

Draft RFC Submission to OMG

A UML Extension Profile for SoC

Revision 1.0a

Submitters:

Fujitsu Limited
IBM Corporation
NEC Corporation

Supported by:

CANON INC.
CATS Co., Ltd.
Metabolics, Ltd.
RICOH COMPANY, LTD.
Toshiba Corporation
UML for SoC Forum
UMTP Japan

Copyright (C) 2004 Fujitsu Limited
Copyright (C) 2004 IBM Corporation
Copyright (C) 2004 NEC Corporation
Copyright (C) 2004 CANON INC.
Copyright (C) 2004 CATS Co., Ltd.
Copyright (C) 2004 Metabolics, Ltd.
Copyright (C) 2004 RICOH COMPANY, LTD.
Copyright (C) 2004 Toshiba Corporation

All rights reserved.

The companies listed above hereby grant to the Object Management Group, Inc. (OMG) and OMG members, permission to copy this document for the purpose of evaluating the technology contained herein during the technology selection process by the appropriate OMG task force. Distribution to anyone not a member of the Object Management Group or for any purpose other than technology evaluation is prohibited.

The material in this document is submitted to the OMG for evaluation. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The companies listed above shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. The information contained in this document is subject to change without notice.

This document contains information which is protected by copyright. All Rights Reserved. Except as otherwise provided herein, no part of this work may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems— without the permission of one of the copyright owners. All copies of this document must include the copyright and other information contained on this page.

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013.

TRADEMARKS

OMG OBJECT MANAGEMENT GROUP, CORBA, Object Request Broker, OMA, UML, Unified Modeling Language, UML CUBE LOGO, MDA, MODEL DRIVEN ARCHITECTURE, MOF, and Meta-Object Facility are registered trademarks or trademarks of Object Management Group, Inc.

2008/08/17 08:02

Contacts:

Takashi Hasegawa
Fujitsu Limited
4-1-1 Kamikodanaka, M/S M13-415,
Nakahara-ku, Kawasaki, 211-8588, Japan
Tel: +81 44 754-2548
Fax: +81 44 754-2473
Email: thasegaw@jp.fujitsu.com

Masato Matsuoka
IBM Japan, Ltd
19-21 Nihonbashi Hakozaeki Cho, Chuoh-ku,
Tokyo 103-8510, Japan
Tel: +81 70 6226-8680
E-mail: E50574@jp.ibm.com

Minoru Tomobe
NEC Corporation
1753 Shimonumabe, Nakahara-ku,
Kawasaki, 211-8666, Japan
Tel: +81 44 435-9486
Fax: +81 44 435-9491
E-mail: tomobe@bu.jp.nec.com

Toshiaki Minami
CANON INC.
3-30-2 Shimomaruko, Ohta-ku,
Tokyo 146-8501, Japan
Tel: +81 3 5732-8460
Fax: +81 3 5482-9314
E-mail: minami.toshiaki@canon.co.jp

Koji Asari
CATS Co., Ltd.

2008/08/17 08:02

2-11-5 Shin-yokohama, Kouhoku-ku,
Yokohama, 222-0033, Japan
Tel: +81 45 473-2816
Fax: +81 45 473-2673
E-mail: asari@zipc.com

Masaki Yamada
Metabolics, Ltd.
576-8 Kokufuhongo, Ooiso-machi,
Kanagawa, 259-0111, Japan
Tel: +81 463 60-2234
Fax: +81 463 60-2266
E-mail: masaki@metabolics.co.jp

Tadayoshi Miyahara
RICOH COMPANY, LTD.
1-3-6 Naka-magome, Ohta-ku,
Tokyo 143-8555, Japan
Tel: +81 3 5742-8840
Fax: +81 3 5742-8749
E-mail: chuugi@iod.ricoh.co.jp

Masami Aihara
Toshiba Corporation
580-1 Horikawa-cho, Saiwai-ku,
Kawasaki, 212-8520, Japan
Tel: +81 44 548-2346
Fax: +81 44 548-8318
E-mail: masami.aihara@toshiba.co.jp

UMTP-Japan
Url: <http://www.umtp-japan.org/>
E-mail: umtp-sec@umtp-japan.org

UML for SoC Forum

2008/08/17 08:02

Url: <http://www.zipc.com/usocf/index.html>

E-mail: usocf@zipc.com

Member List of "UML for SoC Forum":

Takashi Hasegawa, Fujitsu Limited

Minoru Shoji, Fujitsu Limited

Masato Matsuoka, IBM Japan, Ltd. (Rational Software)

Nobuaki Takahashi, IBM Japan, Ltd.

Hisashi Suzuki, IBM Japan, Ltd.

Takayuki Okada, IBM Japan, Ltd.

Minoru Tomobe, NEC Corporation

Toshiaki Minami, CANON INC.

Tetsuji Saito, CANON INC.

Koji Asari, CATS Co., Ltd.

Yuichi Tsukada, CATS Co., Ltd.

Masaki Yamada, Metabolics, Ltd.

Tadayoshi Miyahara, RICOH COMPANY, LTD.

Mutsumi Namba, RICOH COMPANY, LTD.

Satoshi Okada, RICOH COMPANY, LTD.

Masami Aihara, Toshiba Corporation

A UML Extension Profile for SoC(1.0)

2004/05/31

FUJITSU Limited

Minoru Shoji

1 System-on-Chip (SoC): A Brief Introduction

System-on-Chip (SoC) paradigm enables multiple function components, composed of both hardware and software to be integrated on a single silicon chip. The SoC paradigm enables component-based system design. Although SoC paradigm shares the same design principles with component-based software development, SoC design differs from software engineering due to the presence of both synchronous (clocked) and asynchronous hardware components and hardware-software interfaces.

2 What is the UML Extension Profile for SoC ?

Existing UML semantics have a strong focus on software requirements specification and therefore is inadequate for supporting modeling and specification of SoC designs. In order to support modeling and specification of SoC designs, we propose a minimal set of extensions for a UML profile. To augment the semantics for UML-based SoC design, stereotypes for each UML metaclass have to be defined by introducing a special notation and constraint for each SoC element. For this purpose we introduce *SoC structure diagrams*. SoC designers create both class diagrams and SoC structure diagrams for developing an *Executable* system level model.

In the first part of the document we define stereotypes for SoC structure diagram, their notations and constraints. Next, we show an example description of executable system level model corresponding to UML model. This example provides an insight into developing “executable” models and a method of modeling using template modules in SoC structure diagram.

We have chosen SystemC as an executable system level modeling language for illustrating the SoC design profile. However, another suitable system modeling language could be used instead.

