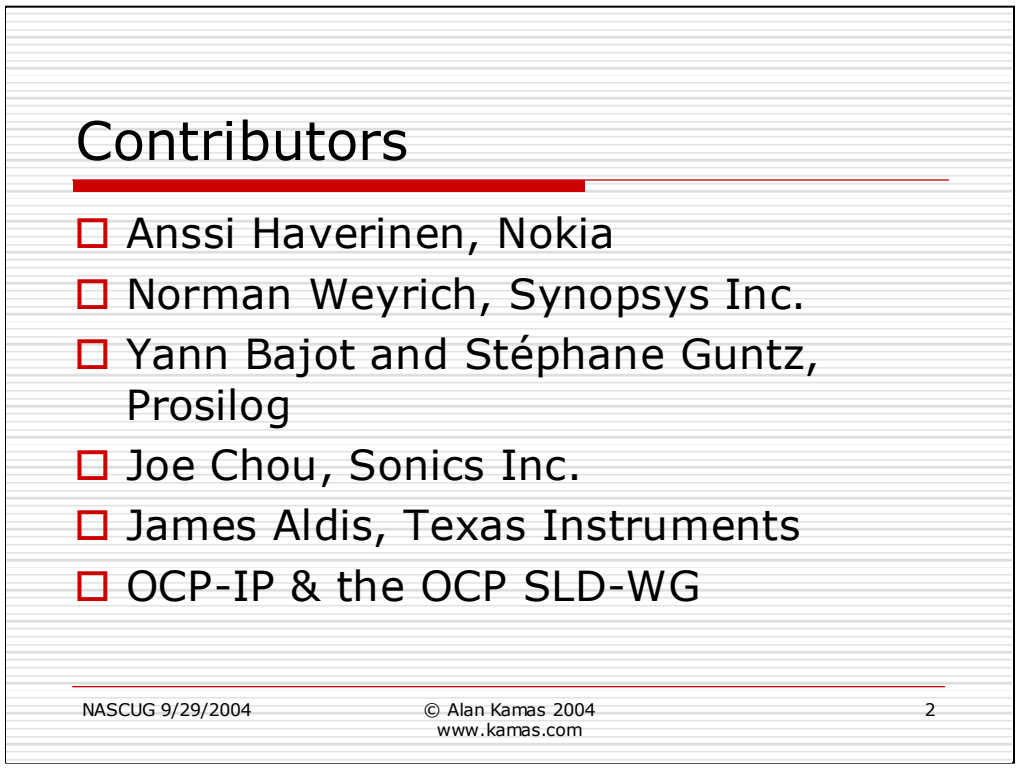
The SystemC OCP Models

An Overview of the SystemC Models
for the Open Core Protocol

Alan Kamas

This talk will give you an overview of the OCP (Open Core Protocol) connection and the set of SystemC models built to simulate it.



The OCP Models are the work of many people.

My work on the project was paid for by Sonics, Inc. Sonics, as with the other companies here, has donated its SystemC work to the project.

In addition to providing code and design, Anssi Haverinen is chair of the OCP System Level Design Working Group which is responsible for the channel models.

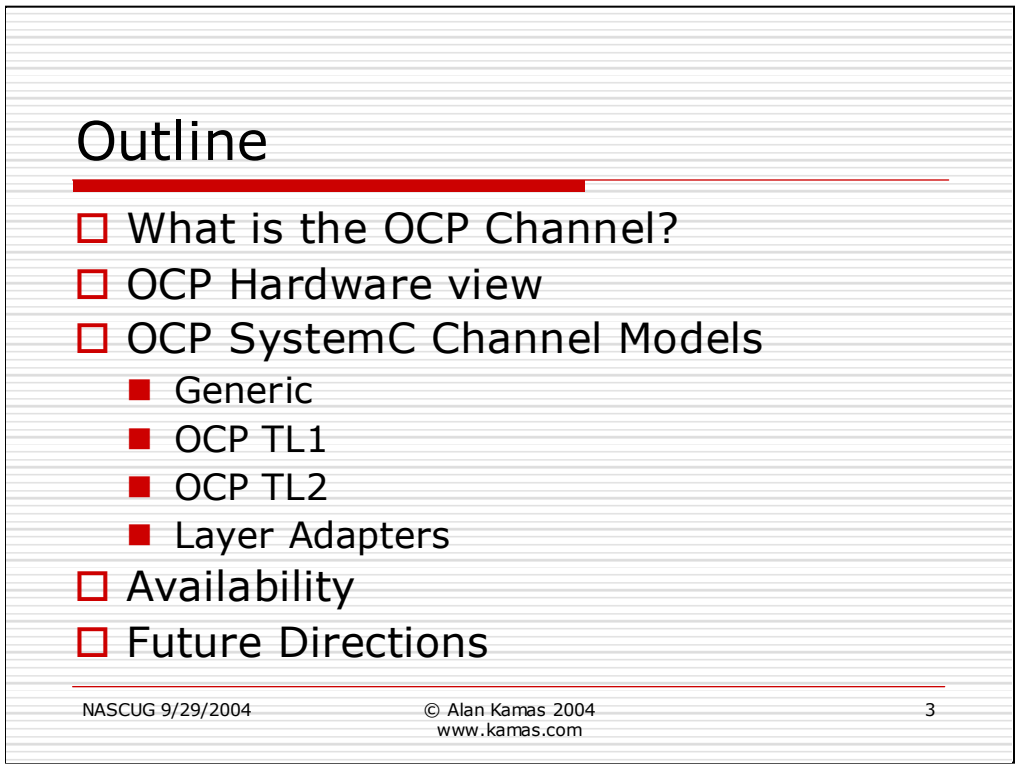
Norman Weyrich (formerly of Synopsys Inc.) was one of the original developers and worked on the generic model.

Yann Bajot and Stephane Guntz have built layer adaptors and contributed to the TL2 models.

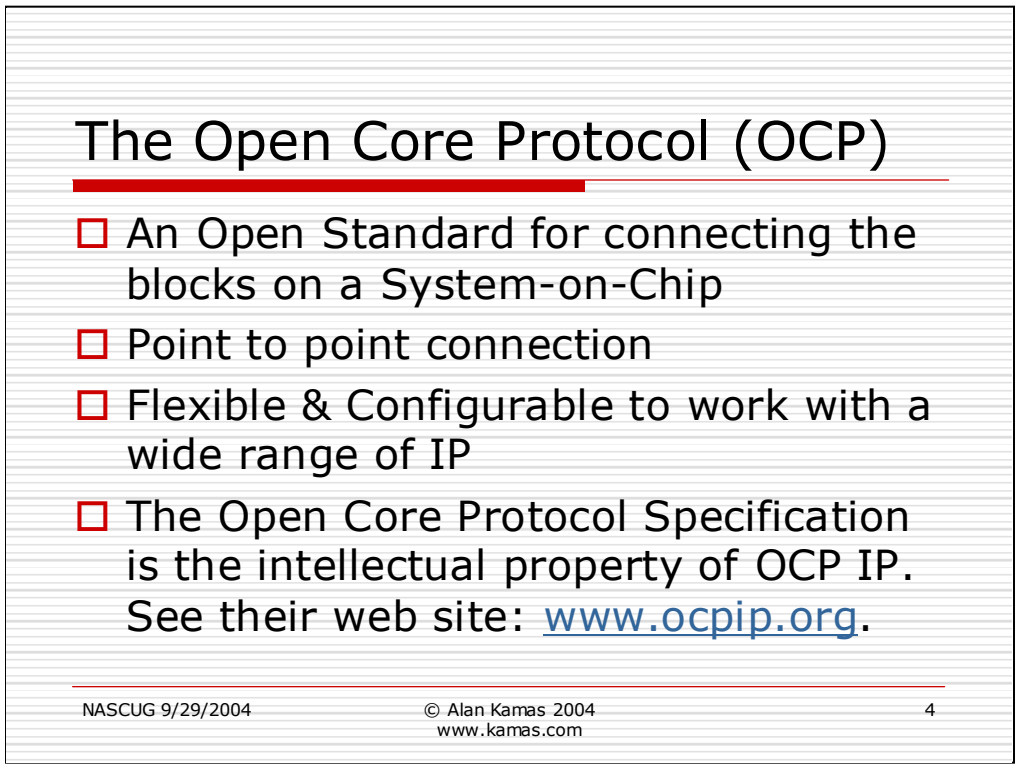
Joe Chou of Sonics has contributed to design and code of the OCP models.

