Chapter 6

Abstract Communication Channels

Zainalabedin Navabi

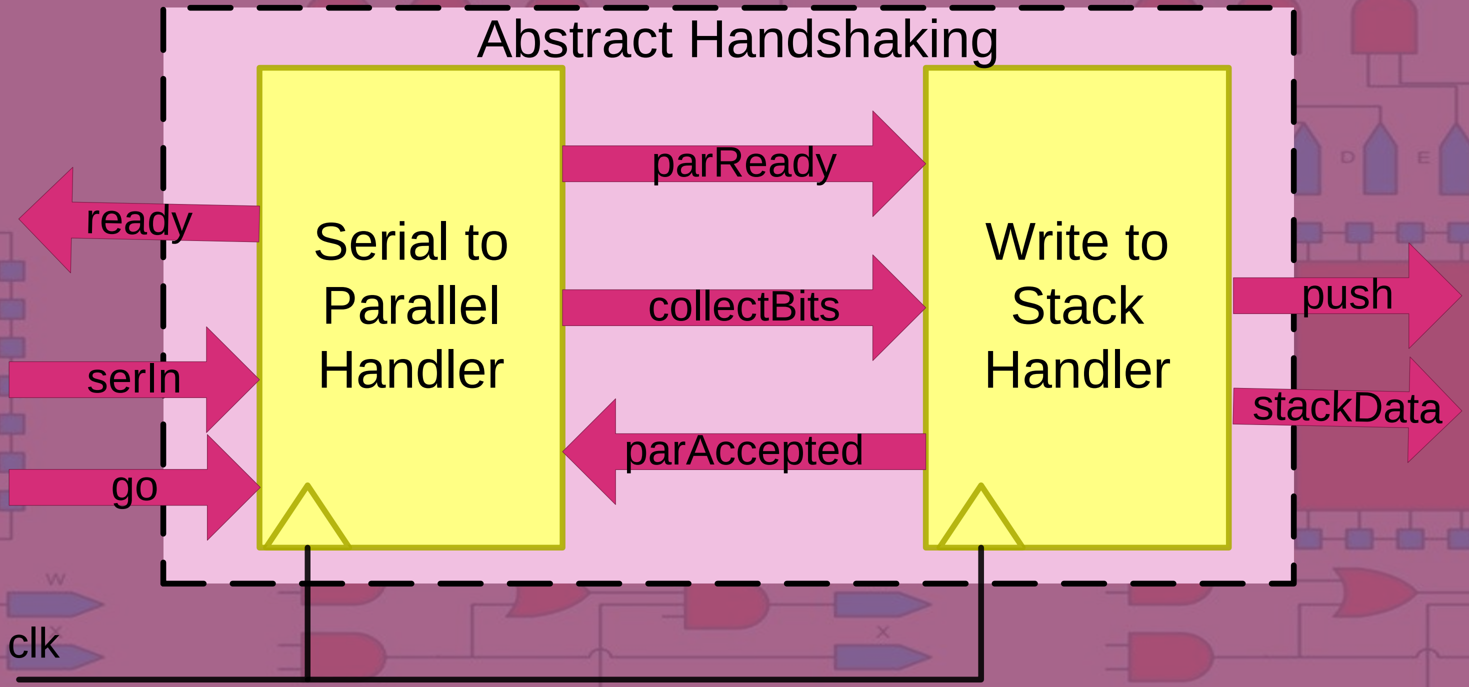
Abstract Channels

- 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

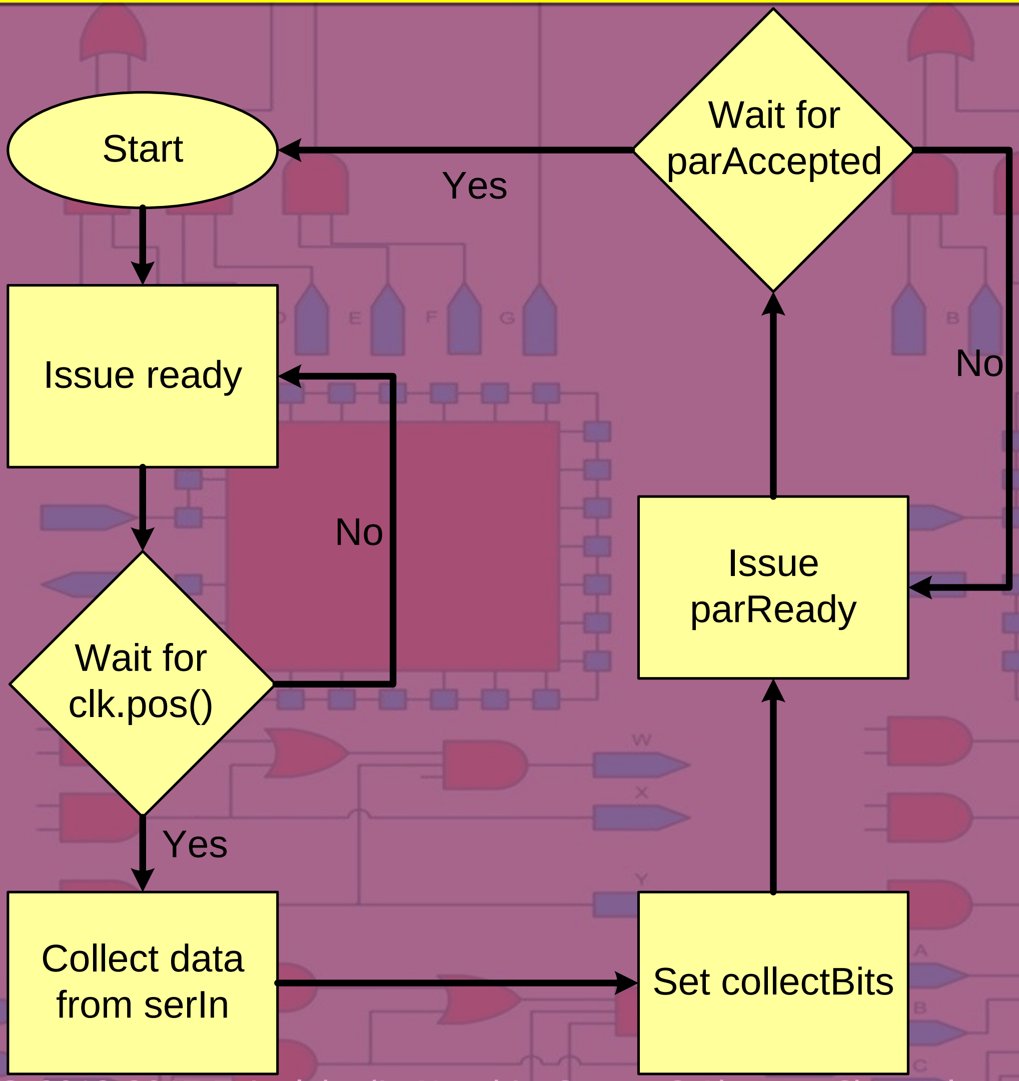
Abstract Channels

- 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

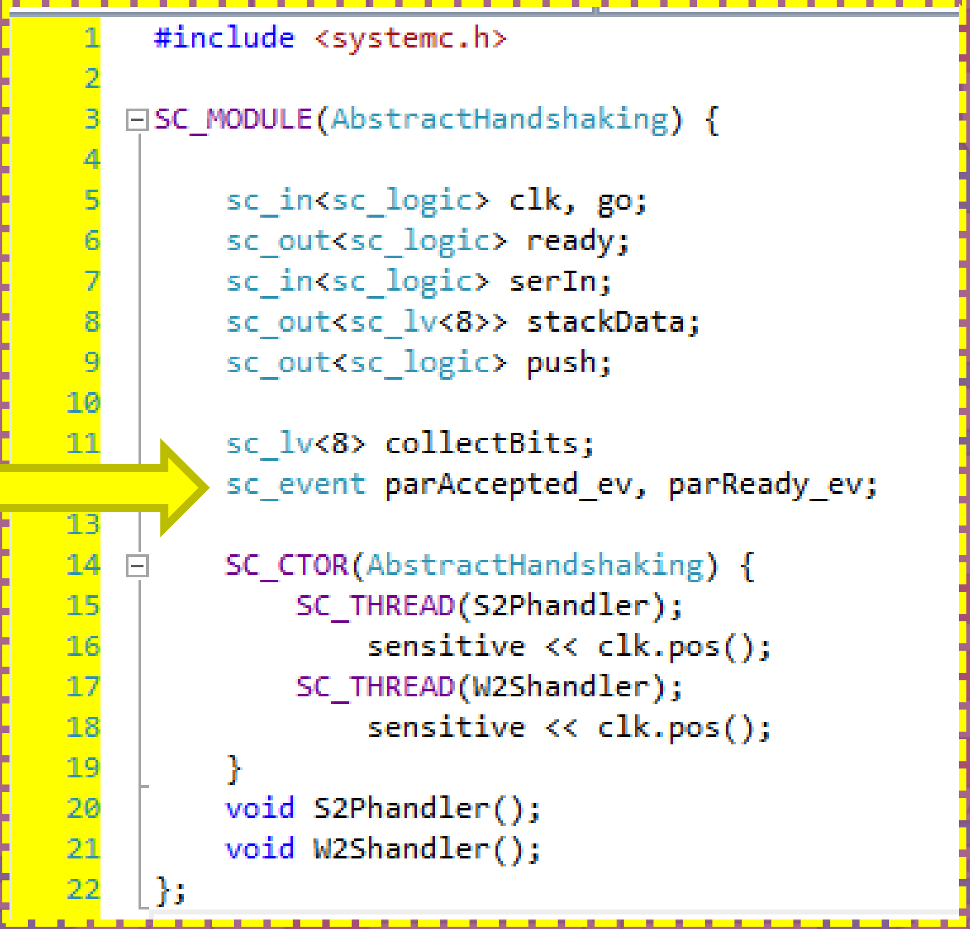


Abstract Handshaking



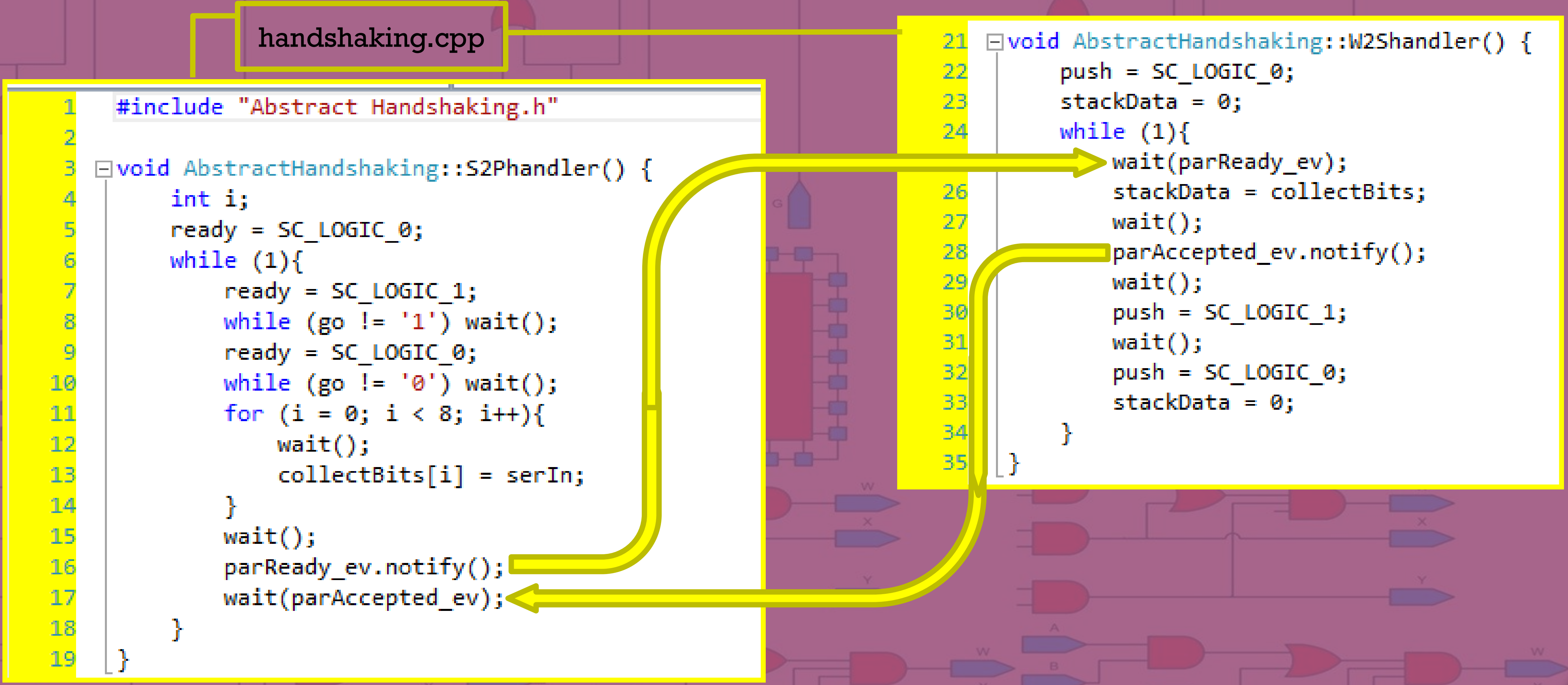
Abstract Handshaking

Abstract Handshaking .h



```
#include <systemc.h>

SC_MODULE(AbstractHandshaking) {

    sc_in<sc_logic> clk, go;
    sc_out<sc_logic> ready;
    sc_in<sc_logic> serIn;
    sc_out<sc_lv<8>> stackData;
    sc_out<sc_logic> push;

    sc_lv<8> collectBits;
    sc_event parAccepted_ev, parReady_ev;

    SC_CTOR(AbstractHandshaking) {
        SC_THREAD(S2Phandler);
            sensitive << clk.pos();
        SC_THREAD(W2Shandler);
            sensitive << clk.pos();
    }
    void S2Phandler();
    void W2Shandler();
};
```

