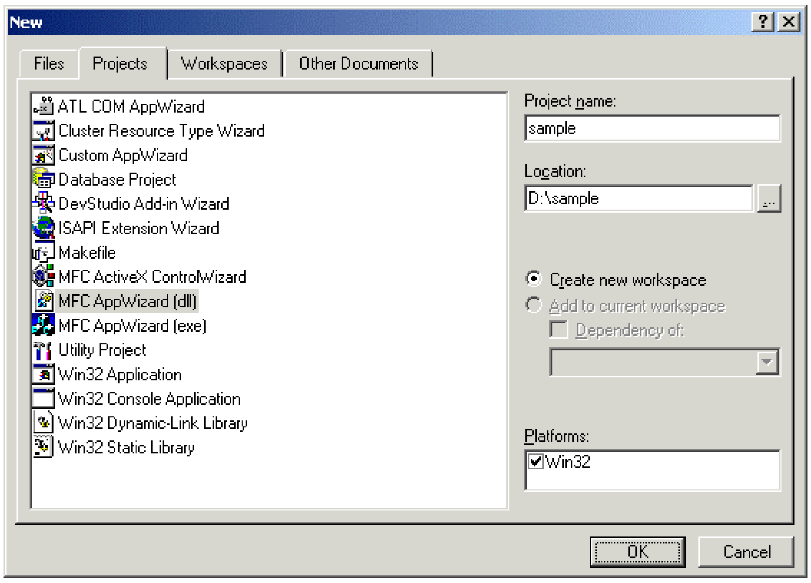
   Output file name:
   C:\MTOOL\PD308SIM\sample.dll

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 under the directory C:¥MTOOL¥PD308SIM.

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].



New

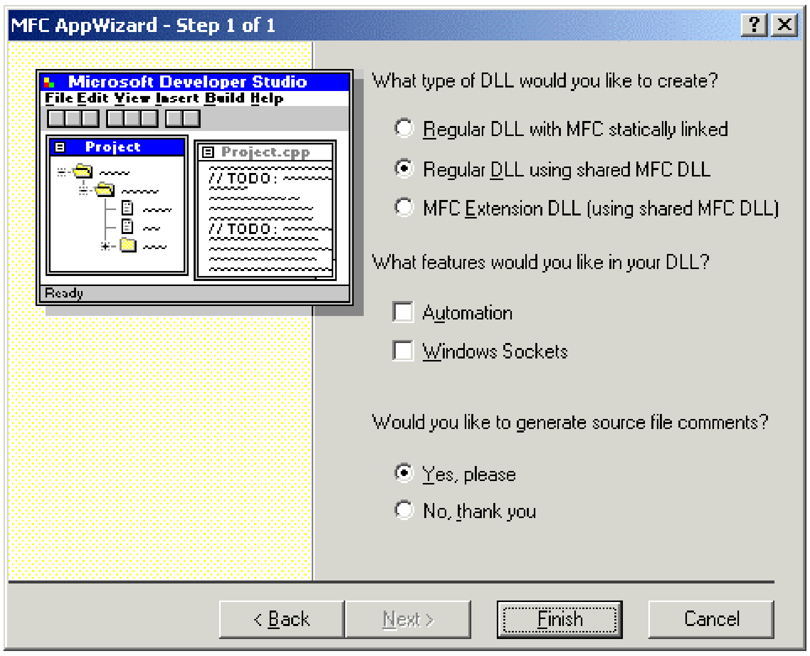
Projects tab: MFC AppWizard (dll)

Project name: sample

Location: D:\sample

[✓ Create new workspace]

Platforms: [✓] Win32



MFC AppWizard - Step 1 of 1

What type of DLL would you like to create?

Regular DLL with MFC statically linked [✓ Regular DLL using shared MFC DLL] MFC Extension DLL (using shared MFC DLL)

What features would you like in your DLL?