James Aldis has contributed work on the channel monitor.

There are many more members of the OCP SLD-WG from many other companies that have contributed to the project.



First we'll cover a brief introduction to OCP channel and how it works at the hardware level. Then we'll look at the SystemC models for the OCP channel covering some of the decisions that went into their design. Finally, we'll take at peek at where the work is headed in the future.

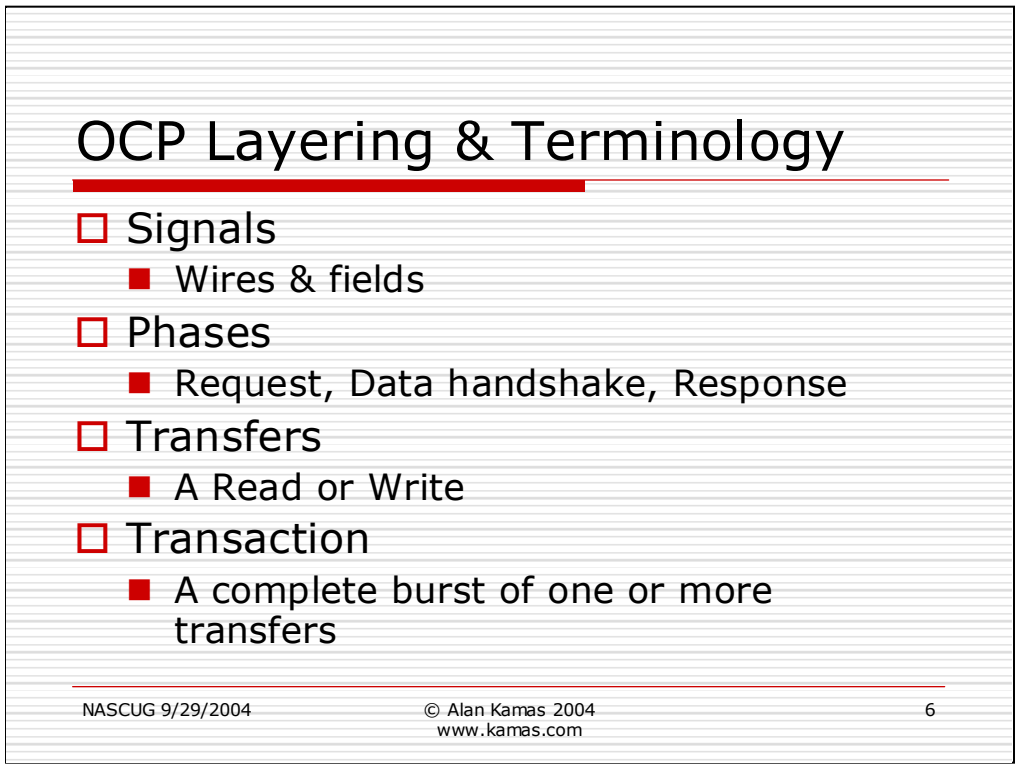


The OCP (Open Core Protocol) is an open standard for connecting the blocks of a system on chip. The Open Core Protocol is point to point – connecting one block to another – and is not a bus specification (connecting many blocks to many blocks). The standard was developed to be configurable and flexible to work with many different types of IP.

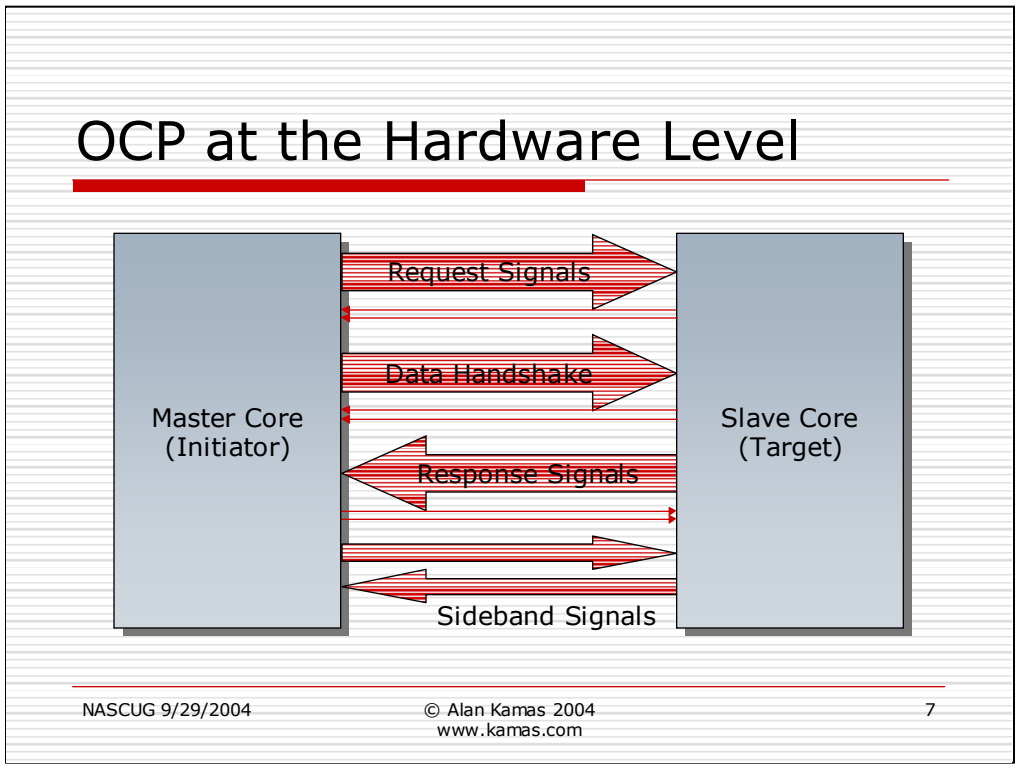
OCP-IP Public Member List

- 3rdeye Technology
- Accent
- Acculent Corporation
- Advanced Architectures
- Alcatel
- Amphion
- ARC International
- Artisan Components
- ATI Technologies
- AXYS Design Automation
- Beach Solutions
- Bitboys
- Broadcom
- Cadence Design Systems
- CAST, Inc.
- Chip Implementation Center
- ControlNet India
- CoWare
- DAFCA
- Denali
- Design And Reuse
- Dolphin Integration
- Duolog
- eASIC
- ECSI
- EDA Cafe
- eInfochips
- Esterel Technologies
- First Silicon Solutions
- Flextronics Semiconductor
- GDA Technologies
- GeoLogic
- HDL Design House
- Hughes Network Systems
- Icera Semiconductor
- IDT
- Infineon
- Imagination Technologies
- Kawasaki
- LIRMM
- LSI Logic
- LTRIM Technologies
- Manhattan Routing, Inc.
- Mentor Graphics
- Micronas
- MIPS Technologies
- National Tsing Hua University
- NEC
- NoBug
- Nokia
- Paradigm Works
- Philips Semiconductors
- PUCRS
- Prosilog
- QThink
- Qualis
- Royal Institute of Technology
- Si2
- Siemens
- Silicon Interfaces
- Silicon Designs International
- Silicon & Software Systems
- Siligence
- Siroyan
- Sonics
- SpiraTech
- STARC
- STMicroelectronics
- Summit Design
- Synergetic Computing Systems
- Synopsys
- Tampere University of Technology
- Technical University of Denmark
- TechOnLine
- Tensilica
- Texas Instruments
- TNI-Valiosys
- Toshiba Semiconductor Group
- Tower Semiconductor
- TranSwitch
- TSMC
- UFCG
- UMC
- University of British Columbia
- UC Berkeley
- Verisity
- Virtual Component Exchange
- Virtual IP Group
- Virtual Silicon
- VSIA
- White Eagle Systems Technology
- WiQuest Communications
- Yamaha Corporation
- YogiTech