3 Stereotypes of SoC profile

Each element of the SoC model corresponds to the stereotyped UML metaclasses as shown in Table 1 below. Constraints defined by these stereotypes are described in accompanying documents titled *SoC profile constraints defined by OCL*, *SoC profile stereotypes*, and *SoC profile stereotype relations(class diagram)*. A detailed description of each SoC Model element is provided in this section.

Note:As two metaclasses are extended using SoCPort, two SoCPort stereotype classifier are drawn in *SoC profile stereotype relations(class diagram)*. For identification purpose, the diagram shows which metaclass is extended by each stereotype.

SoC Model element	Stereotype	UML metaclass
data	data	Class
controller	controller	Class
module	SoCModule	Class
port	SoCPort	Port/Class
protocol interface	SoCInterface	Interface
channel	SoCChannel	Class
connector	SoCConnector	Connector
protocol	SoCProtocol	Collaboration
process	SoCProcess	Operation
module part	SoCModuleProperty	Property
channel part	SoCChannelPropert y	Property
clock port	SoCClock	Port
reset port	SoCReset	Port
clock channel	SoCClockChannel	Property
reset channel	SoCResetChannel	Property
deta type	SoCDataType	Dependency

Table.1 UML metaclasses extended by SoCprofile.

Module

Module is the base class for describing SoC hierarchical class structure.

Tagged value added by this stereotype:

None

Other restrictions:

Only ports, constructors, and destructors are visible external to the module. Modules may consist of template modules and channels. Template parameter also can be used as sub-module, port and channel multiplicity, or as connector connection specifications. At least one member function of a module should not be a process.

(See constraints by OCL)

process

Process is a member function of the module that describes its behavior.

Tagged value added by this stereotype:

None

Other restrictions:

There is no restriction on the number of processes contained in a module.

There is restriction on the number of processes that can access a channel or port depending on the protocol interface realizing the channel.

(See constraints by OCL)

Data

Only the Data class would be the target of communication between modules using the Channel Class.

Tagged value added by this stereotype is:

transportMethod : transportBy

Enumeration value that specifies the instance of the data transmission method between, namely Copy and Pointer. When the value is Copy then module will send the copy of data instance through the channel. When the value is Pointer then module transmits the pointer to the data instance. This pointer method is used when the data class that contains pointer (or reference) as its member and when the data instance should not be copied in order to avoid invalid data access.

Other Restrictions:

(See constraints by OCL)

Controller

The only member function of this class could be the module process that is described later.

protocol interface

It defines the communication between modules, and provides an abstraction for the method of communication. All communication between modules must be realized through protocol interfaces. In many cases, the class that protocol interface can transmits is restricted, and the class must be defined by the "data type" dependency described later. If a protocol interface is template protocol interface, then the dependency may have one template parameter as its target. The "data type" section described later provides a more detailed explanation.

Tagged value added by this stereotype:

Direction: directionKind directionKind is an enumeration type, whose elements are INPUT and OUTPUT, that specifying the incoming/outgoing direction of the protocol interface.

maxProcesses: UnlimitedNatural Specify the maximum number of processes that is accessible "through this protocol interface" to the channel realizing this protocol interface. Those processes may be owned by multiple modules (the number of processes must not exceed maxProcesses attribute value after instantiation of whole module hierarchy).

maxChannels: UnlimitedNatural Specify the maximum number of channels that is connectable to a port instance having this protocol interface as its required interface (the number of channels must not exceed maxChannel attribute value after instantiation of whole module hierarchy).

The default value is 1.

Other restrictions:

(See constraints by OCL)

channel

Realizes communication between modules. Every communication between modules must be realized by channel through protocol interface. Channel must be inherited from protocol interface. Similar to the Module class, Channel may contain Process, Channel, Port and other modules. Its notation and specification of tagged value is also identical to that of module. Such Channels are called "Hierarchical Channels". Simple channels which provide primitive functionality such as hardware signals are called "Primitive Channels".

Tagged value added by this stereotype:

isPrimitive : Boolean Specify if the channel is primitive or not(hierarchical channel). This value is derived by the existence of other modules, processes, and channels contained in the channel. A design tool should be able to alter the channel notation using this value.

Other restrictions:

Each channel must be inherited from at least one INPUT direction protocol interface and one OUTPUT direction protocol interface.

(See constraints by OCL)

protocol

Protocol describes the relationship between protocol interfaces and a channel. As with the UML diagrams, notation is same as that of collaboration diagram.

Tagged value added by this stereotype:

None

Other restrictions:

Each protocol must have (a) at least one incoming protocol interface, (b)

at least one outgoing protocol interface, and (c) at least one channel inherited directly or indirectly from protocol interfaces.

(See constraints by OCL)

port

It defines the protocol interface used for communication external to modules.

Tagged value added by this stereotype:

None

Other restrictions:

(See constraints by OCL)

module part

It is a kind of property (part) typed by the module class. It is usually called as "sub-module".

Tagged value added by this stereotype:

None

Other restrictions:

(See constraints by OCL)

channel part

It is a kind of property (part) typed by the channel class. Typically the channel must be shown as this property, and does not exist independently (channel must occur as property/part of a module).

Tagged value added by this stereotype:

None

Other restrictions:

(See constraints by OCL)

connector

Connector connects ports or connects ports and a channel. If a connector is connected to either of port, module part, or channel part with multiplicity of 1, then the specification of connection between each instances may be defined by OCL. The specification rule is described in the document "SoC profile constraints defined by OCL".

Tagged value added by this stereotype:

None

Other restrictions:

In a module (here we call parent module), ports contained in the module can be connected only to the other ports contained in the module properties (parts). In this case, we call them as a sub-module contained in the parent module. In addition, channel property owned by parent module can be connected only to the ports of sub-modules. Ports of sub-modules can be connected to the ports or channels owned by the parent module. Connection without connectors is prohibited. In addition, when ports are connected directly, the type of protocol interfaces of these ports must be compatible with each other.

(See constraints by OCL)

clock port

This class is inherited from port, and it specifies clock input or clock output port. It is typically used in or after the "structural design" phase, which requires investigation of synchronization schemes.

Tagged values added by this stereotype:

clockDomain : domainName domainName specifies the name of "clock domain", which consists of clock cycle, phase, duty-ratio, and the method of controlling the gated-clock. There is no limitation on the name or the specification of clock domains. Connection between the ClockPorts, or connection between ClockPort and ClockChannel (explained later) with a different clockDomain attribute is prohibited. It is possible to connect different clock domains only if the

connection is through the process that converts the clock domain..

Other restrictions:

(See constraints by OCL)

reset ports

This class is inherited from port, and it specifies reset input or reset output port. It is typically used in or after the "structural design" phase, which requires investigation of reset schemes.