[ ] Automation

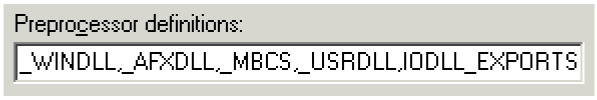
[ ] Windows Sockets

Would you like to generate source file comments?

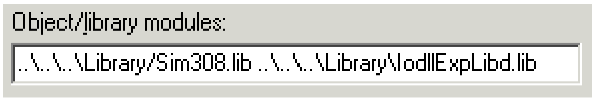
[✓ Yes, please] No, thank you

2. Choose “Setting…” from the Project menu of VC++ and enter the settings described below.

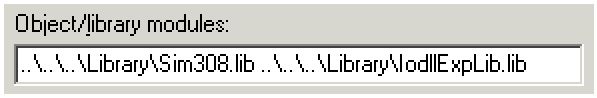
- Select the C/C++ tab and add a definition “IODLL_EXPORTS” to the “Preprocessor definitions:” column of the category “General.”


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

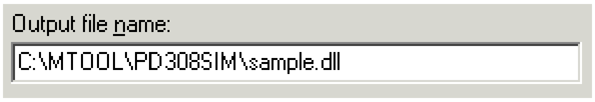
- Select the Link tab and specify the I/O DLL library files sim308.lib and IodllExpLib(d).lib using absolute or relative paths in the “Object/library modules:” column of the category “General.” These library files are stored in the “Library” directory of C:¥MTOOL¥Iodll.
  - If settings are made for Win32 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 C:¥MTOOL¥PD308SIM.


Output file name:
C:\MTOOL\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 infunc.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 under the directory C:¥MTOOL¥PD308SIM.

4. Method for Using the I/O DLL

This chapter describes how to use an I/O DLL in the PDxxSIM 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 PDxxSIM by using PD308SIM 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., “30” or “32R”).

1. Copy the I/O DLL file (.dll file) you want to use into C:¥MTOOL¥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 C:¥MTOOL¥PD308SIM. However, this file is nonexistent if you have never started PD308SIM 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 PD308SIM, 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 PD308SIM.

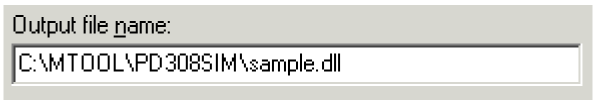
   ```
   [DLLNAME]        <- Delete
   IODLL=Sample     <- Delete
   ```

5. Method for Debugging the I/O DLL

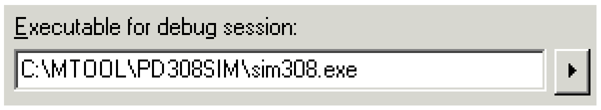
This chapter describes how to debug the I/O DLL you've created.

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 PD3008SIM 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., "30" or "32R").

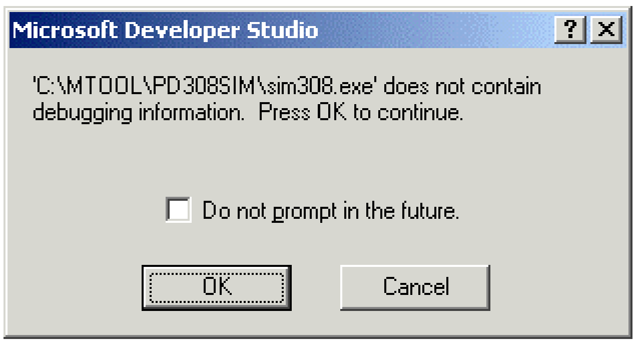
1. Following the procedure described in the preceding section, register the I/O DLL to make it usable.
2. Choose "Setting…" from the Project menu of VC++. Set the "Output file name:" column of the category "General" on the Link tab so that the I/O DLL output destination is the directory C:¥MTOOL¥PD308SIM.


Output file name:
C:\MTOOL\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 C:¥MTOOL¥PD308SIM becomes the debug-time executable file.


Executable for debug session:
C:\MTOOL\PD308SIM\sim308.exe

4. After you finished setting up, choose "Start Debug" -- "Go" from the Build menu of VC++ to start debugging. The dialog box shown below will appear.
   Because the I/O DLL can be debugged, click OK to continue.