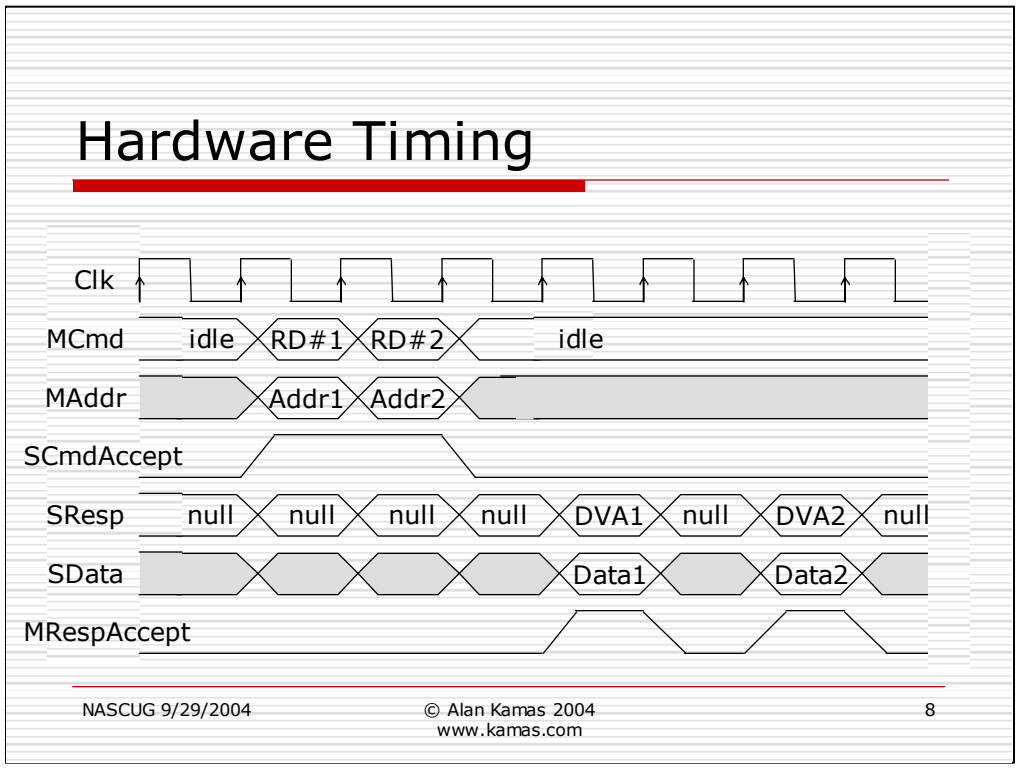
Many companies and universities are involved with the Open Core Protocol



A quick overview of OCP terminology.



So, what does this OCP connection look like? At the hardware level, the Open Core Protocol is a collection of signals between the two cores. There is a handshake path for requests, another (optional) handshake path for request data, and a separate path for responses (which includes the response data). Note that there are also a set of sideband signals between the cores. These signals include interrupt, flags, and error signaling.



One interesting aspect of the OCP connection is that it allows responses to be pipelined. That is, it is possible to send a number of requests before receiving any responses. Here, read request #1 is sent, and then read request #2 is sent and then the response #1 is sent and then response #2. The responses may be sent anytime after the request has been received. Conceivably, this could be a long time after. The only restriction is that responses are returned in the same order as the requests.

Response Pipelining

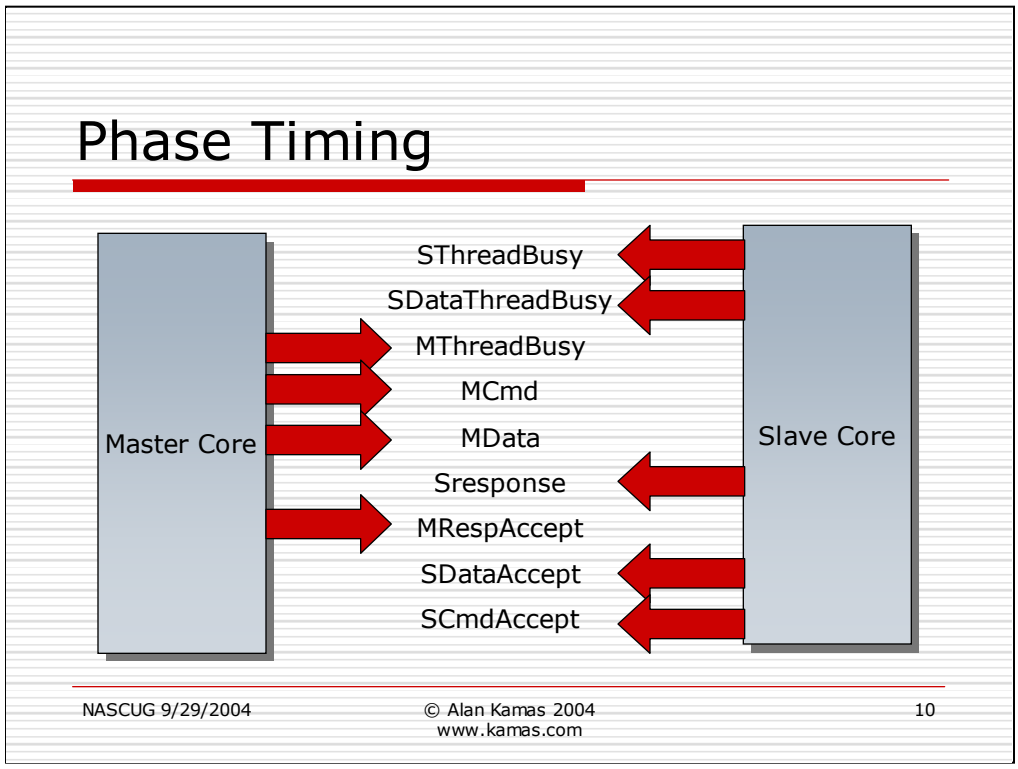
- Next response may not match last request
- Very simple channel models will not work with response pipelining:

```
MyData = channel->read(myAddress);
```


Jumping ahead here, one aspect of pipelined responses is that it limits the type of SystemC model you can have. Blocking calls like this one:

