Chapter 6

Abstract Communication Channels

Zainalabedin Navabi

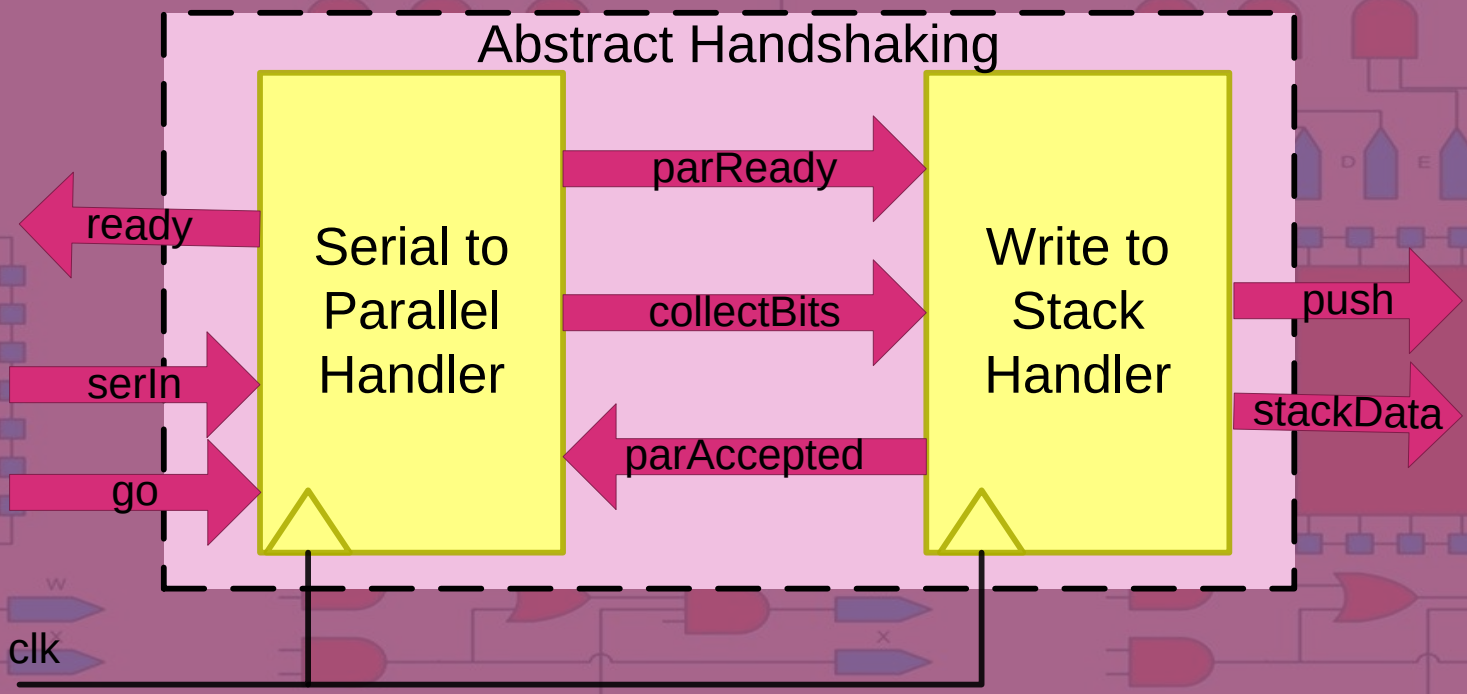
Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface

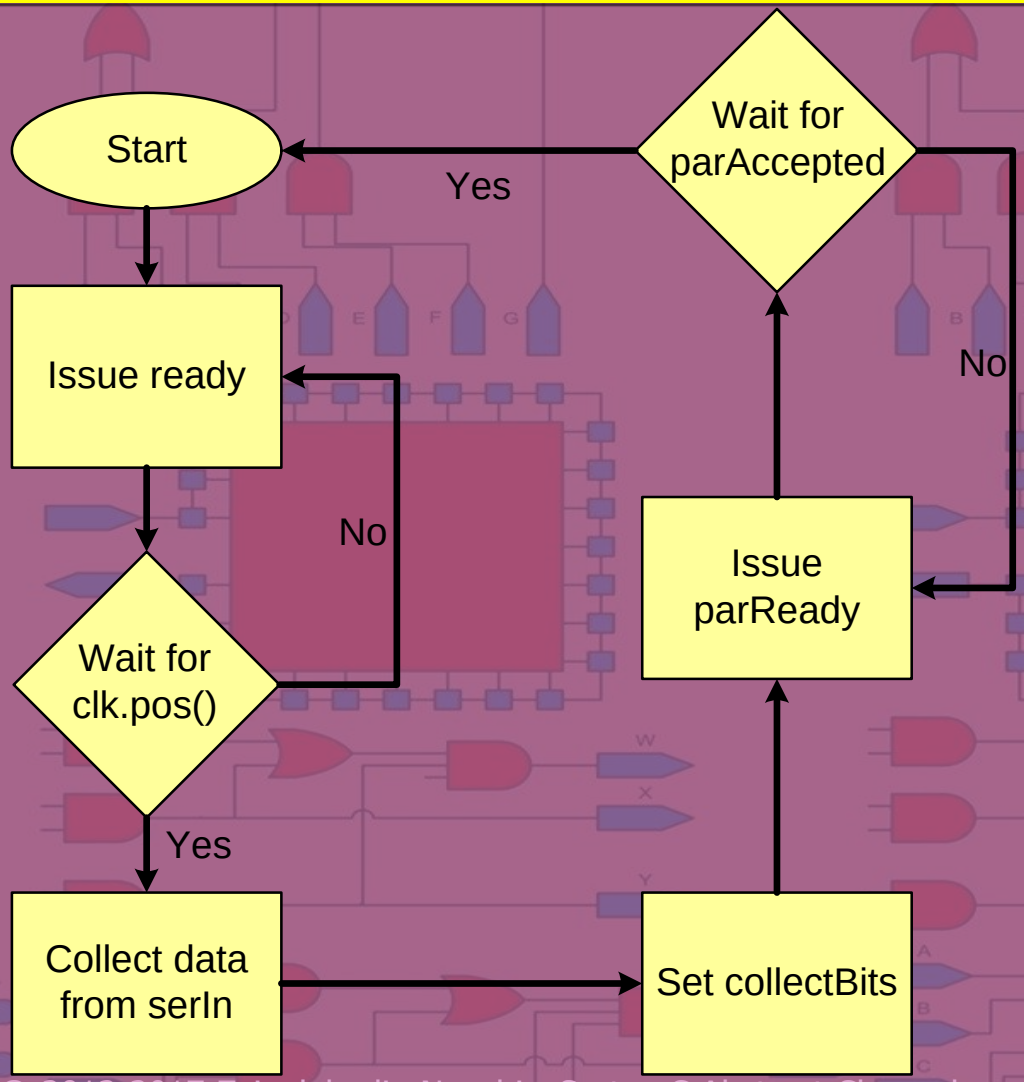
Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface

Abstract Handshaking



Abstract Handshaking



Abstract Handshaking

Abstract Handshaking .h

```
1   #include <systemc.h>
2
3   SC_MODULE(AbstractHandshaking) {
4
5       sc_in<sc_logic> clk, go;
6       sc_out<sc_logic> ready;
7       sc_in<sc_logic> serIn;
8       sc_out<sc_lv<8>> stackData;
9       sc_out<sc_logic> push;
10
11      sc_lv<8> collectBits;
12      sc_event parAccepted_ev, parReady_ev;
13
14      SC_CTOR(AbstractHandshaking) {
15          SC_THREAD(S2Phandler);
16              sensitive << clk.pos();
17          SC_THREAD(W2Shandler);
18              sensitive << clk.pos();
19      }
20      void S2Phandler();
21      void W2Shandler();
22  };
```

Abstract Handshaking

handshaking.cpp

```
1   #include "Abstract Handshaking.h"
2
3   void AbstractHandshaking::S2Phandler() {
4       int i;
5       ready = SC_LOGIC_0;
6       while (1){
7           ready = SC_LOGIC_1;
8           while (go != '1') wait();
9           ready = SC_LOGIC_0;
10          while (go != '0') wait();
11          for (i = 0; i < 8; i++){
12              wait();
13              collectBits[i] = serIn;
14          }
15          wait();
16          parReady_ev.notify();
17          wait(parAccepted_ev);
18      }
19  }
```

```
21  void AbstractHandshaking::W2Shandler() {
22      push = SC_LOGIC_0;
23      stackData = 0;
24      while (1){
25          wait(parReady_ev);
26          stackData = collectBits;
27          wait();
28          parAccepted_ev.notify();
29          wait();
30          push = SC_LOGIC_1;
31          wait();
32          push = SC_LOGIC_0;
33          stackData = 0;
34      }
35  }
```

Abstract Handshaking

Abstract Handshaking_TB.h

```
#include "Abstract Handshaking.h"

SC_MODULE(AbstractHandshakingTB) {
    sc_signal<sc_logic> clk;
    sc_signal<sc_logic> go, serIn;
    sc_signal<sc_logic> ready;
    sc_signal<sc_lv<8>> parOut;
    sc_signal<sc_logic> push;

    AbstractHandshaking* S2W; // Ser to par and Write to stack

    SC_CTOR(AbstractHandshakingTB) {
        S2W = new AbstractHandshaking("Handshaking_TB");
        S2W -> clk(clk);
        S2W -> go(go);
        S2W -> serIn(serIn);
        S2W -> ready(ready);
        S2W -> stackData(parOut);
        S2W -> push(push);

        SC_THREAD(clocking);
        SC_THREAD(serialData);
    }
    void clocking();
    void serialData();
};
```

Abstract Handshaking_TB.cpp

```
#include "Abstract Handshaking_TB.h"

void AbstractHandshakingTB::clocking() {
    int i;
    clk = SC_LOGIC_0;
    for (i = 0; i<=500; i++){
        wait (29, SC_NS);
        clk = SC_LOGIC_1;
        wait (29, SC_NS);
        clk = SC_LOGIC_0;
    }
}
```

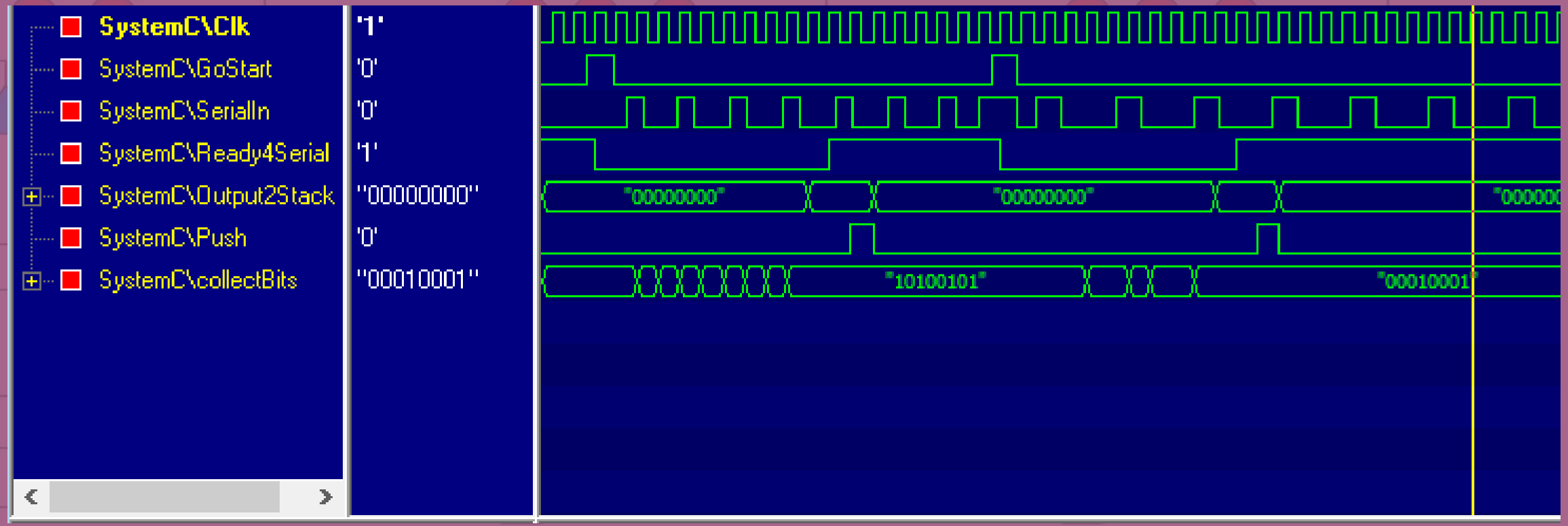
```
void AbstractHandshakingTB::serialData() {
    int i, j;
    go = SC_LOGIC_0;
    serIn = SC_LOGIC_0;
    for (i = 1; i < 5; i++){
        if (ready == SC_LOGIC_1){
            wait(31, SC_NS);
            go = SC_LOGIC_1;
            wait(73, SC_NS);
            go = SC_LOGIC_0;
            for (j = 0; j < 7; j++){
                serIn = SC_LOGIC_0;
                wait(17 * i, SC_NS);
                serIn = SC_LOGIC_1;
                wait(23 * i, SC_NS);
                serIn = SC_LOGIC_0;
                wait(31 * i, SC_NS);
                serIn = SC_LOGIC_1;
            }
        }
        else wait(91, SC_NS);
    }
}
```

Abstract Handshaking

Abstract Handshaking_Main.cpp

```
#include "Abstract Handshaking_TB.h"

int sc_main (int argc, char ** argv)
{
    AbstractHandshakingTB* HSTB1 = new AbstractHandshakingTB("HandshakingTB");

    sc_trace_file* VCDFile;
    VCDFile = sc_create_vcd_trace_file("Handshaking");

    sc_trace (VCDFile, HSTB1->clk, "Clk");
    sc_trace(VCDFile, HSTB1->go, "GoStart");
    sc_trace(VCDFile, HSTB1->serIn, "SerialIn");
    sc_trace(VCDFile, HSTB1->ready, "Ready4Serial");
    sc_trace (VCDFile, HSTB1->parOut, "Output2Stack");
    sc_trace(VCDFile, HSTB1->push, "Push");

    sc_trace(VCDFile, HSTB1->S2W->collectBits, "collectBits");

    sc_start(4500, SC_NS);
    sc_close_vcd_trace_file(VCDFile);
    return 0;
}
```

Abstract Handshaking



Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface

Basics of Channels

- A container class for communication and synchronization
- They implement one or more *interfaces*
- Different channels may implement the same interface in different ways
- A channel implements all the methods of the inherited interface classes

- There are *primitive channels* and *hierarchical channels*

Basics of Channels

- A primitive channel does not contain any hierarchy or processes
- All primitive channels are derived from the base class called *sc_prim_channel*
- SystemC contains several built-in channels:
  - *sc_signal*
  - *sc_mutex*
  - *sc_semaphore*
  - *sc_fifo*

Basics of Channels

Signal type, that is passed to the interface

sc_signal.h

Implements this interface.

For evaluations and accessing sc_signal.

For the sc_signal timing purposes.

sc_signal Class definition

```
64  template <class T>
65  class sc_signal
    : public sc_signal_inout_if<T>,
      public sc_prim_channel
    {
      public: // constructors and destructor:
        sc_signal() { ... }
77      explicit sc_signal( const char* name_ ) { ... }
85      virtual ~sc_signal() { ... }
91
        // interface methods
        virtual void register_port( sc_port_base&
        // get the default event
        virtual const sc_event& default_event() { ... }
        // get the value changed event
        virtual const sc_event& value_changed_event() { ... }
        // read the current value
111     virtual const T& read() { ... }
114     // get a reference to the current value (for tracing)
115     virtual const T& get_data_ref() { ... }
119     // was there an event?
120     virtual bool event() { ... }
```

Basics of Channels

```
SC_MODULE (bCircuit) {
   sc_in <bool> a, b;
   sc_out <int> w;
   …
   sc_signal <bool> d;
   sc_signal <int> e;
   sc_signal <sc_lv <8> > dv;
   …
};
```

sc_signal_in_if is an interface class for the sc_signal class, it is derived from sc_interface

sc_signal is a primitive channel

sc_signal has a template parameter for type

Basics of Channels

- sc_signal methods are:
  - write(..): write value
  - read(): read value
  - event(): was there an event (in bool)
  - default_event(): get the event
- For signals of type bool and sc_logic:
  - posedge_event(): get the event
  - negedge_event()
  - posedge(): was there an event (in bool)
  - negedge()
  - delayed(): get the delayed signal

Basics of Channels

- A mutex is an object used to let multiple program threads share a common resource without colliding
- Any process that needs the resource must *lock()* the mutex waits until lock occurs
- The process *unlocks()* the mutex when done
- Using *trylock()* allows a process to get the mutex if availble (non-blocking)
- There is no event that tells when an *sc_mutex* is freed

Basics of Channels

sc_mutex_if Class definition

```
62  class sc_mutex_if
63  : virtual public sc_interface
64  {
65  public:
66
67      /* ... */
70      virtual int lock() = 0;
71
          // returns -1 if mutex could not be locked
          virtual int trylock() = 0;

          // returns -1 if mutex was not locked by caller
          virtual int unlock() = 0;

78  protected:
79
80      // constructor
81
82      sc_mutex_if() { ... }
85  private:
86
87      // disabled
88      sc_mutex_if( const sc_mutex_if& );
89      sc_mutex_if& operator = ( const sc_mutex_if& );
90  };
```

sc_mutex_if.h

Basics of Channels

sc_mutex Class definition

```
68  class sc_mutex
69  : public sc_mutex_if,
70    public sc_prim_channel
71  {
72  public:
73      // constructors and destructor
74      sc_mutex();
75      explicit sc_mutex( const char* name_ );
76      virtual ~sc_mutex();
77      /* ... */
80      virtual int lock();
81
82      // returns -1 if mutex could not be locked
83      virtual int trylock();
84
85      // returns -1 if mutex was not locked by caller
86      virtual int unlock();
87
88      virtual const char* kind() const { ... }
91  protected:
92      // support methods
93      bool in_use() const { ... }
96  protected:
97      sc_process_b* m_owner;
98      sc_event      m_free;
99  private:
100     // disabled
101     sc_mutex( const sc_mutex& );
102     sc_mutex& operator = ( const sc_mutex& );
103 };
```

sc_mutex.h

Basics of Channels

- *virtual int lock();*
  - If the mutex is unlocked
    - member function *lock()* shall lock the mutex and return
  - If the mutex is locked
    - suspend until the mutex is unlocked (by another process)
- Multiple attempt to lock the mutex in the same delta cycle
  - the process instance that is given the lock in that delta cycle is "non-deterministic"
  - relies on the order in which processes are resume… evaluation phase.

Implements a blocking process

Basics of Channels

- *virtual int trylock();*
  - If the mutex is unlocked
    - Member function *trylock()* shall lock the mutex
    - Shall return the value **0**
  - If the mutex is locked
    - Member function *trylock()* shall immediately return the value **-1**
    - The mutex shall remian locked

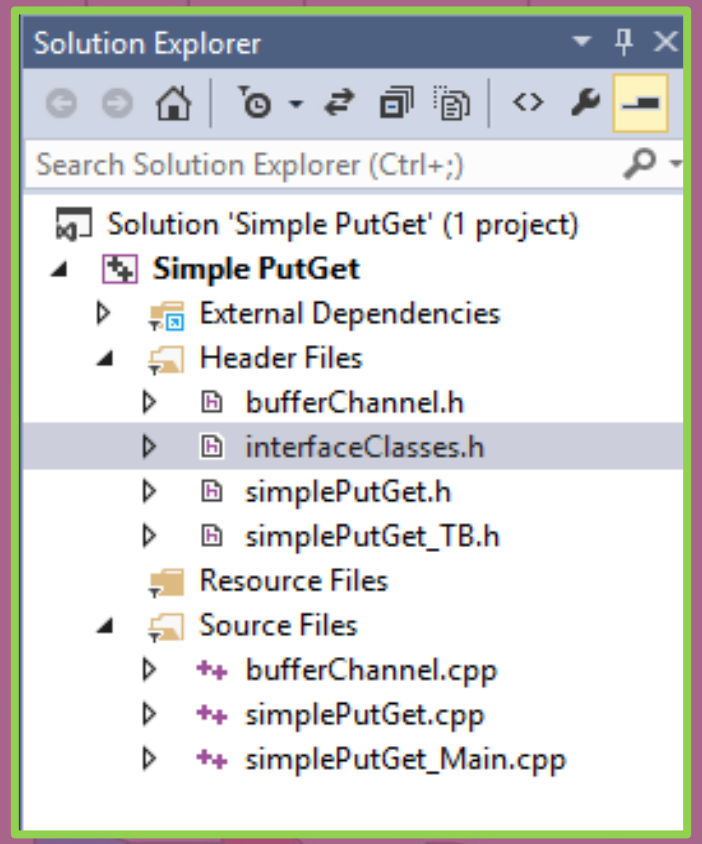
Implements a non-blocking process

Basics of Channels

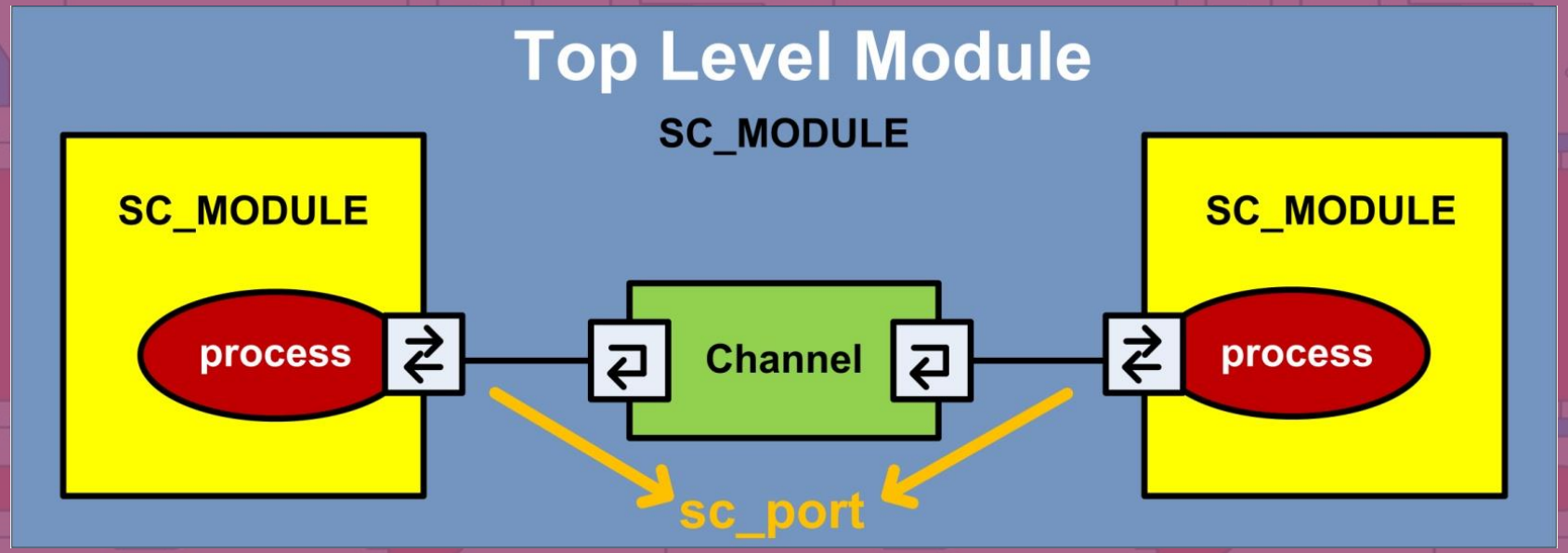
- *virtual int unlock();*
  - If mutex unlocked,
    - member function *unlock* shall return the value **-1**
    - the mutex shall remain unlocked
  - If mutex locked by another process
    - member function *unlock* shall return the value **-1**
    - the mutex shall remain locked
  - If mutex locked by the calling process
    - member function *unlock* shall unlock the mutex
    - shall return the value **0**

Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface



Example 1: Buffer Channel



Example 1: Buffer Channel, *interfaceClasses.h*

interfaceClasses.h

```
1   #include <systemc.h>
2
3   class put_if : virtual public sc_interface
4   {
5       public:
6           virtual void put(sc_lv<8> ) = 0;
7   };
8
9   class get_if : virtual public sc_interface
10  {
11     public:
12          virtual void get(sc_lv<8> &) = 0;
13  };
14
```

Example 1: Buffer Channel, *bufferChannel*

bufferChannel.h

```
#include "interfaceClasses.h"

class buffer : public put_if, public get_if
{
    bool full;
    sc_lv<8> saved;
    sc_event put_event, get_event;
    public:
        buffer() : full(false) {};
        ~buffer() {};
        void put(sc_lv<8> data);
        void get(sc_lv<8> &data);
};
```

bufferChannel.cpp

```
#include "bufferChannel.h"

void buffer::put(sc_lv<8> data) {
        if (full==true) wait(get_event);
        saved=data;
        full=true;
        put_event.notify();
}
void buffer::get(sc_lv<8> &data){
        if (full==false) wait(put_event);
        data = saved;
        full=false;
        get_event.notify();
}
```

Example 1: Buffer Channel, *simplePutGet*

simplePutGet.h

```
1  #include "bufferChannel.h"
2
3  SC_MODULE (transmitter) {
4      sc_port<put_if> out;
5
6      SC_CTOR(transmitter) {
7          SC_THREAD (putting);
8      }
9      void putting();
10 };
11
12 SC_MODULE (receiver) {
13     sc_port<get_if> in;
14
15     SC_CTOR(receiver) {
16         SC_THREAD (getting);
17     }
18     void getting();
19 };
```

simplePutGet.cpp

```
1  #include "simplePutGet.h"
2
3  void transmitter::putting() {
4      int i;
5      sc_lv<8> dataToPut;
6      for (i=0; i<27; i++)
7      {
8          wait(7, SC_NS);
9          dataToPut = (sc_lv<8>) i;
10         out->put(dataToPut);
11         cout << "Data: (" << dataToPut << ") was transmitted at: "
12             << sc_time_stamp() << '\n';
13     }
14 }
15
16 void receiver::getting() {
17     sc_lv<8> dataThatGot;
18     int i; for (i=0; i<27; i++)
19     while (1)
20     {
21         wait(3, SC_NS);
22         in->get(dataThatGot);
23         cout << "Data: (" << dataThatGot << ") was received at: "
24             << sc_time_stamp() << '\n';
25     }
26 }
```

Example 1: Buffer Channel, *simplePutGet_TB*

simplePutGet_TB.h

```
1   #include "simplePutGet.h"
2
3   SC_MODULE (simplePutGet_TB) {
4
5       buffer* BUF1;
6       transmitter* TRS1;
7       receiver* RCV1;
8
9       SC_CTOR(simplePutGet_TB) {
10          BUF1 = new buffer();
11          TRS1 = new transmitter("Transmitter");
12              TRS1->out(*BUF1);
13          RCV1 = new receiver("Receiver");
14              RCV1->in(*BUF1);
15      }
16  };
```

Example 1: Buffer Channel, *bufferChannel_main.cpp*

bufferChannel_main.cpp

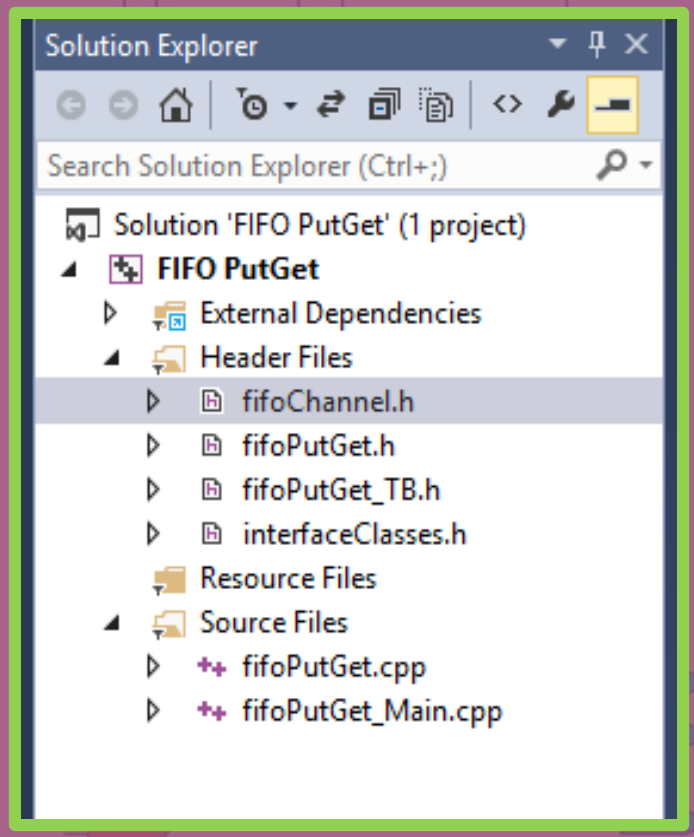
```
1  #include "simplePutGet_TB.h"
2  int sc_main (int argc , char *argv[]) {
3      simplePutGet_TB SPG1("simplePutGet1");
4      sc_start (90, SC_NS);
5      return 0;
6  }
```

Example 1: Buffer Channel, output

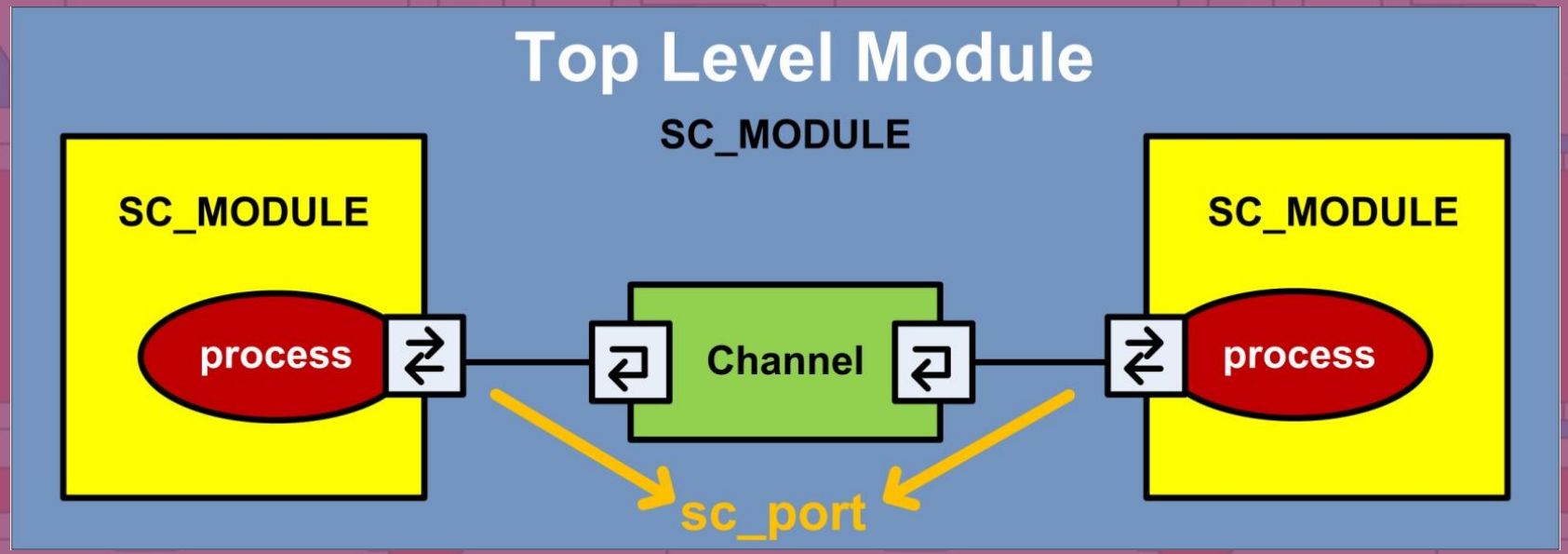
```
        SystemC 2.3.1-Accellera --- Sep 22 2015 07:13:37
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED
Data: (00000000) was transmitted at: 7 ns
Data: (00000000) was received at: 7 ns
Data: (00000001) was transmitted at: 14 ns
Data: (00000001) was received at: 14 ns
Data: (00000010) was transmitted at: 21 ns
Data: (00000010) was received at: 21 ns
Data: (00000011) was transmitted at: 28 ns
Data: (00000011) was received at: 28 ns
Data: (00000100) was transmitted at: 35 ns
Data: (00000100) was received at: 35 ns
Data: (00000101) was transmitted at: 42 ns
Data: (00000101) was received at: 42 ns
Data: (00000110) was transmitted at: 49 ns
Data: (00000110) was received at: 49 ns
Data: (00000111) was transmitted at: 56 ns
Data: (00000111) was received at: 56 ns
Data: (00001000) was transmitted at: 63 ns
Data: (00001000) was received at: 63 ns
Data: (00001001) was transmitted at: 70 ns
Data: (00001001) was received at: 70 ns
Data: (00001010) was transmitted at: 77 ns
```

Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface



Example 2: FiFo Channel



Example 2: FiFo Channel, *interfaceClasses*

template form

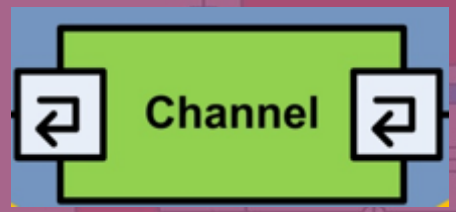
interfaceClasses.h

```
1   #include <systemc.h>
2
3   template <class T>
4   class put_if : virtual public sc_interface
5   {
6       public:
7           virtual void put(T) = 0;
8   };
9
10  template <class T>
11  class get_if : virtual public sc_interface
12  {
13     public:
14         virtual void get(T &) = 0;
15  };
```

Example 2: FiFo Channel, *fifoChannel*

```
1  #include "interfaceClasses.h"
2
3  template <class T, int Max>
4  class fifo : public put_if<T>, public get_if<T>
5  {
6      int size;
7      int elems, head;
8      T queueContents[Max];
9      sc_event put_event, get_event;
10     public:
11         fifo() : size(Max), elems(0), head(0) {};
12         ~fifo() {};
13         void put(T data){
14             if (elems == size) wait(get_event);
15             queueContents[(head + elems) % size] = data;
16             elems = elems + 1;
17             put_event.notify();
18         }
19         void get(T &data){
20             if (elems == 0) wait(put_event);
21             data = queueContents[head];
22             elems = elems - 1;
23             head = (head + 1) % size;
24             get_event.notify();
25         }
26 };
```

fifoChannel.h



Example 2: FiFo Channel, *fifoChannel*

```
1   #include "interfaceClasses.h"
2
3   template <class T, int Max>
4   class fifo : public put_if<T>, public get_if<T>
5   {
6       int size;
7       int elems, head;
8       T queueContents[Max];
9       sc_event put_event, get_event;
10      public:
11          fifo() : size(Max), elems(0), head(0) {};
12          ~fifo() {};
13          void put(T data){
14              if (elems == size) wait(get_event);
15              queueContents[(head + elems) % size] = data;
16              elems = elems + 1;
17              put_event.notify();
18          }
19          void get(T &data){
20              if (elems == 0) wait(put_event);
21              data = queueContents[head];
22              elems = elems - 1;
23              head = (head + 1) % size;
24              get_event.notify();
25          }
26  };
```

Because of the template format, it doesn't have .cpp file and all functions are implemented in .h file

Form 1

fifoChannel.h

Form 2

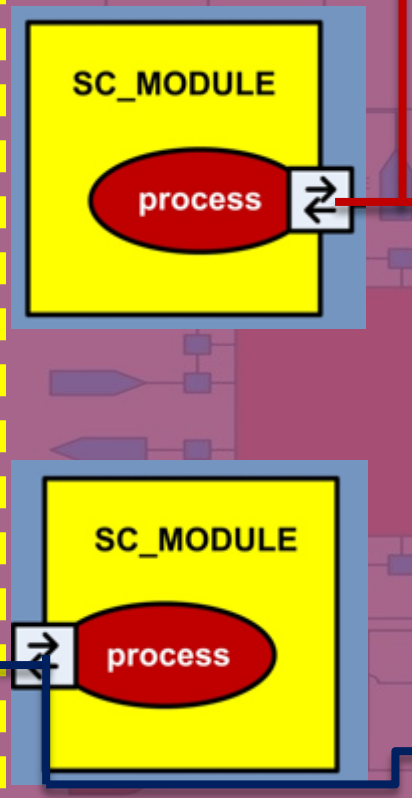
fifoChannel.cpp

```
29  template <class T, int Max>
30  void fifo<T, Max>::put(T data) {
31      if (elems == size) wait(get_event);
32      queueContents[(head + elems) % size] = data;
33      elems = elems + 1;
34      put_event.notify();
35  }
36  template <class T, int Max>
37  void fifo<T, Max>::get(T &data){
38      if (elems == 0) wait(put_event);
39      data = queueContents[head];
40      elems = elems - 1;
41      head = (head + 1) % size;
42      get_event.notify();
43  }
```

Example 2: FiFo Channel, *fifoPutGet*

fifoPutGet.h

```
1   #include "fifoChannel.h"
2
3   SC_MODULE (transmitter) {
4       sc_port<put_if<sc_lv<8>>> out;
5
6       SC_CTOR(transmitter) {
7           SC_THREAD (putting);
8       }
9       void putting();
10  };
11
12  SC_MODULE (receiver) {
13      sc_port<get_if<sc_lv<8>>> in;
14
15      SC_CTOR(receiver) {
16          SC_THREAD (getting);
17      }
18      void getting();
19  };
```

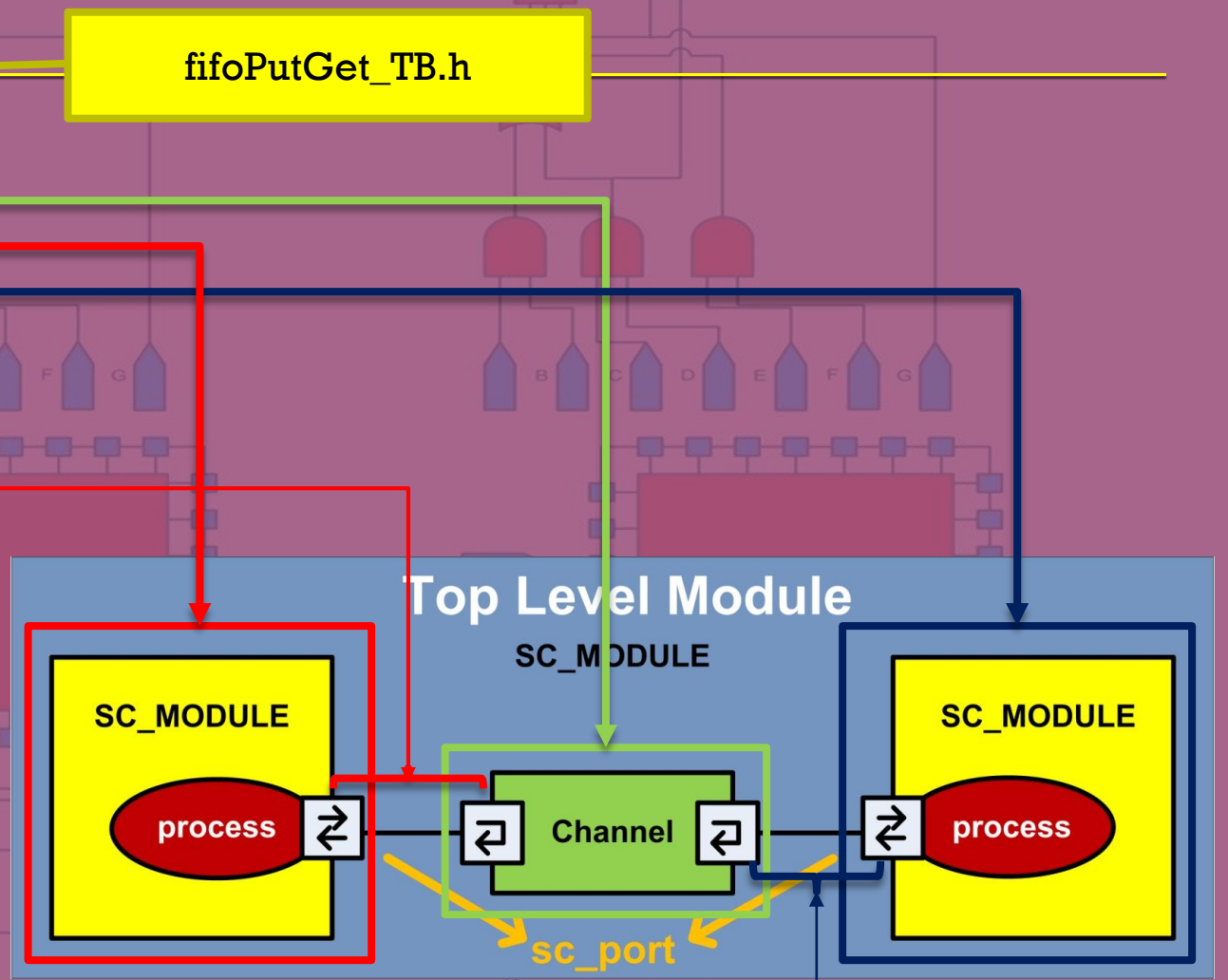



```
1   #include "fifoPutGet.h"
2
3   void transmitter::putting() {
4       int i;
5       sc_lv<8> dataToPut;
6       for (i=0; i<27; i++)
7       {
8           wait(3, SC_NS);
9           dataToPut = (sc_lv<8>) i;
10          out->put(dataToPut);
11          cout << "Data: (" << dataToPut << ") was transmitted at: "
12               << sc_time_stamp() << '\n';
13      }
14  }
15
16  void receiver::getting() {
17      sc_lv<8> dataThatGot;
18      int i; for (i=0; i<27; i++)
19      while (1)
20      {
21          wait(7, SC_NS);
22          in->get(dataThatGot);
23          cout << "Data: (" << dataThatGot << ") was received at: "
24               << sc_time_stamp() << '\n';
25      }
26  }
```

fifoPutGet.cpp

Example 2: FiFo Channel, *fifoPutGet_TB.h*

```
1   #include "fifoPutGet.h"
2
3   SC_MODULE (fifoPutGet_TB) {
4
5       fifo<sc_lv<8>,9> * FIFO1;
6       transmitter* TRS1;
7       receiver* RCV1;
8
9       SC_CTOR(fifoPutGet_TB) {
10          FIFO1 = new fifo<sc_lv<8>,9>;
11          TRS1 = new transmitter("Transmitter");
12          TRS1->out(*FIFO1);
13          RCV1 = new receiver("Receiver");
14          RCV1->in(*FIFO1);
15      }
16  };
17
18  /* ... */
35
36
37  /* ... */
48
49
50  /* ... */
```



Example 2: FiFo Channel, *fifoPutGet_TB.h*

```
18  /* ... */
35
36
37  /* ... */
48
49
50  /* ... */
```

```
38  SC_MODULE(fifoPutGet_TB) {
39
40      fifo<sc_lv<8>, 9> FIFO1;
41      transmitter TRS1;
42      receiver RCV1;
43
44      SC_CTOR(fifoPutGet_TB) : FIFO1(), TRS1("Transmitter"), RCV1("Receiver")
45      {
46          RCV1(FIFO1);
47          TRS1(FIFO1);
48      }
49  };
```

```
/*
fifo* FIFO1;
transmitter* TRS1;
receiver* RCV1;

SC_CTOR(fifoPutGet_TB) {
    FIFO1 = new fifo;
    TRS1 = new transmitter("Transmitter");
    TRS1->out(*FIFO1);
    RCV1 = new receiver("Receiver");
    RCV1->in(*FIFO1);
}
*/
```

```
22  SC_MODULE(fifoPutGet_TB) {
23
24      fifo<sc_lv<8>, 9> FIFO1;
25      transmitter* TRS1;
26      receiver* RCV1;
27
28      SC_CTOR(fifoPutGet_TB) : FIFO1() {
29          TRS1 = new transmitter("Transmitter");
30          TRS1->out(FIFO1);
31          RCV1 = new receiver("Receiver");
32          RCV1->in(FIFO1);
33      }
34  };
```

Example 2: FiFo Channel, *fifoPutGet_main.cpp*

fifoChannel_main.cpp

```
1  #include "fifoPutGet_TB.h"
2  int sc_main (int argc , char *argv[]) {
3      fifoPutGet_TB FPG1("fifoPutGet1");
4      sc_start();
5      return 0;
6  }
```

