OCP Channel Monitors

For OCP Channel Version 2.1.2

Produced by:
OCP-IP SLD Working Group

Original proposal by CoWare, Inc.

1 Contents

2 Revision History

Version	Date	Author	Comments
0.1	10/01/2004	Tim Kogel (CoWare)	Original text-based proposal
0.2	11/05/2004	Tim Kogel (CoWare)	incorporate working group feedback: system monitor support for transaction cancellation and multicast
0.3	02/09/2005	Tim Kogel (CoWare)	Major revision, include documentation for 1st release
1.0	02/19/2005	Anssi Haverinen (Nokia)	Approved for release.
2.1.2	02/24/2006	Tim Kogel (CoWare)	major revision, incorporate new monitor interface for TL1, TL2, and TL3 channels

3 The New Monitor Interfaces

3.1 Motivation

So far the trace monitors are tightly coupled with the TL1 and TL2 channels. In fact the monitors are both members and friend classes of the channels. On the other hand, the performance monitors are using a specific monitor interface to access the current state of the channels. These incompatible concepts make the usage of the different types of monitors not really intuitive. Additionally it is not possible to connect additional, user-defined monitors to the channel without changing the monitor code.

3.2 Overview

In the 2.1.2 release of the OCP channel library the monitor interfaces have been revised to eliminate the limitations mentioned above. The new monitor interface unifies the way monitors are connected to the channels. Additionally an arbitrary number of monitors can be connected to any channel. This enables the user of the OCP channels to develop specific monitors.
In principle the new access interface uses the observer pattern.

3.3 TL1 Monitor Interface Overview

The concept of the new monitor interface is illustrated in Figure 1 by means of the TL1 channel. The channel itself implements a monitor interface, which allows peeking the current state of the monitor. Using the monitor interface external modules can access the channel without modifying it. For this purpose the monitors have an sc_port templated with the monitor interface to connect to the channel.

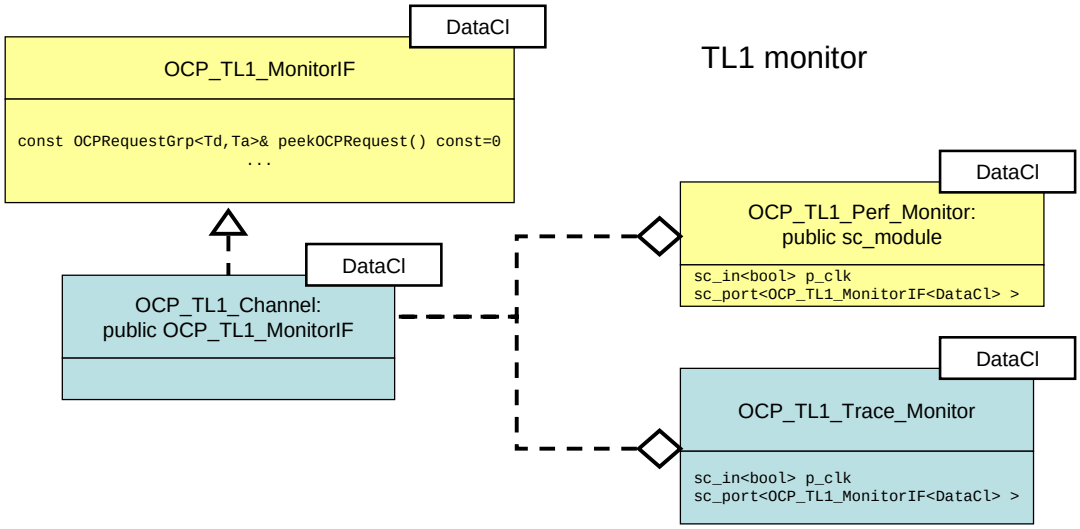

Figure 1: New TL1 Monitor Interface

The TL1 monitors depicted in figure Figure 1 also have a clock input port to access the monitor in every clock cycle. Alternatively the monitors could have OCP master and slave ports to use

```
1    // OCP TL1 channel
2    typedef OCP_TL1_DataCl<OCPCHANNELBit32, OCPCHANNELBit32> data_type;
3    OCP_TL1_Channel_Clocked< data_type > ch0("ocp0");
4
5    // trace monitor
6    OCP_TL1_Trace_Monitor< data_type > tmon0("tmon0","ocp0.trace");
7    tmon0.p_mon(ch0);
8    tmon0.p_clk(clk);
9
10   // performance monitor
11   OCP_Tl1_Perf_Monitor< data_type > pmon0("pmon0",true,false);
12   pmon0.p_mon(ch0);
13   pmon0.p_clk(clk);
```

event-based activation.