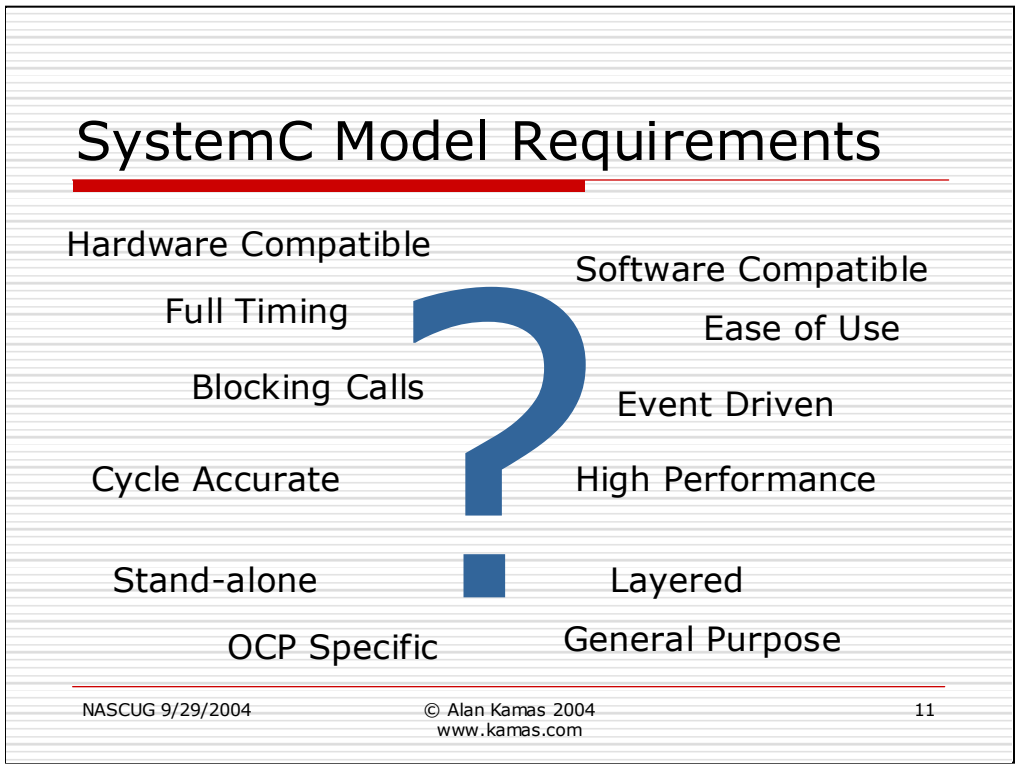
myData = read(myAddress);

Don't always work since the response data you are getting could be from a previous command. You could model the OCP without the separate request and response paths but then you would be missing some of the flexibility and throughput of the OCP connection.

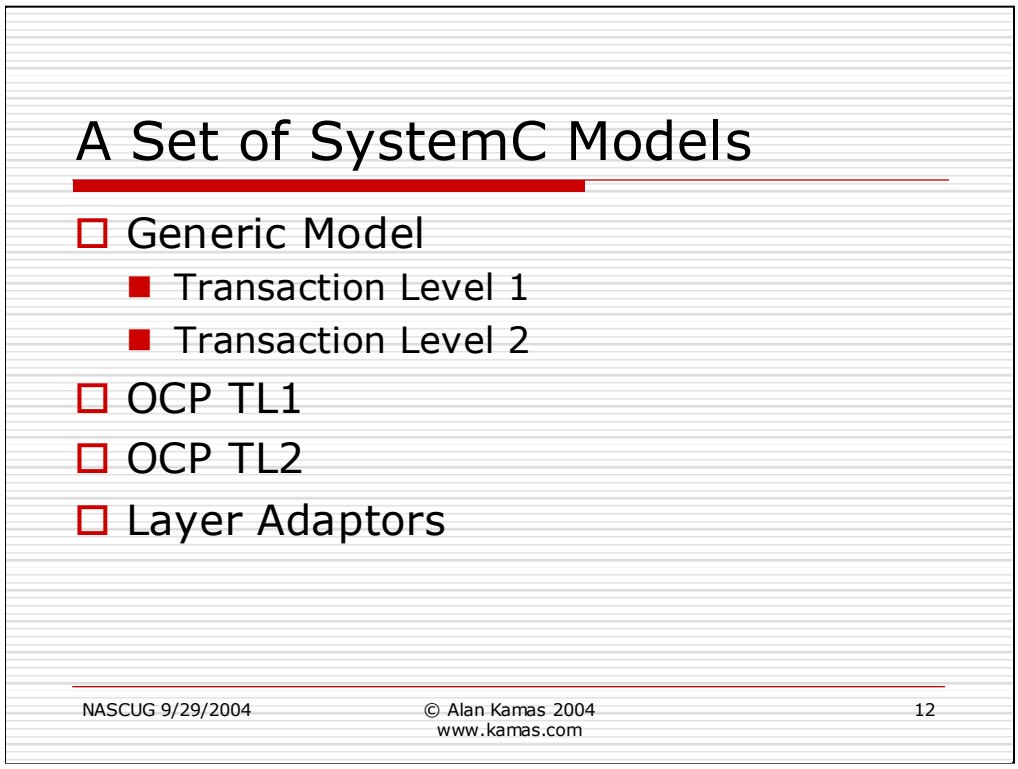


Back to the hardware model.

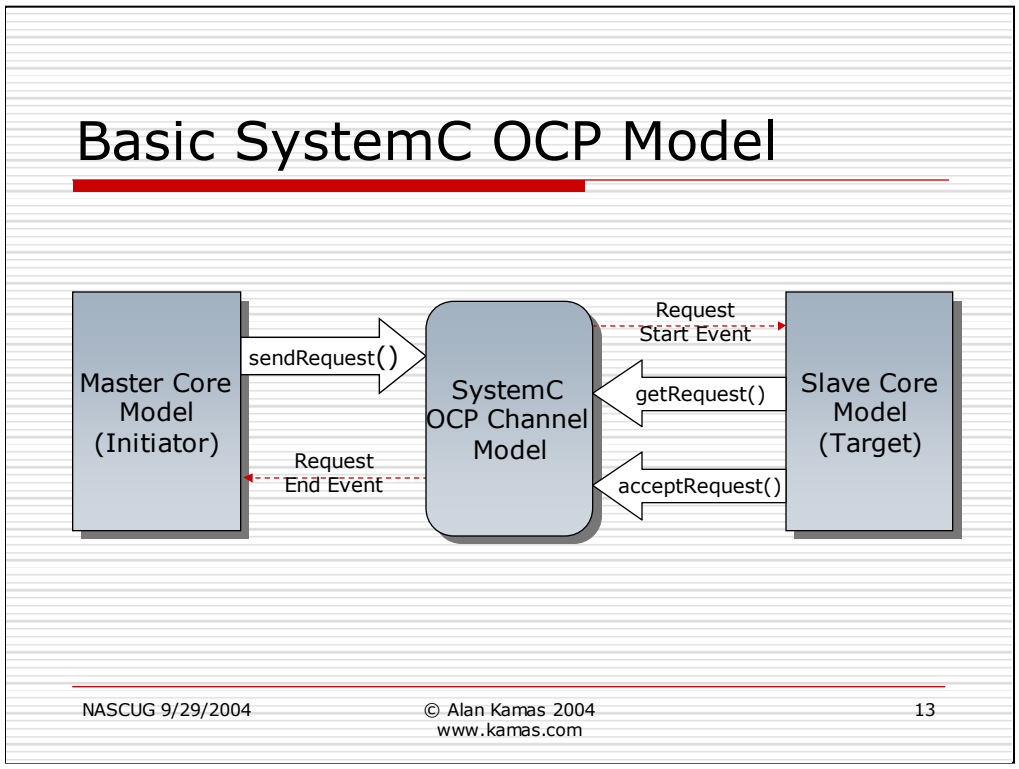
In addition to response pipelining, communications over the OCP connection must follow a certain order. Here the busy signals must be sent before the request. The data must start after the request, and the response comes after that. A cycle accurate model of the OCP connection must follow the same ordering.