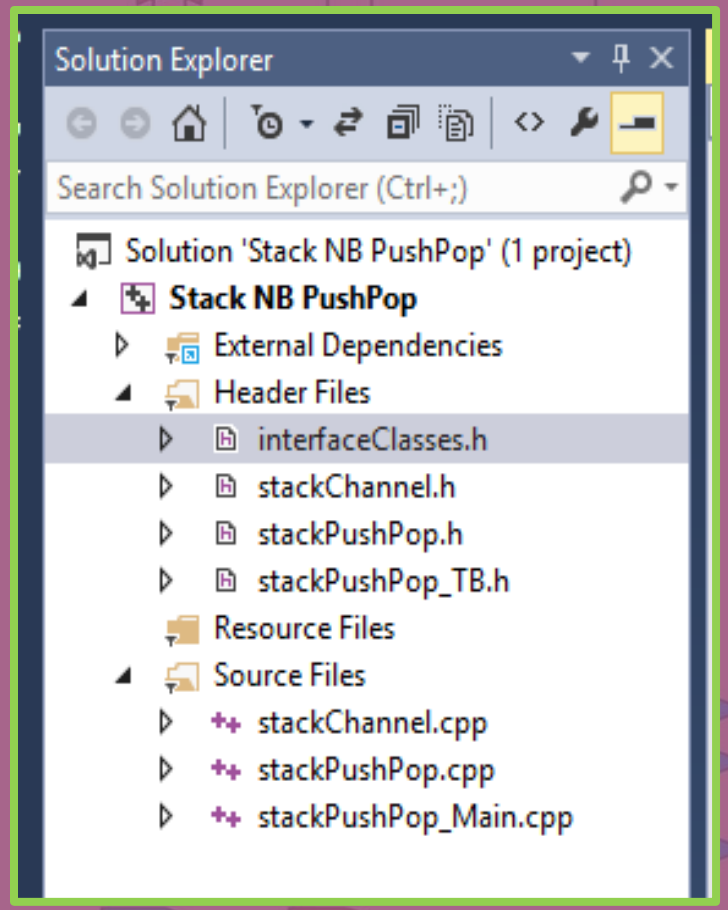
Example 2: FiFo Channel, output

```
        SystemC 2.3.1-Accellera --- Sep 22 2015
        Copyright (c) 1996-2014 by all Contribut
        ALL RIGHTS RESERVED
Data: (00000000) was transmitted at: 3 ns
Data: (00000001) was transmitted at: 6 ns
Data: (00000000) was received at: 7 ns
Data: (00000010) was transmitted at: 9 ns
Data: (00000011) was transmitted at: 12 ns
Data: (00000001) was received at: 14 ns
Data: (00000100) was transmitted at: 15 ns
Data: (00000101) was transmitted at: 18 ns
Data: (00000010) was received at: 21 ns
Data: (00000110) was transmitted at: 21 ns
Data: (00000111) was transmitted at: 24 ns
Data: (00001000) was transmitted at: 27 ns
Data: (00000011) was received at: 28 ns
Data: (00001001) was transmitted at: 30 ns
Data: (00001010) was transmitted at: 33 ns
Data: (00000100) was received at: 35 ns
Data: (00001011) was transmitted at: 36 ns
Data: (00001100) was transmitted at: 39 ns
Data: (00000101) was received at: 42 ns
Data: (00001101) was transmitted at: 42 ns
Data: (00001110) was transmitted at: 45 ns
```

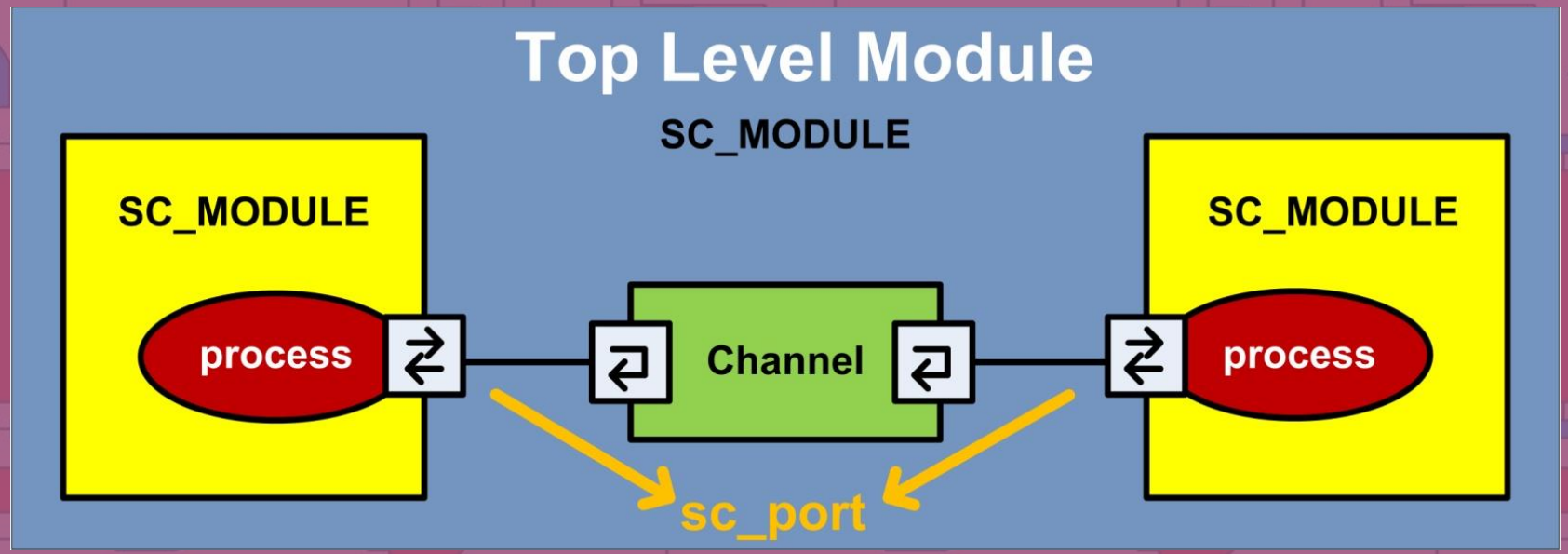
```
Data: (00010100) was transmitted at: 84 ns
Data: (00001100) was received at: 91 ns
Data: (00010101) was transmitted at: 91 ns
Data: (00001101) was received at: 98 ns
Data: (00010110) was transmitted at: 98 ns
Data: (00001110) was received at: 105 ns
Data: (00010111) was transmitted at: 105 ns
Data: (00001111) was received at: 112 ns
Data: (00011000) was transmitted at: 112 ns
Data: (00010000) was received at: 119 ns
Data: (00011001) was transmitted at: 119 ns
Data: (00010001) was received at: 126 ns
Data: (00011010) was transmitted at: 126 ns
Data: (00010010) was received at: 133 ns
Data: (00010011) was received at: 140 ns
Data: (00010100) was received at: 147 ns
Data: (00010101) was received at: 154 ns
Data: (00010110) was received at: 161 ns
Data: (00010111) was received at: 168 ns
Data: (00011000) was received at: 175 ns
Data: (00011001) was received at: 182 ns
Data: (00011010) was received at: 189 ns
Press any key to continue . . .
```

Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface



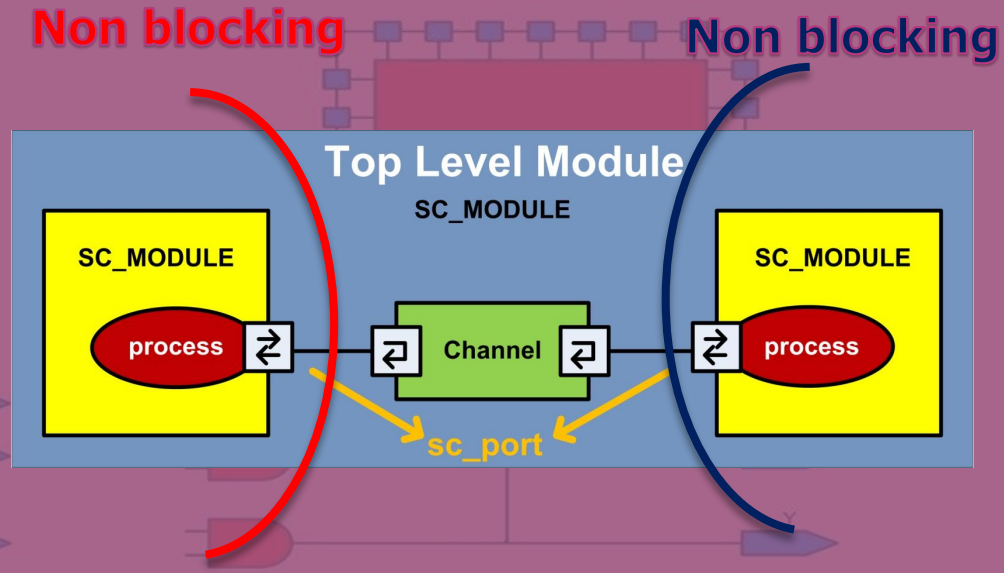
Example 3: Stack Channel



Example 3: Stack Channel, *interfaceClasses.h*

interfaceClasses.h

```
1   #include <systemc.h>
2
3   class stack_push_if: virtual public sc_interface
4       {
5           public:
6               virtual bool nb_push(sc_lv<8> ) = 0;
7               virtual void init() = 0;
8   };
9
10  class stack_pop_if: virtual public sc_interface
11      {
12          public:
13              virtual bool nb_pop(sc_lv<8> &) = 0;
14  };
```



Example 3: Stack Channel, *stackChannel*

```
1   #include "interfaceClasses.h"
2
3   class stack : public stack_push_if, public stack_pop_if {
4   public:
5       stack() {tos=0;};
6       bool nb_push(sc_lv<8> data);
7       void init();
8       bool nb_pop(sc_lv<8> &data);
9
10  private:
11      sc_lv<8> contents[17];
12      int tos;
13  };
```

stackChannel.h

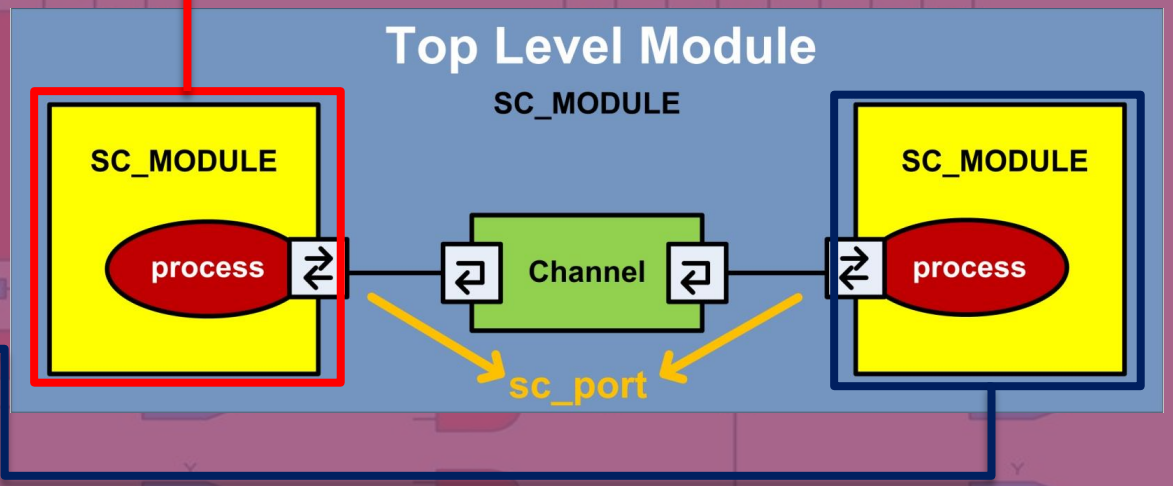
```
1   #include "stackChannel.h"
2
3   bool stack::nb_push(sc_lv<8> data) {
4       if (tos <17)
5           {
6               contents[tos++] = data;
7               return true;
8           }
9           return false;
10  }
11  void stack::init() {
12      tos = 0;
13  }
14  bool stack::nb_pop(sc_lv<8>& data) {
15      if (tos > 0)
16          {
17              data = contents[--tos];
18              return true;
19          }
20          return false;
21  }
```

stackChannel.cpp

Example 3: Stack Channel, *stackPushPop*

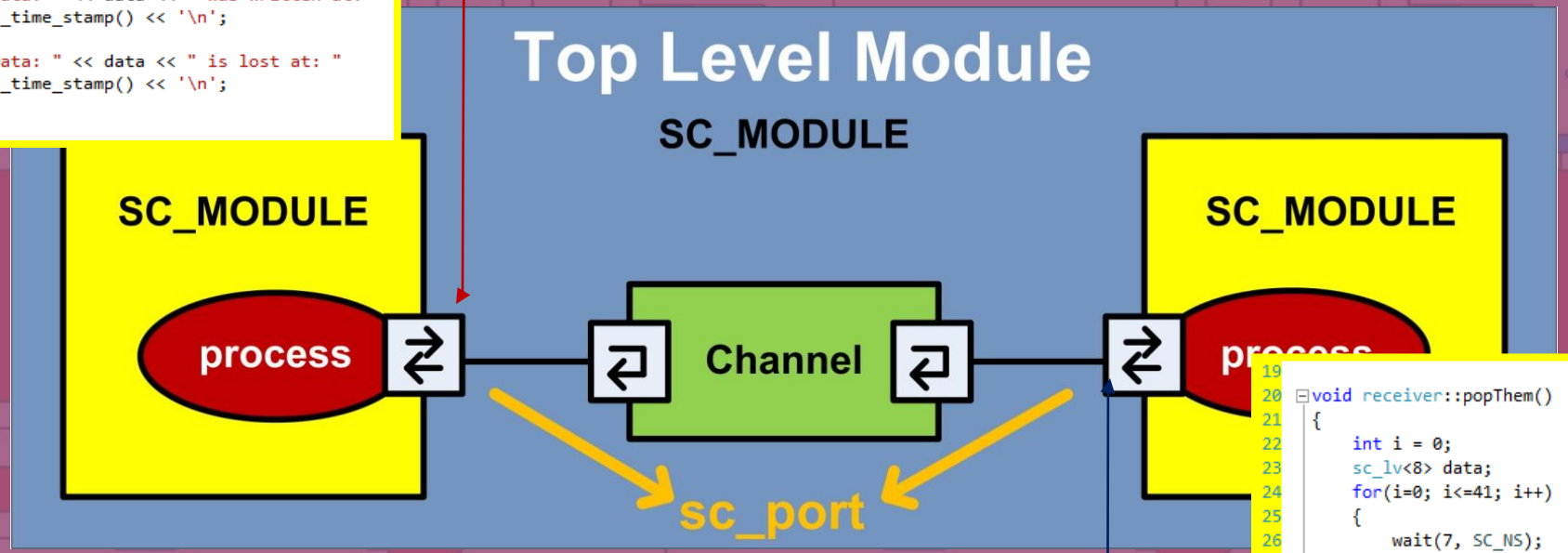
```
1  #include "stackChannel.h"
2
3  SC_MODULE (transmitter) {
4      sc_port<stack_push_if> out; // with if out is a pointer
5
6      SC_CTOR (transmitter)
7      {
8          SC_THREAD(pushSome);
9      }
10     void pushSome();
11 };
12
13 SC_MODULE (receiver) {
14     sc_port<stack_pop_if> in;
15
16     SC_CTOR (receiver)
17     {
18         SC_THREAD(popThem);
19     }
20     void popThem();
21 };
```

stackPushPop.h



Example 3: Stack Channel, *stackPushPop*

```cpp
#include "stackPushPop.h"

void transmitter::pushSome()
{
    int i = 0;
    sc_lv<8> data;
    for(i=0; i<=43; i++)
    {
        data = (sc_lv<8>) i;
        wait(3, SC_NS);
        if (out->nb_push(sc_lv<8> (i)))
            cout << "Data: " << data << " was written at: "
                 << sc_time_stamp() << '\n';
        else
            cout << "Data: " << data << " is lost at: "
                 << sc_time_stamp() << '\n';
    }
}
```




```cpp
void receiver::popThem()
{
    int i = 0;
    sc_lv<8> data;
    for(i=0; i<=41; i++)
    {
        wait(7, SC_NS);
        if (in->nb_pop(data))
            cout << "Data: " << data << " popped at: "
                 << sc_time_stamp() << '\n';
        else
            cout << "No data was poppoed at: "
                 << sc_time_stamp() << '\n';
    }
}
```

Example 3: Stack Channel, *stackPushPop*

```
1   #include "stackPushPop.h"
2
3   void transmitter::pushSome()
4   {
5       int i = 0;
6       sc_lv<8> data;
7       for(i=0; i<=43; i++)
8       {
9           data = (sc_lv<8>) i;
10          wait(3, SC_NS);
11          if (out->nb_push(sc_lv<8> (i)))
12              cout << "Data: " << data << " was written at: "
13                   << sc_time_stamp() << '\n';
14          else
15              cout << "Data: " << data << " is lost at: "
16                   << sc_time_stamp() << '\n';
17      }
18  }
```

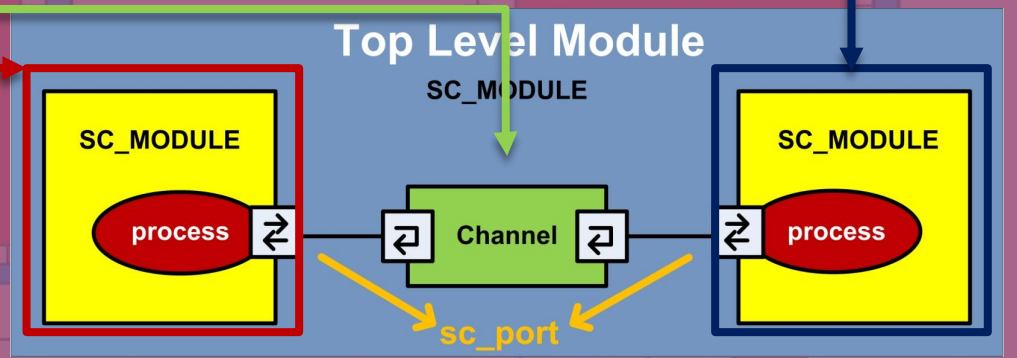
stackPushPop.cpp

```
19
20  void receiver::popThem()
21  {
22      int i = 0;
23      sc_lv<8> data;
24      for(i=0; i<=41; i++)
25      {
26          wait(7, SC_NS);
27          if (in->nb_pop(data))
28              cout << "Data: " << data << " popped at: "
29                   << sc_time_stamp() << '\n';
30          else
31              cout << "No data was poppoed at: "
32                   << sc_time_stamp() << '\n';
33      }
34  }
```

Example 3: Stack Channel, *stackPushPop_TB*

stackPushPop_TB.h

```
1   #include "stackPushPop.h"
2
3   SC_MODULE (stackPushPop_TB) {
4
5       stack* STK1;
6       transmitter* TRS1;
7       receiver* RCV1;
8
9       SC_CTOR(stackPushPop_TB) {
10          STK1 = new stack();
11          TRS1 = new transmitter("Transmitter");
12              TRS1->out(*STK1);
13          RCV1 = new receiver("Receiver");
14              RCV1->in(*STK1);
15      }
16  };
```



Example 3: Stack Channel, *stackPushPop_main*

stackPushPop_main.cpp

```
1  #include "stackPushPop_TB.h"
2  int sc_main (int argc , char *argv[]) {
3      stackPushPop_TB SPP1("stackPushPop1");
4      sc_start();
5      return 0;
6  }
```

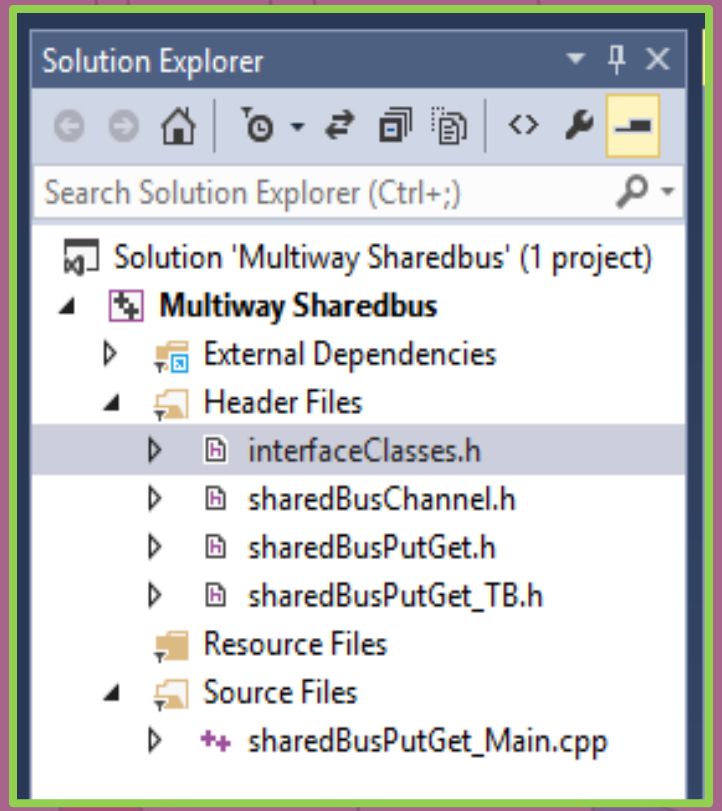
Example 3: Stack Channel, *output*

```
        SystemC 2.3.1-Accellera --- Sep 22 2015 07:13:
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED
Data: 00000000 was written at: 3 ns
Data: 00000001 was written at: 6 ns
Data: 00000001 popped at: 7 ns
Data: 00000010 was written at: 9 ns
Data: 00000011 was written at: 12 ns
Data: 00000011 popped at: 14 ns
Data: 00000100 was written at: 15 ns
Data: 00000101 was written at: 18 ns
Data: 00000101 popped at: 21 ns
Data: 00000110 was written at: 21 ns
Data: 00000111 was written at: 24 ns
Data: 00001000 was written at: 27 ns
Data: 00001000 popped at: 28 ns
Data: 00001001 was written at: 30 ns
Data: 00001010 was written at: 33 ns
Data: 00001010 popped at: 35 ns
Data: 00001011 was written at: 36 ns
Data: 00001100 was written at: 39 ns
Data: 00001100 popped at: 42 ns
Data: 00001101 was written at: 42 ns
Data: 00001110 was written at: 45 ns
```

