The Definitive Guide to SystemC: The SystemC Language

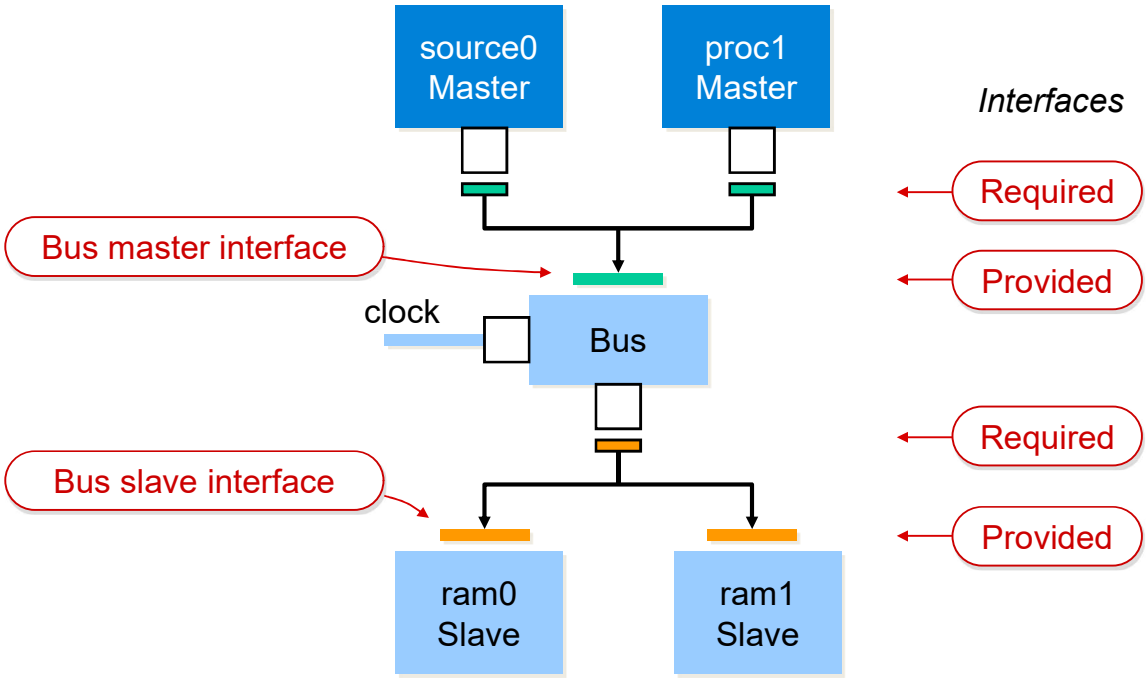
David C Black, Doulos

The SystemC Language

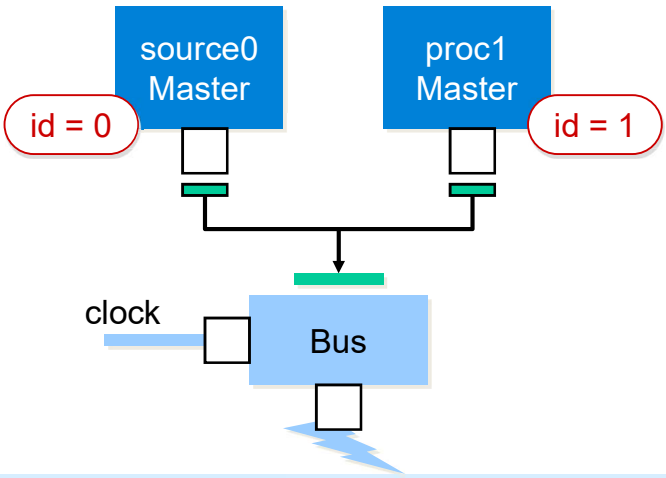
- Introduction to SystemC
- Core Concepts and Syntax
- Bus Modeling
  - Master and slave interfaces
  - Blocking versus non-blocking
  - Multiports
- Odds and Ends