Microsoft Developer Studio
'C:\MTOOL\PD308SIM\sim308.exe' does not contain debugging information. Press OK to continue.

5. To debug the I/O DLL, start PD308SIM 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 PD308SIM.)
   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. (Vector address)
		PD308SIM, PD30SIM --- Vector number.
		PD32RSIM --- EIT vector entry. (Only 0x00000080)
Return value:	None	
Functions:	Notifies the vector number (vector address) 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 longtabsrData, 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(unsigend long address, int length, unsigend 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.

- **Sets memory data**

Function name: int RequestPutMemory(unsigend long address, int length, unsigend 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.
Return value:	int status	
	TRUE	Succeeded.
	FALSE	Error.

Functions: Sets memory data at the specified address.
The write access information performed by this function is not reflected in the GUI output, virtual port output 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. (Vector address)
		PD308SIM, PD30SIM --- Vector number.
		PD32RSIM --- EIT vector entry. (Only 0x00000080)
	int ipl	Priority.
		For PD32RSIM, specify 0 because there is no interrupt 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. (Vector address)
		PD308SIM, PD30SIM --- Vector number.
		PD32RSIM --- EIT vector entry. (Only 0x00000080)
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. Specifying Target Program Symbols in the I/O DLL

This chapter describes how to specify target program symbols in the I/O DLL.

For the symbols defined in the target program to be specified in the I/O DLL, it is necessary that the symbol information be loaded into the I/O DLL using a symbol window which is an extension window of PDxxSIM.

The following explains the procedure for specifying target program symbols in the I/O DLL by using PD308SIM 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., "30" or "32R").

1. Register a symbol window to PD308SIM.
2. In the symbol window, output symbol information to a file.
3. Load the output file into the I/O DLL to register the symbol information.
4. Specify a symbol in the I/O DLL.

Each step is detailed below.

1. Register a symbol window to PD308SIM.

   Follow the procedure described below to register a symbol window to PD308SIM.

(1) Copy the symbol window DLL file "SymbolWindow.dll" from the SymbolWindow¥Pd308sim directory of C:¥MTOOL¥Iodll into C:¥MTOOL¥PD308SIM.

(2) Open the PD308SIMDLL.DEF file existing in C:¥MTOOL¥PD308SIM by using an editor and increment the "WindowCount" counts for the [GENERAL] section.

```
[GENERAL]
ProductName=PD308SIM
WindowCount=9   ->   WindowCount=10
```

(3) Append the following contents to PD308SIMDLL.DEF.

Contents:

```
[Window9]
Title=Symbol Window
ID=50000
Module=SymbolWindow
Initialize=InitializeMDIChildFrame
Create=CreateMDIChildFrame
IsEnableToOpen=IsEnableToOpenMDIChildFrame
GetOptionMenuID=GetOptionMenuID
Append=1
```

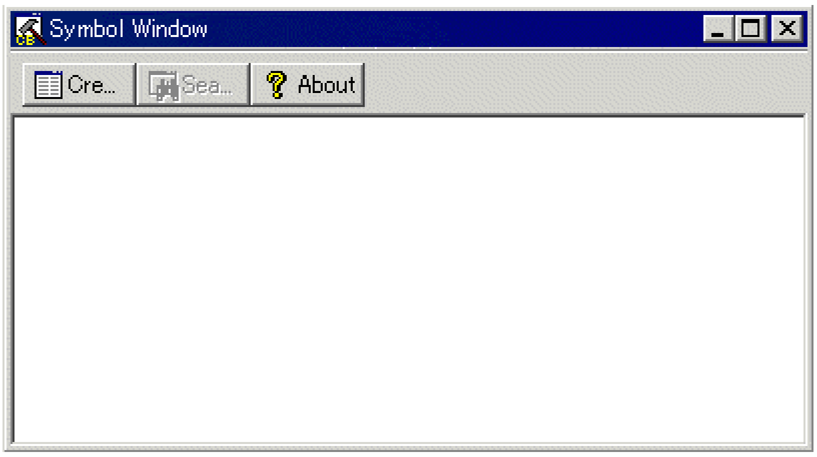
Make sure the number for the section name of the added section "[Window9]" that you set follows those of other windows.