Abstract Handshaking



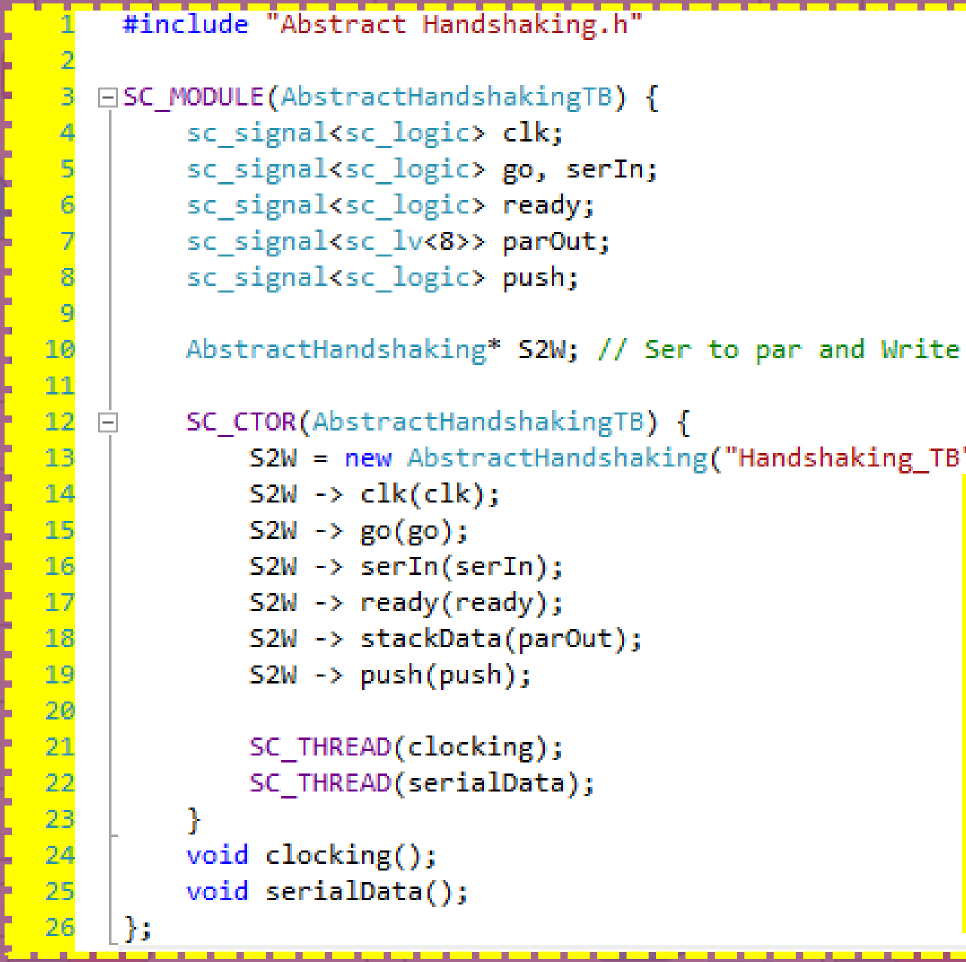
handshaking.cpp

```cpp
#include "Abstract Handshaking.h"

void AbstractHandshaking::S2Phandler() {
    int i;
    ready = SC_LOGIC_0;
    while (1){
        ready = SC_LOGIC_1;
        while (go != '1') wait();
        ready = SC_LOGIC_0;
        while (go != '0') wait();
        for (i = 0; i < 8; i++){
            wait();
            collectBits[i] = serIn;
        }
        wait();
        parReady_ev.notify();
        wait(parAccepted_ev);
    }
}
```

```cpp
void AbstractHandshaking::W2Shandler() {
    push = SC_LOGIC_0;
    stackData = 0;
    while (1){
        wait(parReady_ev);
        stackData = collectBits;
        wait();
        parAccepted_ev.notify();
        wait();
        push = SC_LOGIC_1;
        wait();
        push = SC_LOGIC_0;
        stackData = 0;
    }
}
```

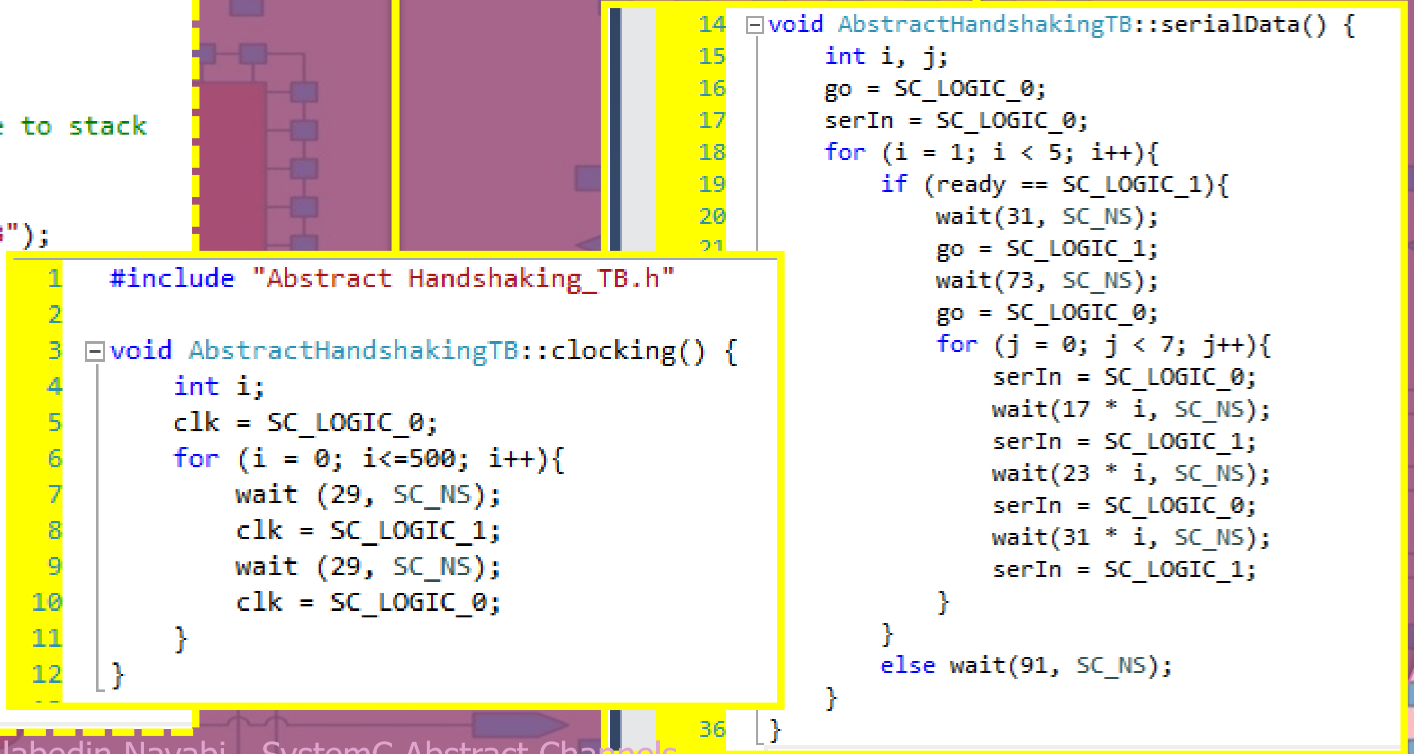
Abstract Handshaking

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

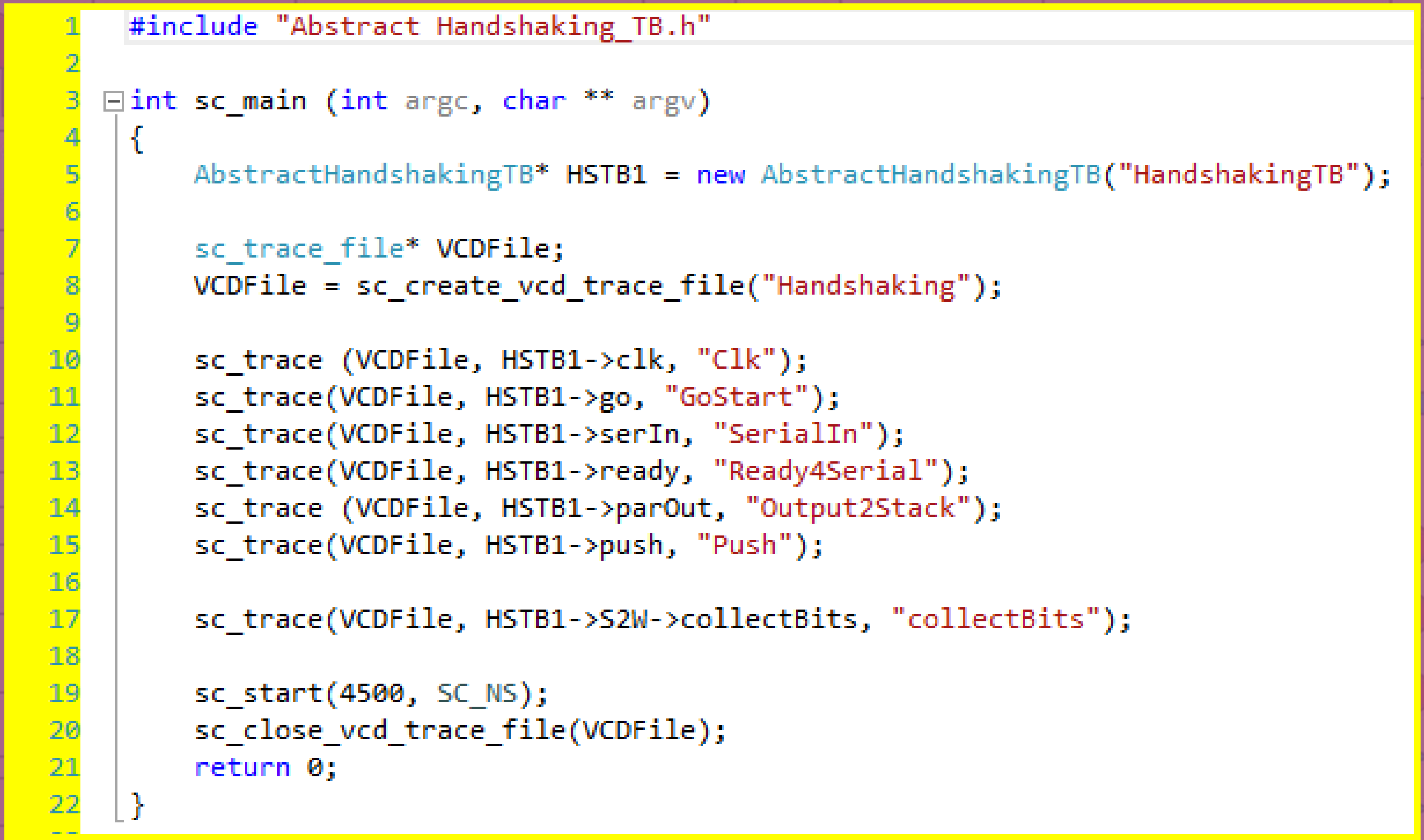


```
#include "Abstract Handshaking_TB.h"

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);
    }
}
```

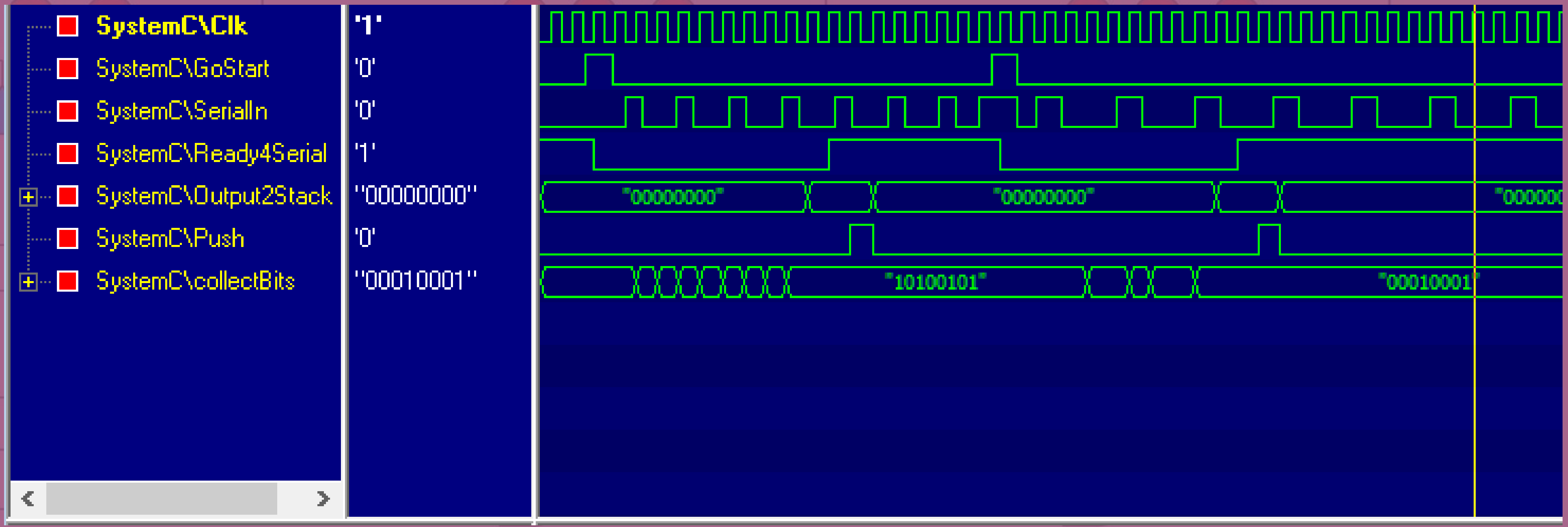
Abstract Handshaking

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



SystemC\Clk '1'
SystemC\GoStart '0'
SystemC\SerialIn '0'
SystemC\Ready4Serial '1'
SystemC\Output2Stack "00000000"
SystemC\Push '0'
SystemC\collectBits "00010001"