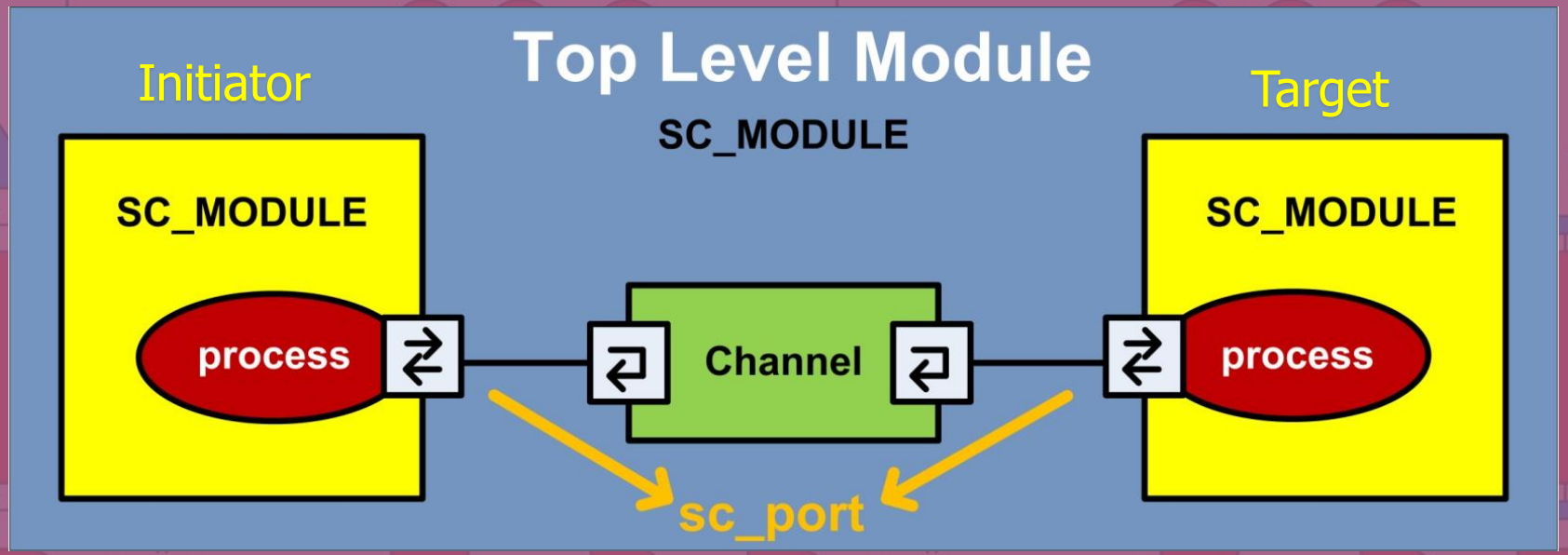
```
Data: 00101001 popped at: 133 ns
Data: 00011011 popped at: 140 ns
Data: 00011001 popped at: 147 ns
Data: 00010111 popped at: 154 ns
Data: 00010101 popped at: 161 ns
Data: 00010100 popped at: 168 ns
Data: 00010010 popped at: 175 ns
Data: 00010000 popped at: 182 ns
Data: 00001110 popped at: 189 ns
Data: 00001101 popped at: 196 ns
Data: 00001011 popped at: 203 ns
Data: 00001001 popped at: 210 ns
Data: 00000111 popped at: 217 ns
Data: 00000110 popped at: 224 ns
Data: 00000100 popped at: 231 ns
Data: 00000010 popped at: 238 ns
Data: 00000000 popped at: 245 ns
No data was poppoed at: 252 ns
No data was poppoed at: 259 ns
No data was poppoed at: 266 ns
No data was poppoed at: 273 ns
No data was poppoed at: 280 ns
No data was poppoed at: 287 ns
No data was poppoed at: 294 ns
Press any key to continue . . .
```

Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface



Example 4: Multi-way shared bus



Example 4: Multi-way shared bus, *interfaceClasses.h*

interfaceClasses.h

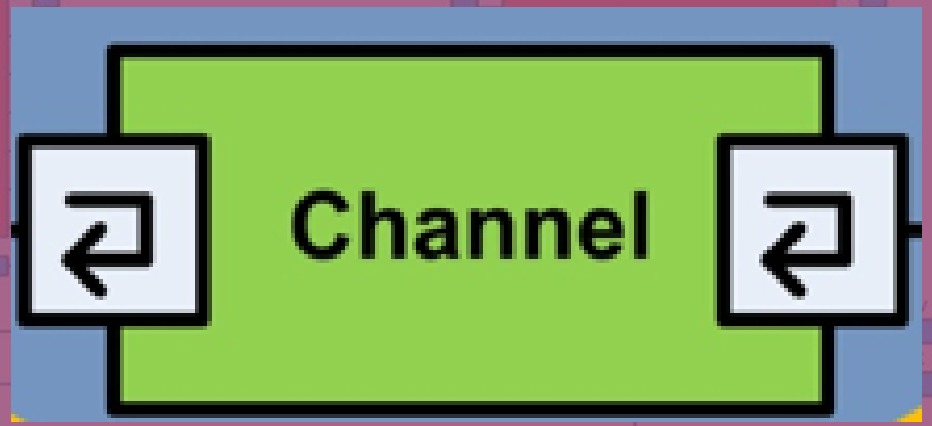
```
1   #include <systemc.h>
2
3   template <class T>
4   class put_if : virtual public sc_interface
5   {
6       public:
7           virtual void put(int initiator, int target, T data) = 0;
8   };
9
10  template <class T>
11  class get_if : virtual public sc_interface
12  {
13     public:
14         virtual void get(int &initiator, int target, T &data) = 0;
15  };
```

Example 4: Multi-way shared bus, *stackChannel*

```
1  #include "interfaceClasses.h"
2
3  template <class T, int numOfInitiators, int numOfTargets>
4  class sharedBus : public put_if<T>, public get_if<T>
5  {
6      int comingFrom, goingTo;
7      T dataPlaced;
8      sc_event dataAvailable[numOfTargets];
9      sc_event dataReceived[numOfTargets];
10
11     sc_mutex busBusy;
12
13     public:
14         sharedBus() : comingFrom(-1), goingTo(-1) {};
15         ~sharedBus() {};
16         void put(int initiator, int target, T data){
17             busBusy.lock();
18             comingFrom = initiator;
19             goingTo = target;
20             dataPlaced = data;
21             dataAvailable[target].notify();
22             wait(dataReceived[target]);
23             busBusy.unlock();
24         }
25         void get(int &initiator, int target, T &data){
26             if (goingTo != target) wait(dataAvailable[target]);
27             initiator = comingFrom;
28             data = dataPlaced;
29             comingFrom = -1;
30             goingTo = -1; // prevent multiple gets of same data
31             dataReceived[target].notify();
32         }
33 };
```

sharedBusChannel.h

Template form

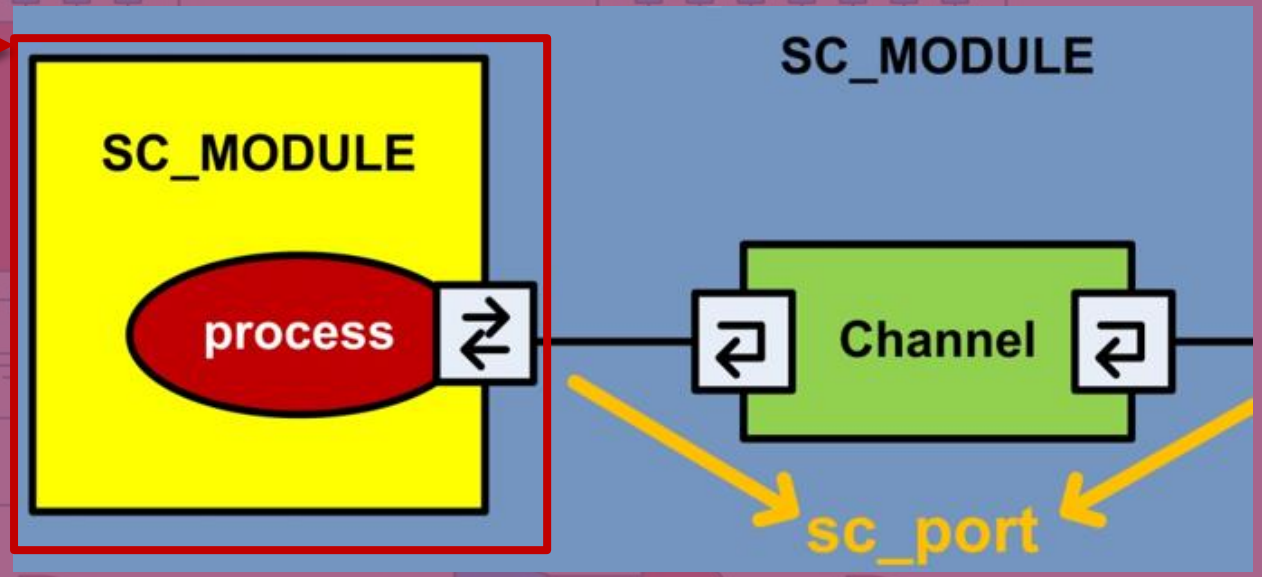


Example 4: Multi-way shared bus,

sharedBusPutGet

```
1   #include "sharedBusChannel.h"
2
3   template <int N, int F>
4   SC_MODULE (initiators) {
5       sc_port<put_if<sc_lv<8>>> out;
6
7       SC_CTOR(initiators) {
8           SC_THREAD (putting);
9       }
10      void putting();
11  };
12
13  template <int N, int F>
14  void initiators<N, F>::putting() {
15      int toTarget;
16      sc_lv<8> transmittedData;
17      ofstream fout("Multiway Sharedbus report.txt", ios::app);
18
19      for (int i = (N * 16); i<(N * 16 + 15); i++)
20      {
21          wait(F, SC_NS);
22          transmittedData = (sc_lv<8>) i;
23          toTarget = rand() % 4; // Total 4 targets
24  ...
29          out->put(N, toTarget, transmittedData);
30  ...
33      }
34  }
35
36  template <int N, int F>
37  SC_MODULE(targets) { ... }
45
46  template <int N, int F>
47  void targets<N, F>::getting() { ... }
```

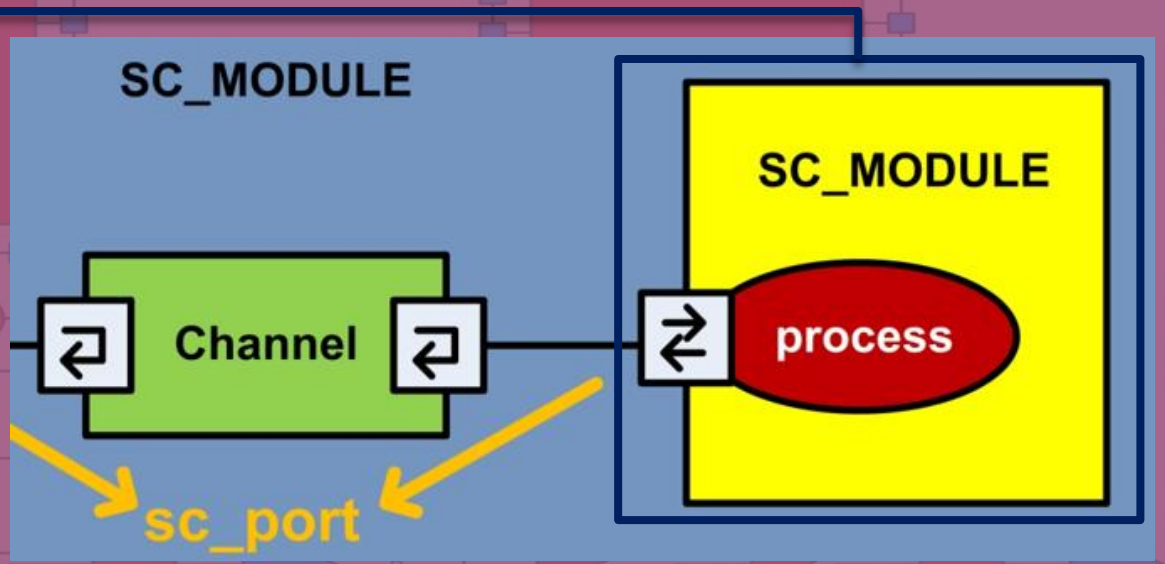
sharedBusPutGet.h



Example 4: Multi-way shared bus, *sharedBusPutGet*

```
1   #include "sharedBusChannel.h"
2
3   template <int N, int F>
4  SC_MODULE (initiators) { ... }
12
13  template <int N, int F>
14 void initiators<N, F>::putting() { ... }
35
36  template <int N, int F>
37 SC_MODULE(targets) {
38      sc_port<get_if<sc_lv<8>>> in;
39
40      SC_CTOR(targets) {
41          SC_THREAD (getting);
42      }
43      void getting();
44  };
45
46  template <int N, int F>
47 void targets<N, F>::getting() {
48      sc_lv<8> receivedData;
49      int dataInitiator;
50      ofstream fout("Multiway Sharedbus report.txt", ios::app);
51
52      while (1)
53      {
54          wait(F, SC_NS);
55 ...
59          in->get(dataInitiator, N, receivedData);
60 ...
62      }
63  }
```

sharedBusPutGet.h



Example 4: Multi-way shared bus, *sharedBusPutGet*

```cpp
#include "sharedBusChannel.h"

template <int N, int F>
SC_MODULE (initiators) {
    sc_port<put_if<sc_lv<8>>> out;

    SC_CTOR(initiators) {
        SC_THREAD (putting);
    }
    void putting();
};

template <int N, int F>
void initiators<N, F>::putting() {
    int toTarget;
    sc_lv<8> transmittedData;
    ofstream fout("Multiway Sharedbus report.txt", ios::app);

    for (int i = (N * 16); i<(N * 16 + 15); i++)
    {
        wait(F, SC_NS);
        transmittedData = (sc_lv<8>) i;
        toTarget = rand() % 4; // Total 4 targets

        cout << "\nInitiator {" << N << "} intends to"
             << " transmit (" << transmittedData << ") at: "
             << sc_time_stamp() << " to: [" << toTarget << "]\n";

        out->put(N, toTarget, transmittedData);
        cout << "Initiator {" << N << "} completed transmtting ("
             << transmittedData << ") at: "
             << sc_time_stamp() << " to: [" << toTarget << "]\n";
    }
}
```

sharedBusPutGet.h

Example 4: Multi-way shared bus, *sharedBusPutGet*

```
1   #include "sharedBusChannel.h"
2
3   template <int N, int F>
4   SC_MODULE (initiators) { ... }
12
13  template <int N, int F>
14  void initiators<N, F>::putting() { ... }
35
36  template <int N, int F>
37  SC_MODULE(targets) {
38      sc_port<get_if<sc_lv<8>>> in;
39
40      SC_CTOR(targets) {
41          SC_THREAD (getting);
42      }
43      void getting();
44  };
```

sharedBusPutGet.h

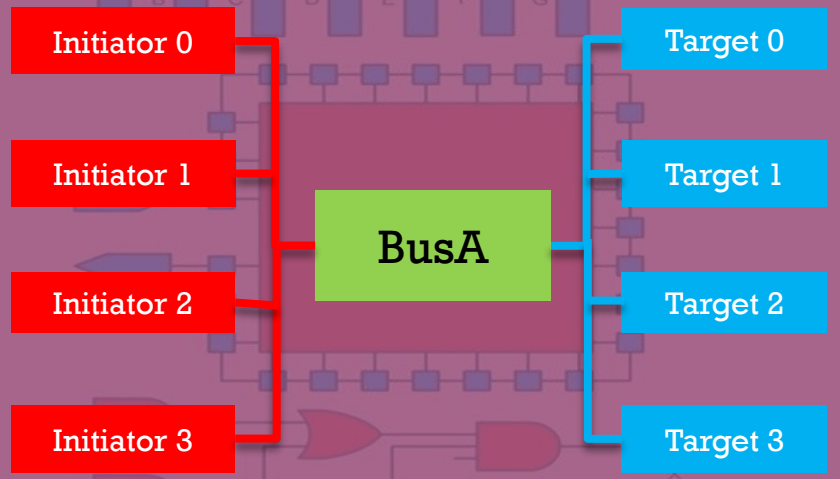
```
45
46  template <int N, int F>
47  void targets<N, F>::getting() {
48      sc_lv<8> receivedData;
49      int dataInitiator;
50      ofstream fout("Multiway Sharedbus report.txt", ios::app);
51
52      while (1)
53      {
54          wait(F, SC_NS);
55
56          cout << "Target [" << N << "] ready to" << " receive something at: "
57               << sc_time_stamp() << '\n';
58
59          in->get(dataInitiator, N, receivedData);
60          cout << "Target [" << N << "] received (" << receivedData << ") at: "
61               << sc_time_stamp() << " from: {" << dataInitiator << "}\n";
62      }
63  }
```

Example 4: Multi-way shared bus,

sharedBusPutGet_TB

```
1   #include "sharedBusPutGet.h"
2
3   SC_MODULE (sharedBusPutGet_TB) {
4
5       sharedBus<sc_lv<8>, 4, 4> * BusA;
6       initiators<0, 3>* INI0;
7       initiators<1, 5>* INI1;
8       initiators<2, 7>* INI2;
9       initiators<3, 4>* INI3;
10      targets<0, 7>* TAR0;
11      targets<1, 4>* TAR1;
12      targets<2, 6>* TAR2;
13      targets<3, 5>* TAR3;
14
15      SC_CTOR(sharedBusPutGet_TB) {
16          BusA = new sharedBus<sc_lv<8>, 4, 4>;
17
18          INI0 = new initiators<0, 3>("Initiator0");
19          INI0->out(*BusA);
20          INI1 = new initiators<1, 5>("Initiator1");
21          INI1->out(*BusA);
22          INI2 = new initiators<2, 7>("Initiator2");
23          INI2->out(*BusA);
24          INI3 = new initiators<3, 4>("Initiator3");
25          INI3->out(*BusA);
26
27          TAR0 = new targets<0, 7>("Target0");
28          TAR0->in(*BusA);
29          TAR1 = new targets<1, 4>("Target1");
30          TAR1->in(*BusA);
31          TAR2 = new targets<2, 6>("Target2");
32          TAR2->in(*BusA);
33          TAR3 = new targets<3, 5>("Target3");
34          TAR3->in(*BusA);
35      }
36  };
```

sharedBusPutGet_TB.h



Example 4: Multi-way shared bus, *sharedBusPutGet_main*

sharedBusPutGet_main.cpp

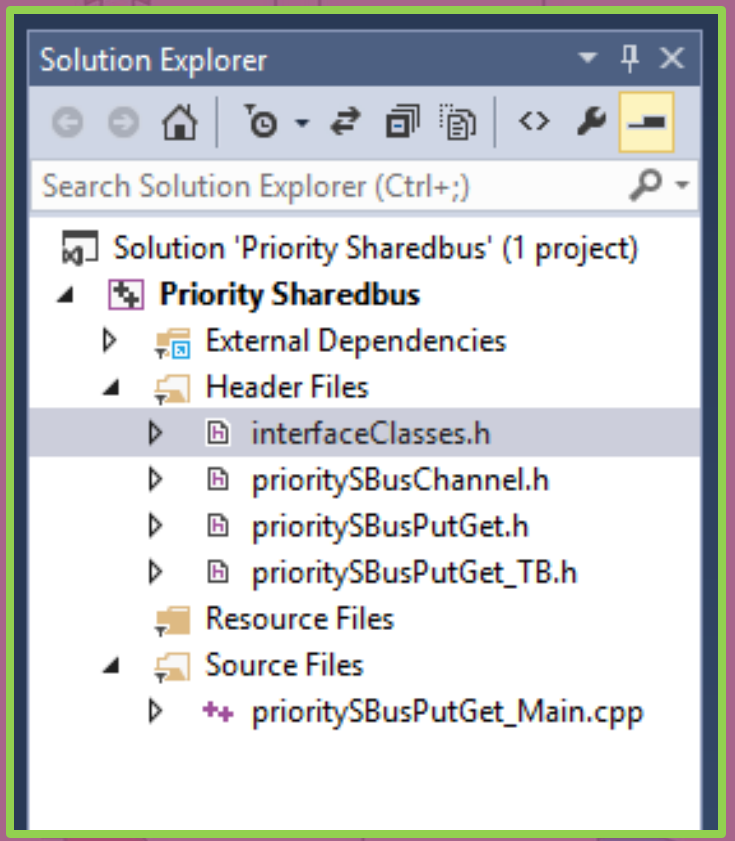
```
1  #include "sharedBusPutGet_TB.h"
2  int sc_main (int argc , char *argv[]) {
3      sharedBusPutGet_TB MultiWay("sharedBusPutGet1");
4      sc_start();
5      return 0;
6  }
```

Example 4: Multi-way shared bus, *output*

```
Target [1] received (00000000) at: 4 ns from: {0}
Initiator {0} completed transmtting (00000000) at: 4 ns to: [1]
Target [3] ready to receive something at: 5 ns

Initiator {1} intends to transmit (00010000) at: 5 ns to: [1]
Target [2] ready to receive something at: 6 ns
Target [0] ready to receive something at: 7 ns

Initiator {2} intends to transmit (00100000) at: 7 ns to: [1]

Initiator {0} intends to transmit (00000001) at: 7 ns to: [3]
Target [1] ready to receive something at: 8 ns
Target [1] received (00110000) at: 8 ns from: {3}
Initiator {3} completed transmtting (00110000) at: 8 ns to: [1]
Target [1] ready to receive something at: 12 ns
Target [1] received (00010000) at: 12 ns from: {1}

Initiator {3} intends to transmit (00110001) at: 12 ns to: [3]
Initiator {1} completed transmtting (00010000) at: 12 ns to: [1]
Target [3] received (00000001) at: 12 ns from: {0}
Initiator {0} completed transmtting (00000001) at: 12 ns to: [3]

Initiator {0} intends to transmit (00000010) at: 15 ns to: [2]
Target [1] ready to receive something at: 16 ns
Target [3] ready to receive something at: 17 ns
```

Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface



Example 5: *Priority shared bus, interfaceClasses.h*

interfaceClasses.h

```
1   #include <systemc.h>
2
3   template <class T>
4   class put_if : virtual public sc_interface
5   {
6       public:
7           virtual void put(int initiator, int target, T data, int delay) = 0;
8   };
9
10  template <class T>
11  class get_if : virtual public sc_interface
12  {
13     public:
14         virtual void get(int &initiator, int target, T &data, int delay) = 0;
15  };
```