Figure 2: TL1 Monitor Example

An example for instantiating the TL1 monitors provided by OCP-IP is depicted in Figure 2. The monitor instantiation and the binding of the monitors to the channels is part of the SystemC netlist. Obviously an arbitrary number of additional monitors can be connected to the channel.

3.4 TL2 Monitor Interface Overview

Ideally the TL2 monitor interface would be just as simple as the TL1 monitor interface. Instead of using clocks, the monitors could use the regular channel events to get activated in case something interesting is happening on the channel.
Unfortunately this would lead to race-conditions between the OCP master and slave components on the one hand side and the attached monitors on the other side. Since the monitors and the OCP components would use the same events, the sequence of activation would not be deterministic. However it is important, that the monitor is activated *before* the OCP components. Otherwise the OCP components could modify the state of the OCP channel before the monitor has a chance to record the respective event. For example an OCP slave could immediately accept the request. The root of the problem is that for performance reasons the TL2 channel (other than the TL1 channel) is not based on the sc_prim_channel.

Consequently the monitor interface of the TL2 channel is one level more complex (see Figure 3). In addition to the peek interface, the TL2 monitor interface allows the registration of monitor objects. For this purpose, the monitors need to implement the TL2 observer interface. Now every observer can subscribe to any set of events in the channel.

Consider the following example: The performance monitor needs to be activated on every request start event. Therefore the performance monitor subscribes itself to the channel using the RegisterRequestStart method. Additionally the performance monitor implements the NotifyRequestStart method. The TL2 channel calls this method in all the respective subscribers (i.e. the monitors) together with the notification of the regular SystemC RequestStartEvent. This ensures that the monitors are always updated before the SystemC components are activated.

The observer interface provides a dummy implementation for all notify functions. Like this the monitors are not forced to implement all the notifications methods. In case the monitor is registered to an event it does not implement the default implementation in the observer interface will issue a warning.