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

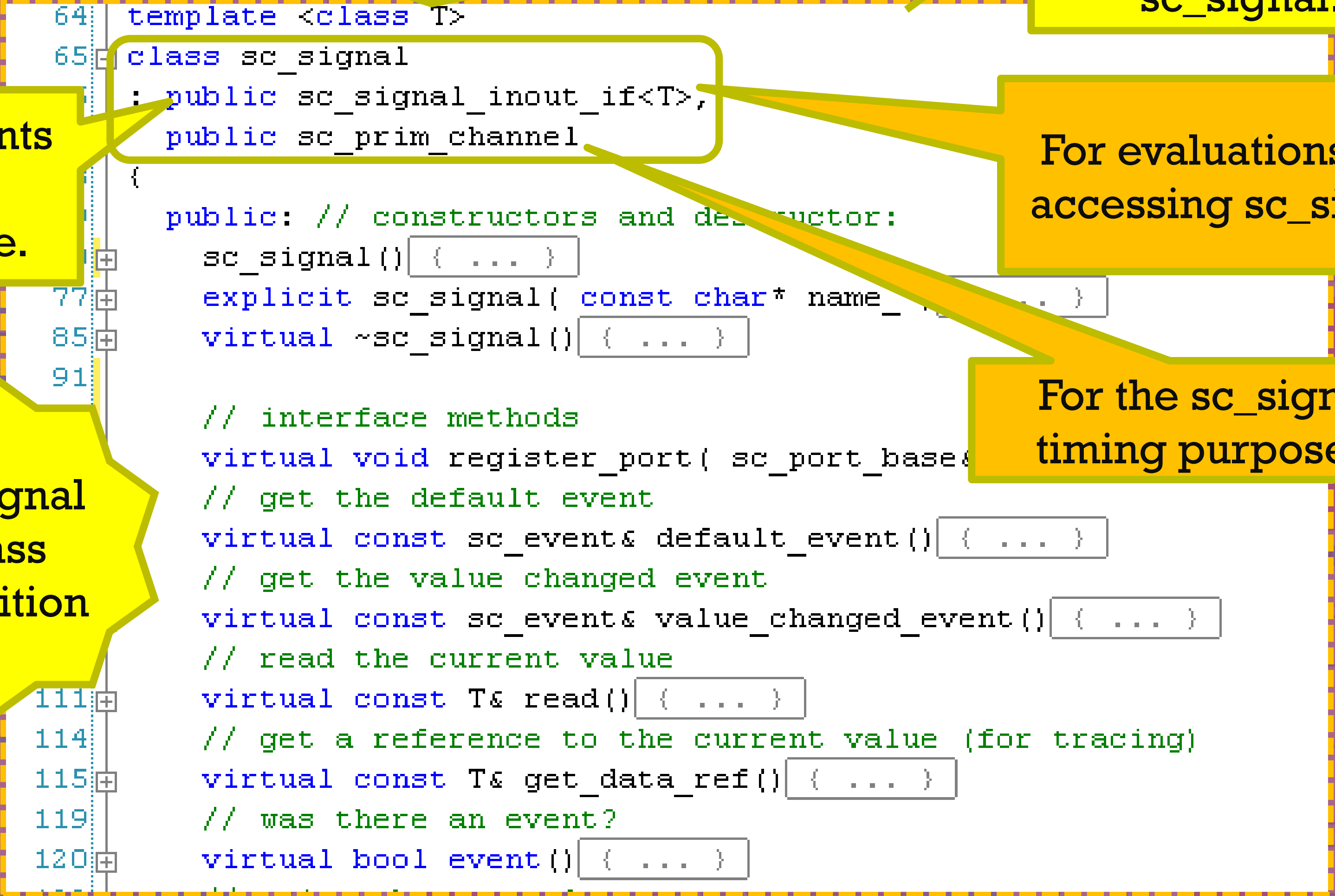
- A container class for communication and synchronization
- They implement one or more *interfaces*
- Different channels may implement the same interface in different ways
- A channel implements all the methods of the inherited interface classes

- 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



```
template <class T>
class sc_signal
: public sc_signal_inout_if<T>,
  public sc_prim_channel
{
  public: // constructors and de      uctor:
    sc_signal() { ... }
    explicit sc_signal( const char* name_       ... }
    virtual ~sc_signal() { ... }

    // 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
    virtual const T& read() { ... }
    // get a reference to the current value (for tracing)
    virtual const T& get_data_ref() { ... }
    // was there an event?
    virtual bool event() { ... }
```

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

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

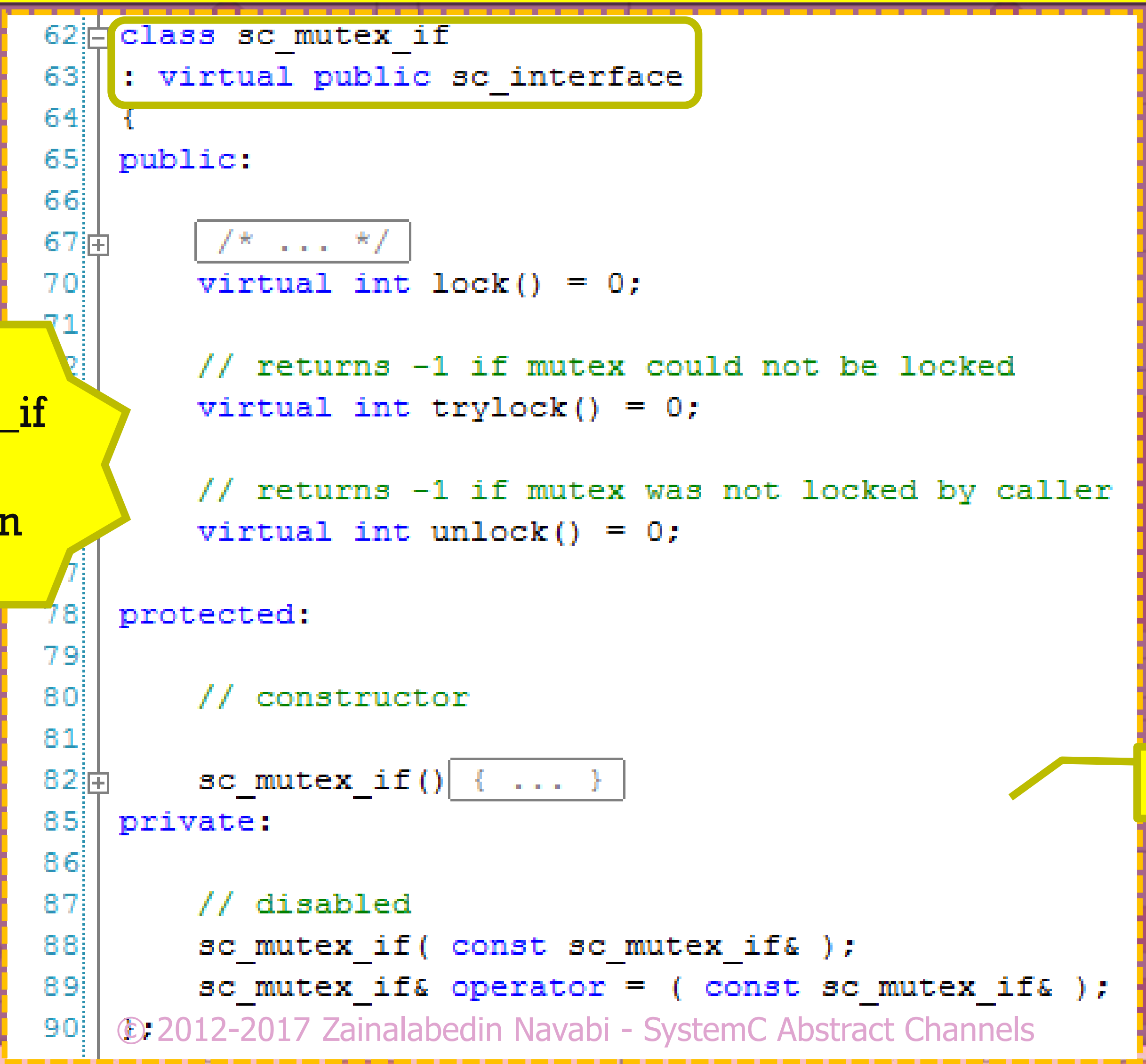
Basics of Channels

- 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



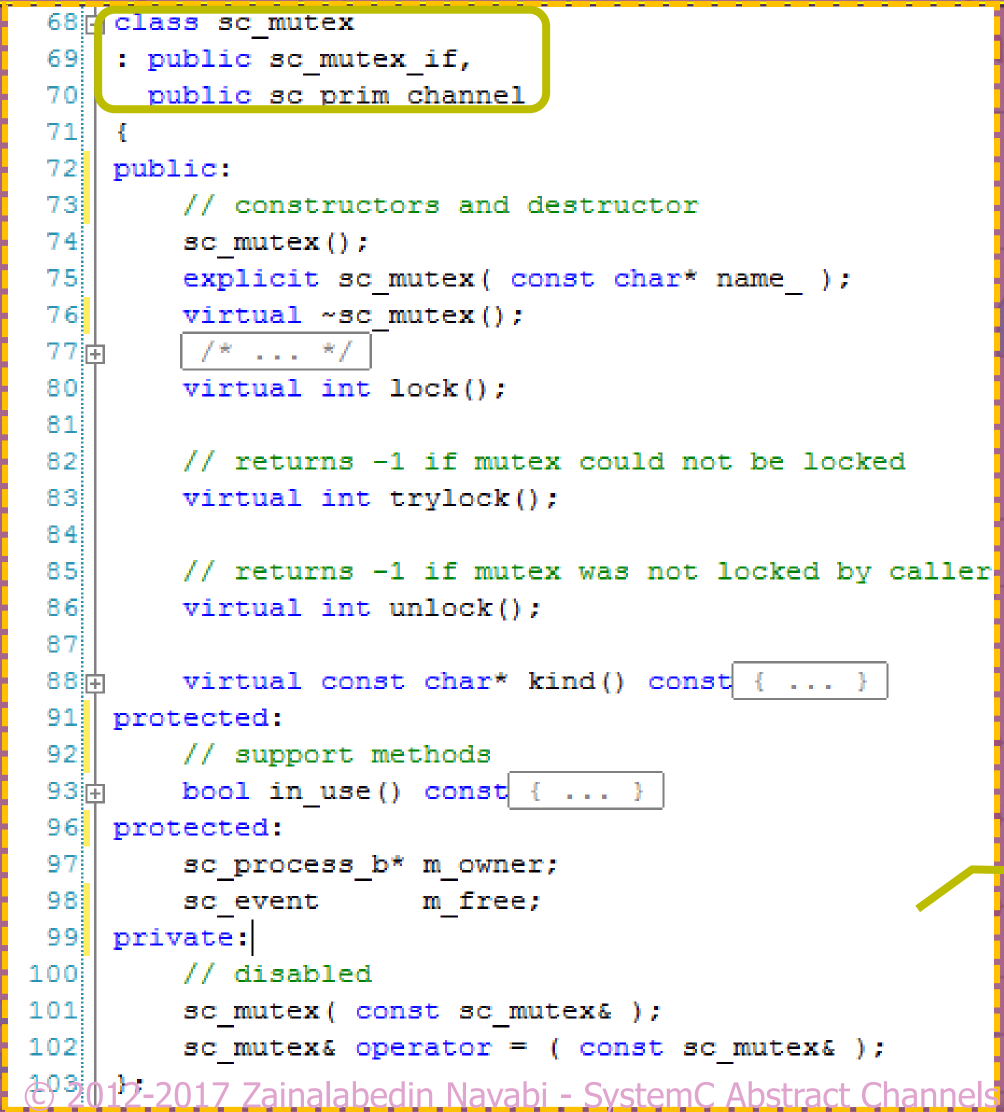
```
class sc_mutex_if
: virtual public sc_interface
{
public:

    /* ... */
    virtual int lock() = 0;

    // 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;

protected:

    // constructor

    sc_mutex_if() { ... }
private:

    // disabled
    sc_mutex_if( const sc_mutex_if& );
    sc_mutex_if& operator = ( const sc_mutex_if& );
};
```

sc_mutex_if Class definition

sc_mutex_if.h

Basics of Channels

sc_mutex Class definition



```
class sc_mutex
: public sc_mutex_if,
  public sc_prim_channel
{
public:
    // constructors and destructor
    sc_mutex();
    explicit sc_mutex( const char* name_ );
    virtual ~sc_mutex();
    /* ... */
    virtual int lock();

    // returns -1 if mutex could not be locked
    virtual int trylock();

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

    virtual const char* kind() const { ... }
protected:
    // support methods
    bool in_use() const { ... }
protected:
    sc_process_b* m_owner;
    sc_event      m_free;
private:
    // disabled
    sc_mutex( const sc_mutex& );
    sc_mutex& operator = ( const sc_mutex& );
};
```