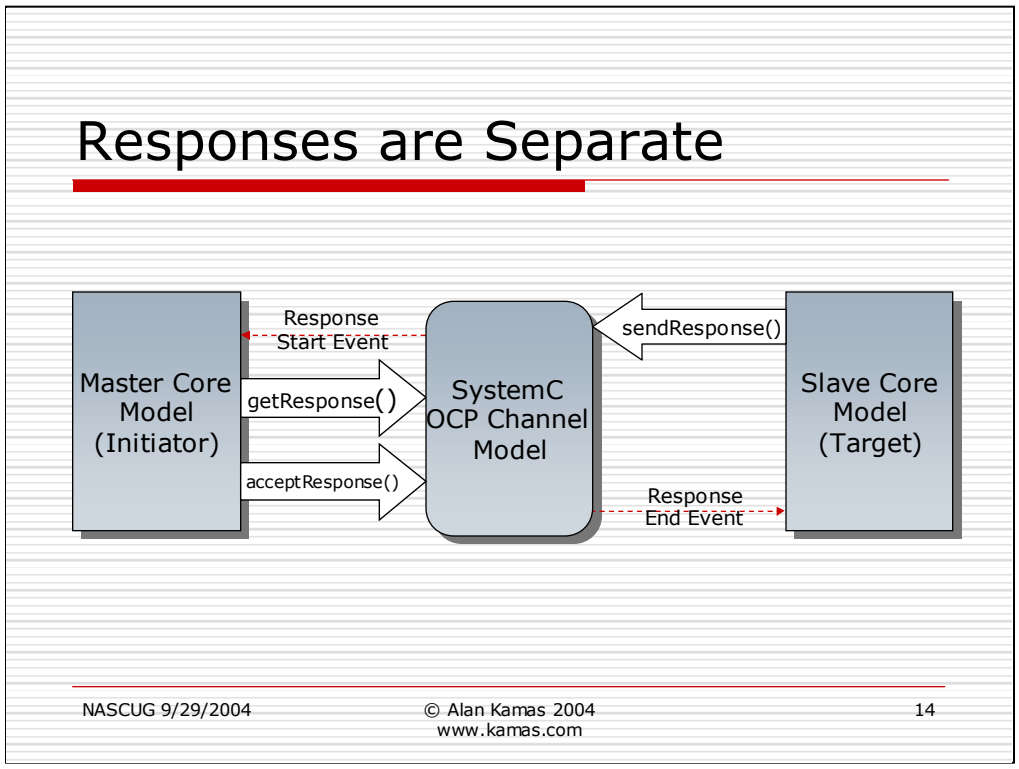
Now that we know a little bit about how the Open Core Protocol works in the hardware, we're ready to consider the SystemC models. The question then becomes, what sort of SystemC model is needed? Is the model for hardware interfacing or architecture exploration? Should it be cycle accurate or should it run at a more abstract level? Should the phases of a transfer be observed or is model performance the priority? Should the model be stand-alone or layered?



The answer is YES! To meet different modeling needs, there are OCP channel models at different abstraction levels layered upon a generic transaction level channel. The OCP channel models include a generic transaction level channel model for sending requests and responses of a templated data type back and forth. Built on top of this are the OCP specific channels. The OCP Transaction Level 1 model is a low level, cycle-accurate model with all of the phases and timing of the hardware. The OCP Transaction Level 2 model runs at a higher level of abstraction by combining the request and data phase and allowing whole bursts to be sent as single TL2 commands.



All of the models have a request path through the channel. The master calls a function (sendRequest) in the channel model through a port that is connected to the channel. The channel takes the request and triggers an event (RequestStartEvent) that the Slave is sensitive to. This may wake up a SystemC process in the slave which then calls the getRequest function in the channel. At a possibly later time, the Slave may call the channel's acceptRequest() function to accept the request. The channel then triggers the RequestEndEvent event. The Master is sensitive to this event as it lets the Master know that old request is finished and a new request may now begin.



A separate path for responses allows for response pipelining.

Generic Channel Model

- Based on initial work of Generic Transaction Level Channel.
- Transfer Data type and format is templated.

Useful for experimentation with channel modeling.

Hooks to build connections with EDA tools

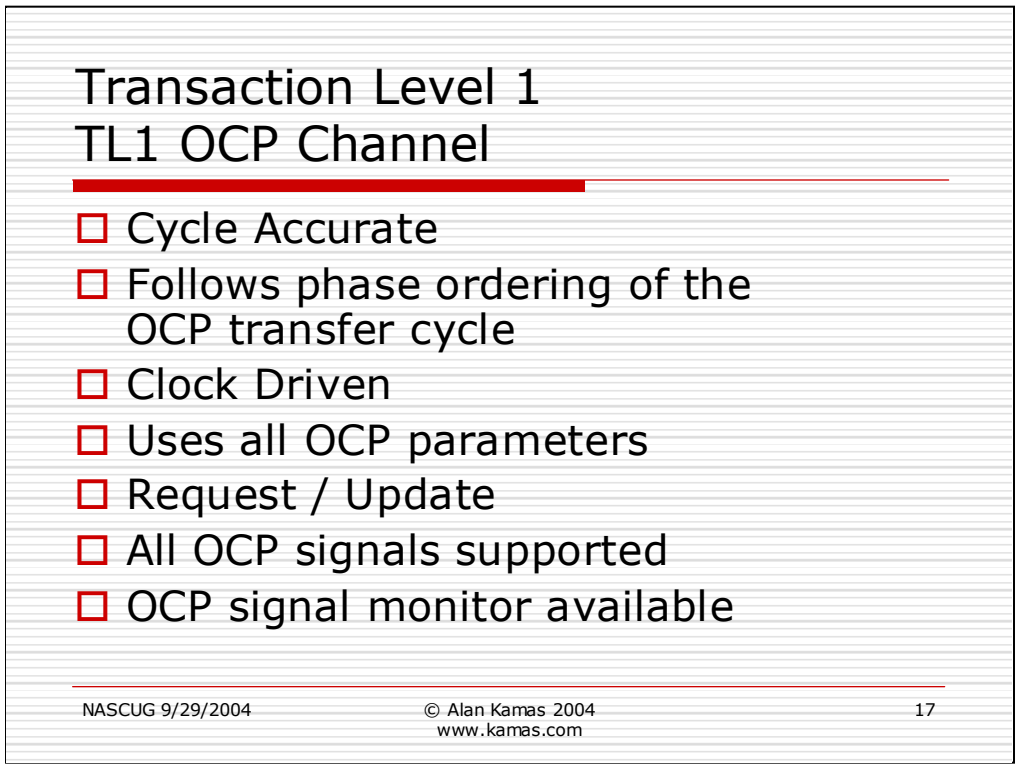
Underlying layer for TL1 and original TL2

Generic Channel Code

```
// Master Core Model Example.
// Send a read command over generic channel
// Channel uses TDataCl as its data class

TDataCl *cDataPtr;
cDataPtr = MasterPort->GetDataCl();
cDataPtr->MputMAddr(Addr);
MasterPort->MputReadRequest();
```


An example of a request being sent over the generic channel. The generic channel is instantiated with a TDataCl class type template. Thus the generic channel can carry any type of data, address, etc. Here, the values for the next request are loaded into the channel and then the "MputReadRequest" command is called to start the new request and to toggle the previously loaded data values unto the channel.