Example Bus Model

Multiple bus masters (modules), shared bus (channel), multiple slaves (channels)
Bus arbitration and memory mapping built into the bus

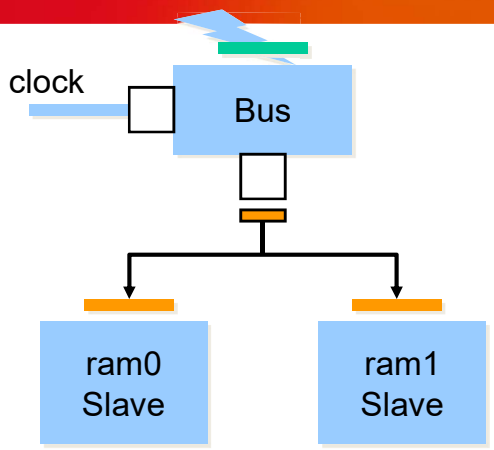


Master Interface Definition




```
class master_if : virtual public sc_interface
{
public:
  virtual void write(sc_uint<8> address, sc_uint<12> data,
                     int id) = 0;
  virtual void read (sc_uint<8> address, sc_uint<12> &data,
                     int id) = 0;
};
```

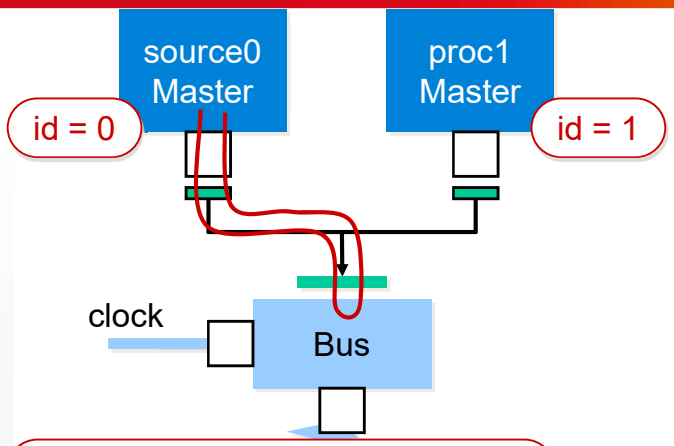
Slave Interface Definition




```
class slave_if : virtual public sc_interface
{
public:
  virtual void slave_write(sc_uint<8> address, sc_uint<12>  data) = 0;
  virtual void slave_read (sc_uint<8> address, sc_uint<12> &data) = 0;
  virtual void get_map(unsigned int &start, unsigned int &size)   = 0;
};
```

Memory map managed within bus channel

Master Write and Read



Runs in the context of the caller

Array of flags

request[0]	bool
request[1]	bool

Array of events

proceed[0]	sc_event
proceed[1]	sc_event

```
void Bus::write(sc_uint<8> address, sc_uint<12> data, int id)
{
  request[id] = true;          // request access to the bus
  wait(proceed[id]);           // wait for permission
  request[id] = false;         // clear the flag
  slave_port[find_port(address)]->slave_write(address, data);
}
```

Blocking and Non-blocking Calls

An Interface Method Call runs in the context of the caller

Important!

ASI terminology:

- A *blocking* method may call wait
- A *blocking* method must be called from a thread process

- A *non-blocking* method must not call wait
- A *non-blocking* method may be called from a thread or method process
- Naming convention nb_*

Bus Controller Process


```
void Bus::control_bus()
{
  int highest;
  for (;;)
  {
    wait(clock->posedge_event());

    // Pick out a master that's made a request
    highest = -1;
    for (int i = 0; i < n_masters; i++)
      if (request[i])
        highest = i;

    // Notify the master with the highest id
    if (highest > -1)
      proceed[highest].notify();
  }
}
```