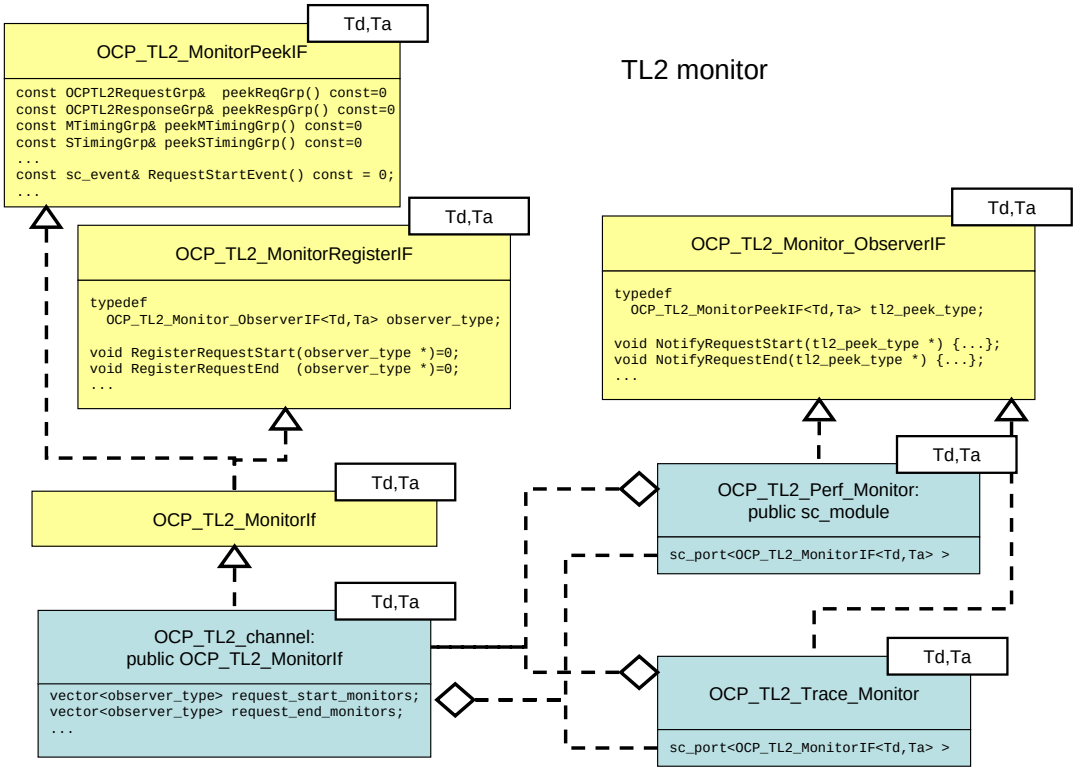

Figure 3: New TL2 Monitor Interface

3.5 Examples

The following examples in the channel package illustrate the usage of the new monitor interface. All these examples are installed at $(OCPROOT)/examples/supplementary

- ocp_tl1_simple illustrates the usage of the TL1 trace monitor and the performance monitor.
- ocp_tl2_simple illustrates the usage of the TL2 trace monitor and the performance monitor.
- ocp_tl3_simple illustrates the usage of the TL3 performance monitor.

Please follow the instruction in the respective README.txt to enable SCV transaction recording.

4 The Performance Monitor

4.1 Motivation

The trace monitor of the SystemC OCP channel primarily addresses the verification of OCP protocol compliance at the cycle accurate TL1 abstraction level. For this purpose the current monitor every cycle dumps the complete status vector of the OCP channel.
For several reasons this trace monitor is not appropriate for the TL2 performance channel. First, the cycle-based status dump is too low level for performance analysis purposes and requires a post-processing step to compile aggregated statistical views. Second, the OCP proprietary dump

format is not compliant with any commercial tool offering, so OCP users need to create their own viewers.
The performance monitor addresses design tasks like architectural modeling and performance simulation. The actual transaction recording is based on the SystemC Verification library [1]. A specific version of the monitor for all channels in the OCP-IP SystemC package is available.
The purpose of this section is to document features, usage, and implementation of the OCP performance monitor. The next section gives a brief overview of the components in the performance monitor. Section 4.3is focused on the user's view of the monitor and defines a set of coding guidelines. Section 4.5 provides more insight into the implementation of the performance monitor.

Note: the SCV library from OSCI is currently not compliant with the SystemC 2.1.v1 library. You can find a compliant library at http://www.greensocs.com/SCVDownload. Additionally you have to add -DSC_USE_SC_STRING_OLD to the compiler flags.

4.2 Overview

The OCP performance monitor enables intuitive performance analysis by means of fast transaction level recording. The analysis instrumentation is based on the SystemC Verification (SCV) standard [1].