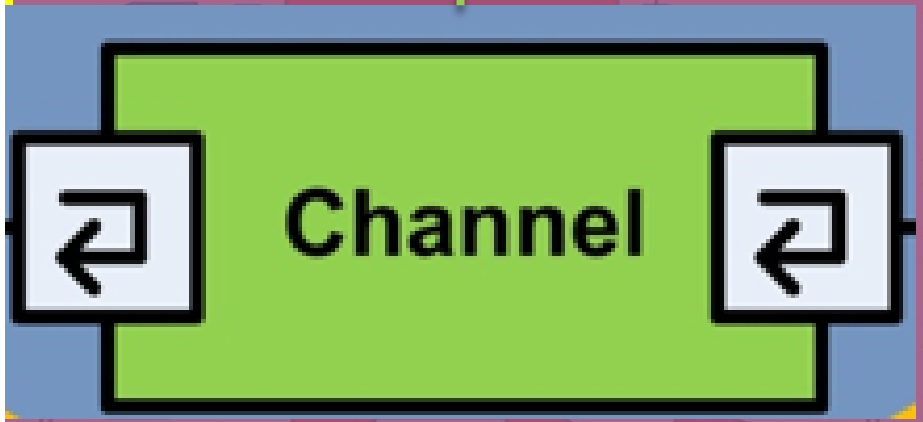
Example 5: *Priority shared bus,prioritySBusChannel.h*

Template form

prioritySBusChannel.h

```
3   // data type, number of Inititiators, number of Targets
4   template <class T, int nI, int nT>
5   class prioritySBus : public put_if<T>, public get_if<T>
6   {
7       int comingFrom, goingTo;
8       T dataPlaced;
9       sc_event dataAvailable[nT];
10      sc_event dataReceived[nT];
11      sc_event busReleased;
12
13      bool requestingI[nI]; // requesting Initiators
14      bool busBusy;
15
16      int priority(int, bool*);
17
18      public:
19          prioritySBus() : comingFrom(-1), goingTo(-1), busBusy(false) {};
20          ~prioritySBus() {};
21
```

Handling Priorities

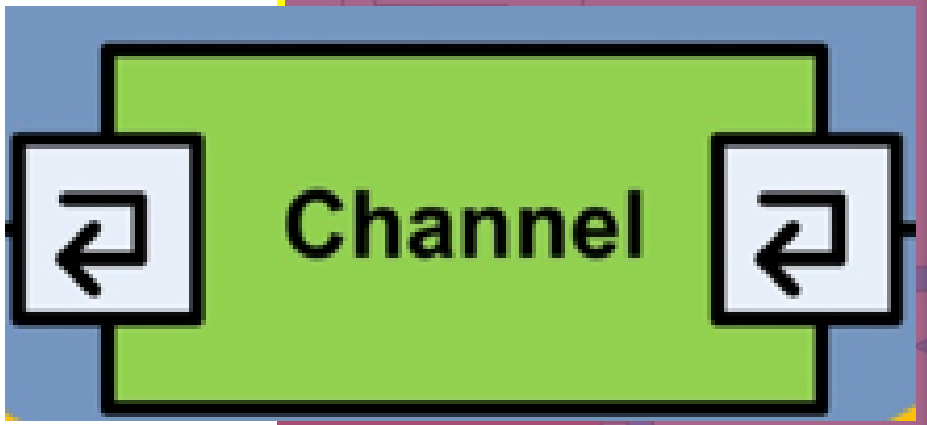


Example 5: *Priority shared bus,*

prioritySBusChannel.h

Template form

```
4   template <class T, int nI, int nT>
5   class prioritySBus : public put_if<T>, public get_if<T>
6   {
7       int comingFrom, goingTo;
8       T dataPlaced;
9       sc_event dataAvailable[nT];
10      sc_event dataReceived[nT];
11      sc_event busReleased;
12
13      bool requestingI[nI]; // requesting Initiators
14      bool busBusy;
15
16      int priority(int, bool*);
17
18      public:
19          prioritySBus() : comingFrom(-1), goingTo(-1), busBusy(false) {};
20          ~prioritySBus() {};
21
22          void put(int initiator, int target, T data, int delay) { ... }
38          void get(int &initiator, int target, T &data, int delay) { ... }
47  };
```



prioritySBusChannel.h

Example 5: Priority shared bus, prioritySBusChannel.h

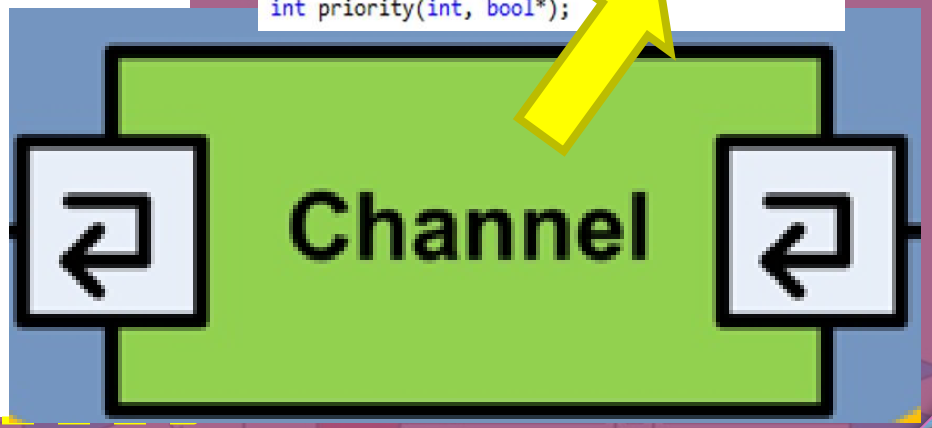
```
50  template <class T, int nI, int nT>
51  int prioritySBus<T, nI, nT>::priority(int totalCandidates, bool* thoseRequesting)
52  {
53      // 0 has highest priority
54      int found = -1;
55      for (int i = totalCandidates - 1; i >= 0; i--){
56          if (*(thoseRequesting + i)) found = i;
57      }
58      return found;
59  }
```

Example 5: *Priority shared bus, prioritySBusChannel.h*

```
22          void put(int initiator, int target, T data, int delay){
23              wait(delay, SC_NS);
24              requestingI[initiator] = true;
25              while (busBusy || (initiator != priority(nI, requestingI))) {
26                  wait(busReleased);
27              }
28              requestingI[initiator] = false;
29              busBusy = true;
30              comingFrom = initiator;
31              goingTo = target;
32              dataPlaced = data;
33              dataAvailable[target].notify();
34              wait(dataReceived[target]);
35              busBusy = false;
36              busReleased.notify();
37          }
38          void get(int &initiator, int target, T &data, int delay){
39              if (goingTo != target) wait(dataAvailable[target]);
40              initiator = comingFrom;
41              data = dataPlaced;
42              wait(delay, SC_NS);
43              comingFrom = -1;
44              goingTo = -1; // prevent multiple gets of same data
45              dataReceived[target].notify();
46          }
47      };
```

prioritySBusChannel.h

```
int comingFrom, goingTo;
T dataPlaced;
sc_event dataAvailable[nT];
sc_event dataReceived[nT];
sc_event busReleased;

bool requestingI[nI]; // requesting Initiators
bool busBusy;

int priority(int, bool*);
```

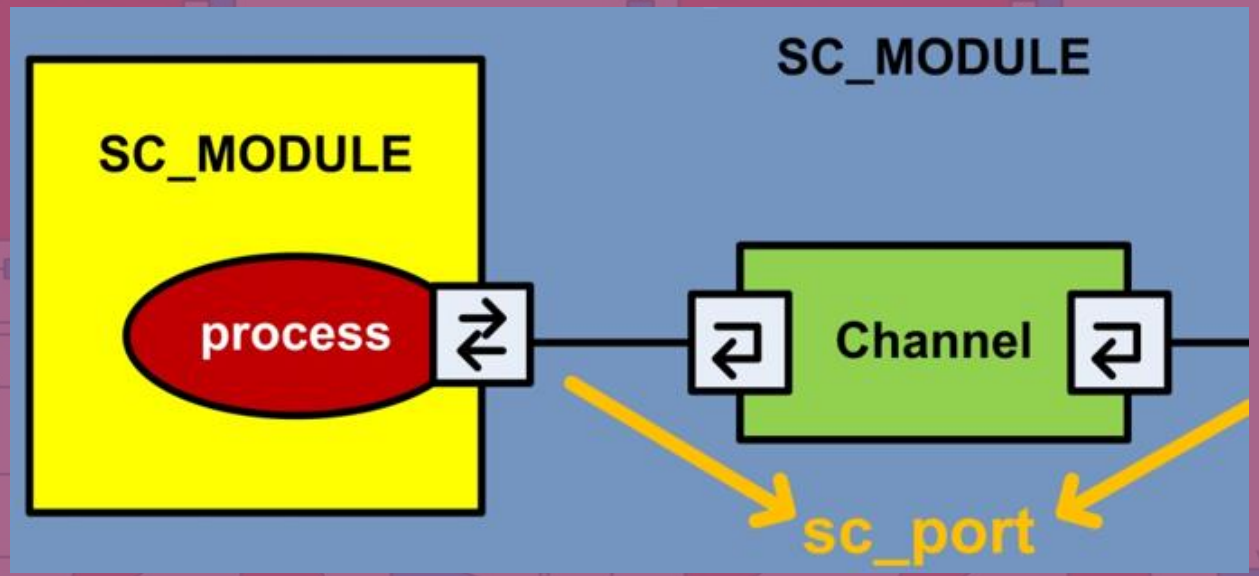


Example 5: *Priority shared bus,*

prioritySBusPutGet.h

```
1   #include "prioritySBusChannel.h"
2
3   // initiator Number, Frequency, Delay
4   template <int N, int F, int D>
5   SC_MODULE (initiators) {
6       sc_port<put_if<sc_lv<8>>> out;
7
8       SC_CTOR(initiators) {
9           SC_THREAD (putting);
10      }
11      void putting();
12  };
13
14  template <int N, int F, int D>
15  void initiators<N, F, D>::putting() {
16      int toTarget;
17      sc_lv<8> tData; // transmitted Data
18
19      for (int i = (N * 16); i<(N * 16 + 15); i++)
20      {
21          wait(F, SC_NS);
22          tData = (sc_lv<8>) (rand() % 256);
23          toTarget = 0;
24  ...
29          out->put(N, toTarget, tData, D);
30  ...
32      }
33  }
34
35  // target Number, Frequency, Delay
36  template <int N, int F, int D>
37  SC_MODULE(targets) { ... }
45
46  template <int N, int F, int D>
47  void targets<N, F, D>::getting() { ... }
63
```

prioritySBusPutGet.h

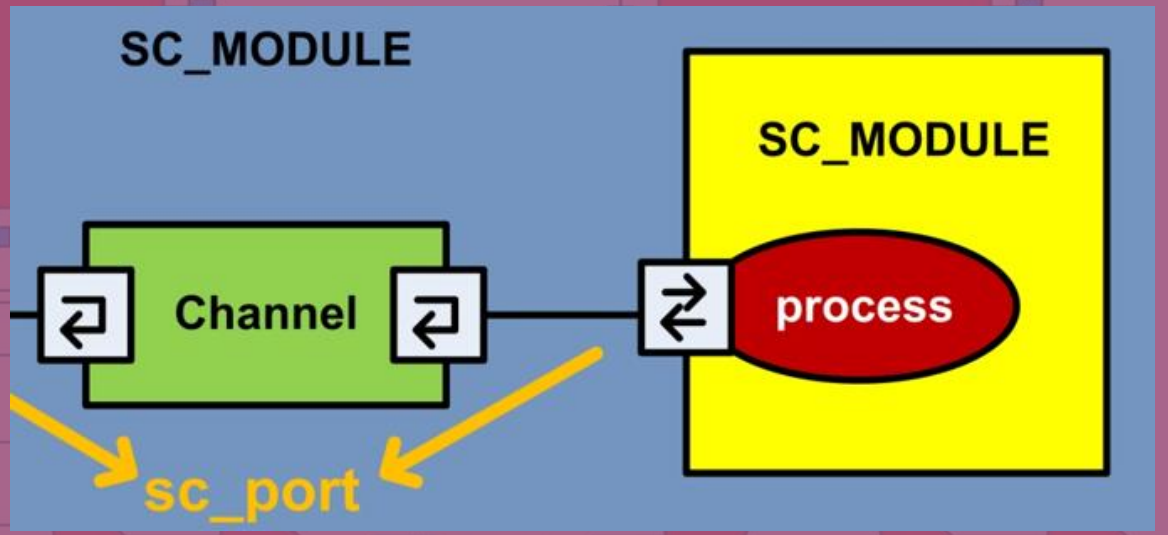


Example 5: *Priority shared bus,*

prioritySBusPutGet.h

```
1   #include "prioritySBusChannel.h"
2
3   // initiator Number, Frequency, Delay
4   template <int N, int F, int D>
5   SC_MODULE (initiators) { ... }
13
14  template <int N, int F, int D>
15  void initiators<N, F, D>::putting() { ... }
34
35  // target Number, Frequency, Delay
36  template <int N, int F, int D>
37  SC_MODULE(targets) {
38      sc_port<get_if<sc_lv<8>>> in;
39
40      SC_CTOR(targets) {
41          SC_THREAD (getting);
42      }
43      void getting();
44  };
45
46  template <int N, int F, int D>
47  void targets<N, F, D>::getting() {
48      sc_lv<8> rData; // received Data
49      int dataInitiator;
50
51      while (1)
52      {
53          wait(F, SC_NS);
54  ...
58          in->get(dataInitiator, N, rData, D);
59  ...
61      }
62  }
```

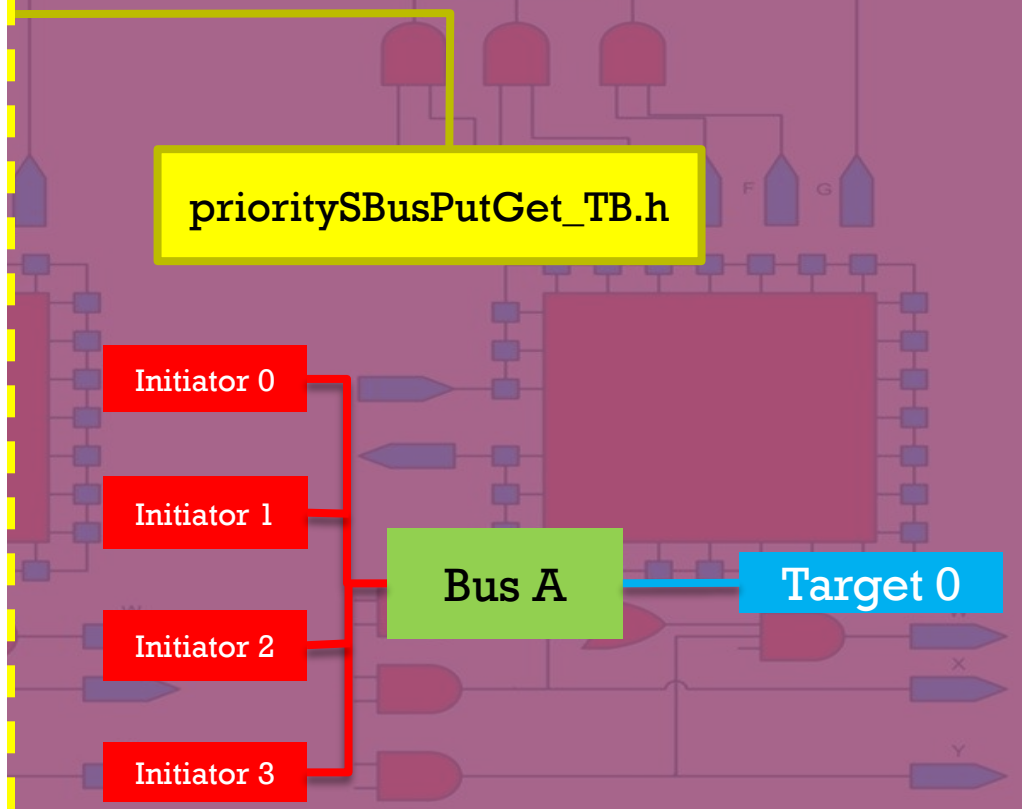
prioritySBusPutGet.h



Example 5: *Priority shared bus,*

prioritySBusPutGet_TB.h

```
1   #include "prioritySBusPutGet.h"
2
3   SC_MODULE(prioritySBusPutGet_TB) {
4
5       prioritySBus<sc_lv<8>, 4, 1> * BusA;
6       initiators<0, 9, 0>* INI0;
7       initiators<1, 11, 0>* INI1;
8       initiators<2, 7, 0>* INI2;
9       initiators<3, 15, 0>* INI3;
10      targets<0, 17, 13>* TAR0;
11
12      SC_CTOR(prioritySBusPutGet_TB) {
13          BusA = new prioritySBus<sc_lv<8>, 4, 1>;
14
15          INI0 = new initiators<0, 9, 0>("Initiator0");
16          INI0->out(*BusA);
17          INI1 = new initiators<1, 11, 0>("Initiator1");
18          INI1->out(*BusA);
19          INI2 = new initiators<2, 7, 0>("Initiator2");
20          INI2->out(*BusA);
21          INI3 = new initiators<3, 15, 0>("Initiator3");
22          INI3->out(*BusA);
23
24          TAR0 = new targets<0, 17, 13>("Target0");
25          TAR0->in(*BusA);
26      }
27  };
```



Example 5: *Priority shared bus,prioritySBusPutGet_main*

prioritySBusPutGet_main.cpp

```
1  #include "prioritySBusPutGet_TB.h"
2
3  int sc_main (int argc , char *argv[]) {
4      prioritySBusPutGet_TB PriorityBus ("prioritySBusPutGet1");
5      sc_start();
6      return 0;
7  }
8
```

Example 5: *Priority shared bus,output*

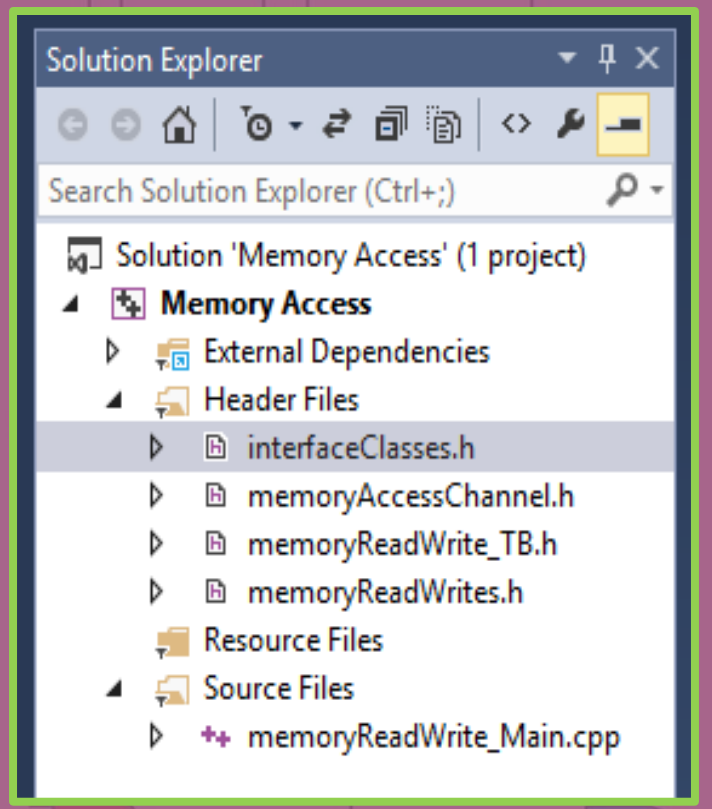
```
Initiator {0} intends to transmit (00101001) at: 9 ns to: [0]

Initiator {1} intends to transmit (00101001) at: 11 ns to: [0]

Initiator {3} intends to transmit (00101001) at: 15 ns to: [0]
Target [0] ready to receive something at: 17 ns
Target [0] received (00101001) at: 30 ns from: {2}
Initiator {2} completed transmtting (00101001) at: 30 ns to: [0]

Initiator {2} intends to transmit (00100011) at: 37 ns to: [0]
Target [0] ready to receive something at: 47 ns
Target [0] received (00101001) at: 60 ns from: {0}
Initiator {0} completed transmtting (00101001) at: 60 ns to: [0]

Initiator {0} intends to transmit (00100011) at: 69 ns to: [0]
Target [0] ready to receive something at: 77 ns
Target [0] received (00101001) at: 90 ns from: {1}
Initiator {1} completed transmtting (00101001) at: 90 ns to: [0]

Initiator {1} intends to transmit (00100011) at: 101 ns to: [0]
Target [0] ready to receive something at: 107 ns
Target [0] received (00100011) at: 120 ns from: {0}
Initiator {0} completed transmtting (00100011) at: 120 ns to: [0]
```

Example 5: *Priority shared bus*,output

```
Initiator {0} intends to transmit (10111110) at: 129 ns to: [0]
Target [0] ready to receive something at: 137 ns
Target [0] received (00100011) at: 150 ns from: {1}
Initiator {1} completed transmtting (00100011) at: 150 ns to: [0]

Initiator {1} intends to transmit (10111110) at: 161 ns to: [0]
Target [0] ready to receive something at: 167 ns
Target [0] received (10111110) at: 180 ns from: {0}
Initiator {0} completed transmtting (10111110) at: 180 ns to: [0]

Initiator {0} intends to transmit (10000100) at: 189 ns to: [0]
Target [0] ready to receive something at: 197 ns
Target [0] received (10111110) at: 210 ns from: {1}
Initiator {1} completed transmtting (10111110) at: 210 ns to: [0]

Initiator {1} intends to transmit (10000100) at: 221 ns to: [0]
Target [0] ready to receive something at: 227 ns
Target [0] received (10000100) at: 240 ns from: {0}
Initiator {0} completed transmtting (10000100) at: 240 ns to: [0]

Initiator {0} intends to transmit (11100001) at: 249 ns to: [0]
Target [0] ready to receive something at: 257 ns
Target [0] received (10000100) at: 270 ns from: {1}
Initiator {1} completed transmtting (10000100) at: 270 ns to: [0]
```

Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface



Example 6: *Memory Access*, *interfaceClasses.h*

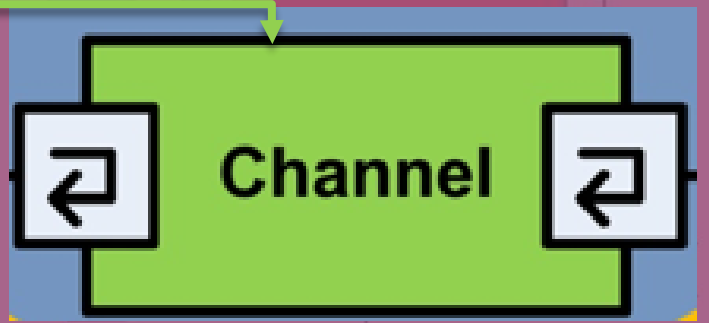
interfaceClasses.h

```
1   #include <systemc.h>
2
3   template <class T>
4   class requestMem_if : virtual public sc_interface
5   {
6   public:
7       virtual void requestMem(int initiator, T &data, T &address, bool rwbar,int delay) = 0;
8
9   };
10
11  template <class T>
12  class memRespond_if : virtual public sc_interface
13  {
14  public:
15     virtual void memForward(int &initiator, T &data, T &address, bool &rwbar) = 0;
16     virtual void memBackward(T &data, int delay) = 0;
17  };
```