The Transaction Level One

This OCP SystemC model attempts to fully capture the timing and ordering of the hardware OCP connection. It is built on top of the generic channel with a data class for full OCP support and OCP specific commands for ease of use.

OCP TL1 Code

```
// Send a write request over the OCP TL1
// Channel (using blocking commands)

OCPRequestGrp<Td,Ta> req;
req.MCmd = OCP_MCMD_WR;
req.MAddr = 0x0401;
MasterPort->startOCPRequestBlocking(req);
OCPDataHSGrp<Td> datahs;
datahs.MData = myData;
MasterPort->startOCPDataHSBlocking(datahs);
```


Here is a sequence of commands to send a write request over the OCP TL1 channel. Note that here the channel uses an OCP specific data structure to hold the request. This structure supports all of the OCP request signals including MCmd (command type), and MAddr (the address). The OCP TL1 channel model also has an independent data path. Here, the write data is sent after the write command.

OCP TL1 Code – non blocking

```
// Send a read request over the OCP TL1
   Channel (using non-blocking commands)

OCPRequestGrp<Td> req;
req.MCmd = OCP_MCMD_RD;
req.MThreadID = 1;
req.MAddr = 0x0401;
if (MasterPort->sendOCPRequest(req)) {
   cout << "Request Sent" << endl;
}
```


The OCP TL1 interface supports both blocking and non-blocking calls.

Transaction Level 2
TL2 OCP Channel

- Models OCP specific data flow through the channel
- Faster / Greater Throughput
- Commands to send an entire burst of data at a time.
- Request & Response only – no Data Handshake
- Timing Approximate

The OCP Transaction Level 2 channel is designed to work at a higher abstraction level. The phase structure is simplified, consolidating the data handshake phase into the request group. The request and response groups are expanded so that an entire burst of OCP transfers may be sent with a single command. The TL2 channel has better performance and greater throughput but the downside is that the timing will not be cycle accurate as it is in TL1.

OCP TL2 Channel Code

```
// Send a write burst over the OCP TL2
   Channel (using blocking commands)
Td myData[4];
myData[0]=0; myData[1]=1; myData[2]=2;
   myData[3]=3;
OCPRequestGrp<Td> req;
req.MCmd = OCP_MCMD_WR;
req.MAddr = startAddr;
req.MDataPtr = myData;
MPort->sendOCPRequestBlocking(req,4,true);
```


Here a burst write request of 4 data words is sent over the OCP TL2 channel. The whole burst is sent as a single command. Note that, as with the OCP TL1 channel, both blocking and non-blocking calls are available.

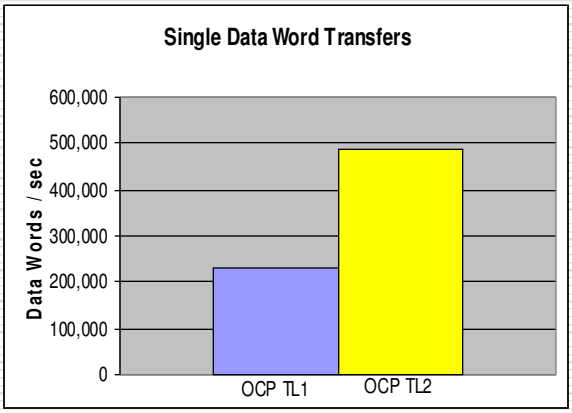
Performance OCP TL1 vs TL2

One at a time

Write of one data word then Read of one data word.

OCP TL1: 232,000 data words / sec

OCP TL2: 486,000 data words / sec



NASCUG 9/29/2004 © Alan Kamas 2004 www.kamas.com 22

In this benchmark test the master sends a write command and then a read command. This sequence is looped 10 million times.

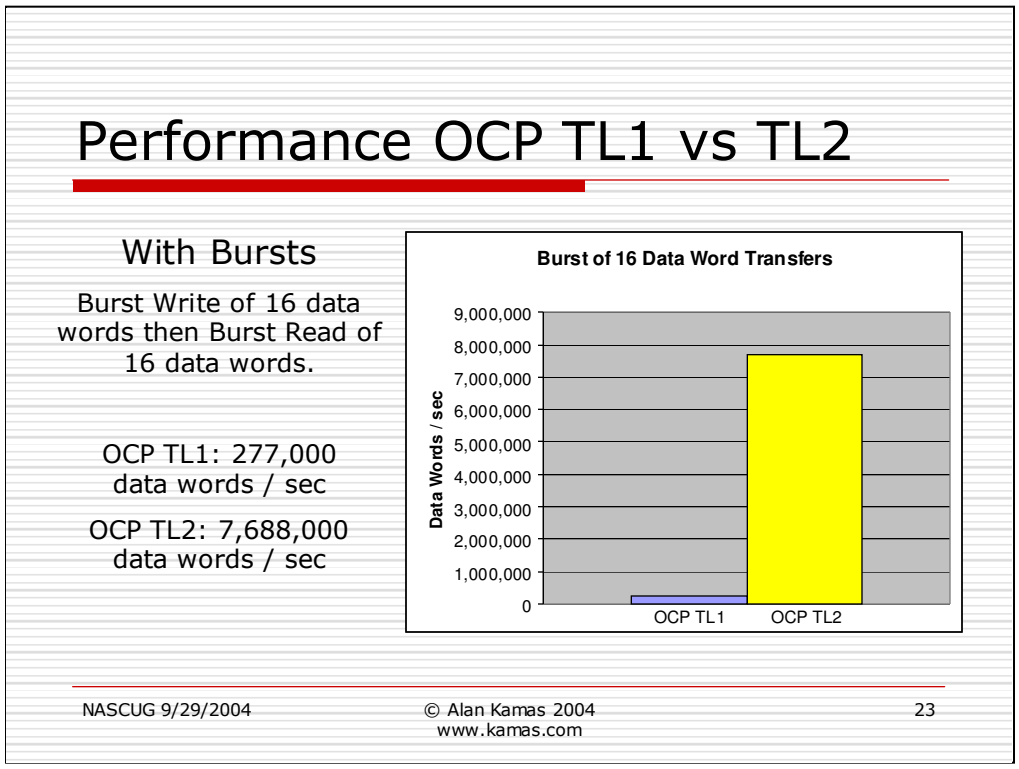
As you can see from the chart, the TL2 channel has about twice the throughput of the TL1 channel.

For TL1: data handshake is used for the writes, which slows it down. Also, the TL1 master uses one SystemC thread method (to handle requests) while the TL2 models are completely event driven.

Here are the test details:

In each of these tests, a simple master is connected to a simple core through the OCP channel model. The OCP Channel has data handshake with command, data, and response accept. For writes, the command goes first and then the data goes in the next cycle.

These tests were run on a dual processor Pentium III 1.26GHz machine. All simulations ran on a single processor. The tests were compiled under Linux with gcc 2.96 using the "-O" flag and the standard OSCI SystemC library. The first test is a single data word write command followed by a single data word read.