To remove the registered symbol window, restore the contents of PD308SIMDLL.DEF that you modified.

2. In the symbol window, output symbol information to a file.

   Below shows a process.

(1) Start PD308SIM and when it has started up, choose [Add-In Windows] -> [Symbol Window] from the Optional Windows menu to open the symbol window.

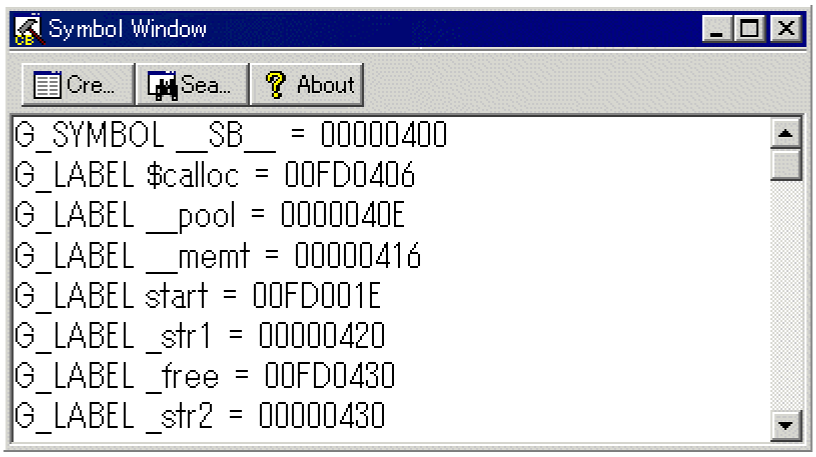


(2) Download the target program into PD308SIM. After downloading, click the “Create Symbol List” button in the toolbar of the symbol window and select the file to which to output the symbol information.

Note:

Of the output symbol information, the information on local symbols is that of module files whose scopes are set in PD308SIM. (The term “scope” refers to the effective range of local symbols/ local labels.) To output information on the local symbols/ local labels included in other module files, change scope settings in PD308SIM. For details, see the description of the *scope* command in “PD308SIM Help,” the online help for PD308SIM.

(3) When symbol information is output to a file, the symbol information is displayed in the symbol window.



```
G_SYMBOL __SB__ = 00000400
G_LABEL $calloc = 00FD0406
G_LABEL __pool = 0000040E
G_LABEL __memt = 00000416
G_LABEL start = 00FD001E
G_LABEL _str1 = 00000420
G_LABEL _free = 00FD0430
G_LABEL _str2 = 00000430
```

- The following are displayed in the contents.

  Symbol type Symbol name = Symbol address

  Symbol types are displayed in order of G_SYMBOL (global symbols), L_SYMBOL (local symbols), G_LABEL (global labels) and L_LABEL (local labels). No symbol types are displayed unless they exist in the target program.
- To search for symbols, click the “Search Symbol” button in the toolbar and enter the symbol name you want to search.
- If the target program is downloaded again, the symbol information being displayed in the symbol window is updated. In this case, symbol information is written over the file to which symbol information was previously output. To change symbol information output files, click the “Create Symbol List” button and specify a symbol file again.

  Note:

  Once a symbol information output file is selected (i.e., symbol information is displayed in the symbol window), the symbol information is automatically updated the next time you download the target program.

  If you download the target program while no symbol information is being displayed in the symbol window, no symbol information is updated. Therefore, click the “Create Symbol List” button to output symbol information.

3. Load the output file into the I/O DLL to register the symbol information.
   Below shows a process.

- To register the symbol information to the I/O DLL, use the RegistrationSymbolList and FreeSymbolList functions. Below shows a specification of the function.