Example 6: *Memory Access*, *memoryAccessChannel.h*

```
1   #include "interfaceClasses.h"
2
3   // data and address type, number of Inititiators
4   template <class T, int nI>
5   class memoryAccess : public requestMem_if<T>, public memRespond_if<T>
6   {
7     int comingFrom;
8     T incomingData;
9     T incomingAddress;
10    T outgoingData;
11    bool read;
12    bool memoryRequested;
13    sc_event memoryCalledFor;
14    sc_event memoryCompleted;
15
16    sc_mutex busBusy;
17
18    public:
19
20    memoryAccess() : comingFrom(-1), memoryRequested(false) {};
21    ~memoryAccess() {};
22
23    void requestMem(int initiator, T &data, T &address, bool rwbar, int delay) { ... }
43
44    void memForward(int &initiator, T &data, T &address, bool &rwbar) { ... }
53    void memBackward(T &data, int delay) { ... }
59  };
```

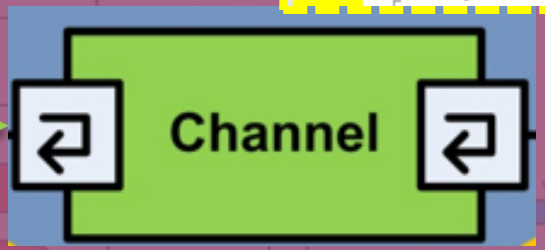
memoryAccessChannel.h



Example 6: *Memory Access*, *memoryAccessChannel.h*

```
23      void requestMem(int initiator, T &data, T &address, bool rwbar, int delay){
24          wait(delay, SC_NS);
25          busBusy.lock();
26          comingFrom = initiator;
27          if (!rwbar) incomingData = data;
28          incomingAddress = address;
29          read = rwbar;
30          memoryRequested = true;
31          memoryCalledFor.notify();
32          wait(memoryCompleted);
33          if (rwbar) data = outgoingData;
34      ...
41          busBusy.unlock();
42      }
```

```
44      void memForward(int &initiator, T &data, T &address, bool &rwbar){
45          if (!memoryRequested) wait(memoryCalledFor);
46          memoryRequested = false;
47          initiator = comingFrom;
48          if (!read) data = incomingData;
49          address = incomingAddress;
50          rwbar = read;
51          // Ready for Backward operation
52      }
53      void memBackward(T &data, int delay){
54          outgoingData = data;
55          wait(delay, SC_NS);
56          comingFrom = -1;
57          memoryCompleted.notify();
58      }
```



memoryAccessChannel.h

Example 6: *Memory Access*, *memoryAccessChannel.h*

```
23  void requestMem(int initiator, T &data, T &address, bool rwbar, int delay){
24      wait(delay, SC_NS);
25      busBusy.lock();
26      comingFrom = initiator;
27      if (!rwbar) incomingData = d
28      incomingAddress = address;
29      read = rwbar;
30      memoryRequested = true;
31      memoryCalledFor.notify();
32      wait(memoryCompleted);
33      if (rwbar) data = outgoingDa
34  ...
41      busBusy.unlock();
42  }
```

```
44  void memForward(int &initiator, T &data, T &address, bool &rwbar){
45      if (!memoryRequested) wait(memoryCalledFor);
46      memoryRequested = false;
47      initiator = comingFrom;
48      if (!read) data = incomingData;
49      address = incomingAddress;
50      rwbar = read;
51      // Ready for Backward operation
52  }
53  void memBackward(T &data, int delay){
54      outgoingData = data;
55      wait(delay, SC_NS);
56      comingFrom = -1;
57      memoryCompleted.notify();
58  }
```

Initiator

Target

memoryAccessChannel.h

Example 6: *Memory Access*, *memoryReadWrites.h*

```
1   #include "memoryAccessChannel.h"
2
3   // initiator Number, Frequency, Dela
4   template <int N, int F, int D>
5   SC_MODULE (initiators) {
6       sc_port<requestMem_if<sc_lv<8>>>
7
8       SC_CTOR(initiators) {
9           SC_THREAD (requesting);
10      }
11      void requesting();
12  };
```

memoryReadWrites.h

```
14  template <int N, int F, int D>
15  void initiators<N, F, D>::requesting() {
16      sc_lv<8> tData; // transmitted Data
17      sc_lv<8> tAddress; // transmitted Address
18      bool rwbar;
19
20      for (int i = 0; i < 15; i++)
21      {
22          wait(F, SC_NS);
23          rwbar = (bool) (rand() % 2);
24          tData = (rwbar?((sc_lv<8>)255) : (sc_lv<8>) (rand() % 256));
25          tAddress = (sc_lv<8>) (rand() % 256);
26  ...
29          if (!rwbar) cout << " Data:" << tData;
30  ...
32          out->requestMem(N, tData, tAddress, rwbar, D);
33  ...
36      }
37  }
```

Example 6: *Memory Access,*
memoryReadWrites.h

```
3   // initiator Number, Frequency, Delay
4   template <int N, int F, int D>
5   SC_MODULE (initiators) { ... }
13
14  template <int N, int F, int D>
15  void initiators<N, F, D>::requesting() { ... }
38
39  // memory Number, Frequency, Delay
40  template <int N, int F, int D>
41  SC_MODULE(memory) {
42      sc_lv<8>* mem;
43
44      sc_port<memRespond_if<sc_lv<8>>> in;
45
46      SC_CTOR(memory) {
47          mem = new sc_lv<8>[256];
48          for (int i = 0; i < 256; i++) mem[i] = (sc_lv<8>) i;
49
50          SC_THREAD (responding);
51      }
52      void responding();
53      virtual ~memory(){
54          delete[] mem;
55      }
56  };
```

memoryReadWrites.h

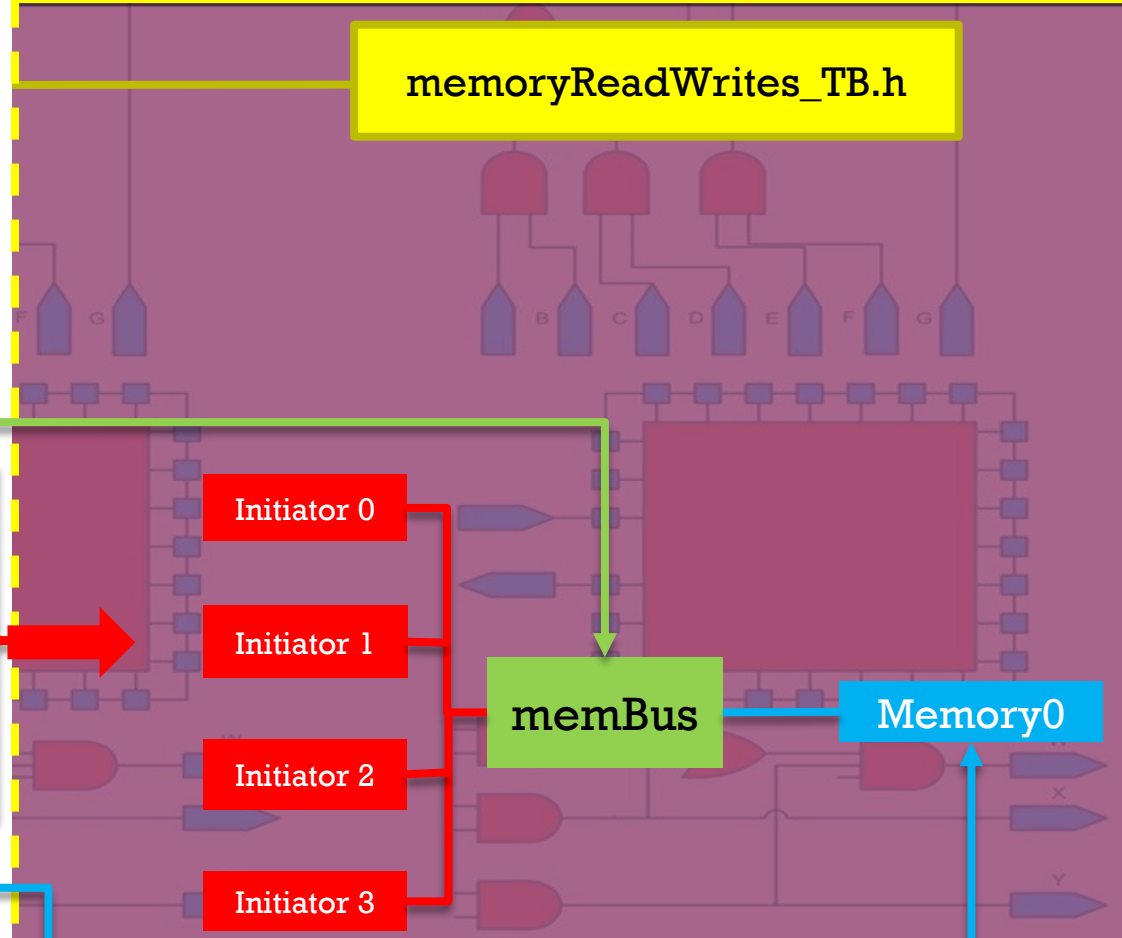
Example 6: *Memory Access*, *memoryReadWrites.h*

memoryReadWrites.h

```
57
58   template <int N, int F, int D>
59  void memory<N, F, D>::responding() {
60       sc_lv<8> rData;     // received Data
61       sc_lv<8> rAddress; // received address
62       bool rwbar;         // read write
63       sc_lv<8> sData;     // sending data back
64       int dataInitiator;
65
66       while (1)
67       {
68           wait(F, SC_NS);
69           in->memForward(dataInitiator, rData, rAddress, rwbar);
70               if (rwbar) sData = mem[rAddress.to_uint()];
71               else mem[rAddress.to_uint()] = rData;
72           in->memBackward(sData, D);
73  ...
76       }
77   }
78
```

Example 6: *Memory Access,*
memoryReadWrites_TB.h

```
1   #include "memoryReadWrites.h"
2
3   SC_MODULE(memoryAccess_TB) {
4
5       memoryAccess<sc_lv<8>, 4> * memBus;
6       initiators<0, 9, 0>* INI0;
7       initiators<1, 1111, 0>* INI1;
8       initiators<2, 117, 0>* INI2;
9       initiators<3, 1115, 0>* INI3;
10      memory<0, 17, 0>* MEM0;
11
12      SC_CTOR(memoryAccess_TB) {
13          memBus = new memoryAccess<sc_lv<8>, 4>;
14
15          INI0 = new initiators<0, 9, 0>("Initiator0");
16          INI0->out(*memBus);
17          INI1 = new initiators<1, 1111, 0>("Initiator1");
18          INI1->out(*memBus);
19          INI2 = new initiators<2, 117, 0>("Initiator2");
20          INI2->out(*memBus);
21          INI3 = new initiators<3, 1115, 0>("Initiator3");
22          INI3->out(*memBus);
23
24          MEM0 = new memory<0, 17, 0>("Memory0");
25          MEM0->in(*memBus);
26      }
27  };
28
```



Example 6: *Memory Access*, *memoryReadWrites_main*

memoryReadWrites_main.cpp

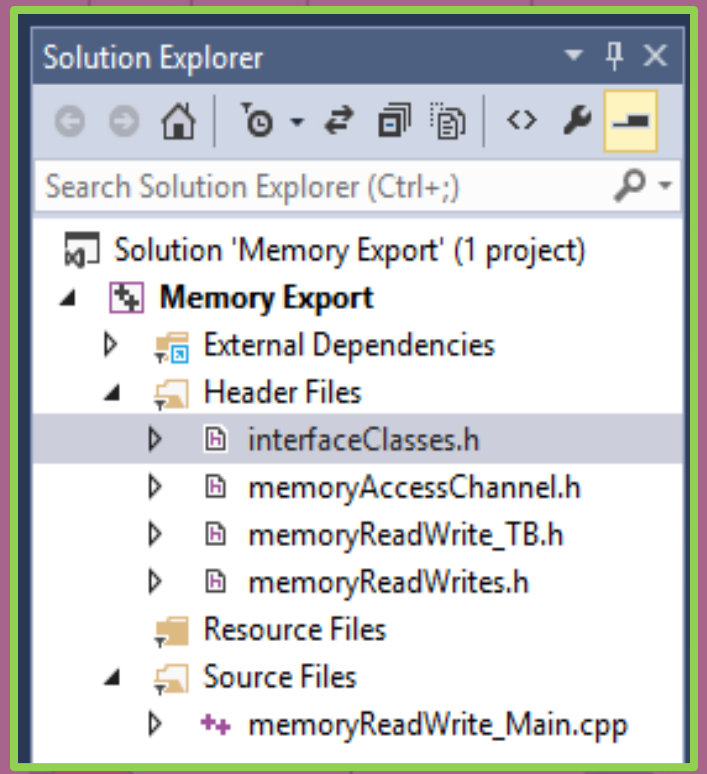
```
1  #include "memoryReadWrite_TB.h"
2
3  int sc_main (int argc , char *argv[]) {
4      memoryAccess_TB MemoryAccess ("memoryAccess1");
5      sc_start();
6      return 0;
7  }
8
```

Example 6: *Memory Access, output*

```
Initiator {0} intends to read from Address:00100011 at: 9 ns
Memory READ Data:00100011 Address:00100011 requested by {0}
Reading 0: incomingAddress:35, outgoingData:00100011
Initiator {0} completed rwbar:1 Data:00100011 Address:00100011 at: 17 ns

Initiator {0} intends to write to Address:11100001 Data:10000100 at: 26 ns
Memory WROTE Data:10000100 Address:11100001 requested by {0}
Writing 0: incomingData:10000100, incomingAddress:225
Initiator {0} completed rwbar:0 Data:10000100 Address:11100001 at: 34 ns

Initiator {0} intends to write to Address:10101110 Data:11010110 at: 43 ns
Memory WROTE Data:11010110 Address:10101110 requested by {0}
Writing 0: incomingData:11010110, incomingAddress:174
Initiator {0} completed rwbar:0 Data:11010110 Address:10101110 at: 51 ns

Initiator {0} intends to write to Address:01001001 Data:10010000 at: 60 ns
Memory WROTE Data:10010000 Address:01001001 requested by {0}
Writing 0: incomingData:10010000, incomingAddress:73
Initiator {0} completed rwbar:0 Data:10010000 Address:01001001 at: 68 ns

Initiator {0} intends to read from Address:11110001 at: 77 ns
Memory READ Data:11110001 Address:11110001 requested by {0}
Reading 0: incomingAddress:241, outgoingData:11110001
Initiator {0} completed rwbar:1 Data:11110001 Address:11110001 at: 85 ns
```

Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface



Example 7: *Memory Export, interfaceClasses.h*

interfaceClasses.h

```
1   #include <systemc.h>
2
3   template <class T>
4   class requestMem_if : virtual public sc_interface
5   {
6   public:
7       virtual void requestMem(int initiator, T &data, T &address, bool rwbar, int delay) = 0;
8
9   };
10
11  template <class T>
12  class memRespond_if : virtual public sc_interface
13  {
14  public:
15     virtual void memForward(int &initiator, T &data, T &address, bool &rwbar) = 0;
16     virtual void memBackward(T &data, int delay) = 0;
17  };
```

Example 7: *Memory Export*, *memoryAcessChannel.h*

```
1   #include "interfaceClasses.h"
2
3   // data and address type, number of Inititiators
4   template <class T, int nI>
5   class memoryAccess : public requestMem_if<T>, public memRespond_if<T>
6   {
7       int comingFrom;
8       T incomingData;
9       T incomingAddress;
10      T outgoingData;
11      bool read;
12      bool memoryRequested;
13      sc_event memoryCalledFor;
14      sc_event memoryCompleted;
15
16      sc_mutex busBusy;
17
18  public:
19      memoryAccess() : comingFrom(-1), memoryRequested(false) {};
20      ~memoryAccess() {};
21
22      void requestMem(int initiator, T &data, T &address, bool rwbar, int delay) { ... }
41
42      void memForward(int &initiator, T &data, T &address, bool &rwbar) { ... }
51      void memBackward(T &data, int delay) { ... }
57  };
```

memoryAcessChannel.h

Example 7: *Memory Export*, *memoryAcessChannel.h*

```
1   #include "interfaceClasses.h"
2
3   // data and address type, number of Inititiators
4   template <class T, int nI>
5   class memoryAccess : public requestMem_if<T>, public memRespond_if<T>
6   {
7   ...
17
18  public:
19      memoryAccess() : comingFrom(-1), memoryRequested(false) {};
20      ~memoryAccess() {};
21
22      void requestMem(int initiator, T &data, T &address, bool rwbar, int delay){
23          wait(delay, SC_NS);
24          busBusy.lock();
25          comingFrom = initiator;
26          if (!rwbar) incomingData = data;
27          incomingAddress = address;
28          read = rwbar;
29          memoryRequested = true;
30          memoryCalledFor.notify();
31          wait(memoryCompleted);
32          if (rwbar) data = outgoingData;
33  ...
39          busBusy.unlock();
40          }
```

memoryAcessChannel.h

Example 9: *Memory Export*, *memoryAcessChannel.h*

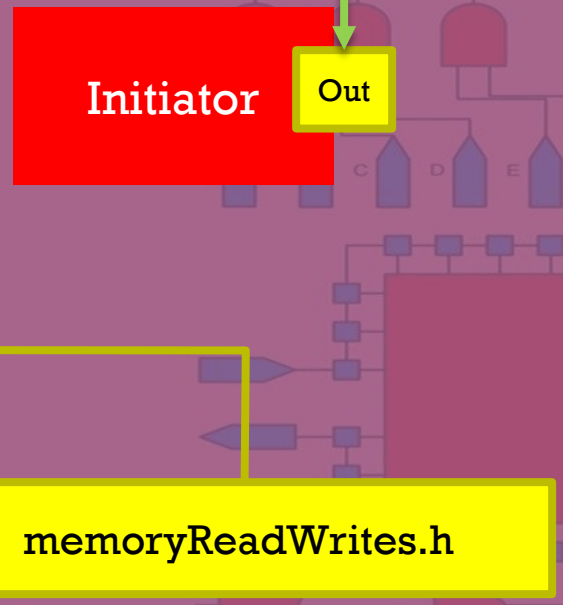
```
42      void memForward(int &initiator, T &data, T &address, bool &rwbar){
43          if (!memoryRequested) wait(memoryCalledFor);
44          memoryRequested = false;
45          initiator = comingFrom;
46          if (!read) data = incomingData;
47          address = incomingAddress;
48          rwbar = read;
49          // Ready for Backward operation
50      }
51      void memBackward(T &data, int delay){
52          outgoingData = data;
53          wait(delay, SC_NS);
54          comingFrom = -1;
55          memoryCompleted.notify();
56      }
57  };
58
```

memoryAcessChannel.h

Example 7: *Memory Export,*

memoryReadWrites.h

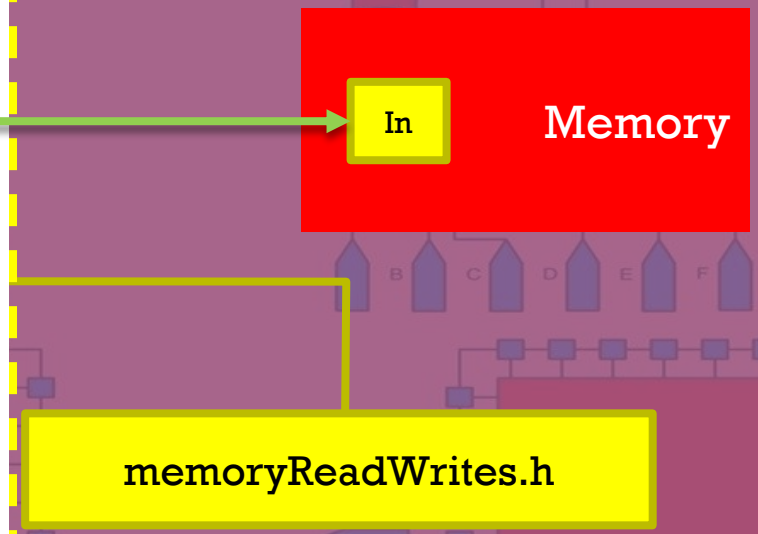
```
1   #include "memoryAccessChannel.h"
2
3   // initiator Number, Frequency, Delay
4   template <int N, int F, int D>
5   SC_MODULE (initiators) {
6       sc_port<requestMem_if<sc_lv<8>>> out;
7
8       SC_CTOR(initiators) {
9           SC_THREAD (requesting);
10      }
11      void requesting();
12  };
13
14  template <int N, int F, int D>
15  void initiators<N, F, D>::requesting() {
16      sc_lv<8> tData; // transmitted Data
17      sc_lv<8> tAddress; // transmitted Address
18      bool rwbar;
19
20      for (int i = 0; i < 15; i++)
21      {
22          wait(F, SC_NS);
23          rwbar = (bool) (rand() % 2);
24          tData = (rwbar?((sc_lv<8>)255) : (sc_lv<8>) (rand() % 256));
25          tAddress = (sc_lv<8>) (rand() % 256);
26  ...
36      }
37  }
38
39  // memory Number, Frequency, Delay
40  template <int N, int F, int D>
41  SC_MODULE(memory) { ... }
59
60  template <int N, int F, int D>
61  void memory<N, F, D>::responding() { ... }
80
```



Example 7: *Memory Export,*

memoryReadWrites.h

```
4   template <int N, int F, int D>
5   SC_MODULE (initiators) { ... }
13
14  template <int N, int F, int D>
15  void initiators<N, F, D>::requesting() { ... }
38
39  // memory Number, Frequency, Delay
40  template <int N, int F, int D>
41  SC_MODULE(memory) {
42      sc_lv<8>* mem;
43
44      sc_export<memRespond_if<sc_lv<8>>> in;
45      memoryAccess<sc_lv<8>, 4>* memBus;
46
47      SC_CTOR(memory) {
48          mem = new sc_lv<8>[256];
49          for (int i = 0; i < 256; i++) mem[i] = (sc_lv<8>) i;
50
51          memBus = new memoryAccess<sc_lv<8>, 4>;
52          SC_THREAD (responding);
53      }
54      void responding();
55      virtual ~memory(){
56          delete[] mem;
57      }
58  };
```