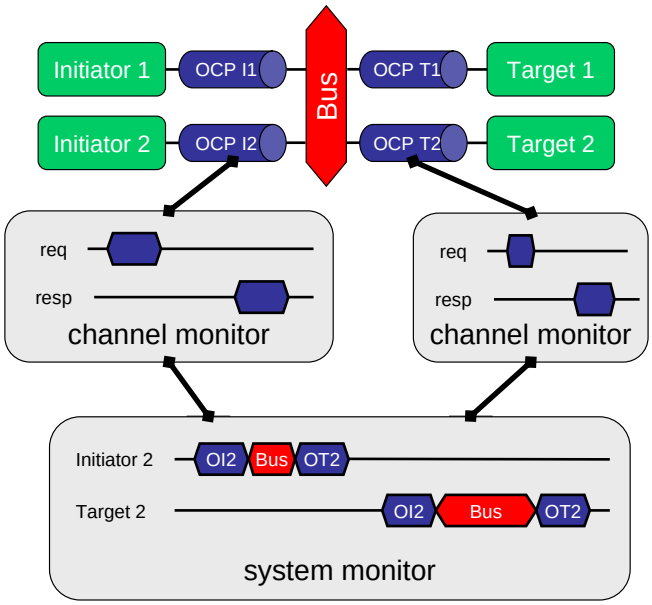

Figure 4: Performance Monitor Overview

As depicted in Figure 4, the performance monitor comprises two hierarchy levels

- The *channel monitor* is attached to a single OCP channel. It records the local transactions on this channel.
- The *system monitor* is attached to every channel monitor. It collects the individual transactions from all channels and compiles them into aggregated transactions. These aggregated transactions show the relation between the individual phases. This enables the system-level analysis of the communication in terms of latency, throughput and bottlenecks.

4.3 The Performance Monitor Module

This section is dedicated to the user view of the performance monitor. It describes the features of the monitor and the actions required by the user to instantiate and to configure the monitor. This section also defines a set of SystemC coding guidelines the user has to follow in order to obtain correct and meaningful results. We use the TL2 version of the performance monitor to discuss the features and usage.

The purpose of the SystemC monitor module is to configure the transaction recording for a specific channel. The available configuration parameters are specified as constructor arguments.

```
14 template <class Tdata, class Taddr>
15 class OCP_TL2_Perf_ Monitor: public sc_module {
16 public:
17   OCP_TL2_Monitor_Port<Tdata,Taddr> p_ocp;
18
19   OCP_TL2_Perf_Monitor (sc_module_name name,
20                       bool channel_recording  = true,
21                       bool system_recording   = true,
22                       bool master_is_node     = false,
23                       bool slave_is_node      = false,
24                       unsigned int  max_nbr_of_threads = 0,
25                       bool burst_recording    = true,
26                       bool attribute_recording= true);
27 };
```

The monitor has one registration port, which has to be bound to the OCP channel.
Figure 5

The list below describes in the constructor arguments:

- **`sc_module_name:`** mandatory constructor argument for all SystemC modules; specifies the name of the monitor module
- **`channel_recording:`** en-/disables the actual transaction recording for this particular channel
- **`system_recording:`** en-/disables the registration of this channel monitor to the system monitor
- **`master_is_node:`** specifies whether or not the SystemC module connected to the master interface of the OCP channel is a communication node. This parameter is only relevant for the system monitor and imposes some requirements on the behavior of the SystemC module (see section 4.3.1.2).
- **`slave_is_node:`** specifies whether or not the SystemC module connected to the slave interface of the OCP channel is a communication node. This parameter is only relevant for the system monitor and imposes some requirements on the behavior of the SystemC module (see section 4.3.1.2).
- **`max_nbr_of_threads:`** a value bigger than zero enables the per-thread recording feature of the channel monitor. If the value is zero, all threads are written into the same transaction stream. If the value is bigger than zero it specifies the maximum number of

OCP threads. In that case the connected master and slave modules are not allowed to send OCP transactions with a thread identifier bigger than `max_nbr_of_threads-1`.

- **`burst_recording:`** By default the channel monitor records every transaction on the OCP channel. This configuration parameter enables the separate recording of complete OCP bursts.
- **`attribute_recording:`** By default the channel monitor records only the address attribute of the OCP transactions. This configuration parameter enables recording all attributes of the request and response transactions.

An illustration of the features of the performance monitor can be found in [2].

4.3.1 SystemC Modeling Guidelines

This section defines a set of SystemC coding guidelines the user has to follow in order to obtain correct and meaningful results.