Tagged value added by this stereotype:

resetSpec : resetSpecName resetSpecName specifies the name of "reset specification", which consists of reset timing specification (synchronous / asynchronous reset, polarity etc.). There is no restriction on the name or the specification of reset specification. Connection between the resetPorts, or connection between resetPort and resetChannel (explained later) with a different resetSpec attribute is prohibited. It is possible to connect different reset specifications only if it is through a process that converts reset specifications..

Other restrictions:

(See constraints by OCL)

clock channel

This is a property (part) defining the channel instance(s) that transmits the execution events to synchronous processes. It is typically used in or after "structural design" phase, which requires consideration of synchronization schemes.

Tagged value added by this stereotype :

clockDomain : domainName domainName specifies the name of "clock domain". Refer to the clockPort for the definition of clock domain. Connection to a clockPort is limited

to that with same clockDomain attribute.

Other restrictions :

(See constraints by OCL)

reset channel

This is a property(part) defining channel instance(s) that transmits the reset events to processes. It is typically used in or after the "structural design" phase, which requires the consideration of resetting schemes.

Tagged value added by this stereotype:

resetSpec : resetSpecName ResetSpecName specifies the name of "reset specification". Refer to the resetPort for the definition of reset specification. Connection to a resetPort is limited to that with same resetSpec attribute.

Othre restrictions :

(See constraints by OCL)

data type

This is an extension of UML dependency and points at a data class which the protocol interfaces transmit. If the protocol interface is a template, then this dependency may point to one of the template parameter classes.

Tagged value added by this stereotype:

None

Other restrictions:

(See constraints by OCL)

4 SoC Structure diagram

SoC structure diagram is used to describe the structure of SoC. The following elements are used:

- Module and Module Part
- Port (Single or Multiple)
- Protocol Interface
- Channel Part
- Connector
- Protocol
- Process
- Clock Port
- Reset Port
- Clock Channel
- Reset Channel
- Data Type

This section explains the notations in SoC structure diagrams.

4.1 Module and module part

Module hierarchy is explained by a module and other module parts contained in it as shown in figure 1. When module part hierarchy of module is displayed using SoC structure diagram, the expanded form is typically used (in this diagram, elements other than property name and class name of module in the structure diagram are omitted). The method of expanding every module in a hierarchy is not defined in UML specification, although it is necessary for large-scale system (SoC) specification notation. However, it is also difficult for designers to operate whole hierarchy expressed in expanded form drawn on one diagram for large scale system design. There should be alternative method of suppressing the expansion of module for selected parts of a hierarchy. To express the module whose hierarchy is suppressed on a diagram, there should be an icon as shown in figure 2. This diagram shows sub-module C2 shown in figure 1, without expanding the sub-module hierarchy.

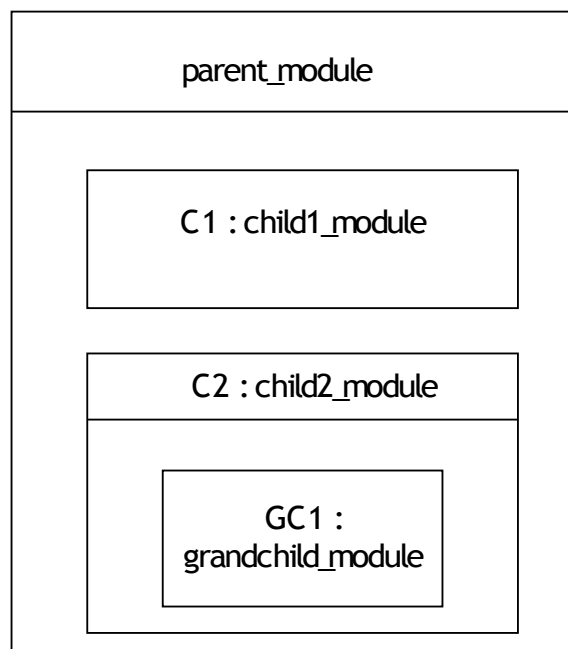


Fig.1 module hierarchy in structure diagram

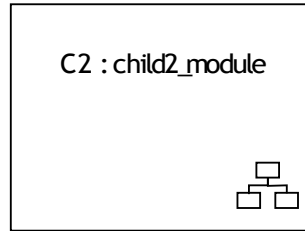


Fig.2 The module which abbreviates sub module hierarchy

The initialization parameters of sub-modules and channels owned by a module must be passed as the module's parameter, and passed to the sub-modules and channels during the module's construction.

Module can also be parameterized (templated). The notation of these parameterized module instances follow the UML notation. Templated module is shown in uninstantiated form in structure diagram in the same way as templated class. Template parameter (which may be a constant) must be specified in some form by the designer (tools will not assume the parameter by the port types because it is too complex) when it is instantiated.

Furthermore, it is possible to configure the module hierarchy using the constructor parameter. In this case, uninstantiated form is used on SoC structure diagram. Constructor parameter is specified in SoC structure diagram as shown in the following figure.

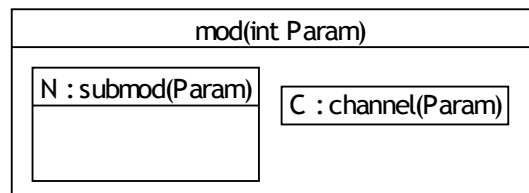


Fig.3 An example of constructor arguments usage

Sequence diagrams may be used to define complex sequences during module construction as shown in figure 4. This diagram shows an example of a constructor sequence that reads a file for each module instance name. To construct module hierarchy with these sequences, the module may have pointers to module parts or channel parts instead of

having instances as its members. These members are assigned the address of instances when instantiated in the constructor body. In that case, the module part or channel part must be marked by attributes. The attribute specification depends on the SoC design tools and therefore is not part of this extension profile.