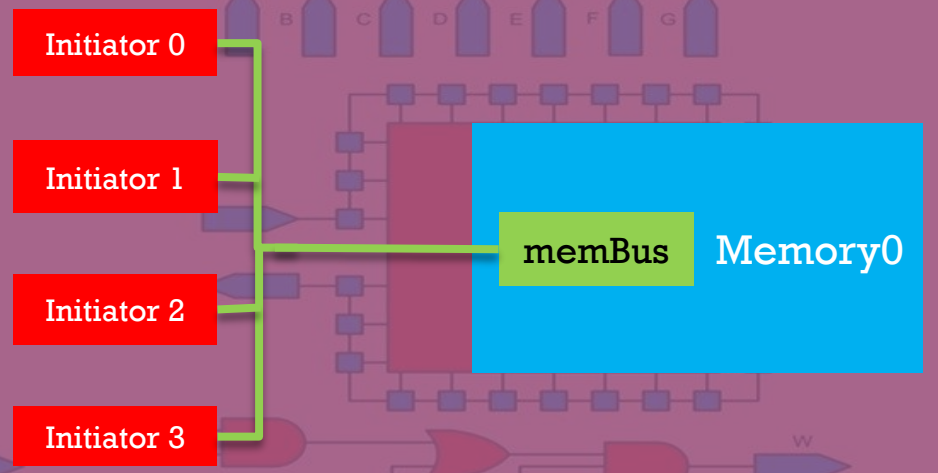
Example 7: *Memory Export*, *memoryReadWrites.h*

```
59
60   template <int N, int F, int D>
61 void memory<N, F, D>::responding() {
62       sc_lv<8> rData;    // received Data
63       sc_lv<8> rAddress; // received address
64       bool rwbar;        // read write
65       sc_lv<8> sData;    // sending data back
66       int dataInitiator;
67
68       while (1)
69       {
70           wait(F, SC_NS);
71           in->memForward(dataInitiator, rData, rAddress, rwbar);
72               if (rwbar) sData = mem[rAddress.to_uint()];
73               else mem[rAddress.to_uint()] = rData;
74           in->memBackward(sData, D);
75   ...
78       }
79   }
```

memoryReadWrites.h

Example 7: *Memory Export*, *memoryReadWrites.h*

```
1   #include "memoryReadWrites.h"
2
3   SC_MODULE(memoryAccess_TB) {
4
5       initiators<0, 9, 0>* INI0;
6       initiators<1, 1111, 0>* INI1;
7       initiators<2, 117, 0>* INI2;
8       initiators<3, 1115, 0>* INI3;
9       memory<0, 17, 0>* MEM0;
10
11      SC_CTOR(memoryAccess_TB) {
12
13          MEM0 = new memory<0, 17, 0>("Memory0");
14          MEM0->in(*MEM0->memBus);
15
16          INI0 = new initiators<0, 9, 0>("Initiator0");
17          INI0->out(*MEM0->memBus);
18          INI1 = new initiators<1, 1111, 0>("Initiator1");
19          INI1->out(*MEM0->memBus);
20          INI2 = new initiators<2, 117, 0>("Initiator2");
21          INI2->out(*MEM0->memBus);
22          INI3 = new initiators<3, 1115, 0>("Initiator3");
23          INI3->out(*MEM0->memBus);
24      }
25  };
```



Example 7: *Memory Export*, *memoryReadWrites_main*

memoryReadWrites_main.cpp

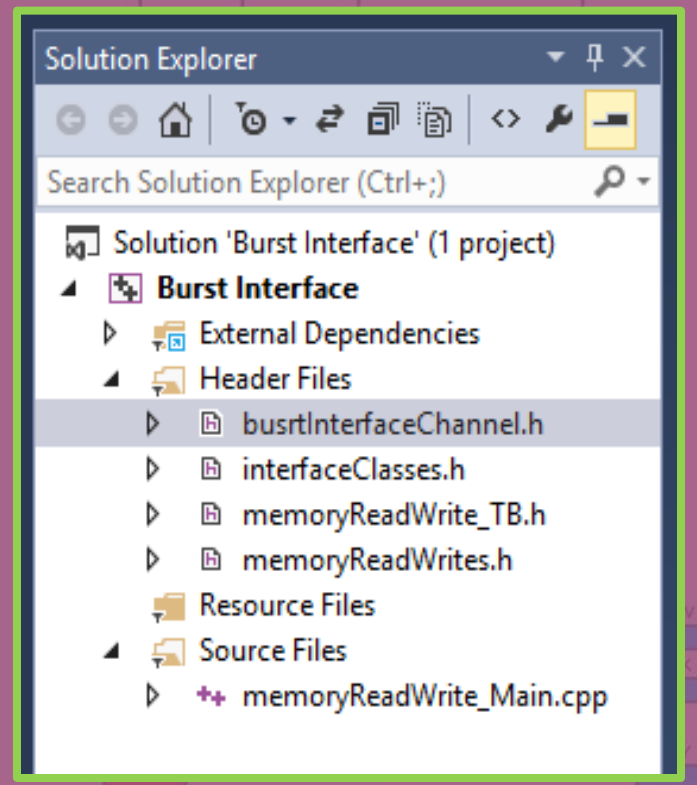
```
1  #include "memoryReadWrite_TB.h"
2  int sc_main (int argc , char *argv[]) {
3      memoryAccess_TB MemoryAccess ("memoryAccess1");
4      sc_start();
5      return 0;
6  }
7
```

Example 7: *Memory Export*, output

```
Initiator {0} intends to read from Address:00100011 at: 9 ns
Memory READ Data:00100011 Address:00100011 requested by {0}
Reading 0: incomingAddress:35, outgoingData:00100011
Initiator {0} completed rwbar:1 Data:00100011 Address:00100011 at: 17 ns

Initiator {0} intends to write to Address:11100001 Data:10000100 at: 26 ns
Memory WROTE Data:10000100 Address:11100001 requested by {0}
Writing 0: incomingData:10000100, incomingAddress:225
Initiator {0} completed rwbar:0 Data:10000100 Address:11100001 at: 34 ns

Initiator {0} intends to write to Address:10101110 Data:11010110 at: 43 ns
Memory WROTE Data:11010110 Address:10101110 requested by {0}
Writing 0: incomingData:11010110, incomingAddress:174
Initiator {0} completed rwbar:0 Data:11010110 Address:10101110 at: 51 ns

Initiator {0} intends to write to Address:01001001 Data:10010000 at: 60 ns
Memory WROTE Data:10010000 Address:01001001 requested by {0}
Writing 0: incomingData:10010000, incomingAddress:73
Initiator {0} completed rwbar:0 Data:10010000 Address:01001001 at: 68 ns

Initiator {0} intends to read from Address:11110001 at: 77 ns
Memory READ Data:11110001 Address:11110001 requested by {0}
Reading 0: incomingAddress:241, outgoingData:11110001
Initiator {0} completed rwbar:1 Data:11110001 Address:11110001 at: 85 ns
```

Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface



Example 8: *Burst interface*, *interfaceClasses.h*

interfaceClasses.h

```
1   #include <systemc.h>
2
3   template <class Ti, class Ta>
4   class requestMem_if : virtual public sc_interface
5   {
6       public:
7           virtual void requestMem(int initiator, Ti &data, Ta &address, bool rwbar, int
              delay) = 0;
8
9   };
10
11  template <class Tt, class Ta>
12  class memRespond_if : virtual public sc_interface
13  {
14     public:
15          virtual void memForward(int &initiator, Tt &data, Ta &address, bool &rwbar) = 0;
16          virtual void memBackward(Tt &data, int delay) = 0;
17  };
```

Example 8: *Burst interface*, *burstinterfaceChannel.h*

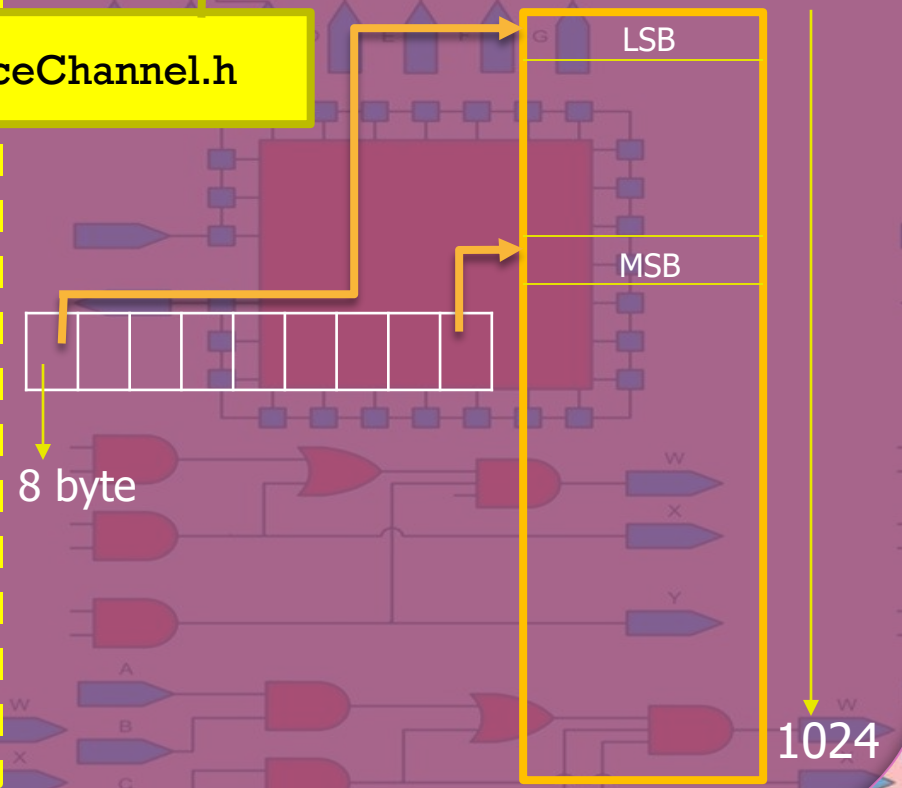
```
1   #include "interfaceClasses.h"
2
3   // data and address type, number of Inititiators
4   template <class Ti, class Tt, class Ta, int nI>
5   class memoryAccess : public requestMem_if<Ti, Ta>, public memRespond_if<Tt, Ta>
6   {
7       int comingFrom;
8
9       Ti incomingData;
10      Tt incomingSegments;
11      Ta incomingAddress;
12      Tt outgoingData;
13      bool read;
14      bool memoryRequested;
15      sc_event memoryCalledFor;
16      sc_event memoryCompleted;
17
18      sc_mutex busBusy;
19
20      public:
21      memoryAccess() : comingFrom(-1), memoryRequested(false) {};
22          ~memoryAccess() {};
23
24      void requestMem(int initiator, Ti &data, Ta &address, bool rwbar, int delay) { ... }
53
54      void memForward(int &initiator, Tt &data, Ta &address, bool &rwbar) { ... }
63      void memBackward(Tt &data, int delay) { ... }
69  };
70
```

burstinterfaceChannel.h

Example 8: *Burst interface*, *burstinterfaceChannel.h*

```
20      public:
21      memoryAccess() : comingFrom(-1), memoryRequested(false) {};
22          ~memoryAccess() {};
23
24      void requestMem(int initiator, Ti &data, Ta &address, bool rwbar, int delay){
25          int burstLength;
26          wait(delay, SC_NS);
27          busBusy.lock();
28          burstLength = data.length() / outgoingData.length();
29          for (int j = burstLength; j > 0; j--)
30          {
31              comingFrom = initiator;
32              // Big Endian (Data byte Ends (its LSB) in Bigger address location)
33              if (!rwbar) incomingSegments = data.range(j*outgoingData.length()-1,
34                                                        (j-1)*outgoingData.length());
35              incomingAddress = (address.range(address.length()-1 , 3) ,
36                                               (sc_lv<3>)(burstLength - j));
37              read = rwbar;
38              memoryRequested = true;
39              memoryCalledFor.notify();
40              wait(memoryCompleted);
41              if (rwbar) data.range(j*outgoingData.length()-1,
42                                    (j-1)*outgoingData.length()) = outgoingData;
43  ...
50          }
51          busBusy.unlock();
52      }
```

burstinterfaceChannel.h



Example 8: *Burst interface*, *burstinterfaceChannel.h*

```
54      void memForward(int &initiator, Tt &data, Ta &address, bool &rwbar){
55          if (!memoryRequested) wait(memoryCalledFor);
56          memoryRequested = false;
57          initiator = comingFrom;
58          if (!read) data = incomingSegments;
59          address = incomingAddress;
60          rwbar = read;
61          // Ready for Backward operation
62      }
63      void memBackward(Tt &data, int delay){
64              outgoingData = data;
65              wait(delay, SC_NS);
66              comingFrom = -1;
67              memoryCompleted.notify();
68          }
69  };
```

burstinterfaceChannel.h

Example 8: *Burst interface,*

memoryReadWrites.h

```
1   #include "busrtInterfaceChannel.h"
2
3   // initiator Number, Frequency, Delay
4   template <int N, int F, int D>
5   SC_MODULE (initiators) {
6       sc_port<requestMem_if<sc_lv<64>, sc_lv<16> >> out;
7
8       SC_CTOR(initiators) {
9           SC_THREAD (requesting);
10      }
11      void requesting();
12  };
13
14  template <int N, int F, int D>
15  void initiators<N, F, D>::requesting() {
16      sc_lv<64> tData; // transmitted Data
17      sc_lv<64> dataToWrite;
18      sc_lv<16> tAddress; // transmitted Address (start)
19      bool rwbar;
20
21      for (int i = 0; i < 15; i++)
22      {
23          wait(F, SC_NS);
24          rwbar = (bool) (rand() % 2);
25          dataToWrite.range(63, 32) = rand();
26          dataToWrite.range(31, 0) = rand();
27
28          tData = rwbar ? ((sc_lv<64>)255) : dataToWrite;
29          tAddress = ((sc_lv<13>)(rand() % 512), "000"); // From ...000 to ...111
30      ...
36          out->requestMem(N, tData, tAddress, rwbar, D);
37      ...
40      }
41  }
```

memoryReadWrites.h

Example 8: *Burst interface*, *memoryReadWrites.h*

```
43   // memory Number, Frequency, Delay
44   template <int N, int F, int D>
45 ⊞SC_MODULE(memory) { ... }
62
63   template <int N, int F, int D>
64 ⊞void memory<N, F, D>::responding() { ... }
83
```

```
3    // initiator Number, Frequency, Delay
4    template <int N, int F, int D>
5  ⊞SC_MODULE (initiators) { ... }
13
14   template <int N, int F, int D>
15 ⊞void initiators<N, F, D>::requesting() { ... }
42
43   // memory Number, Frequency, Delay
44   template <int N, int F, int D>
45 ⊟SC_MODULE(memory) {
46       sc_lv<8>* mem;
47
48       sc_port<memRespond_if<sc_lv<8>, sc_lv<16> >> in;
49
50   ⊟   SC_CTOR(memory) {
51           mem = new sc_lv<8>[4096]; // Addressable memory
52           for (int i = 0; i < 4096; i++)
53               mem[i] = (sc_lv<8>) (i%256);
54
55           SC_THREAD (responding);
56       }
57       void responding();
58   ⊟   virtual ~memory(){
59           delete[] mem;
60       }
61   };
```

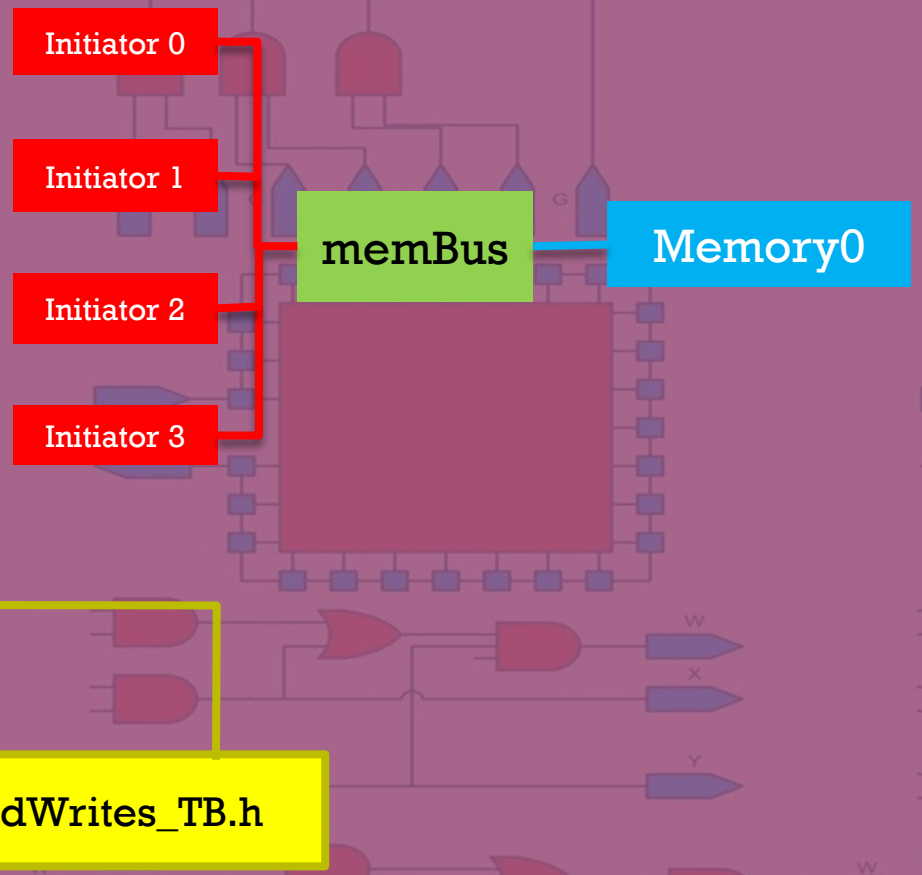
Example 8: *Burst interface*, *memoryReadWrites.h*

```
43  // memory Number, Frequency, Delay
44  template <int N, int F, int D>
45  SC_MODULE(memory) { ... }
62
63  template <int N, int F, int D>
64  void memory<N, F, D>::responding() { ... }
83
```

```
63  template <int N, int F, int D>
64  void memory<N, F, D>::responding() {
65      sc_lv<8> rData;     // received Data
66      sc_lv<16> rAddress; // received address
67      bool rwbar;         // read write
68      sc_lv<8> sData;     // sending data back
69      int dataInitiator;
70
71      while (1)
72      {
73          wait(F, SC_NS);
74          in->memForward(dataInitiator, rData, rAddress, rwbar);
75              if (rwbar) sData = mem[rAddress.range(11 , 0).to_uint()];
76              else mem[rAddress.range(11, 0).to_uint()] = rData;
77          in->memBackward(sData, D);
78  ...
81      }
82  }
```

Example 8: *Burst interface*, *memoryReadWrites_TB.h*

```
1   #include "memoryReadWrites.h"
2
3   SC_MODULE(memoryAccess_TB) {
4
5       memoryAccess<sc_lv<64>, sc_lv<8>, sc_lv<16>, 4> * memBus;
6       initiators<0, 9, 0>* INI0;
7       initiators<1, 1111, 0>* INI1;
8       initiators<2, 117, 0>* INI2;
9       initiators<3, 1115, 0>* INI3;
10      memory<0, 17, 0>* MEM0;
11
12      SC_CTOR(memoryAccess_TB) {
13          memBus = new memoryAccess<sc_lv<64>, sc_lv<8>, sc_lv<16>, 4>;
14
15          INI0 = new initiators<0, 9, 0>("Initiator0");
16          INI0->out(*memBus);
17          INI1 = new initiators<1, 1111, 0>("Initiator1");
18          INI1->out(*memBus);
19          INI2 = new initiators<2, 117, 0>("Initiator2");
20          INI2->out(*memBus);
21          INI3 = new initiators<3, 1115, 0>("Initiator3");
22          INI3->out(*memBus);
23
24          MEM0 = new memory<0, 17, 0>("Memory0");
25          MEM0->in(*memBus);
26      }
27  };
28
```



Example 8: *Burst interface*, *memoryReadWrites_main*

```cpp
#include "memoryReadWrite_TB.h"

int sc_main (int argc , char *argv[]) {
    memoryAccess_TB MemoryAccess ("memoryAccess1");
    sc_start();
    return 0;
}

```

memoryReadWrites_main.cpp

Example 8: *Burst interface, output*

```
Writing 3: incomingData:00000000, incomingAddress:1776
Memory WROTE Data:00000000 Address:0000011011110001 requested by {3}
Writing 3: incomingData:00000000, incomingAddress:1777
Memory WROTE Data:01001101 Address:0000011011110010 requested by {3}
Writing 3: incomingData:01001101, incomingAddress:1778
Memory WROTE Data:10110111 Address:0000011011110011 requested by {3}
Writing 3: incomingData:10110111, incomingAddress:1779
Memory WROTE Data:00000000 Address:0000011011110100 requested by {3}
Writing 3: incomingData:00000000, incomingAddress:1780
Memory WROTE Data:00000000 Address:0000011011110101 requested by {3}
Writing 3: incomingData:00000000, incomingAddress:1781
Memory WROTE Data:00010101 Address:0000011011110110 requested by {3}
Writing 3: incomingData:00010101, incomingAddress:1782
Memory WROTE Data:01000111 Address:0000011011110111 requested by {3}
Writing 3: incomingData:01000111, incomingAddress:1783
Initiator {3} completed rwbar:0 Data:0000000000000000100110110110111000000000000
0000000101010100111 Address:0000011011110000 at: 10658 ns

Initiator {1} intends to read from Address:0000111001000000 at: 11341 ns
Memory READ Data:01000000 Address:0000111001000000 requested by {1}
Reading 1: incomingAddress:3648, outgoingData:01000000
Memory READ Data:01000001 Address:0000111001000001 requested by {1}
Reading 1: incomingAddress:3649, outgoingData:01000001
Memory READ Data:01000010 Address:0000111001000010 requested by {1}
Reading 1: incomingAddress:3650, outgoingData:01000010
```