4.3.1.1 Channel Monitor

Delayed Acceptance of Requests and Responses. In essence the performance monitor records the start and end of any request and response transaction on the OCP channel. The corresponding events in the new OCP TL2 performance channel are the `RequestStartEvent`, `RequestEndEvent`, `ResponseStartEvent` and `ResponseEndEvent`. The user should ensure a non-zero delay between these events in order to record meaningful transactions. Hence the user should not use automatic or immediate acceptance of OCP transactions.
The new OCP TL2 performance channel provides dedicated API delayed `acceptRequest` and `acceptResponse` methods, which take a delay as a function parameter. The delay can be specified as a `sc_time` object or as an integer number representing the number of clock cycles. Users of the layered OCP channels should call `wait()` between receiving and accepting a transaction.

4.3.1.2 System Monitor

Forwarding of the Transaction Handle Member. The system monitor essentially requires the preservation of the new `TrHandle` member in the transaction data structures throughout the end-to-end transaction.
This means any *communication node* like a bus has to forward the value of this member in the request and the response path. In case the complete transaction data-structure is copied nothing extra needs to be done, since the `TrHandle` member is copied in the copy-constructor of the request- and response-groups. A communication node is also supposed to forward all incoming transactions. In case it consumes a transaction it has to cancel this transaction through the `cancelTransaction` method of the system monitor.
The system monitor relies on the *slave modules* to forward the `TrHandle` member from the request to the response data-structure. Additionally, the response behavior has to be compliant with the 2.x version of the OCP protocol, i.e. a slave is not supposed to send a response in case of `OCP_MCMD_IDLE, OCP_MCMD_WR` and `OCP_MCMD_BCST` type of transactions.

4.3.1.3 Example

The OCP methodology package available on ocpip.org contains an examples which fully support the performance monitor:

- scv_ocp_tl2_slave.cpp copies the TrHandle member from the request to the response data structure
- ocp_tl2_perf_bus.cpp copies the TrHandle member from the OCP slave port to the OCP master port in the request path and vice versa for the response path
- the top-level netlist main_bus_2m_3s.cpp instantiates and binds performance monitors to all OCP channels.

4.4 Instantiation and Binding

The immediate impact on the user code is restricted to the structural SystemC code, which also instantiates the OCP channels. The complete code of a simple point-to-point system is listed below. The extra code required to use the performance monitor is highlighted by a grey background. The following enumeration discusses the required additions to the code.

```
1  // performance monitor include
2  #include "ocp_tl2_perf_monitor.h"
3
4  #include "ocp_tl2_channel.h"
5  #include "ocp_tl2_master.h"
6  #include "ocp_tl2_slave.h"
7
8  int sc_main(int, char*[])
9  {
10   scv_tr_text_init();
11   scv_tr_db db("ocp_db");
12   scv_tr_db::set_default_db(&db);
13
14   typedef unsigned int Ta;
15   typedef unsigned int Td;
16   // Creates the OCP TL2 channel
17   OCP_TL2_Channel<Ta,Td>  ch0("ch0");
18   // Set the OCP parameters for this channel
19   MapStringType  ocpParamMap;
20   readMapFromFile("ocpParams", ocpParamMap);
21   ch0.setConfiguration(ocpParamMap);
22   // specify monitor parameters
23   bool channel_recording  = true;
24   bool system_recording   = false;
25   unsigned int  max_nbr_of_threads = 4;
26   bool burst_recording    = true;
27   bool attribute_recording    = false;
28   // Creates the performance monitor
29   OCP_TL2_Perf_Monitorl<Ta,Td> mon0("mon0", channel_recording,
   system_recording, false, false, max_nbr_of_threads, burst_recording,
   attribute_recording);
30
31   // bind monitor port to the channel
32   mon0.p_ocp(ch0);
33
34   // Creates masters and slaves
35   tl2_slave <Ta,Td> sl1("sl1");
36   tl2_master<Ta,Td> ms1("ms1");
37
38   // Connect masters and slaves using OCP channel
39   ms1.ocp(ch0);
40   sl1.ocp(ch0);
41   // Starts simulation
42   sc_start(2000, SC_NS);
43   return(0);
44 }
```

Figure 6

- **Include (lines 1-2):** The user must include the monitor header file.
- **SCV setup (lines 10-12):** The SCV transaction recording needs to be initialized. This example uses the text based recording of the publicly available SCV library. Additionally the recording database needs to be specified.
- **Instantiation (lines 22-29):** One performance monitor module must be instantiated for each OCP channel that is to be monitored. Naturally the template arguments of the monitor must match with the observed channel. In case of more complex systems with communication nodes special attention must be paid to the **`master_is_node`** and **`slave_is_node`** parameters (see section 4.3.1.1).
- **Binding (lines 31-32):** Finally the monitor port needs to be bound to the channel.