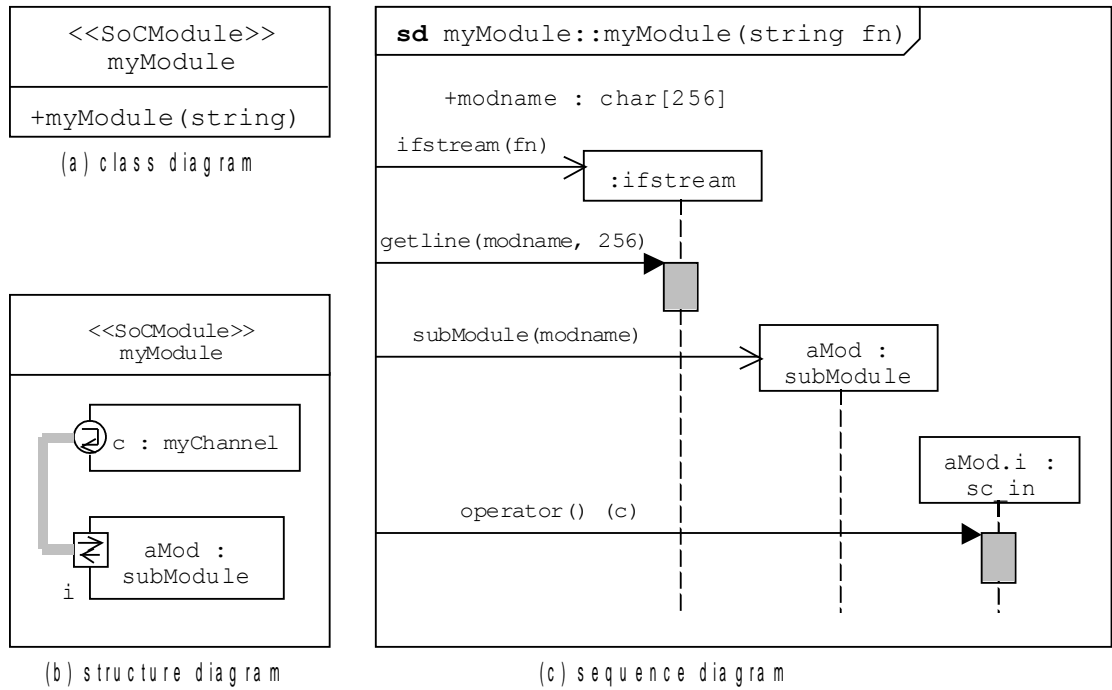


Fig. 4 Complex constructor definition example

4.2 Port and the property

The module class has to consist of a process member function that defines a branch or a merge when a branch or a merge has to be realized to/from a port. This definition is similar to the definition of typical module. Single port is the port with multiplicity equals one, and is shown as a rectangle two opposite-facing arrows as shown in Figure 5 below.

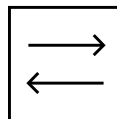


Fig.5 Icon of Port

With the multi port that has multiplicity of 2 or more, another icon is used. The following icon is used for multi port.

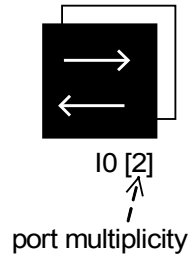


Fig.6 Icon of Multi Port

4.3 Protocol Interface

Protocol interface on the structure diagram is expressed by the circle with U shaped arrow in it, written on a rectangle showing a channel as shown in figure 7.

Protocol interface must have a dependency (Data Type dependency) to a data class. If a protocol interface is inherited from another protocol interface, either general interface or specialized interface must have the dependency. In other words, there should be exactly one data type dependency when referred to by ports or channels.

As the design changes, modification and implementation of protocol interface become necessary. For this purpose, modification and implementation of protocol are expressed in the generalization of protocol and protocol interface just as it is done in software modeling.

4.4 Channel, channel part and its property

In SoC structure diagram, channel parts are displayed as owned property (part) of a module, and are drawn in a rectangle. In the case of channel part typed by primitive channel, it is possible to omit the rectangle and express the channel only by a text string next to the connector consisting of the instance name and class name as shown in figure 7. In this case, a

connector must be drawn using a black line. For hierarchical channels, it is possible to specify the internal structure of the channel using same notation as modules as shown below (except that it is drawn in dotted line instead of continuous line).

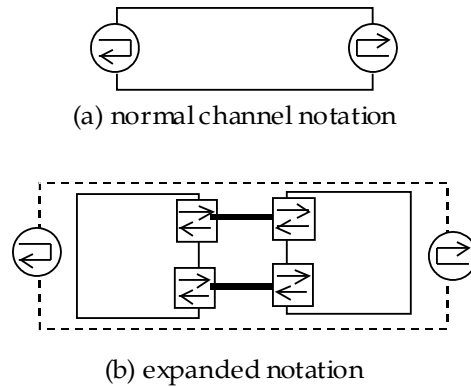


Fig. 7 Hierarchical channel notation variations

4.5 Connector

connector is expressed as a line drawn between a channel and port, or between ports. As shown in figure 7, connector between ports or connector omitting rectangle, which shows a channel, must be drawn in a black line, and connector between port and (protocol interface of) channel must be drawn in thick gray line. Connection index must be shown in a tagged value notation. .

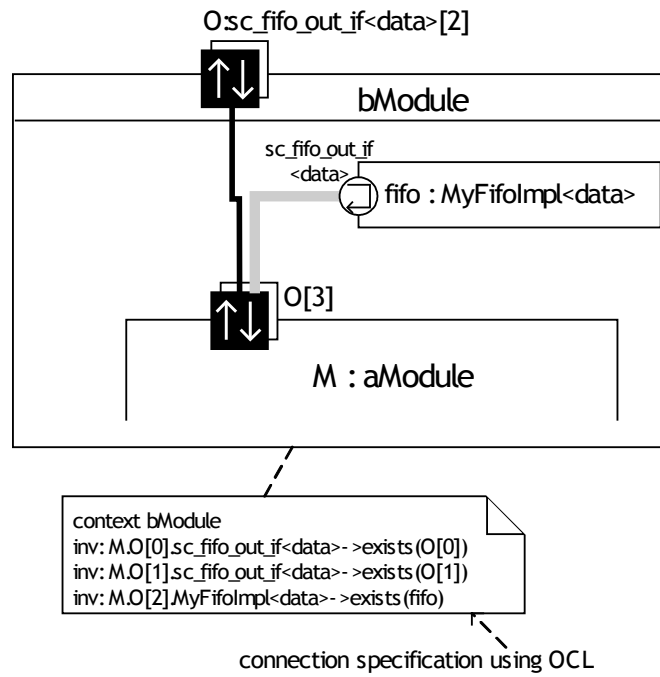


Fig.8:Specification of connecting instances

4.6 Protocol

Protocol is a kind of Collaboration, expressed as in the following figure.

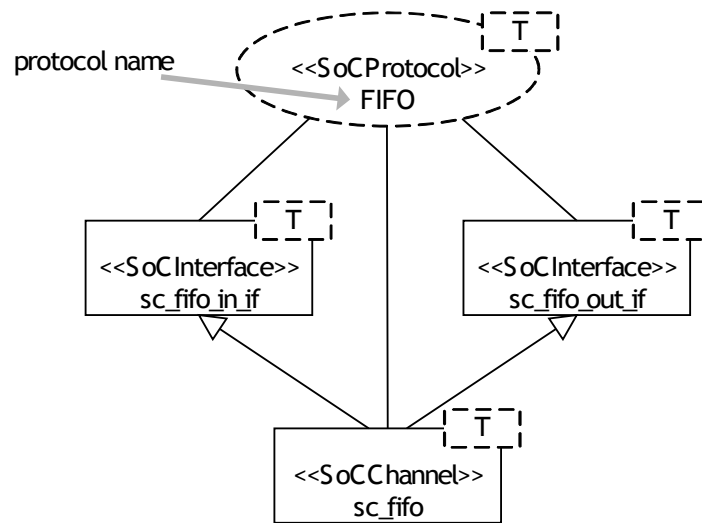


Fig.9:Protocol

4.7 Process

Process is a kind of Operation described below.