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

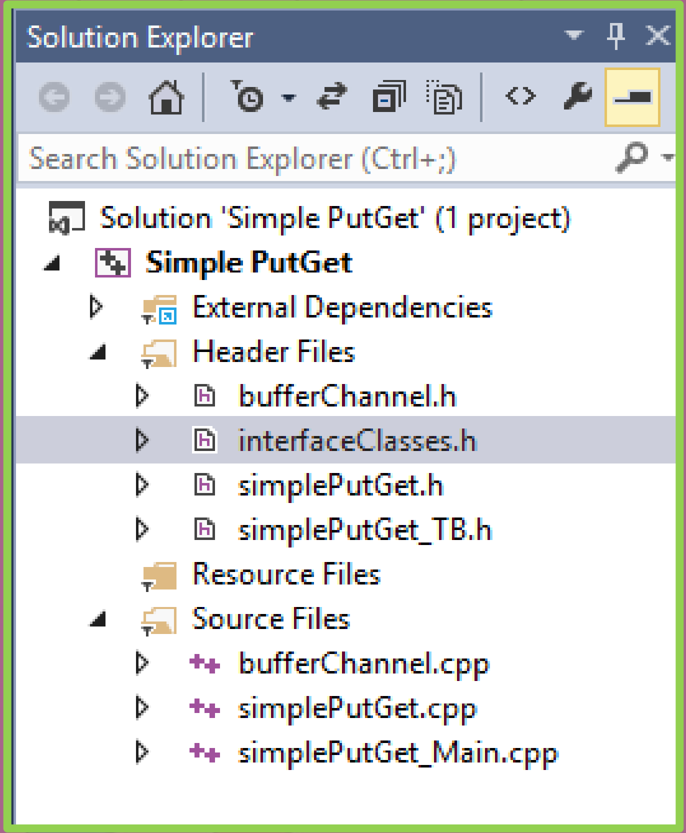
Implements a non-blocking process

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**

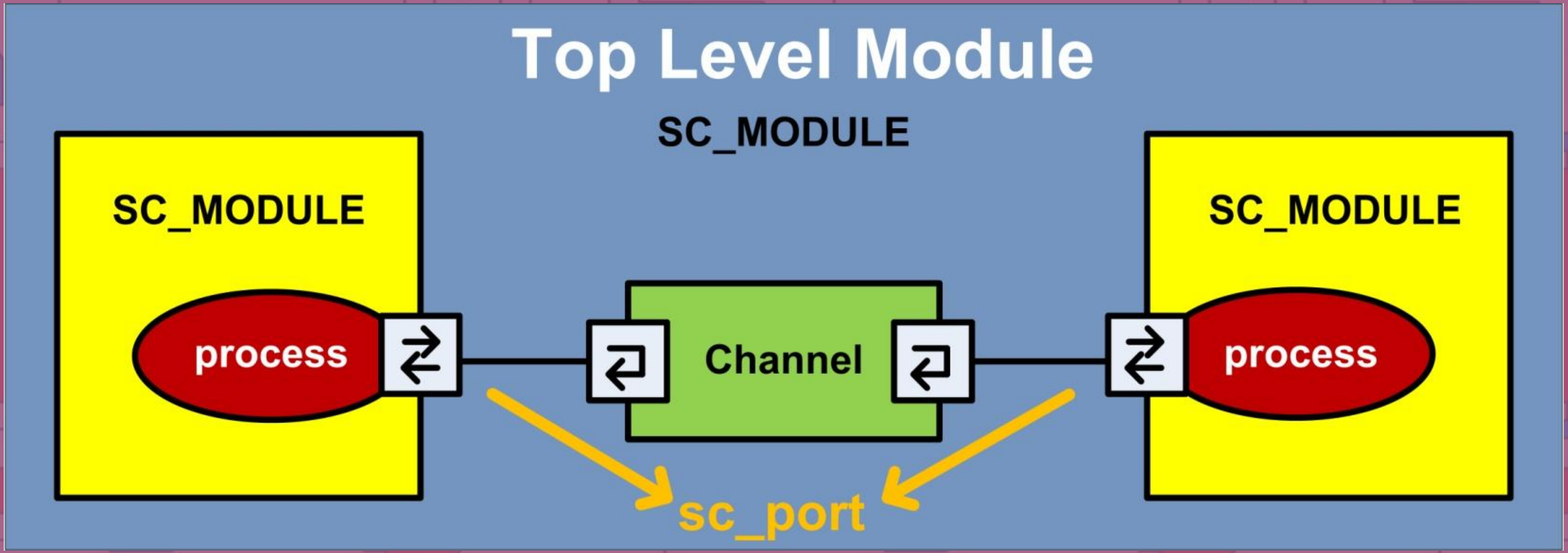
Abstract Channels

- 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



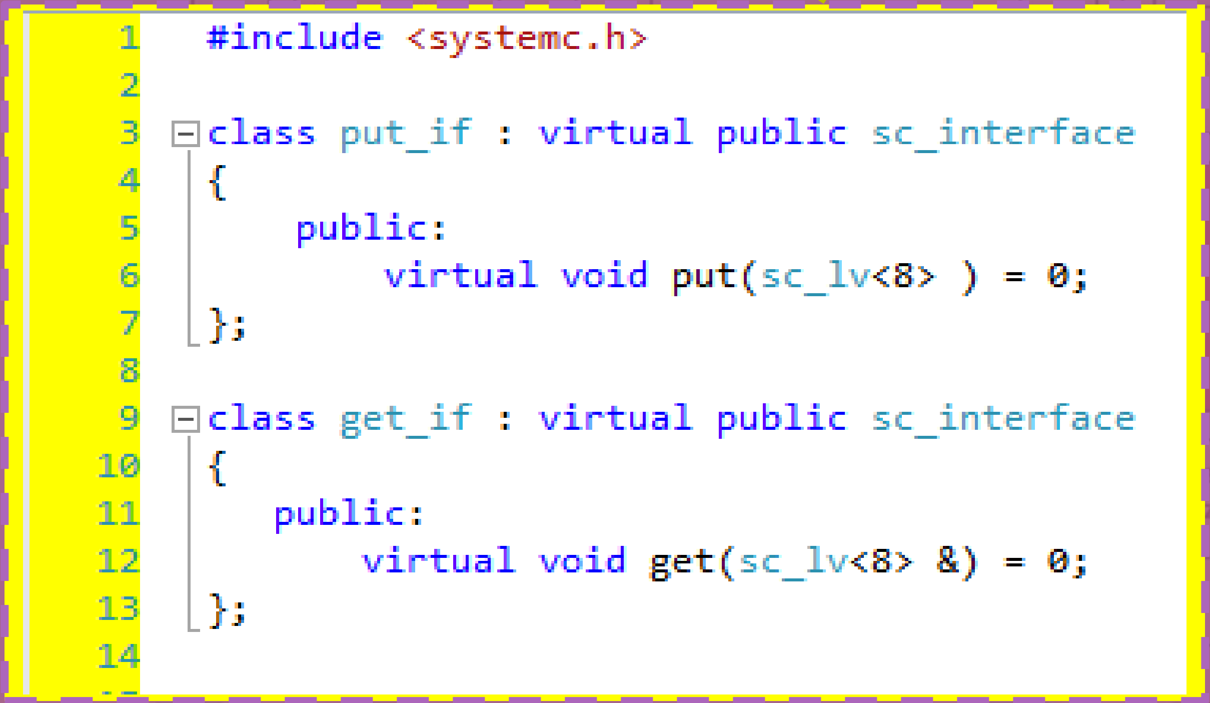
Solution 'Simple PutGet' (1 project)
- Simple PutGet
  - External Dependencies
  - Header Files
    - bufferChannel.h
    - interfaceClasses.h
    - simplePutGet.h
    - simplePutGet_TB.h
  - Resource Files
  - Source Files
    - bufferChannel.cpp
    - simplePutGet.cpp
    - simplePutGet_Main.cpp

Example 1: Buffer Channel



Example 1: Buffer Channel, *interfaceClasses.h*

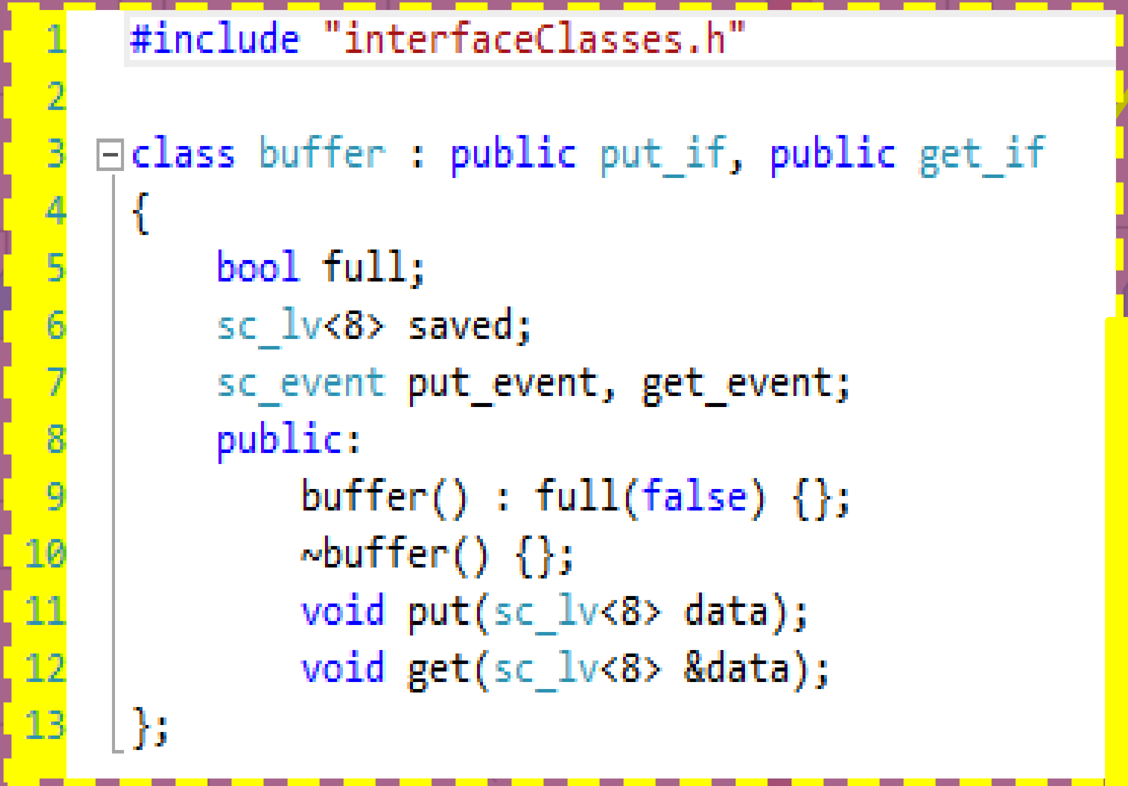
interfaceClasses.h



```
#include <systemc.h>

class put_if : virtual public sc_interface
{
    public:
        virtual void put(sc_lv<8> ) = 0;
};

class get_if : virtual public sc_interface
{
   public:
       virtual void get(sc_lv<8> &) = 0;
};

```

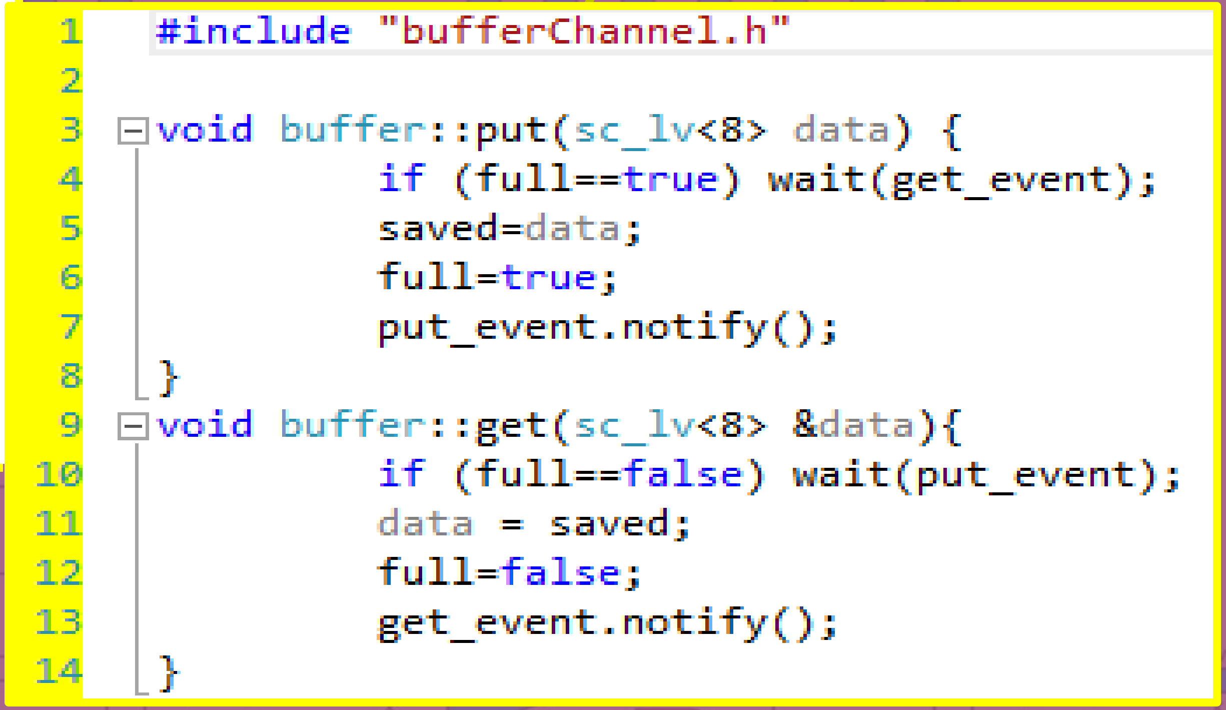
Example 1: Buffer Channel, *bufferChannel*

bufferChannel.h



```cpp
#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);
};
```