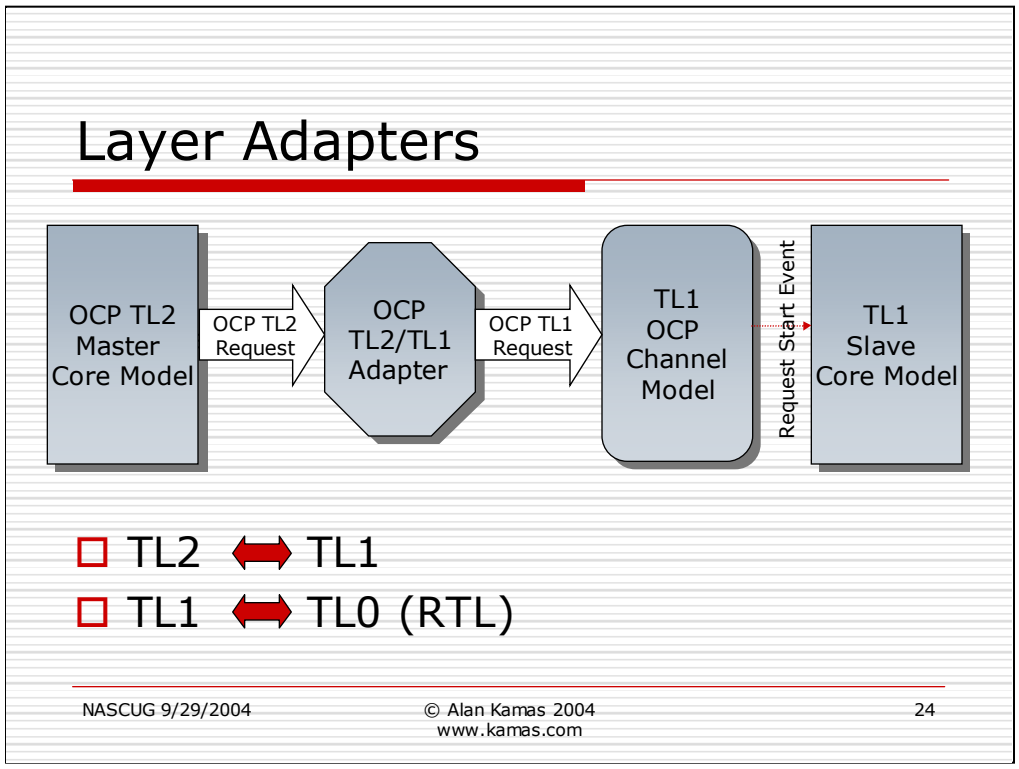
The real speed up from the TL2 channel is how it handles bursts. Here the master sends a burst 16 write command followed by a burst 16 read command, and the sequence is repeated.

The OCP TL1 channel sends each command of the burst individually. A burst 16 read is 16 read commands and 16 responses. The OCP TL2 channel allows the cores to send the whole burst as one command.

Here are the test details:

In each of these tests, a simple master is connected to a simple core through the OCP channel model. The OCP Channel has data handshake with command, data, and response accept. For writes, the command goes first and then the data goes in the next cycle.

These tests were run on a dual processor Pentium III 1.26GHz machine. All simulations ran on a single processor. The tests were compiled under Linux with gcc 2.96 using the "-O" flag and the standard OSCI SystemC library. This second test is a burst 16 data word write command followed by a burst 16 data word read.



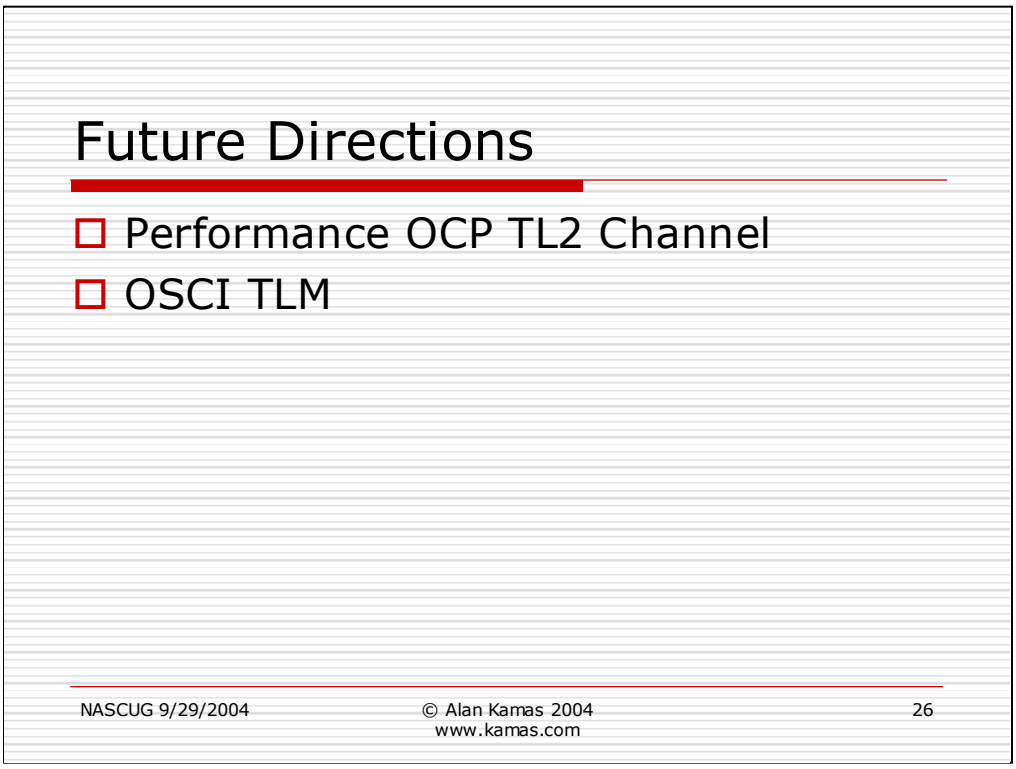
One problem with multiple channel model types is getting cores written for different models to work together.

The layer adapters convert the interface from one OCP layer to another. Here a Core model has been written to work with the OCP TL2 channel. The layer adapter allows it to work with an OCP TL1 channel.

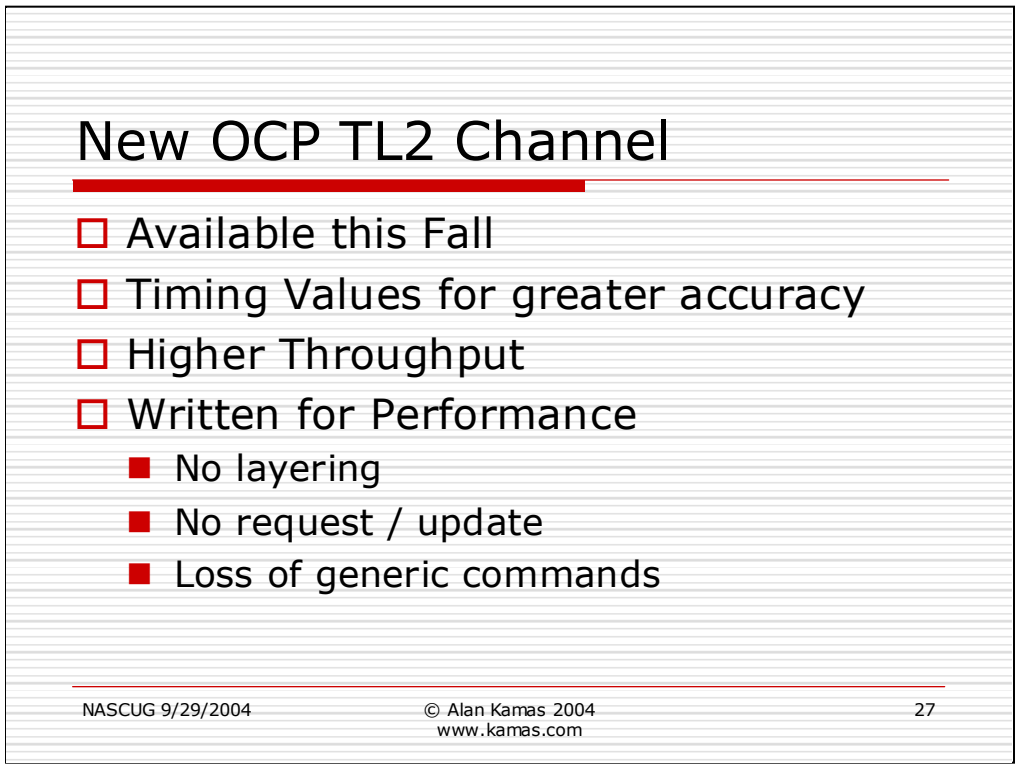
The layer adapters may even be chained together allowing an OCP TL2 core to communicate with an RTL (TL0) channel.