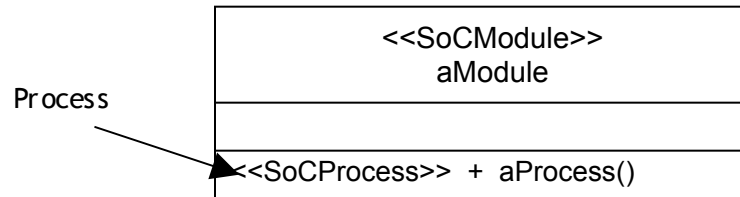


Fig.10 Process

4.8 Clock Port

The clock port uses same notation as port, with a difference of having an attribute specific to clock port. Example of the clock port is shown in Figure 11.

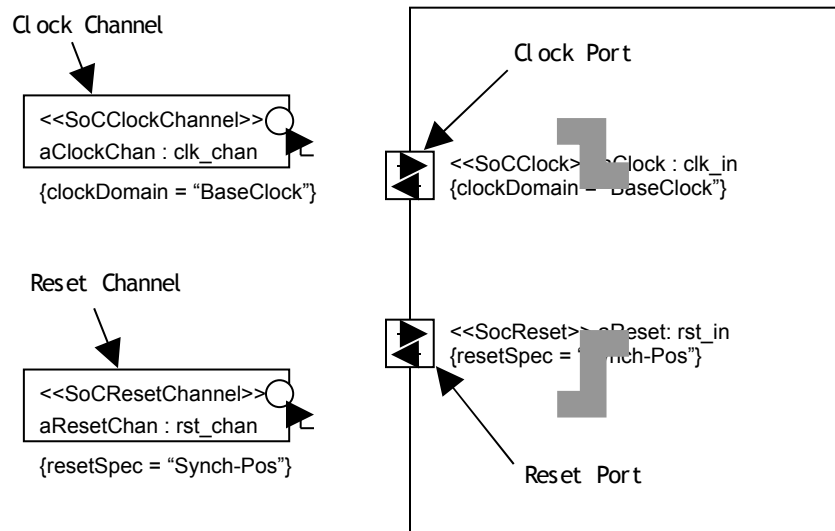


Fig.11 Clock and reset (channel/port)

4.9 Reset Port

The reset port uses same notation as port except for having an attribute specific to reset port. Example of the reset port is shown in Figure 11.

4.10 Clock Channel

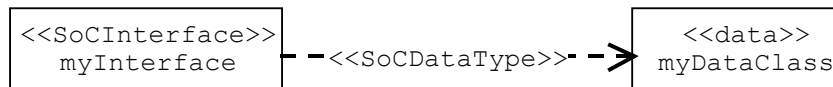
The clock channel uses same notation as channel part except for having an attribute specific to clock channel. Example of the clock channel is shown in Figure 11.

4.11 Reset Channel

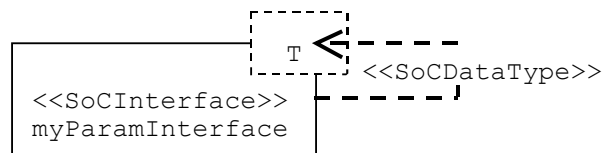
The reset channel uses same notation as channel part except for having an attribute specific to reset channel. Example of the reset channel is shown in Figure 11.

4.12 Data Type

Data Type dependency has the same notation with normal dependency as shown below.



(a) SoCDataType dependency



(b) SoCDataType dependency pointing at template parameter

Fig. 12 Data Type dependency example

5 The SystemC description corresponding to the extended profile

The SystemC description corresponding to the UML model with extended profile is shown here. The SystemC description consists of "the module description file", which describes the declaration of module, instantiation of the sub module, the channel, and the port, and "the data class description file", which defines the data class. A detailed configuration of the files is not specified in this document

5.1 Module description file

A header file will be created for each module. The header file includes module definition, port declaration, channel declaration, member function (which becomes a process) declaration and module constructor definition. It does also have "#include" as part of the necessary header (for sub modules and channels). In addition, initialization parameter is the list of parameters. The number of parameters depends on module.

Module definitions are as shown below:

```
[ template < templetParameterList > ]
SC_MODULE (className) {
  [ (port declaration) ]
  [ (sub module declaration) ]
  [ (channel declaration) ]
  [ (process function declaration) ]
  [ (declaration of user defined member variable) ]
  [ (declaration and definition of user defined member
function, and so on) ]
SC_HAS_PROCESS (className);
className (sc_module_name name [,
initializationParameters ]): Sc_module (name) [, (port
initialization) ] [, (initialization of user defined member
variables) ] {
  [ (Definition of sub module generation) ]
  [ (definition of channel generation) ]
  [ (definition of connection) ]
  [ (process generation definition) ]
  [ (initialization of user defined member variable, and so
on) ]
}
};

[ template < templetParameterList > ]
void className [ < templetParameterList > ]: :
memberFunctionName () {
```

```

    // contents of process function
}

```

[Definition of user defined member function, and so on]

◇ The port declaration is described as below

```

sc_port<      protocolInterfaceClassName      [      <
data_type_dataClassName > ] > instanceName [ ' [ '
portMultiplicity      '      ]      '      ] ;

```

Each port should have this description. When the port multiplicity is 1 (in case of the single port), the port multiplicity and the bracket [] shall be omitted. When the protocol interface is template interface, the dataClassName must be specified as a template parameter. If not parameterized, the data_type_dataClassName and <> shall be omitted.

The clock port is described as below.

```

sc_in_clk instanceName [ ' [ ' portMultiplicity ' ] ' ] ;

```

Declaration of the reset port is similar to normal port declaration.

◇ The sub module declaration is described as below

```

subModuleClassName * subModuleInstanceName [ '
[      'multiplicity'      ]      '      ] ;

```

Each sub module should have this description. When the sub module multiplicity is 1, the multiplicity and the bracket [] shall be omitted.

◇ The channel declaration is described as below.

```

channelName [ < data_type_dataClassName > ] *
channelInstanceName [ ' [ ' multiplicity ' ] ' ] ;

```

Each channel should have this description. When the channel multiplicity is 1, the multiplicity and the bracket [] shall be omitted.