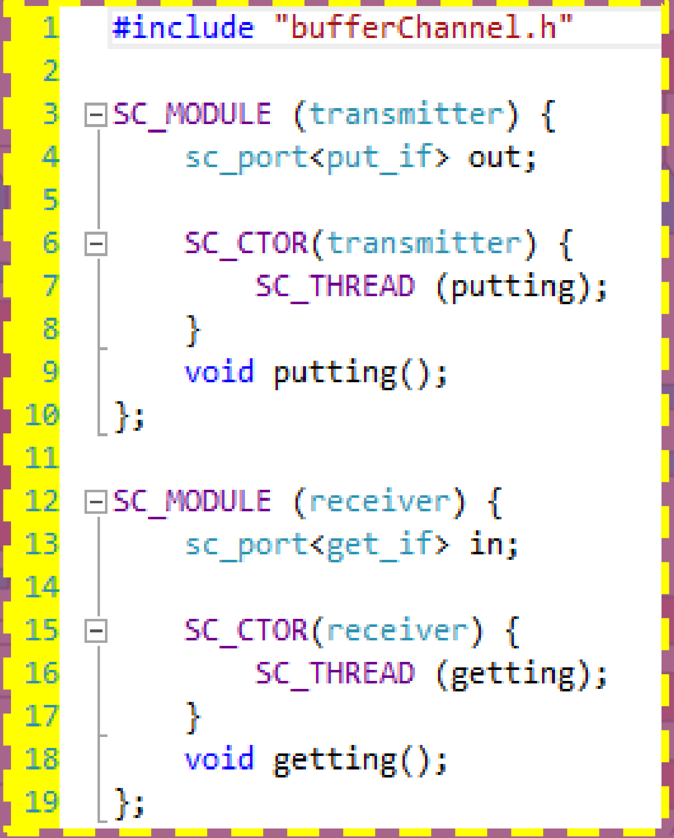
bufferChannel.cpp



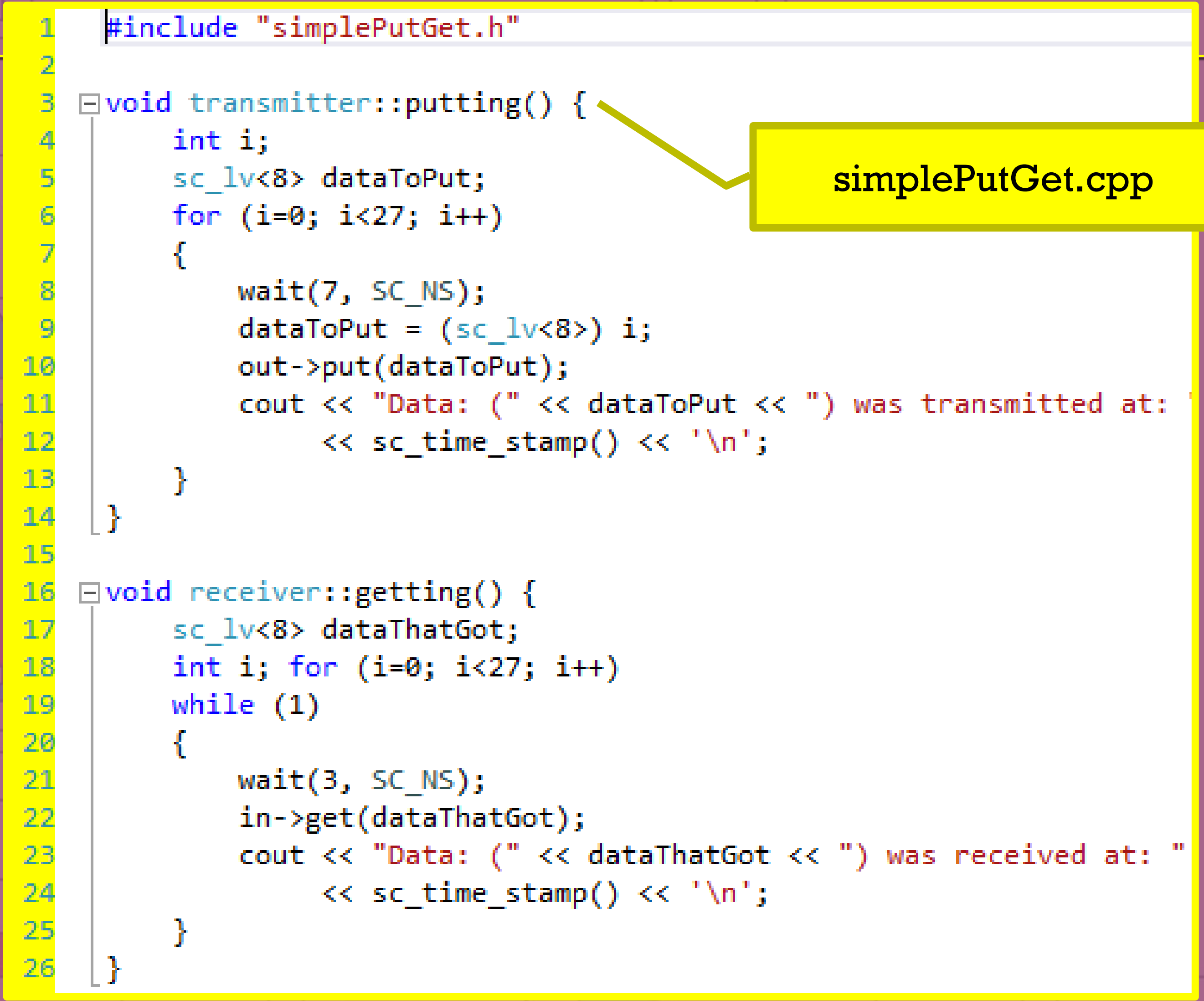
```cpp
#include "bufferChannel.h"

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



```
#include "bufferChannel.h"

SC_MODULE (transmitter) {
    sc_port<put_if> out;

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

SC_MODULE (receiver) {
    sc_port<get_if> in;

    SC_CTOR(receiver) {
        SC_THREAD (getting);
    }
    void getting();
};
```

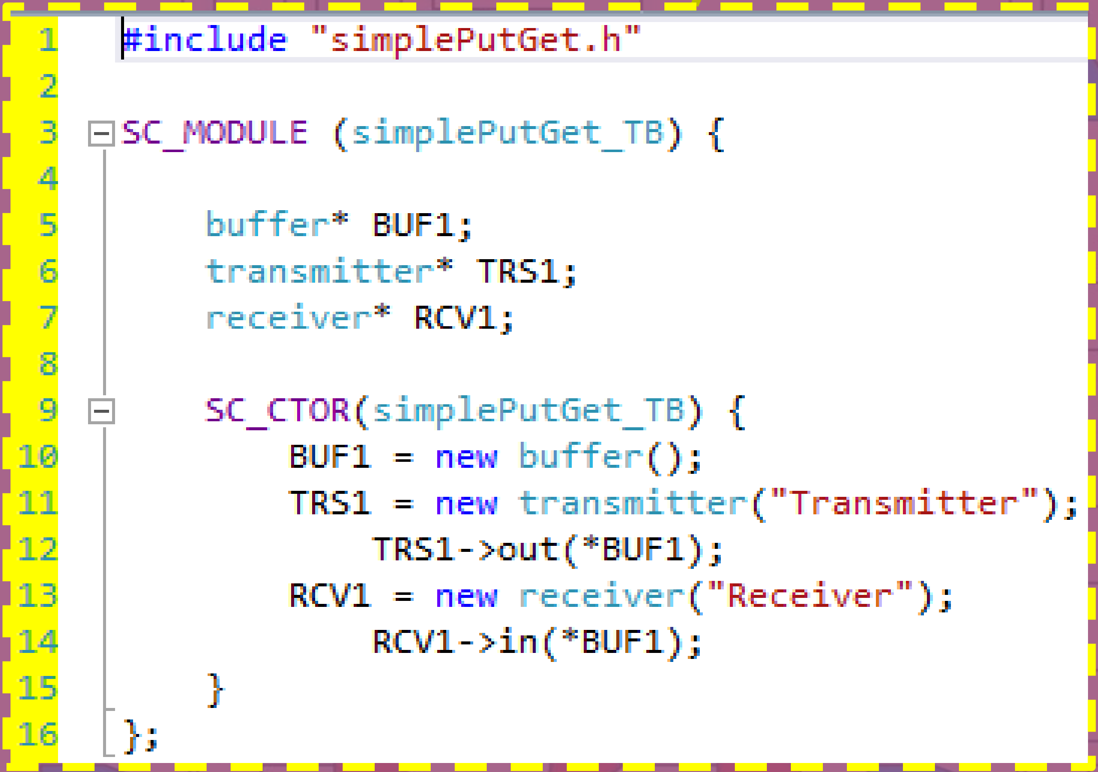


simplePutGet.cpp

```
#include "simplePutGet.h"

void transmitter::putting() {
    int i;
    sc_lv<8> dataToPut;
    for (i=0; i<27; i++)
    {
        wait(7, SC_NS);
        dataToPut = (sc_lv<8>) i;
        out->put(dataToPut);
        cout << "Data: (" << dataToPut << ") was transmitted at: "
             << sc_time_stamp() << '\n';
    }
}

void receiver::getting() {
    sc_lv<8> dataThatGot;
    int i; for (i=0; i<27; i++)
    while (1)
    {
        wait(3, SC_NS);
        in->get(dataThatGot);
        cout << "Data: (" << dataThatGot << ") was received at: "
             << sc_time_stamp() << '\n';
    }
}
```

Example 1: Buffer Channel, *simplePutGet_TB*

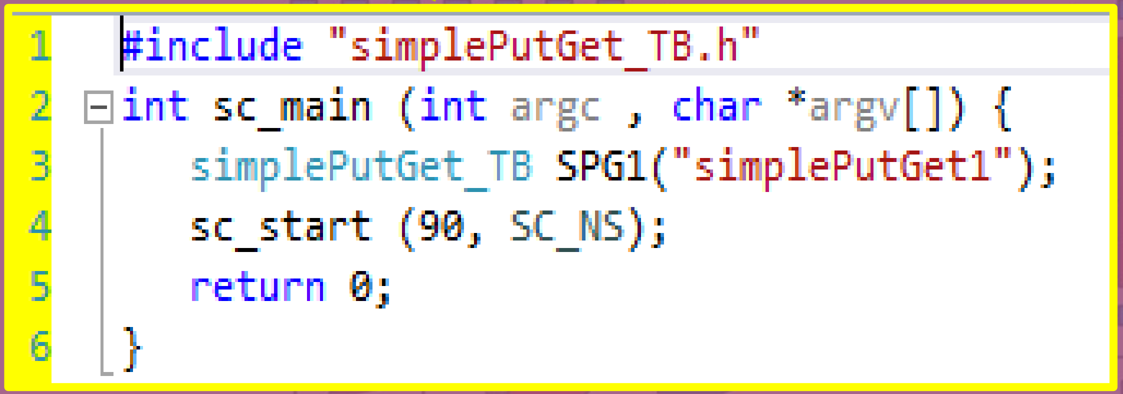
simplePutGet_TB.h



```
#include "simplePutGet.h"

SC_MODULE (simplePutGet_TB) {

    buffer* BUF1;
    transmitter* TRS1;
    receiver* RCV1;

    SC_CTOR(simplePutGet_TB) {
        BUF1 = new buffer();
        TRS1 = new transmitter("Transmitter");
            TRS1->out(*BUF1);
        RCV1 = new receiver("Receiver");
            RCV1->in(*BUF1);
    }
};
```

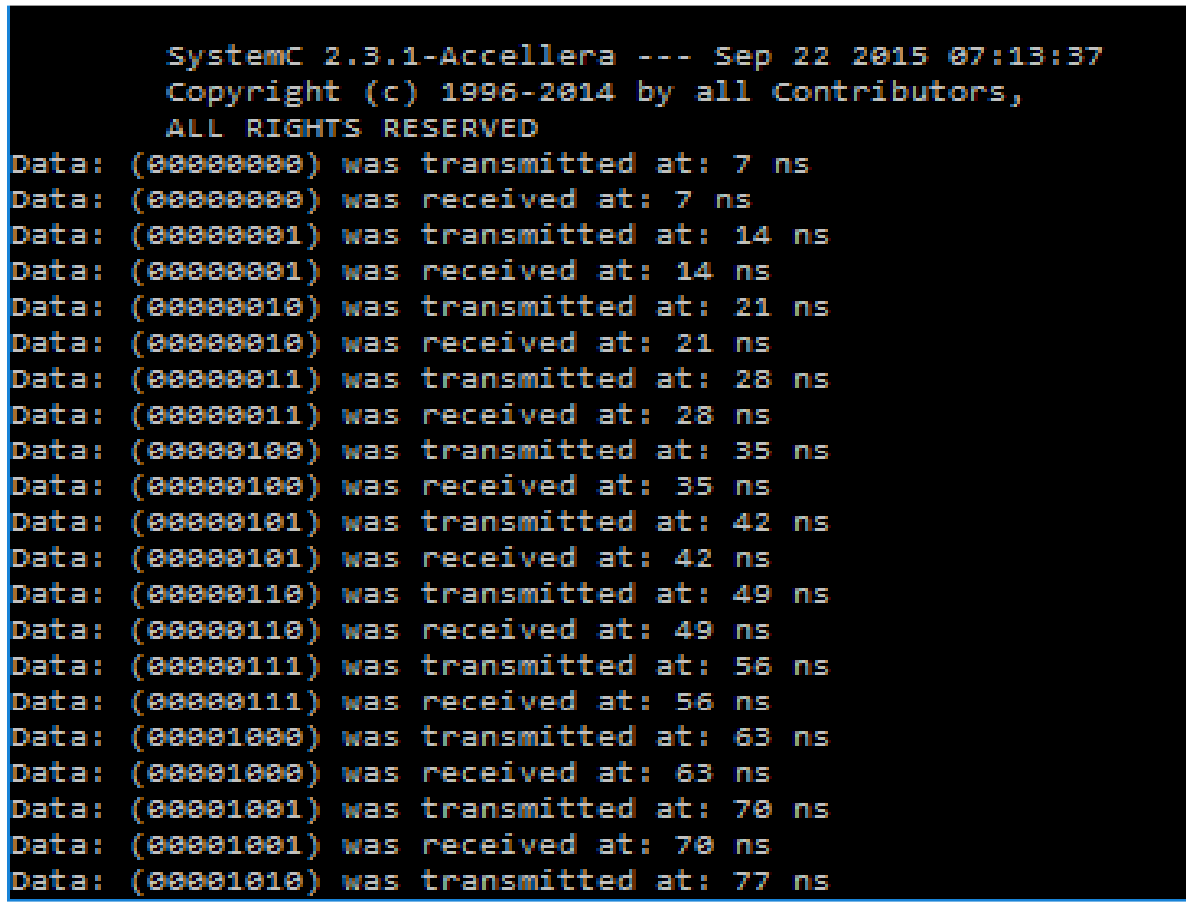
Example 1: Buffer Channel, *bufferChannel_main.cpp*

bufferChannel_main.cpp



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

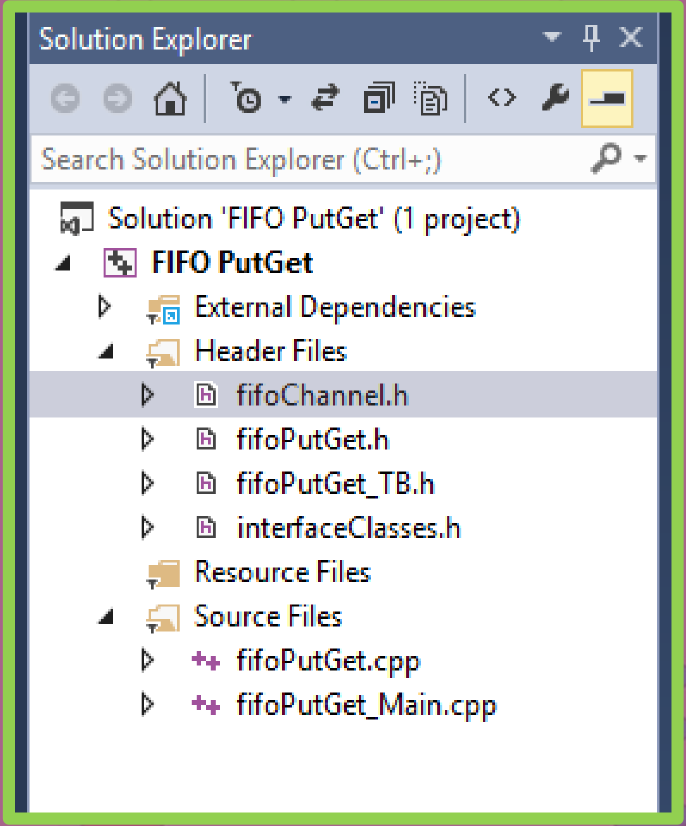
Example 1: Buffer Channel, output



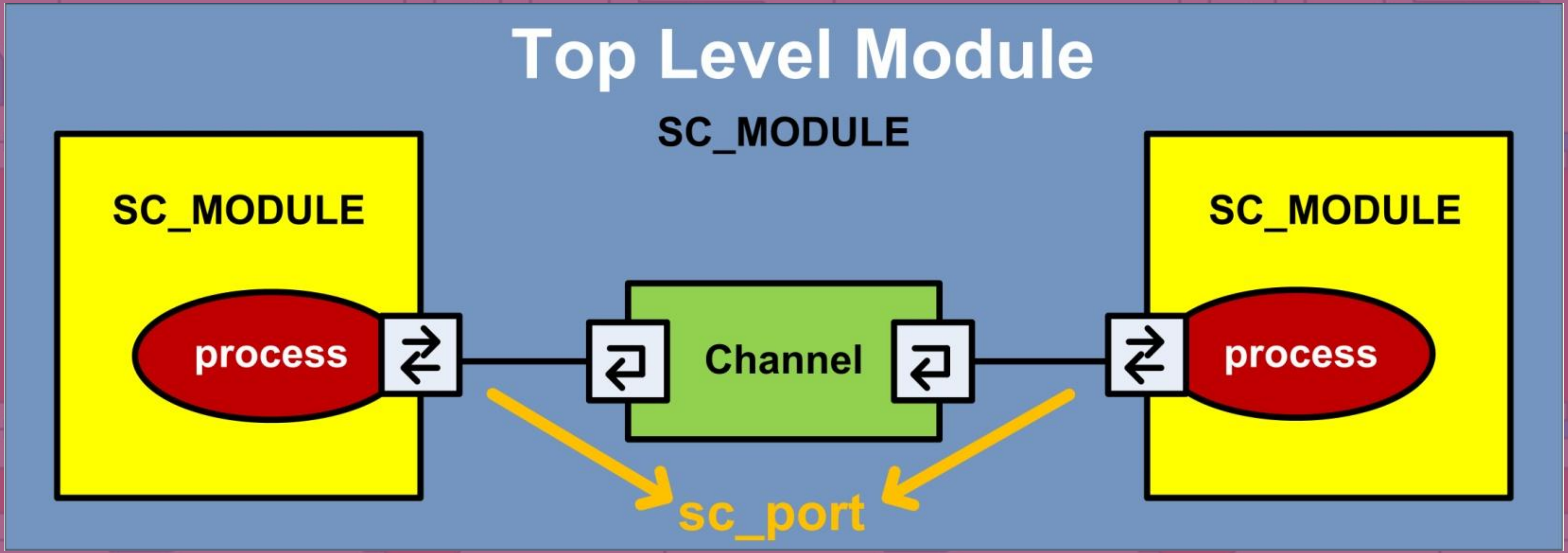
```
        SystemC 2.3.1-Accellera --- Sep 22 2015 07:13:37
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED
Data: (00000000) was transmitted at: 7 ns
Data: (00000000) was received at: 7 ns
Data: (00000001) was transmitted at: 14 ns
Data: (00000001) was received at: 14 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