[Registration of symbol information]

Function name:	BOOL RegistrationSymbolList(const char * pszFile)	
Parameter:	const char * pszFile	Symbol file name. Specify the file with absolute path. Can't use a network pathname.
Return value:	BOOL status	
	TRUE	Succeeded.
	FALSE	Error.

An error dialog is displayed if an error occurs.

Below show the contents of the error.

Error message	Contents
A symbol file can't open.	The symbol file cannot be opened. Check whether the specified symbol file is correct and whether it exists.
Not enough memory to open a symbol file.	The symbol file cannot be opened for lack of memory. Check whether sufficient memory is available.
The format of the symbol file is wrong.	The symbol file has an invalid format. Create a symbol file correctly again.

Functions: This function loads the created symbol file into the symbol window to register the symbol information. The symbol information registered by this function is retained in memory. Therefore, whenever the symbol information becomes unnecessary, call the FreeSymbolList function to delete the symbol information (to free up memory space).

[Delete of symbol information]

Function name:	void FreeSymbolList(void)
Parameter:	None
Return value:	None
Functions:	Delete the symbol information (to free up memory space).

Below shows a use example.

```
void NotifyReset(void)
{
      // When reset, load symbol information to register.
      RegistrationSymbolList( "d:¥¥sample¥¥test.sym" );

      return;
}
void NotifyEnd(void)
{
      // Delete the symbol information loaded into the I/O DLL when exiting PDxxSIM.
      FreeSymbolList( );

      return;
}
```

Note:

The RegistrationSymbolList function registers only the symbol information included in one symbol file. If this function is called twice or more, it deletes the previous symbol information (to free up memory space) before registering new symbol information. If the same symbol file is specified, the function inspects the file’s timestamp and only when it has been updated, re-registers the symbol information.

4. Specify a symbol in the I/O DLL.
   Below shows a process.

- To convert the symbol to a value in the I/O DLL, use the ChangeSymtToVal function. Below shows a specification of the function.

[Convert value of the symbol]

Function name:	BOOL ConvertSymToVal(const char * pszSym, unsigned long * ulAddr, int nType)	
Parameter:	const char * pszSym	Symbol name.
	unsigned long * ulAddr	Pointer in which the converted symbol value is stored.
	int nType	Symbol types which are given priority when searched.
	TYPE_SYMBOL	Symbol.
	TYPE_LABEL	Label.
Return value:	BOOL status	
	TRUE	Succeeded.
	FALSE	A symbol isn't found.
Functions:	This function searches for the corresponding value for the symbol and stores the found value in ulAddr. The table below shows what search priority will be assigned to each symbol type when specified by nType.	

priority level	TYPE_SYMBOL	TYPE_LABEL
level 1	Local symbol	Local label
level 2	Global symbol	Global label
level 3	Local label	Local symbol
level 4	Global label	Global symbol

Attention:

For the symbol name to be specified for the pszSym parameter of the ConvertSymToVal function, be sure to use one that is displayed in the symbol window. For example, if the symbol name displayed in the symbol window is prefixed by an underbar as in the case of “_strl,” the symbol name specified for the pszSym parameter must also be prefixed by an underbar.

Below shows a use example.

```
void NotifyPreReadMemory(unsigned long address, int length)
{
      unsigned long s_addr;
      unsigned long data;

      // When the address 0x3e0 is read, store the value of the symbol “strl” in 0x3e0
      if ( address != 0x3e0 ) {
            return;
      }
      if ( ConvertSymToVal("_str1", &s_addr, TYPE_LABEL ) == FALSE ) {
            return;
      }
      RequestGetMemory(s_addr, length, &data);
      RequestPutMemory(address, length, data);
}
```

9. 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 PDxxSIM.
2. Only one I/O DLL can be specified in PDxxSIM. You cannot specify multiple I/O DLLs.

PDxxSIM I/O DLL Kit User's Manual

Rev. 1.00
May 1, 2003
REJ10J0063-0100Z

PDxxSIM I/O DLL Kit
User's Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ10J0063-0100Z