The clock channel and the reset channel are declared as below

```

clockChannelName * clockChannelInstanceName ;

```

◇ The process function declaration is described as below.

```

void memberFunctionName ();

```

Also the sub modules, the channels and the processes are declared in the module constructor. The arguments for the constructor will be given from the initialization

parameter for the constructor.

✧ The port initialization is described as below.
`portInstanceName ("portInstanceName") [, portInstanceName ("portInstanceName")]`

✧ The sub module is instantiated as below
`submoduleInstanceName [' [' index '] '] = new submoduleClassName ("instanceName", initialization parameters);`

Each sub module should have this description. When the sub module multiplicity is 1, the index and the bracket [] shall be omitted.

✧ The channel is instantiated as below
`channelInstanceName [' [' index '] '] = new channelName [< data_type_dataClassName >] (initialization parameters);`

Each channel should have this description. When the channel multiplicity is 1, the index and the bracket [] shall be omitted. The clock channel is instantiated as below. Clock parameter will be given from the "clock domain" specification shown as the clockDomain tagged value.

```
clockChannelInstanceName = new
clockChannelName ("clockChannelInstanceName", clock
parameter);
```

✧ The connection between the channels or the ports is defined, with according to the connection specification of the module, as below
`submoduleInstanceName [' [' subModuleIndex '] '] -> portInstanceName [' [' subModulePortIndex '] '] (portInstanceName [' [' portIndex '] '], or * channelInstanceName [' [' channelIndex '] ']);`

Each connection should have this description. When the connection is to the sub module with multiplicity 1, the sub module index and the bracket [] shall be omitted. When the connection is to the sub-module port with multiplicity 1, the port index and the bracket [] shall be omitted. When the connection is to the module port with multiplicity 1, the port index and the bracket [] shall be omitted. When the connection is to the channel with

multiplicity 1, the channel index and the bracket [] shall be omitted.

◇ The process instantiation is described as below
SC_THREAD (**memberFunctionName**);

5.2 Data class description file

The header file is created in every class with the <<data>> stereotype. The description is a little different depending on the transport Method tagged value, as shown below.

```
typedef dataClassName* data_type_dataClassName; (When  
transportMethod is Pointer)  
typedef dataClassName data_type_dataClassName; (When  
transportMethod is Copy)
```

6 SystemC description example

Here we will show several examples of the actual UML diagrams for SoC designs and the generated SystemC codes from UML. The first example is the description of a module for sending and receiving data using a FIFO, the second example is a structured design model with "clock" and "reset".

6.1 Description example 1

Shown below is the UML class diagram. Here func() member function of user_class1 and func() member function of user_class2 are processes. And data1, data2, data3 are input/output data between the processes.

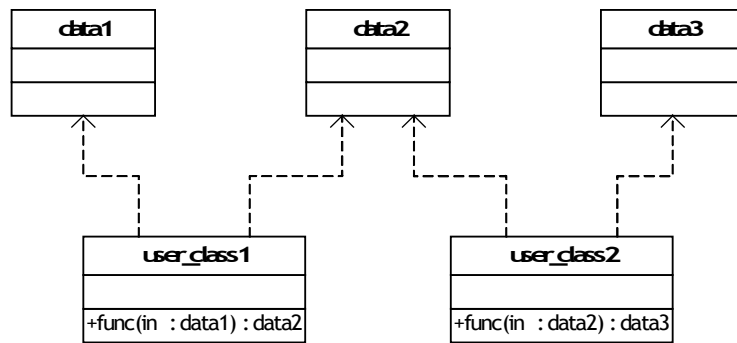


Fig.13 Class Diagram of result of requirements analysis

First, for this class diagram, <<Data>> <<Controller>> stereotypes are specified. Here after, all the tagged value transportMethod of the class with <<data>> stereotype are Pointers.

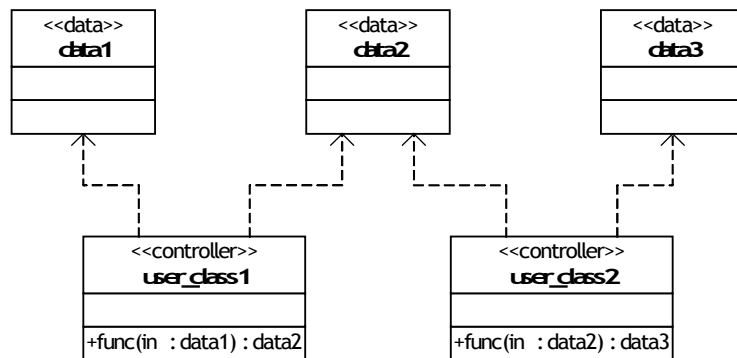


Fig.14 Class diagram after the stereotype specified

The following figure is the structure diagram. All the necessary values and attributes are specified as notes.

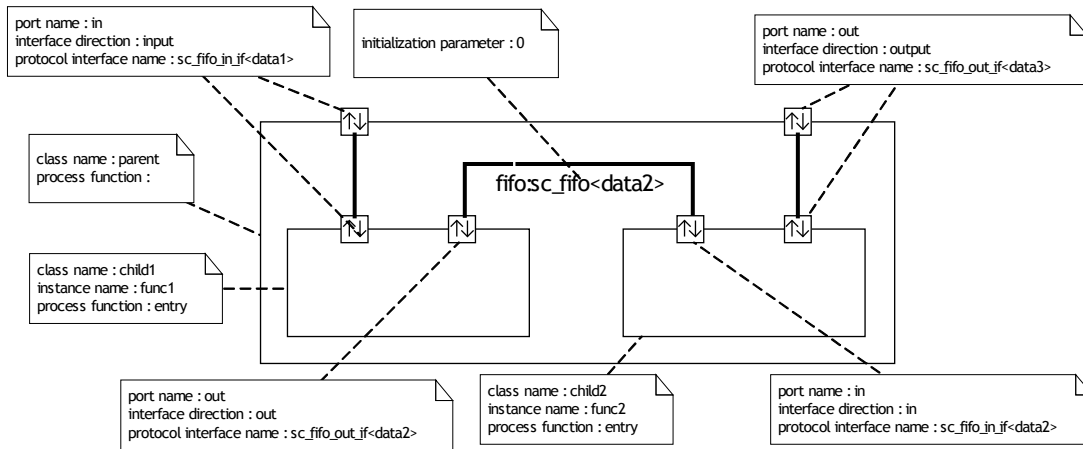


Fig. 15 Structure diagram and attribute of each object, tagged value

The channel is described in the UML diagram as shown below. Here the FIFO channel of SystemC (sc_fifo) is used as a channel with buffer size = 16 (default value of sc_fifo class). The buffer size is given as an initialization parameter at the instantiation of the channel. (In the figure below, attributes, and/or template arguments of buffer size are not shown.)