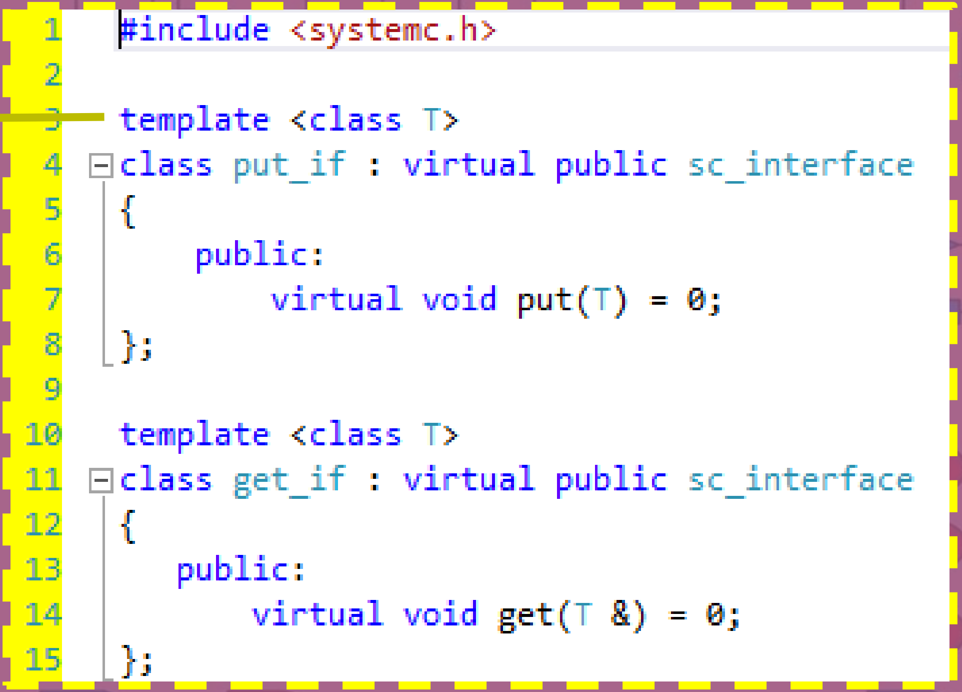
- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using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



Example 2: FiFo Channel



Example 2: FiFo Channel, *interfaceClasses*

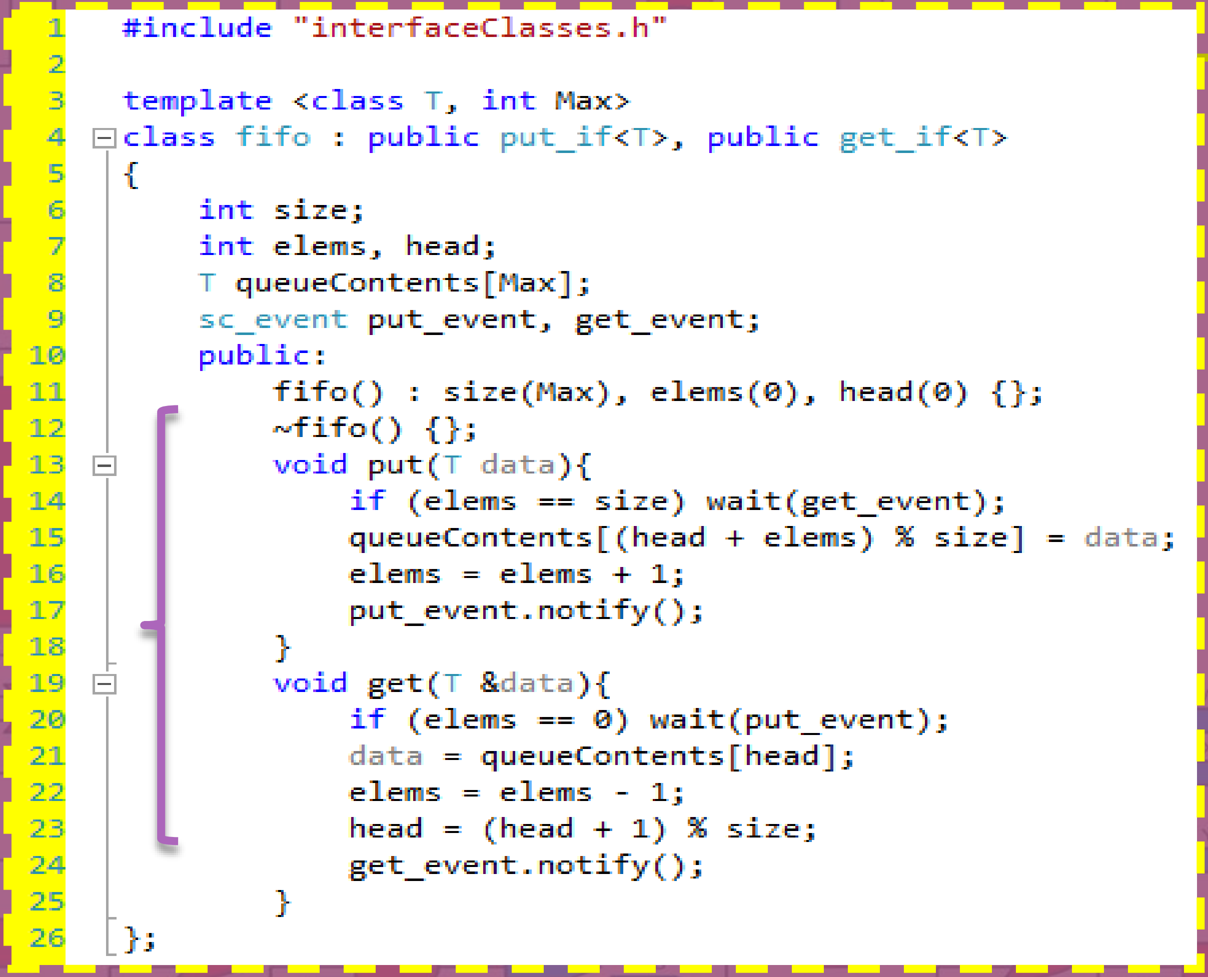


template form

interfaceClasses.h

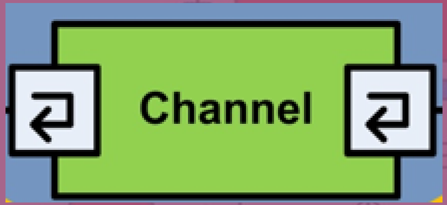
```cpp
#include <systemc.h>

template <class T>
class put_if : virtual public sc_interface
{
    public:
        virtual void put(T) = 0;
};

template <class T>
class get_if : virtual public sc_interface
{
   public:
       virtual void get(T &) = 0;
};
```

Example 2: FiFo Channel, *fifoChannel*

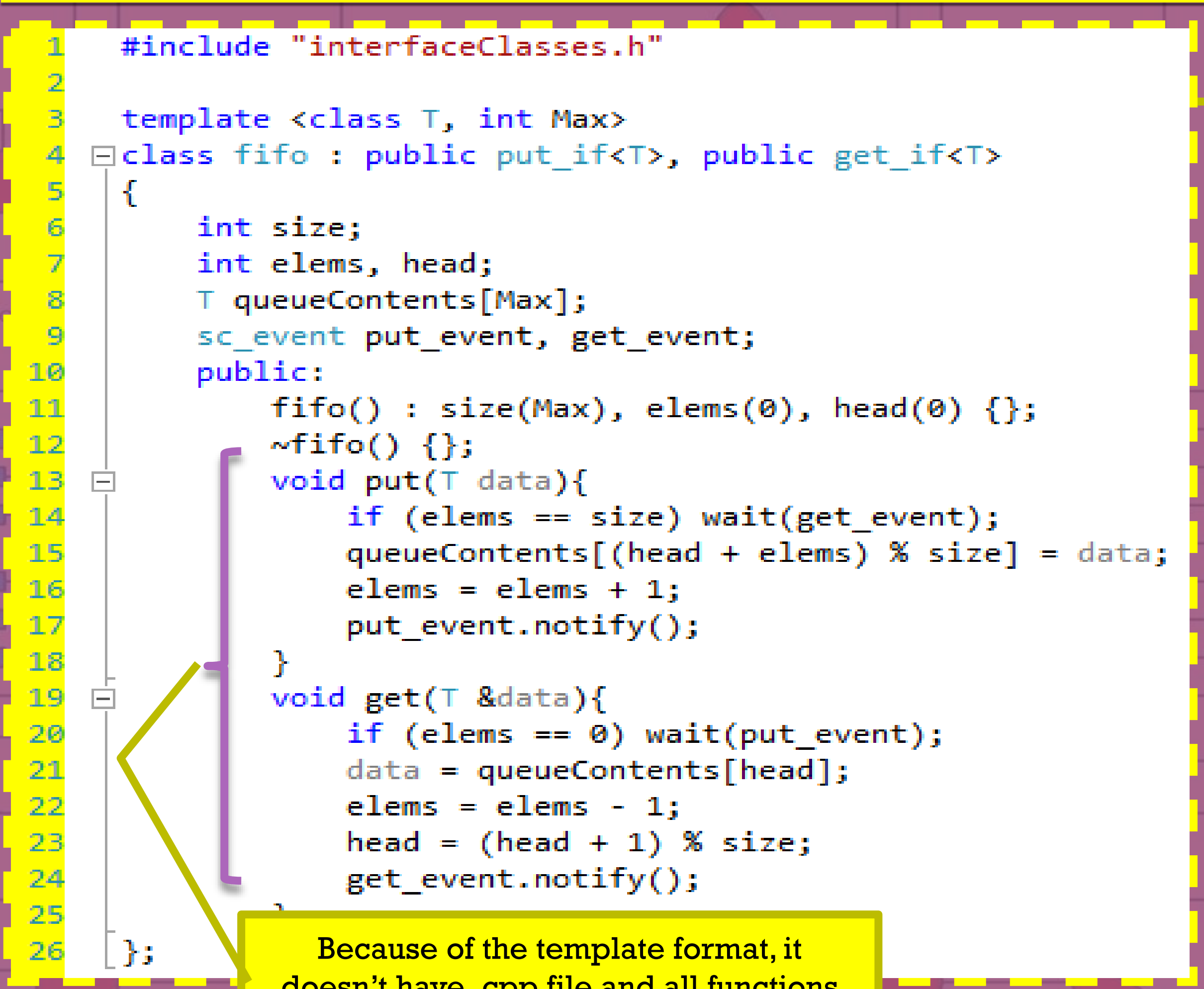


```
#include "interfaceClasses.h"

template <class T, int Max>
class fifo : public put_if<T>, public get_if<T>
{
    int size;
    int elems, head;
    T queueContents[Max];
    sc_event put_event, get_event;
    public:
        fifo() : size(Max), elems(0), head(0) {};
        ~fifo() {};
        void put(T data){
            if (elems == size) wait(get_event);
            queueContents[(head + elems) % size] = data;
            elems = elems + 1;
            put_event.notify();
        }
        void get(T &data){
            if (elems == 0) wait(put_event);
            data = queueContents[head];
            elems = elems - 1;
            head = (head + 1) % size;
            get_event.notify();
        }
};
```

fifoChannel.h



Example 2: FiFo Channel, *fifoChannel*



```
#include "interfaceClasses.h"

template <class T, int Max>
class fifo : public put_if<T>, public get_if<T>
{
    int size;
    int elems, head;
    T queueContents[Max];
    sc_event put_event, get_event;
    public:
        fifo() : size(Max), elems(0), head(0) {};
        ~fifo() {};
        void put(T data){
            if (elems == size) wait(get_event);
            queueContents[(head + elems) % size] = data;
            elems = elems + 1;
            put_event.notify();
        }
        void get(T &data){
            if (elems == 0) wait(put_event);
            data = queueContents[head];
            elems = elems - 1;
            head = (head + 1) % size;
            get_event.notify();
        }
};
```

