Understanding Specialized Ports and TLM Interconnects
by Anna Keist, Bill Bunton and David Black

Electronic System Level Services

www.ESLX.com

Presented at the 7th Meeting of the North American SystemC Users Group
Version 1.15
Outline

- Goal
- Review
  - C++ Inheritance
  - C++ Abstract and Interface Classes
  - SystemC Interface
  - SystemC Channel
  - SystemC Port
- Definition of A Specialized Port
- Examples
  - A Real System Application of Specialized Ports
  - Implementation uses TLM 1.0
- How to
  - Designing & using Specialized Ports
- Summary
Goal

Demystify specialized ports by

- Provide the necessary C++ and SystemC background
- Showing examples from a real system
- Incorporate TLM 1.0 standard examples
Basic SystemC Review

C++ Interfaces
SC Interfaces
SC Channels
SC Ports
C++ Inheritance

- Derived class inherits from one or more base classes
- Derived class may override or add methods/data

```cpp
class Base {
    public:
        int read(void) { /* implement */ }
        void write(int v) { /* implement */ }
    }

class Derived : public Base {
    public:
        void write(int v) { /* override */ }
        int foo() { /* add method */ }
    }
```
C++ Abstract & Interface Classes

- Keyword `virtual` in declaration allows polymorphism
- Pure virtual method (\(= 0\)) means no implementation
  - Creates an `abstract` class
  - Compels definition in derived class
- All pure virtual and no data defines an `interface` class (aka API)

```cpp
class Abstract {
    public:
        virtual void write(int v) = 0;
};
class Derived : public Abstract {
    public:
        void write(int v) { /* implement */ }
};
```
SystemC Interface

- Defines only pure virtual methods, no data
- Inherits from sc_interface
- Methods implemented in channels inherit the interface

```cpp
template < typename T >
class tlm_blocking_put_if :
    public virtual sc_interface {

    public:
        virtual void put(const T &t) = 0;
    };
```

Pure virtual method

```
INTERFACE
CHANNEL
```

```
sc_interface
```

©2007 ESLX, Inc.
SystemC Channel

- Provides safe communications between processes
- Implements one or more SC Interfaces

```cpp
class PV_tlm_channel :
  public put_if<packet>,
  public get_if<packet>,
  public sc_channel {

public:
  void put() { ... }
  void get() { ... }
};
```
SystemC Port

- `sc_port<INTERFACE> PORTNAME;`
- Act like pointers to channels outside the `sc_module`
- Polymorphic due to binding to the interface
Definition of A Specialized Port

- A class derived from `sc_port<IF>` providing extra functionality.

"A class derived from a template class `sc_port` that passes a particular type as the first argument to template `sc_port`, and which provides convenience functions for accessing ports of that specific type."


- SystemC provides some predefined partially specialized ports
  - `sc_in<T>, sc_out<T>
  - `sc_fifo_in<T>, sc_fifo_out<T>
  - These primarily provide event finders to allow static sensitivity

Examples
(using TLM 1.0)
A Real System Application

EX

CPU

WAN Model

LAN Model

Phone Model

Router
(Addr Decode)

m0_to_ram

m1_to_ram

m2_to_ram

m3_to_ram

Arbiter

Flash

RAM

RAM

specialized

sc_port<T>

router_port

initiator_port

sc_export<T>
Router port - focus
**Router port - characteristics**

- **router_port** example of TLM utilities is a specialized port
  - Provides some useful functionality for routers
  - Remains a templated class

- Builds up a 1-1 map from port name to port id.
  - Can use `get_port_index(name, id)` to access this mapping
  - Use for building an address map.

- Can also print out the names of everything port is connected to
  - Iterating through the map
  - Useful debugging feature whether or not routing

- Accomplished by overriding the binding operators.
Simply inherit from `sc_port<T>`

```
    //FILE: tlm1.0/utils/router_port.h
    ...
    template < typename IF >
    class router_port
    : public sc_port< IF , 0 >
    {
        ...
    }
```
Functionality not be possible without inheritance

```cpp
// Override binding operator
void operator() (sc_export<IF> &exp) {
    // Assign index in map
    name_map()[exp.name()] = (*this).size();
    sc_port<IF>::operator() (exp);//rest of binding
}

bool get_port_index(const string &port_name, int &p) {
    map<string, int>::iterator i
        = name_map().find(port_name);
    if (i == name_map().end()) { return false;}
    p = (*i).second;
    return true;
}
```

...
Router Port - Binding & usage

- Not much different from ordinary port
  - Except for use of get_port_index() method

```c++
//testbench.h
router_port< tlm_transport_if< REQ, RSP >> r_port;

//testbench.cpp
router->r_port(Mem_Flash->target_port);// binding

//router.h
RSP transport( const REQ &req ) {
  REQ new_req = req;
  return r_port[port_index]->_transport(new_req);
}
```
Initiator Port - focus
**Initiator Port - characteristics**

- **initiator_port** from TLM 1.0 examples is a specialized port
  - Adds convenience methods, read() and write()
  - In some sense represents 2nd layer of TLM
- Shared by several masters in the design
  - Justifies design of the specialized port
//FILE: tlm1.0/examples/basic_protocol/basic_initiator.h

template < class ADDRESS, class DATA >
class basic_initiator_port
 : public sc_port<
    tlm_transport_if<
        basic_request<ADDRESS,DATA>,basic_response<DATA>
    > , N >,
    public virtual basic_if<ADDRESS , DATA >
{
    // Convenience method
    virtual status write( const ADDRESS &a, const DATA &d )
    {
        // implementation uses TLM channel's transport()
    }
};
Designing Specialized Ports

- Determine what functionality needs to be encapsulated
  - Does the required functionality need to specialize `sc_port`?
  - Will more than one module use the convenience provided?
  - Would a simple port encapsulation suffice?

- Create a header file declaring new port class
  - Suggest naming with `_port` suffix
  - Inherit publicly from `sc_port<T>`
  - Declare convenience methods

- Peer review

- Define methods in separate implementation file
Using Specialized ports

- Include the specialized port's header file
- Instantiate the port(s)
- Call methods as needed using `operator()`
Summary

- **Specialized Ports**
  - Simply: A class derived from `sc_port<IF>` providing extra functionality.
  - Required in select situations (e.g. `router_port`)
  - Convenient in other situations (e.g. `initiator_port`)

- **TLM 1.0 Example**
  - Defines and uses two specialized ports
    - Both provide extra functionality, `get_named_index()`, `read()`, `write()`

- **Discussed how to design and implement**
  - Determine what you need
Questions?
Layered TLM API Architecture

User Layer

Convenience Interface

Protocol Layer

Transport Interface

Transport Layer

OSCI TLM 1.0 Standard