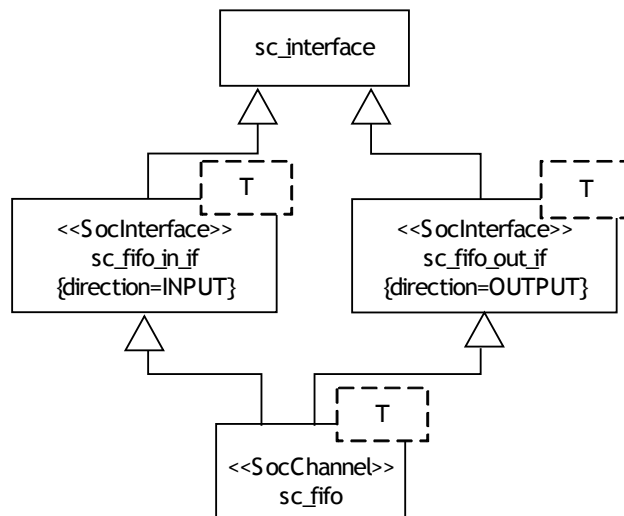


Fig. 16 FIFO Protocol Interface and Channel

2008/08/17 08:02

The generated SystemC source code from the UML diagram above is shown below. Please note that the user specified portion and the portion automatically generated using the tool are not distinguished here using comments. And also, the include statements which usually exists in the header file are ommited.

```
(parent.h)
#include "child1.h"
#include "child2.h"
#include "data_type_data1.h"
#include "data_type_data2.h"
#include "data_type_data3.h"
SC_MODULE(parent) {
    sc_port< sc_fifo_in_if< data_type_data1 > > in;
    sc_port< sc_fifo_out_if< data_type_data3 > > out;

    child1* func1;
    child2* func2;

    sc_fifo< data_type_data2 > *fifo;

    SC_HAS_PROCESS(parent);
    parent(sc_module_name name)
    : sc_module(name), in("in"), out("out") {
        func1 = new child1("func1");
        func2 = new child2("func2");

        fifo = new sc_fifo< data_type_data2 >(0);

        func1->in(in);
        func1->out(*fifo);
        func2->in(*fifo);
        func2->out(out);
    }
};
```

```
(child1.h)
#include "data_type_data1.h"
#include "data_type_data2.h"
SC_MODULE(child1) {
    sc_port< sc_fifo_in_if< data_type_data1 > > in;
    sc_port< sc_fifo_out_if< data_type_data2 > > out;

    void entry();

    SC_HAS_PROCESS(child1);
    child1(sc_module_name name)
    : sc_module(name), in("in"), out("out") {
        SC_THREAD(entry);
```

```

    }
};
void child1::entry() {

}

(child2.h)
#include "data_type_data2.h"
#include "data_type_data3.h"
SC_MODULE(child2) {
    sc_port< sc_fifo_in_if< data_type_data2 > > in;
    sc_port< sc_fifo_out_if< data_type_data3 > > out;

    void entry();

    SC_HAS_PROCESS(child2);
    child2(sc_module_name name)
    : sc_module(name), in("in"), out("out") {
        SC_THREAD(entry);
    }
};

(data_type_data1.h)
typedef data1* data_type_data1;

(data_type_data2.h)
typedef data2* data_type_data2;

(data_type_data3.h)
typedef data3* data_type_data3;

```

6.2 Description example 2

Here is the SystemC description example of the clock and reset. In this example, the UML structure diagram and the corresponding SystemC description for the module (and its process), which has a synchronous reset and a clock, are shown. The first figure is the structure diagram, and the clockDomain = "Clock" specifies the clock with the positive edge trigger, and the reset parameter "Synch" specifies the synchronous reset. Please note that the channels of data input/output are not shown in this figure.

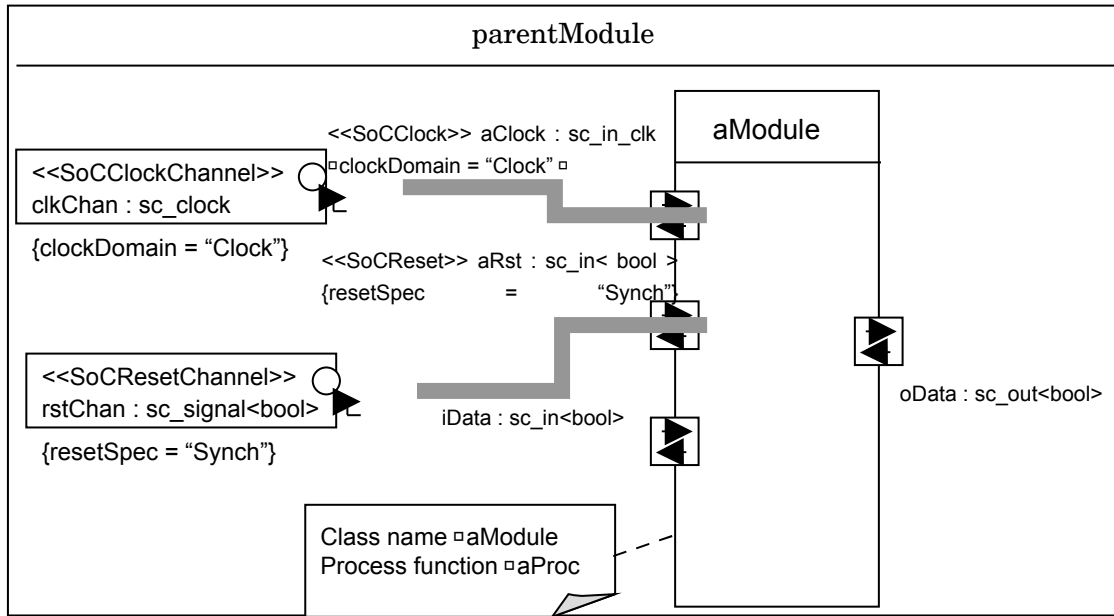


Fig. 17 Clock, Reset example

The SystemC description for the structure diagram follows below. Again, please note that the data channel is not shown here.

```

SC_MODULE(aModule) {
    sc_in_clk aClock;
    sc_in<bool> aRst;
    sc_in<bool> iData;
    SC_HAS_PROCESS(aModule);
    inline void aProc();
    aModule(sc_module_name name) :
        sc_module(name),
        aClock(aClock, aRst(aRst, iData(aData {
            SC_CTHREAD(aProc, aClock.pos());
            watching(aRst.delayed() == true);
        }
        }
        );
    void aModule::aProc() {
        while (true) {
            wait();
            bool const b = iData.read();
            oData.write(b);
        }
    }
}

SC_MODULE(parentModule) {
    sc_signal *clkChan;
    sc_signal<bool> *rstChan;
    aModule *aMod;
    parentModule(sc_module_name name) : sc_module(name) {
        clkChan = new sc_clock(*clkChan 10, SC_NS);
        rstChan = new sc_signal<bool>(*rstChan);
        aMod = new aModule(*Module);
        aMod->aClock(*clkChan);
        aMod->aRst(*rstChan);
        /* data channel instantiation and bindings here
    }
};