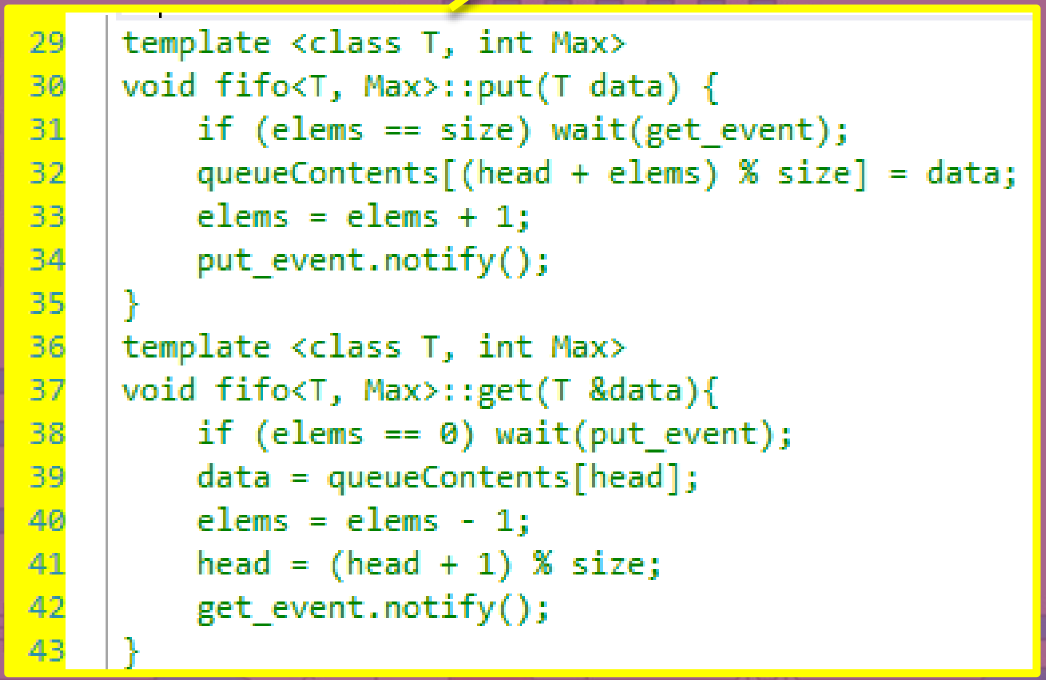
Because of the template format, it doesn't have .cpp file and all functions are implemented in .h file

Form 1

fifoChannel.h

Form 2

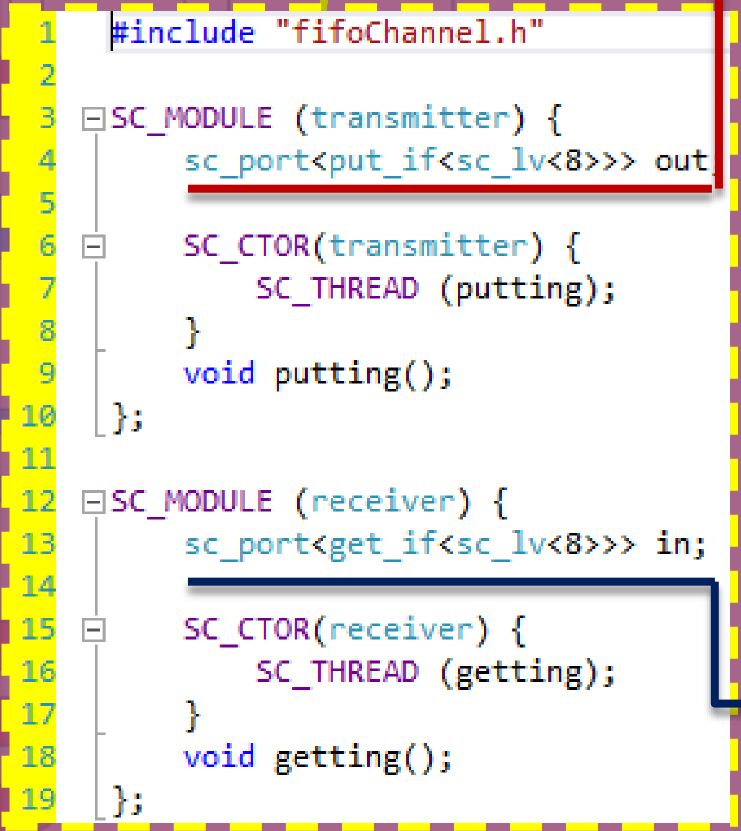
fifoChannel.cpp



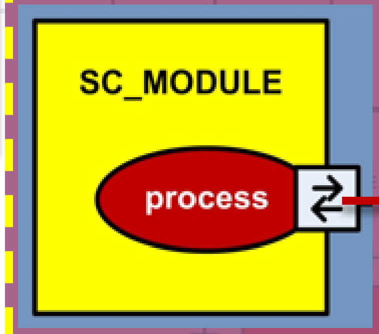
```
template <class T, int Max>
void fifo<T, Max>::put(T data) {
    if (elems == size) wait(get_event);
    queueContents[(head + elems) % size] = data;
    elems = elems + 1;
    put_event.notify();
}
template <class T, int Max>
void fifo<T, Max>::get(T &data){
    if (elems == 0) wait(put_event);
    data = queueContents[head];
    elems = elems - 1;
    head = (head + 1) % size;
    get_event.notify();
}
```

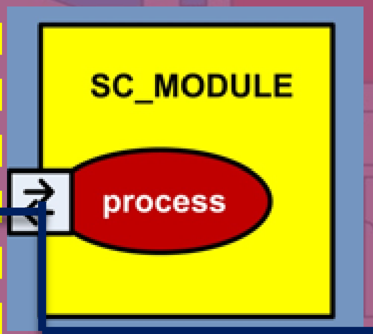
Example 2: FiFo Channel, *fifoPutGet*

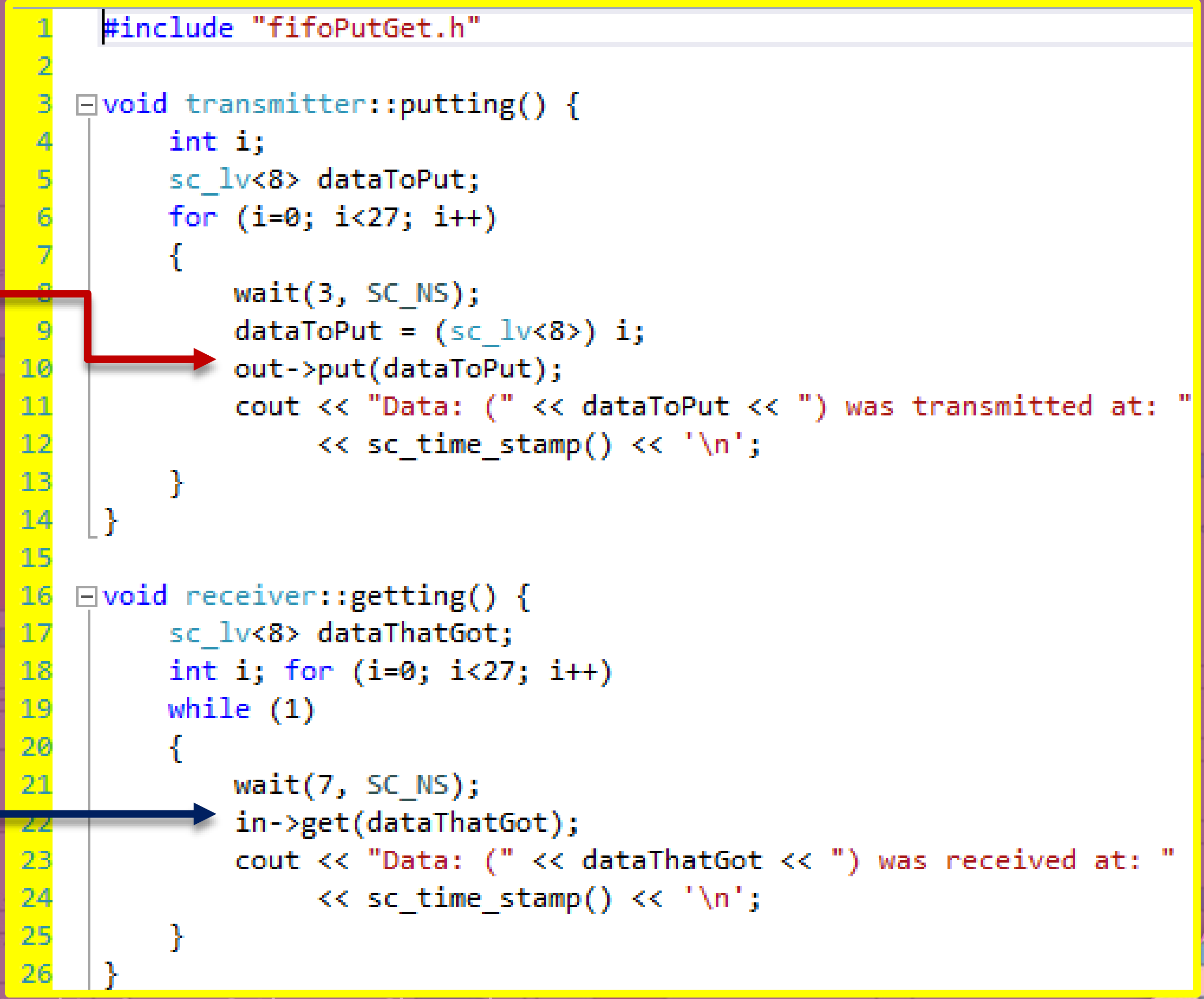
fifoPutGet.h



```
#include "fifoChannel.h"

SC_MODULE (transmitter) {
    sc_port<put_if<sc_lv<8>>> out

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

SC_MODULE (receiver) {
    sc_port<get_if<sc_lv<8>>> in;

    SC_CTOR(receiver) {
        SC_THREAD (getting);
    }
    void getting();
};
```



