Transaction Level Modeling with SystemC

Thorsten Grötker
Engineering Manager
Synopsys, Inc.
Outline

- Abstraction Levels
- SystemC Communication Mechanism
- Application 1: Generic Transaction Level Communication Channel
- Application 2: Transaction Level Modeling of the AMBA AHB/APB Protocol
Abstraction Levels:

- **Transaction Level**
  - Layer 3: Message Layer
    - Model (un-)timed functionality
    - Point-point communication
  - Layer 2: Transaction Layer
    - Model/analyze SoC architecture, start SW development
    - Estimated timing
  - Layer 1: Transfer Layer
    - Cycle true but faster than RTL
    - Detailed analysis, develop low-level SW

- **Pin Level**
  - Layer 0: Register Transfer Level
## SystemC Language Architecture

### Methodology-specific channels

#### Elementary Channels
- Signal, Timer, Mutex, Semaphore, FIFO, etc.

- **Time**
- **Concurrency**
- **Modules**
- **Processes**
- **Interfaces**
- **Ports**
- **Channels**
- **Events**
- **Event-driven sim. kernel**

### Data Types
- 4-valued logic (0, 1, X, Z)
- 4-valued logic-vectors
- Bits and bit-vectors
- Arbitrary-precision integers
- Fixed-point numbers
- C++ user-defined types
- C++ built-in types (int, char…)

### C++ Language Standard
SystemC Communication Mechanism: How?

- Separate Functionality from Communication
  - Functionality: implemented in Modules
  - Communication: implemented in Channels

- Interface Method Calls (IMC)
  - The collection of a fixed set of communication Methods is called an Interface
  - Channels implement one or more Interfaces
  - Modules can be connected via their Ports to those Channels which implement the corresponding Interface ( sc_port<interface> )
**Primitive Channel**

Module1

```
Module1::process()
{
    port->write(42);
}
```

Module2

```
Module2::process()
{
    x = port->read();
}
```
Hierarchical Channel

Channels can be hierarchical, i.e. they can contain processes, ports, modules and channels.
Transaction Level Modeling Example

Heavy use of interface method call scheme.

Bus and slaves are implemented as hierarchical channels.
Outline

- Abstraction Levels
- SystemC Communication Mechanism
- Application 1: Generic Transaction Level Communication Channel
- Application 2: Transaction Level Modeling of the AMBA AHB/APB Protocol
Generic Transaction-Level Channel

- **Background**
  - Developed by Nokia and Synopsys in cooperation with TI and Sonics for OCP

- **Requirements**
  - Point-to-point connection
    (module ↔ generic channel ↔ module)
    - Data passing and synchronization
  - Interface independent of communication protocol
    - Protocol to be implemented by the user
    - Can be used for Layers 1, 2, and 3
Generic Channel Interface (simplified)

- **Initiator ("Master")**
  - TdataCl* GetDataCl();
  - bool MputRequest*();
  - bool MgetResponse*(bool release);
  - void Mrelease(time);

- **Target ("Slave")**
  - TdataCl* GetDataCl();
  - bool SgetRequest*(bool release);
  - bool SputResponse*();
  - void Srelease(time);

**TdataCl:**
- template parameter
- depends on application and communication protocol being used
Flow of Events

```c
data = port->GetDataCl();
data = port->GetDataCl();

Master

Slave

NB: data type is likely to depend on the protocol.

data->input = 42;
MputWriteRequestBlocking();

SgetRequestBlocking(true);
int mem = data->input;
if (mem != 0)
    data->response = true;
else
    data->response = false;
SputResponseBlocking();

MgetResponseBlocking();
if (data->response) ...
```
The bus protocol has to be implemented by the user; is it not part of the generic interface.
Transaction-Level Modeling of the AMBA AHB/APB Protocol

- **Background**
  - implemented in the spirit of “simple_bus”
  - developed in close cooperation with ARM

- **Requirements**
  - cycle-accurate (Layer 1)
  - optimal simulation speed
  - ease of use (blocking interface)
  - integration of low-level models (non-blocking interface)
  - enable SW debugging (debug interface)
Generic Interfaces

M1
M2
M3

BUS

S1
S2

Arbiter
Decoder

C
Dedicated AMBA Interfaces

M1  M2  M3
Arbiter
Decoder
BUS
S1  S2
Dedicated AMBA Interfaces

Master → Bus Communication: Interface Method Call

Arbiter
Decoder

M1  M2  M3

BUS

S1  S2
Dedicated AMBA Interfaces

Bus → \{Arbiter, Decoder, Slave\}: Interface Method Call
AHB Master-Bus Interface

- **Blocking**
  - `bool burst_read(id, data, start_address, burst_length, burst_mode, num_bytes);`
  - `bool burst_write(id, data, start_address, burst_length, burst_mode, num_bytes);`

- **Non-Blocking**
  - `void request(id);`
  - `bool has_grant(id);`
  - `void init_transaction(id, read_mode, address, ...);`
  - `void set_data(id, data);`
  - `bool response(id, status);`
  - `...`

Note that this interface is rather protocol-specific.
AHB Interfaces (cont’d)

- **Direct Bus Interface (Debugging)**
  - bool direct_read(address, data, num_bytes);
  - bool direct_write(address, data, num_bytes);

- **Bus-Slave Interface**
  - void set_data(data);
  - void control_info(burst_mode, transfer_type, ...);
  - void read(address, num_bytes);
  - void write(address, num_bytes);
  - bool response(status);
  - ...

AMBA AHB Components

- AHB bus with external arbiter and decoder
- Interconnection matrix for AMBA multilayer support
- AHB bus monitor
- Processors (ARM 926, ARM 946)
- Pin Level adaptors
- Example masters and slaves
- Example AHB platforms
AMBA APB Components

- AHB-APB bridge with APB bus
- APB bus monitor
- Pin Level adaptors
- APB peripherals
  - Remap and pause controller
  - Interrupt controller
  - APB timer
- Example AHB/APB platforms
Summary

- Two examples out of a continuum of styles
- Generic Transaction-Level Channel
  - Not limited to bus-based systems
  - Can be used for layers 1, 2, and 3
  - Protocol-specific code inlined in connected modules (or partly hidden in methods of TdataCl)
  - Protocol reflected in TdataCl and use of generic interface
- AMBA Example
  - Protocol reflected in interface
  - Optimal simulation speed
    - No intermediate channel objects
    - No process on slave side required (arbiter, address decoder, or simple memories can be accessed via simple function calls)
Same functionality can be modeled in both styles

```
port->transaction_A(foo);
port->transaction_B(bar);
```

vs.

```
data->transaction_type = A;
data->input = foo;
port->MputRequest();
...
data->transaction_type = B;
data->input = bar;
port->MputRequest();
...
```

Communication refinement typically requires code changes or use of adaptors in both cases

Both approaches can be combined
Further information

- **SystemC and Transaction-Level Modeling**
  - OSCI website: www.systemc.org
  - “System Design with SystemC”
- **Generic Transaction-Level Channel White Paper**
  - www.ocpip.org/data/systemc.pdf
- **AMBA models**
  - www.synopsys.com