4.5 Overview of the Performance Monitor Implementation

The implementation of the performance monitor is separated into the channel monitor for local point-to-point transaction recording and the system monitor for global transaction recording. The SystemC monitor module is responsible to instantiate this channel monitor object with the correct configuration parameters and to perform the registration. In case no transaction recording is required no SystemC monitor module is bound to the TL2 channel. In this way the configuration of the transaction recording is a property of the SystemC module hierarchy.
The transaction recording itself is implemented as a set of pure C++ objects.
The channel monitor as well as the system monitor contain the SCV specific objects for transaction recording. In this way the TL2 channel itself remains completely independent from the SCV library.
The channel monitor implementation contains a pointer to the system monitor. Since only one instance of the system monitor exists, all channel monitor objects retrieve the pointer to the single system monitor from the system monitor registry.
To support cancellation of transactions in a bus node, the pointer to the system monitor object can also be retrieved by arbitrary bus nodes.

In the 2.1.2 release the implementation has been cleaned up. The separation of the performance monitor into interface and implementation has been removed. The new monitor interface already enables the implementation of arbitrary user-defined monitors. The TL1 and TL2 performance monitors share the same implementation. The TL3 performance monitor has much simpler implementation because of the templated transaction data structure.

4.5.1 Channel Monitor

The transaction recording in the performance monitor is limited to simple start and end of request and response phases. Hence the SCV implementation is rather simple and requires only a limited set of member variables:

- One scv_tr_stream object per request and response for independent recording of transactions
- One scv_tr_handle object per request and response as a handle to the currently ongoing transaction
- A number of scv_tr_generator objects to create specific entries in the database, e.g. to differentiate read and write transactions.
- Additional scv objects are instantiated in case the burst recording and/or the thread recording feature is enabled

The transaction recording can be done at the TL2 chunk level or at the OCP burst level. This is achieved by maintaining separate scv_stream objects for chunk recording and burst recording.
Note: The current implementation of the SCV library is quite limited with respect to transaction recording. For example the recording is text based and supports only a single database, i.e. all transactions are recorded in a single file. Implementation of an improved implementation of the SCV transaction recording interface is beyond the current scope of the performance monitor.

4.5.2 System Monitor

The OCP TL2 system monitor collects the transactions from all OCP channels to enable system level performance analysis. The basic idea to register all modules to be either leaf components (initiators, targets) or intermediate communication nodes (buses, bridges). System level transactions start with an initiator request and end ether when the request reaches the target (write, broadcast) or when the response reaches again the initiator. Whenever the transaction reaches a communication node, a new phase of the transaction starts.
To enable this kind of system level transaction recording, an additional 'transaction-handle' member is required in the OCP data structure to trace the streaming of transaction through the system. This transaction-handle needs to be forwarded by all communication nodes and slave modules (see section 4.3.1.2).

Obviously the implementation of the system monitor is more complex than the channel monitor. Basically, the system monitor maintains a map of scv_transaction_handle objects for all ongoing transactions. Every time a transaction recording method is called, the system monitor either start or ends a transaction or starts or ends a new phase of the transaction.

5 Future Work

The new access interface is most likely only an intermediate step towards a standardized analysis framework for verification and architecture exploration. The OSCI TLM working group is currently standardizing on the basic analysis instrumentation concepts. Once this activity is finished the OCP-IP SLD working group will most likely update the monitor interface to the new standard.

6 Related Documentation

[1] SystemC Verification Standard Specification, Version 1.0e, May 16, 2003
[2] A New SCV Compliant Transaction Recording Monitor for the SystemC OCP Channel, presentation at the OCP-IP pavilion at DATE 2005