Getting the OCP Channel Models

- Available for download from OCP-IP:
  http://www.ocpip.org/socket/systemc
- Click on the license
- Fill in the form
- Read the email
- Download the Channel Models



A quick look into the future for the OCP channel models



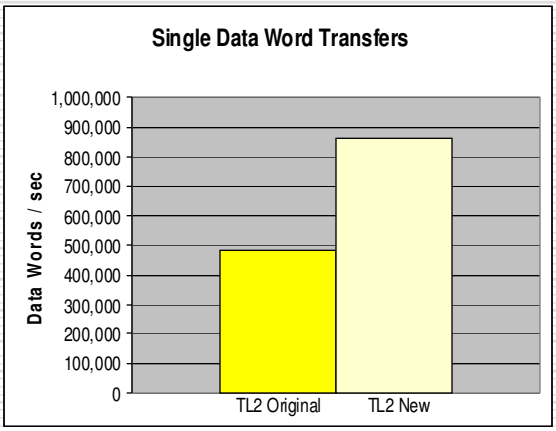
There will be a new performance version of the OCP TL2 available this Fall. The new TL2 channel adds timing points that are passed through the channel so that approximately cycle accurate timing may be achieved at the TL2 level. The new OCP TL2 channel was rewritten with performance as the goal and runs faster, however because it is not layered, it does not support all of generic level commands.

New TL2 Channel Performance

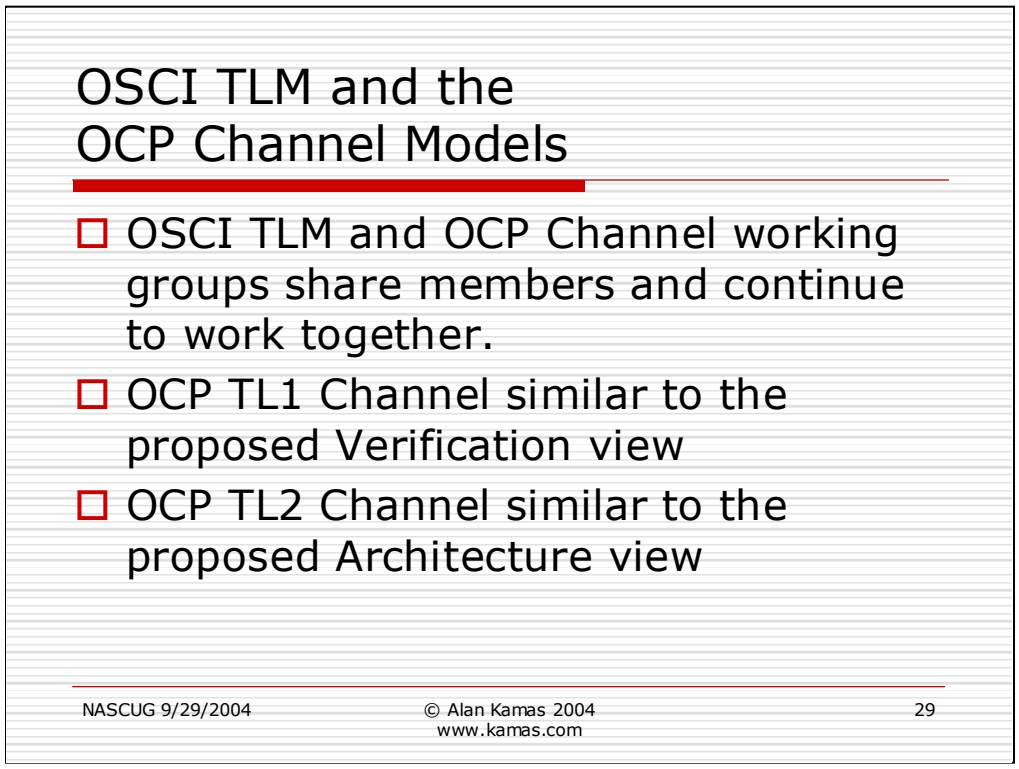
Write of one data word then Read of one data word.

OCP TL2 Orig: 486,000 data words / sec

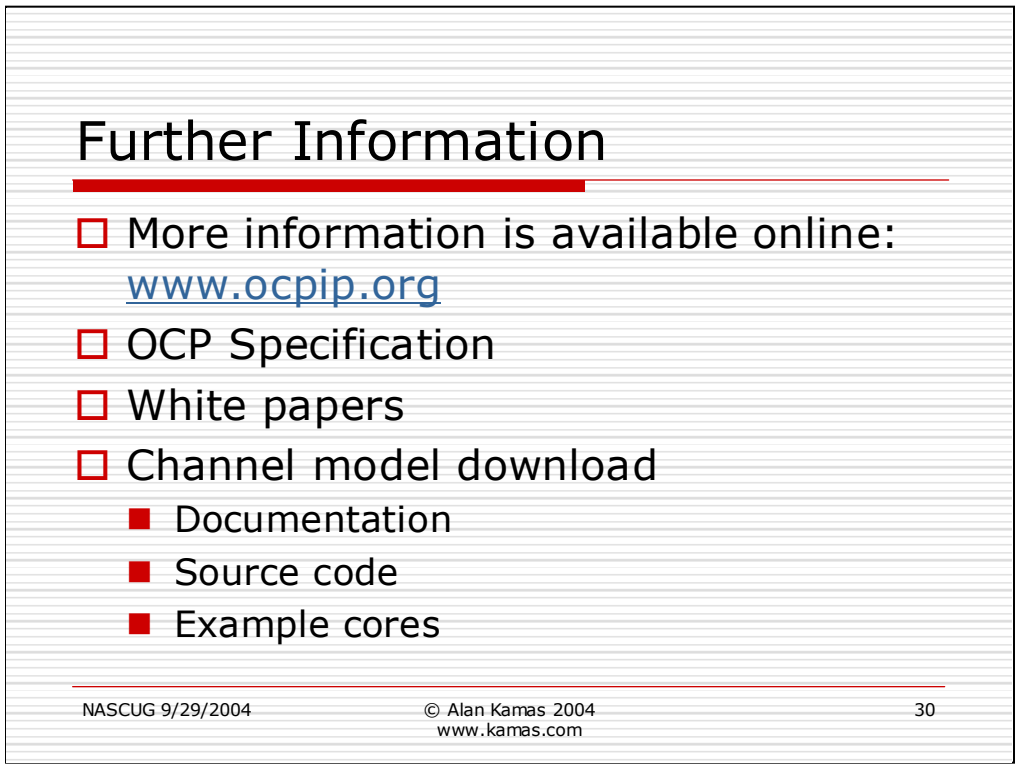
OCP TL2 New: 865,000 data words / sec



The new TL2 channel has about 75% more throughput then the original channel. This is true for burst transfers as well.



The Open SystemC Initiative is also working on developing transaction level models in SystemC.



There is more information available online.