Abstract Channels

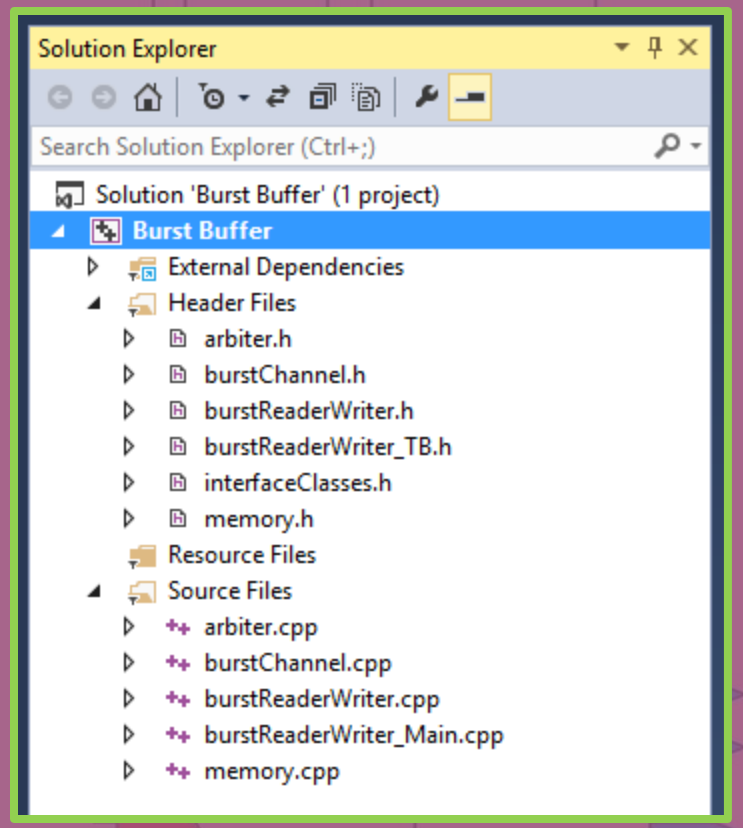
- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface

Hierarchical Channels

- Can contain complex behavior
- Implemented as modules in SystemC
  - Derived from sc_module
- More importantly, a platform for defining interfaces

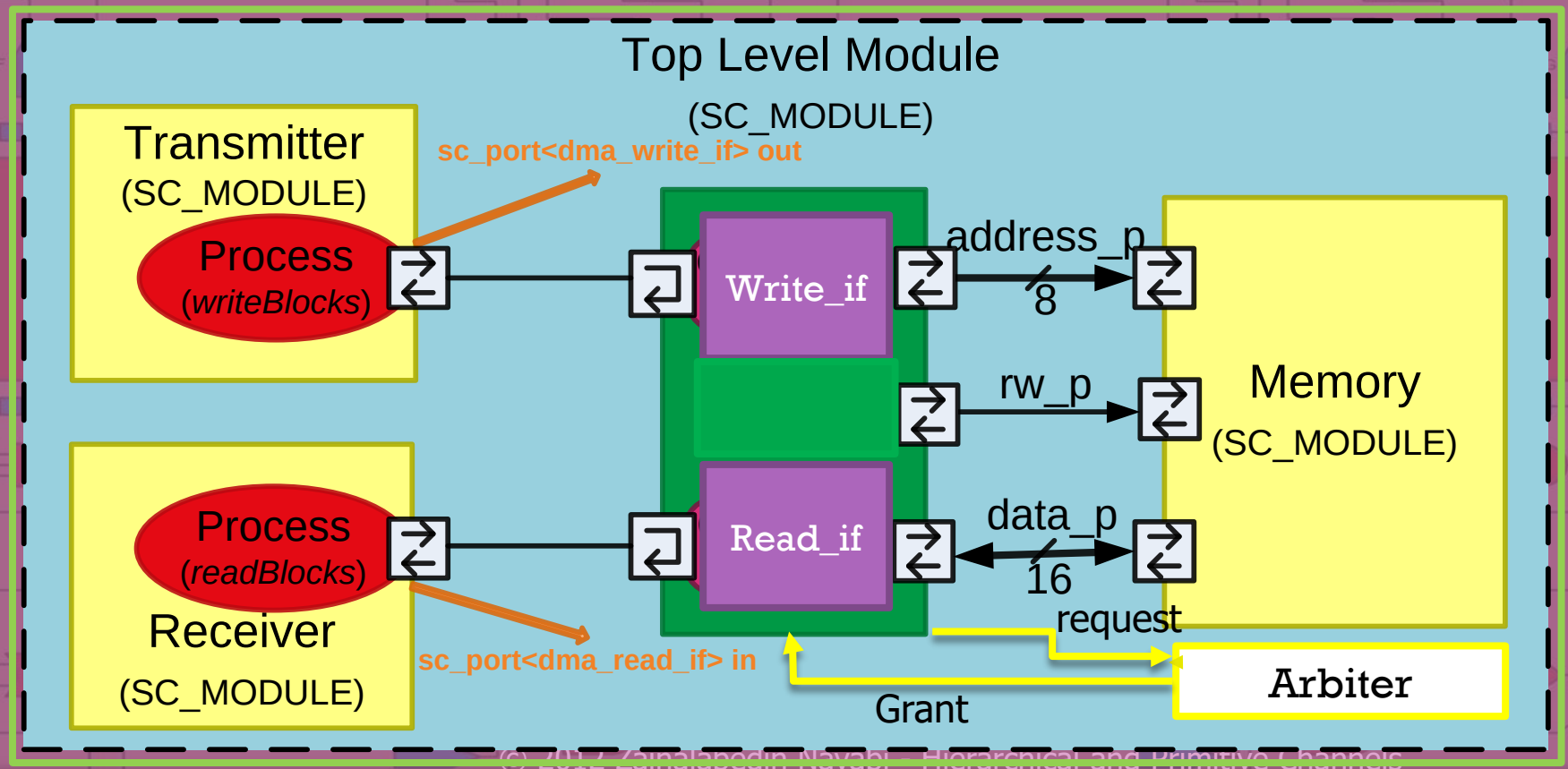
Abstract Channels

- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using sc_port and sc_export
    - Burst interface handler
  - Hierarchical Channels
    - Burst buffer with RTL interface



Burst Buffer

- Burst Channel
  - Port on one side and interface on the other



Example :Burst Buffer, *interfaceClasses*

interfaceClasses.h

```
1   #include <systemc.h>
2
3   class burst_write_if: virtual public sc_interface
4       {
5           public:
6               virtual void burstWrite(sc_lv<13> address, sc_lv<64> data ) = 0;
7   };
8
9   class burst_read_if: virtual public sc_interface
10      {
11          public:
12              virtual void burstRead(sc_lv<13> address, sc_lv<64>& data) = 0;
13  };
14
```

Example :Burst Buffer, *Buffer Channel.h*

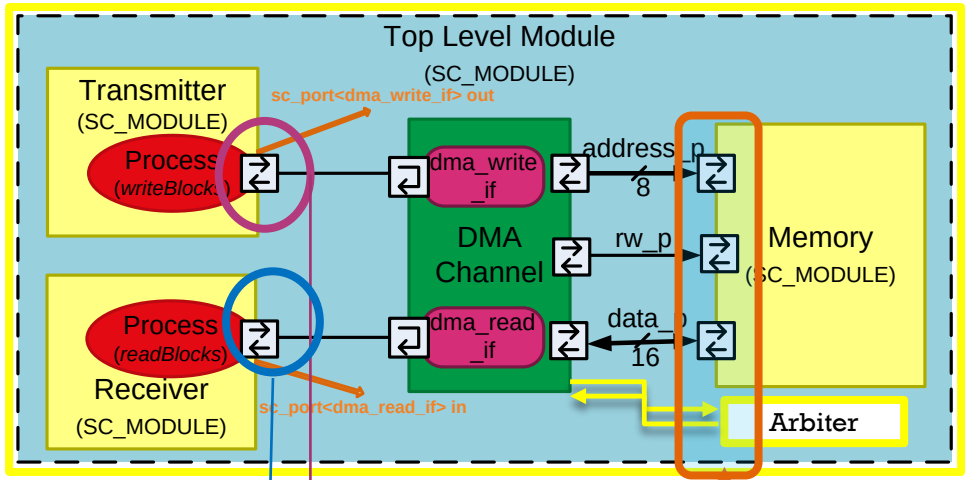
```
1   #include "interfaceClasses.h"
2
3   class burstBuffer : public sc_channel, public burst_write_if, public burst_read_if {
4
5   public:
6       sc_out_rv<16> memAddress;
7       sc_inout_rv<8> memData;
8       sc_out_resolved rwbar, cs;
9       sc_in_resolved memReady;
10      sc_out_resolved busRequest;
11      sc_in<sc_logic> busGrant;
12
13      sc_mutex burstChannelBusy;
14
15      burstBuffer (sc_module_name NAME): sc_channel(NAME){};
16  //  ~burstBuffer(){};
17
18      virtual void burstWrite(sc_lv<13> initAddress, sc_lv<64> initData);
19      virtual void burstRead(sc_lv<13> initAddress, sc_lv<64>& initData);
20  };
21
```

BufferChannel.h

Example :Burst Buffer, *Buffer Channel.h*

BufferChannel.h

```
1  #include "interfaceClasses.h"
2
3  class burstBuffer : public sc_channel, public burst_write_if, public burst_read_if {
4
5  public:
6      sc_out_rv<16> memAddress;
7      sc_inout_rv<8> memData;
8      sc_out_resolved rwbar, cs;
9      sc_in_resolved memReady;
10     sc_out_resolved busRequest;
11     sc_in<sc_logic> busGrant;
12
13     sc_mutex burstChannelBusy;
14
15     burstBuffer (sc_module_name NAME): sc_channel(NAME){};
16 //  ~burstBuffer(){};
17
18     virtual void burstWrite(sc_lv<13> initAddress, sc_lv<64> initData);
19     virtual void burstRead(sc_lv<13> initAddress, sc_lv<64>& initData);
20 };
21
```



```cpp
#include "burstChannel.h"

void burstBuffer::burstWrite(sc_lv<13> initAddress, sc_lv<64> initData ) {
    sc_lv<16> byteAddress;
    sc_lv<8> byteData;

    burstChannelBusy.lock();

    busRequest->write(SC_LOGIC_1);
    wait(busGrant->posedge_event());

    for (int i = 0; i<8; i++) {
        byteAddress = (initAddress, (sc_lv<3>)i);
        byteData = initData.range(i*8+7, i*8);

        memAddress->write(byteAddress);
        memData->write(byteData);
        cs->write(SC_LOGIC_1);
        rwbar->write(SC_LOGIC_0);

        wait(memReady->posedge_event());
        cs->write(SC_LOGIC_0);
        wait(memReady->negedge_event());

        memAddress->write("ZZZZZZZZZZZZZZZZ");
        memData->write("ZZZZZZZZ");
        cs->write(SC_LOGIC_Z);
        rwbar->write(SC_LOGIC_Z);

        wait(1, SC_NS);
    }
    busRequest->write(SC_LOGIC_0);
    wait(busGrant->negedge_event());
    busRequest->write(SC_LOGIC_Z);

    burstChannelBusy.unlock();
}

void burstBuffer::burstRead(sc_lv<13> initAddress, sc_lv<64>& initData) { ... }
```

BufferChannel.cpp

```
1   #include "burstChannel.h"
2
3   void burstBuffer::burstWrite(sc_lv<13> initAddress, sc_lv<64> initData ) { ... }
38
39  void burstBuffer::burstRead(sc_lv<13> initAddress, sc_lv<64>& initData) {
40      sc_lv<16> byteAddress;
41      sc_lv<8> byteData;
42
43      burstChannelBusy.lock();
44
45      busRequest->write(SC_LOGIC_1);
46      wait(busGrant->posedge_event());
47      for (int i = 0; i<8; i++) {
48          byteAddress = (initAddress, (sc_lv<3>)i);
49          memAddress->write(byteAddress);
50          cs->write(SC_LOGIC_1);
51          rwbar->write(SC_LOGIC_1);
52          wait(memReady->posedge_event());
53          byteData = memData->read();
54          initData.range(i * 8 + 7, i * 8) = byteData;
55
56          cs->write(SC_LOGIC_0);
57          wait(memReady->negedge_event());
58
59          memAddress->write("ZZZZZZZZZZZZZZZ");
60          memData->write("ZZZZZZZZ");
61          cs->write(SC_LOGIC_Z);
62          rwbar->write(SC_LOGIC_Z);
63
64          wait(1, SC_NS);
65      }
66      busRequest->write(SC_LOGIC_0);
67      wait(busGrant->negedge_event());
68      busRequest->write(SC_LOGIC_Z);
69
70      burstChannelBusy.unlock();
71  }
```

BufferChannel.cpp

Example :Burst Buffer, *Arbiter.h*

Arbiter.h

```
1   #include <systemc.h>
2
3   SC_MODULE(arbiter) {
4       sc_in<sc_lv<4>> request;
5       sc_out<sc_lv<4>> grant;
6       int clockDelay;
7       sc_lv<4> granted;
8
9       SC_HAS_PROCESS(arbiter);
10      arbiter(sc_module_name NAME, int D) : sc_module(NAME), clockDelay(D) {
11          SC_THREAD(arbitration);
12          sensitive << request;
13      }
14      ~arbiter() {}
15
16      void arbitration();
17  };
```

Example :Burst Buffer, *Arbiter.cpp*

Arbiter.cpp

```
1   #include "arbiter.h"
2
3   void arbiter::arbitration() {
4       while (1) {
5           wait(clockDelay, SC_NS);
6           for (int i = 3; i >= 0; i--){
7               if (request->read()[i] == SC_LOGIC_1) granted[i] = SC_LOGIC_1;
8               else granted[i] = SC_LOGIC_0;
9           }
10          grant->write(granted);
11          wait();
12      }
13  }
14
```

Example :Burst Buffer, *Memory.h*

Memory.h

```
1   #include <systemc.h>
2
3   SC_MODULE (memory) {
4       sc_in_rv<16> addressBus;
5       sc_inout_rv<8> dataBus;
6       sc_in_resolved rwbar, cs;
7       sc_out_resolved memReady;
8       int memActivePart, memDelay;
9       sc_lv<8> *mem;
10
11      SC_HAS_PROCESS(memory);
12
13      memory(sc_module_name NAME, int P=1024, int D=9);
14      ~memory() {delete []mem;}
15
16      void memReadWrite();
17  };
```

Example :Burst Buffer,

Memory.cpp

```cpp
#include "memory.h"

memory::memory(sc_module_name NAME, int P, int D) :
        sc_module(NAME), memActivePart(P), memDelay(D) {
    mem = new sc_lv<8>[memActivePart];
    for (int i=0; i< memActivePart; i++) {
        mem[i] = sc_lv<8>(i);
    }
    SC_THREAD(memReadWrite);
}
void memory::memReadWrite() {
    while (1) {
        wait(cs->posedge_event());
        wait(memDelay, SC_NS);
        if (addressBus->read().to_uint() <= memActivePart){
            if (rwbar->read() == SC_LOGIC_1){ // Read operation
                dataBus = *(mem + addressBus->read().to_uint());
                cout << "Reading-" << *(mem + addressBus->read().to_uint())
                    << " from address: ";
            }
            else{
                *(mem + addressBus->read().to_uint()) = dataBus;
                cout << "Writing-" << dataBus << " to address: ";
            }
        }
        memReady->write(SC_LOGIC_1);
        wait(cs->negedge_event());
        dataBus = "ZZZZZZZZ";
        wait(1, SC_NS);
        memReady->write(SC_LOGIC_0);
        cout << addressBus->read().to_uint() << " at:"
            << sc_time_stamp() << '\n';
    }
}
```

Memory.cpp

Example :Burst Buffer, *Burst Reader Writer.h*

```
1   #include "burstChannel.h"
2
3   SC_MODULE (writer) {
4       sc_port<burst_write_if> out; // with if out is a pointer
5
6       SC_CTOR (writer)
7       {
8           SC_THREAD(writeBlocks);
9       }
10      void writeBlocks();
11  };
12
13  SC_MODULE (reader) {
14      sc_port<burst_read_if> in;
15
16      SC_CTOR (reader)
17      {
18          SC_THREAD(readBlocks);
19      }
20      void readBlocks();
21  };
22
```

BurstReaderWriter.h

Example :Burst Buffer,

Burst Reader Writer.cpp

```
1   #include "burstReaderWriter.h"
2
3   void writer::writeBlocks()
4   {
5       sc_lv<64> dataToWrite;
6       sc_lv<13> startAddress;
7
8       for (int j = 0; j <= 2; j++){
9
10          startAddress = (sc_lv<13>)(rand() % 127);
11          dataToWrite.range(63, 32) = rand();
12          dataToWrite.range(31,  0) = rand();
13      ...
15          out->burstWrite(startAddress, dataToWrite);
16      ...
18          wait(13, SC_NS);
19      }
20  }
21
22  void reader::readBlocks()
23  {
24      sc_lv<64> dataRead;
25      sc_lv<13> startAddress;
26
27      for (int j = 0; j <= 2; j++){
28
29          startAddress = (sc_lv<13>)(rand() % 127);
30      ...
31          in->burstRead(startAddress, dataRead);
32      ...
34          wait(53, SC_NS);
35      }
36  }
37
```

BurstReaderWriter.cpp

Example :Burst Buffer, *Burst Reader Writer TestBench*

```
1  #include "burstReaderWriter.h"
2  #include "memory.h"
3  #include "arbiter.h"
4
5  SC_MODULE(burstReaderWriter_TB) {
6      sc_signal_rv<16> memAddressBus;
7      sc_signal_rv<8> memDataBus;
8      sc_signal_resolved memRwbar, memCs;
9      sc_signal_resolved memReady;
10     sc_signal<sc_lv<4>> req;
11     sc_signal<sc_lv<4>> gnt;
12     sc_signal_resolved req0;
13     sc_signal<sc_logic> gnt0;
14
15     burstBuffer* BBChannel;
16     writer* WR1;
17     reader* RD1;
18     memory* MEM1;
19     arbiter* ARB1;
20
21     SC_CTOR(burstReaderWriter_TB) { ... }
49     void setRequest(){ sc_lv<4> r; r = 0; r[0] = req0; req.write(r); }
50     void setGrant(){gnt0 = gnt.read()[0]; }
51 };
52
```

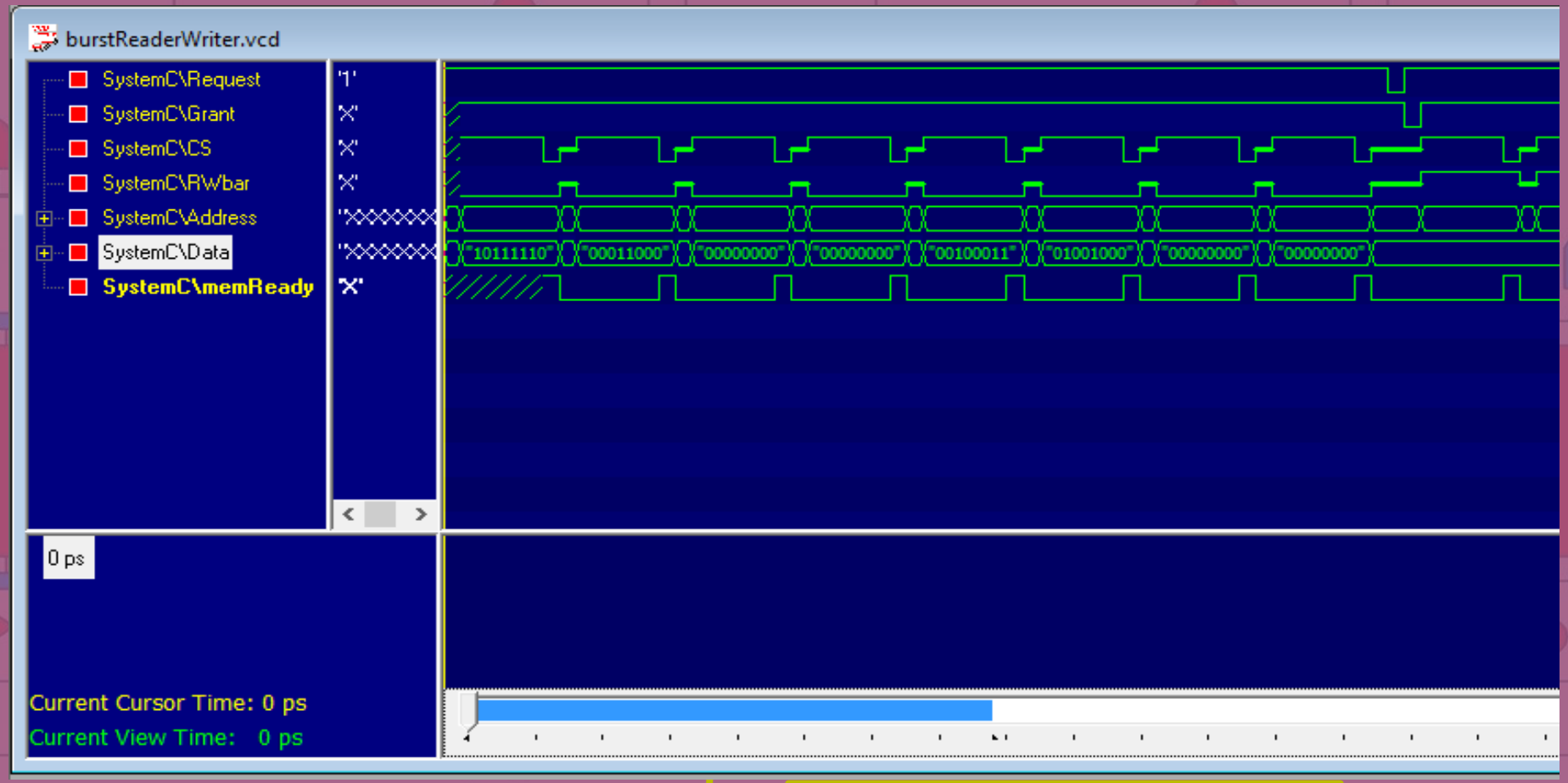
BurstReaderWriter_tb.h

Example :Burst Buffer, *Burst Reader Writer Main.cpp*

```cpp
#include "burstReaderWriter_TB.h"
int sc_main (int argc , char *argv[]) {
    burstReaderWriter_TB SPP1("burstReaderWriter");

    sc_trace_file *wf = sc_create_vcd_trace_file("burstReaderWriter");
    // Dump memory signals
    sc_trace(wf, SPP1.req0, "Request");
    sc_trace(wf, SPP1.gnt0, "Grant");
    sc_trace(wf, SPP1.memCs, "CS");
    sc_trace(wf, SPP1.memRwbar, "RWbar");
    sc_trace(wf, SPP1.memAddressBus, "Address");
    sc_trace(wf, SPP1.memDataBus, "Data");
    sc_trace(wf, SPP1.memReady, "memReady");

    sc_start();// 6200, SC_NS);
    sc_close_vcd_trace_file(wf);
    return 0;
}

```

BurstReaderWriter_Main.cpp

Example :Burst Buffer, Output



Trace results

Summary

- Handling handshaking in an abstract way
- Enclose communications in channels
- Connect modules through channels
- Develop your own channels or use existing
- Primitive channels
- Hierarchical channels