```

Fig. 18 Clock, Reset in SystemC description example

7 Modeling guidelines

This section describes the basic method of transmitting data instances via channels. As described in the data stereotype discussion, there are two modes in transmission of the data instances. In the first part, SystemC example of Copy mode is shown. Next example of Pointer mode is shown. These examples show how modules aCopyMod or aPointerMod receive the instance of Data type, then how these modules transfer the received instances to another module. The timing of creation and deletion of value is critical to pointer mode. Therefore we recommend that Copy mode is adopted. However, when pointers or references are mandatory because of implementation algorithms Pointer mode must be selected. When the pointer mode is used, application of following rules is recommended to avoid problems during model creation: 1) sender of the data instance allocates the memory area for the data instance, initialize the area with values, and send the pointer pointing at the area, 2) receiver deallocates the area pointed by the pointer after its use of received data.

Transmission example of Copy mode:

```
typedef Data data_type_Data;
SC_MODULE(aCopyMod) {
    sc_port< sc_fifo_in_if< data_type_Data > > in;
    sc_port< sc_fifo_out_if< data_type_Data > > out;
    void entry();
    aCopyMod(sc_module_name name) :
        sc_module(name), in("in"), out("out")
    {
        SC_THREAD(entry);
    }
};
void aCopyMod::entry() {
    while (true) {
        Data t = in.read();
        Data nt = // The value nt which based on the value t is
        created here.
        out.write(nt);
    }
}
```

Fig. 19 Example of the data transmission by copy mode

Transmission example of Pointer mode:

2008/08/17 08:02

```
typedef Data* data_type_Data;
SC_MODULE(aPointerMod) {
    sc_port< sc_fifo_in_if< data_type_Data > > in;
    sc_port< sc_fifo_out_if< data_type_Data > > out;
    void entry();
    aPointerMod(sc_module_name name) : sc_module(name),
    in("in"), out("out") {
        SC_THREAD(entry);
    }
};
void aPointerMod::entry() {
    while (true) {
        data_type_Data t = in.read();
        data_type_Data nt = new Data(); // Here, the Data type value nt
        // is instantiated using the value t.
        delete t; // The read value t is deleted here.
        out.write(nt);
    }
}
```

Fig.20 Example of the data transmission by Pointer mode

8 Notation of module hierarchy and conversion to the SystemC description

In SoC design it is essential that the configuration precisely conforms to a specification. This section explains how this configuration must be captured by the design entry tools.

As shown in the following diagram, implementing some functionality with sub modules, and channels connecting them is widely employed.

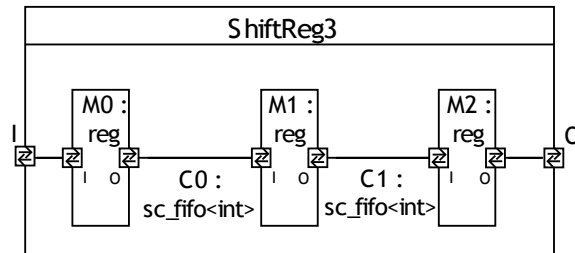


Fig.21 Example of the module which has systematic structure

In SoC design, it is typical to construct a module structure using an algorithm parameterized by template parameters or constructor arguments. This section explains how this kind of structure is described using the structure diagram.

The following diagram shows the same module using a template parameter for specifying the number of owning modules. Connection specification between each instances are described using OCL as invariants of ShiftReg. By allowing this kind of description, designers are able to define the highly re-usable modules.

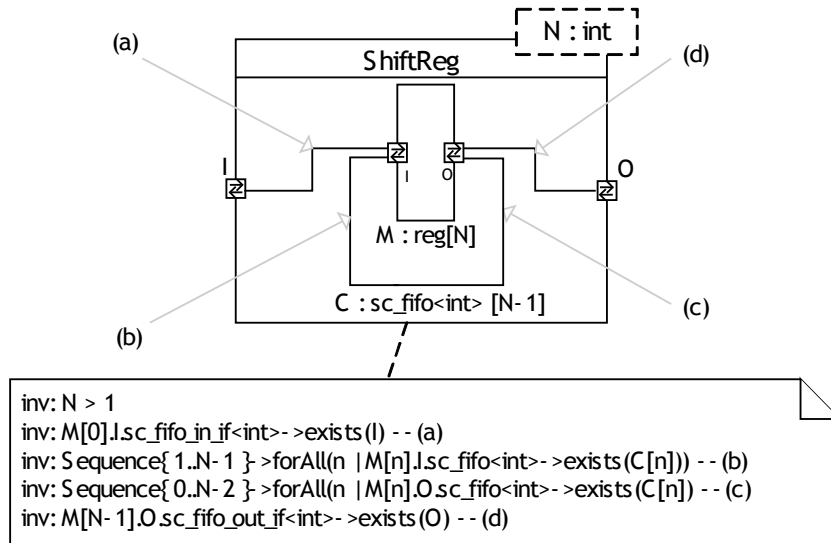


Fig.22 Example of template module

The SystemC description corresponding to the diagram is shown below, (note that include statements or any declarations, that is required but not relevant to the explanation of the notation, are abbreviated).

```

template <int N>
struct ShiftReg : public sc_module {
    sc_port< sc_fifo_in_if<int> > I;
    sc_port< sc_fifo_out_if<int> > O;

    reg *M[N];
    sc_fifo<int> *C[N-1];

    ShiftReg(sc_module_name name) : sc_module(name) {
        assert(N>0);
        for (int i = 0; i < N; i++) {
            stringstream regname;
            regname << "reg[" << i << "]";
            M[i] = new reg(regname.str().c_str());
        }
        for (int i = 0; i < N-1; i++) {
            C[i] = new sc_fifo<int>;
        }
        for (int i = 0; i < N; i++) {
            if (i == 0) {

```

2008/08/17 08:02

```
        M[i]->I(I);
    }
    if (i > 0) {
        M[i]->I(*C[i-1]);
    }
    for (int i = 0; i < N; i++) {
        if (i == N-1) {
            M[i]->O(O);
        }
        if (i < N-1) {
            M[i]->O(*C[i]);
        }
    }
};
```

End of the Document