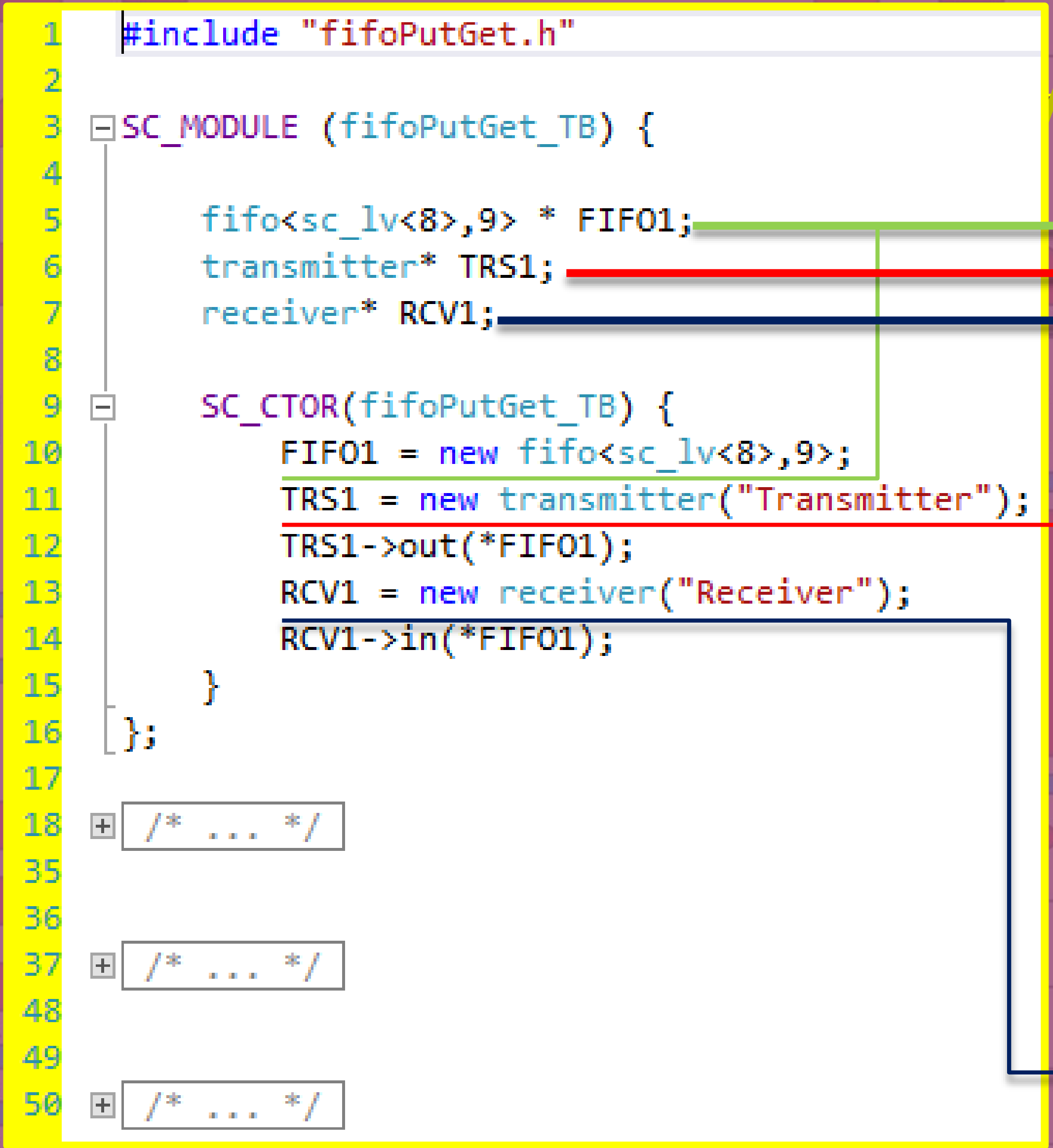




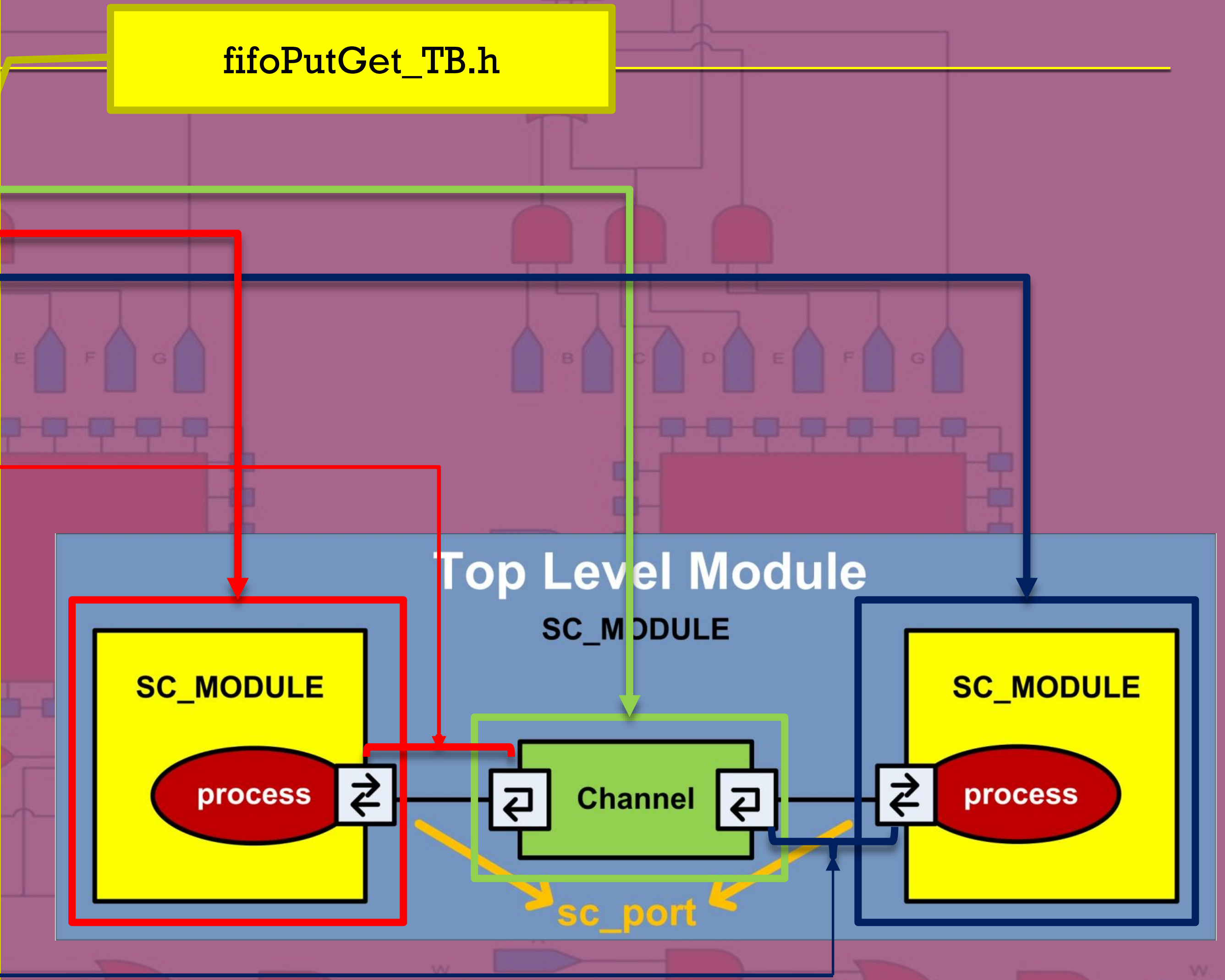
```
#include "fifoPutGet.h"

void transmitter::putting() {
    int i;
    sc_lv<8> dataToPut;
    for (i=0; i<27; i++)
    {
        wait(3, SC_NS);
        dataToPut = (sc_lv<8>) i;
        out->put(dataToPut);
        cout << "Data: (" << dataToPut << ") was transmitted at: "
            << sc_time_stamp() << '\n';
    }
}

void receiver::getting() {
    sc_lv<8> dataThatGot;
    int i; for (i=0; i<27; i++)
    while (1)
    {
        wait(7, SC_NS);
        in->get(dataThatGot);
        cout << "Data: (" << dataThatGot << ") was received at: "
            << sc_time_stamp() << '\n';
    }
}
```

fifoPutGet.cpp

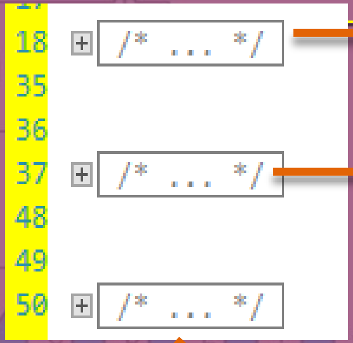
Example 2: FiFo Channel, *fifoPutGet_TB.h*



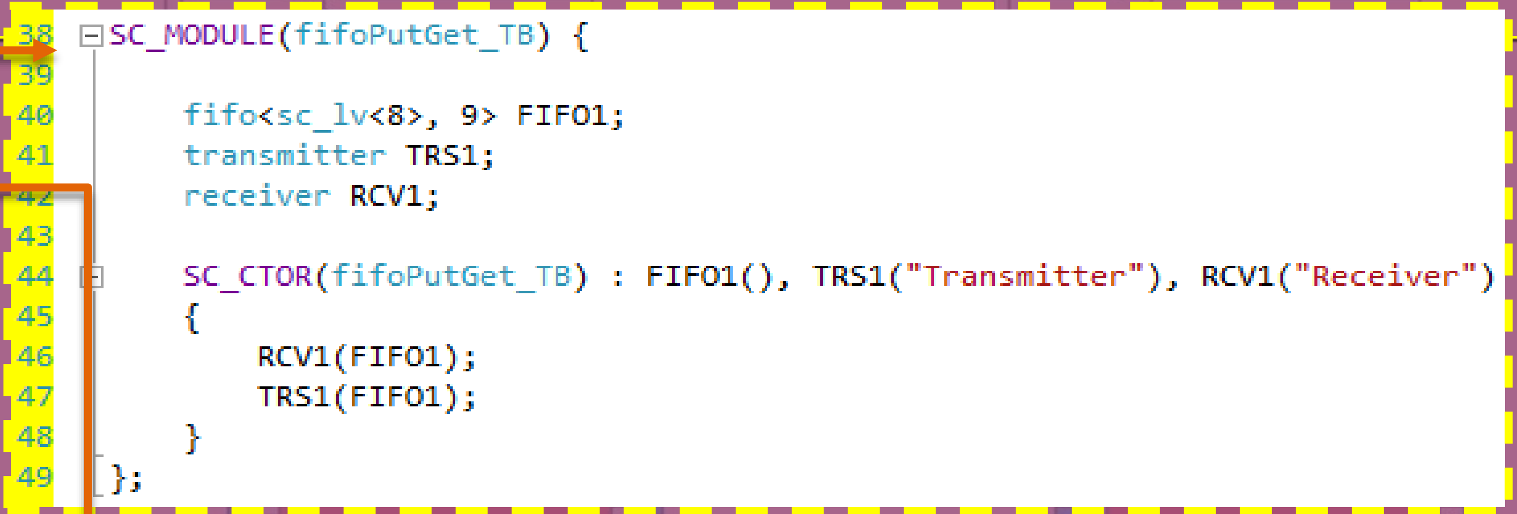
```
#include "fifoPutGet.h"

SC_MODULE (fifoPutGet_TB) {

    fifo<sc_lv<8>,9> * FIFO1;
    transmitter* TRS1;
    receiver* RCV1;

    SC_CTOR(fifoPutGet_TB) {
        FIFO1 = new fifo<sc_lv<8>,9>;
        TRS1 = new transmitter("Transmitter");
        TRS1->out(*FIFO1);
        RCV1 = new receiver("Receiver");
        RCV1->in(*FIFO1);
    }
};

/* ... */


/* ... */


/* ... */
```



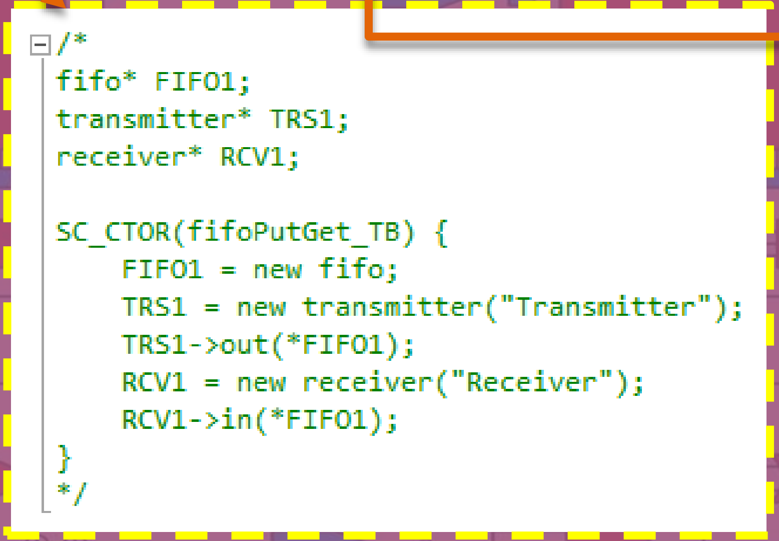
Example 2: FiFo Channel, *fifoPutGet_TB.h*



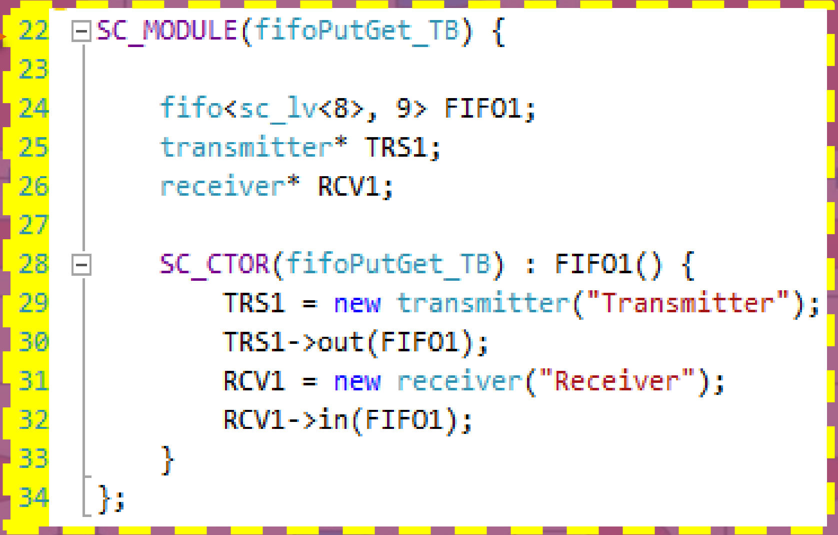
```
/* ... */


/* ... */


/* ... */
```



```
SC_MODULE(fifoPutGet_TB) {

    fifo<sc_lv<8>, 9> FIFO1;
    transmitter TRS1;
    receiver RCV1;

    SC_CTOR(fifoPutGet_TB) : FIFO1(), TRS1("Transmitter"), RCV1("Receiver")
    {
        RCV1(FIFO1);
        TRS1(FIFO1);
    }
};
```



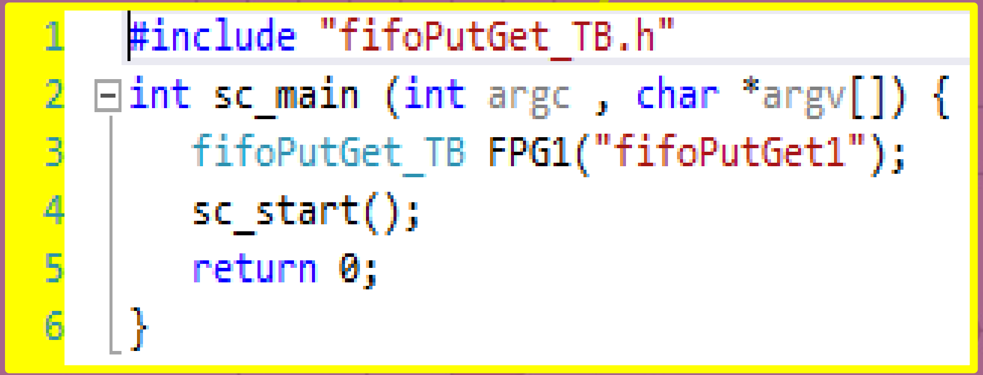
```
/*
fifo* FIFO1;
transmitter* TRS1;
receiver* RCV1;

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



```
SC_MODULE(fifoPutGet_TB) {

    fifo<sc_lv<8>, 9> FIFO1;
    transmitter* TRS1;
    receiver* RCV1;

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

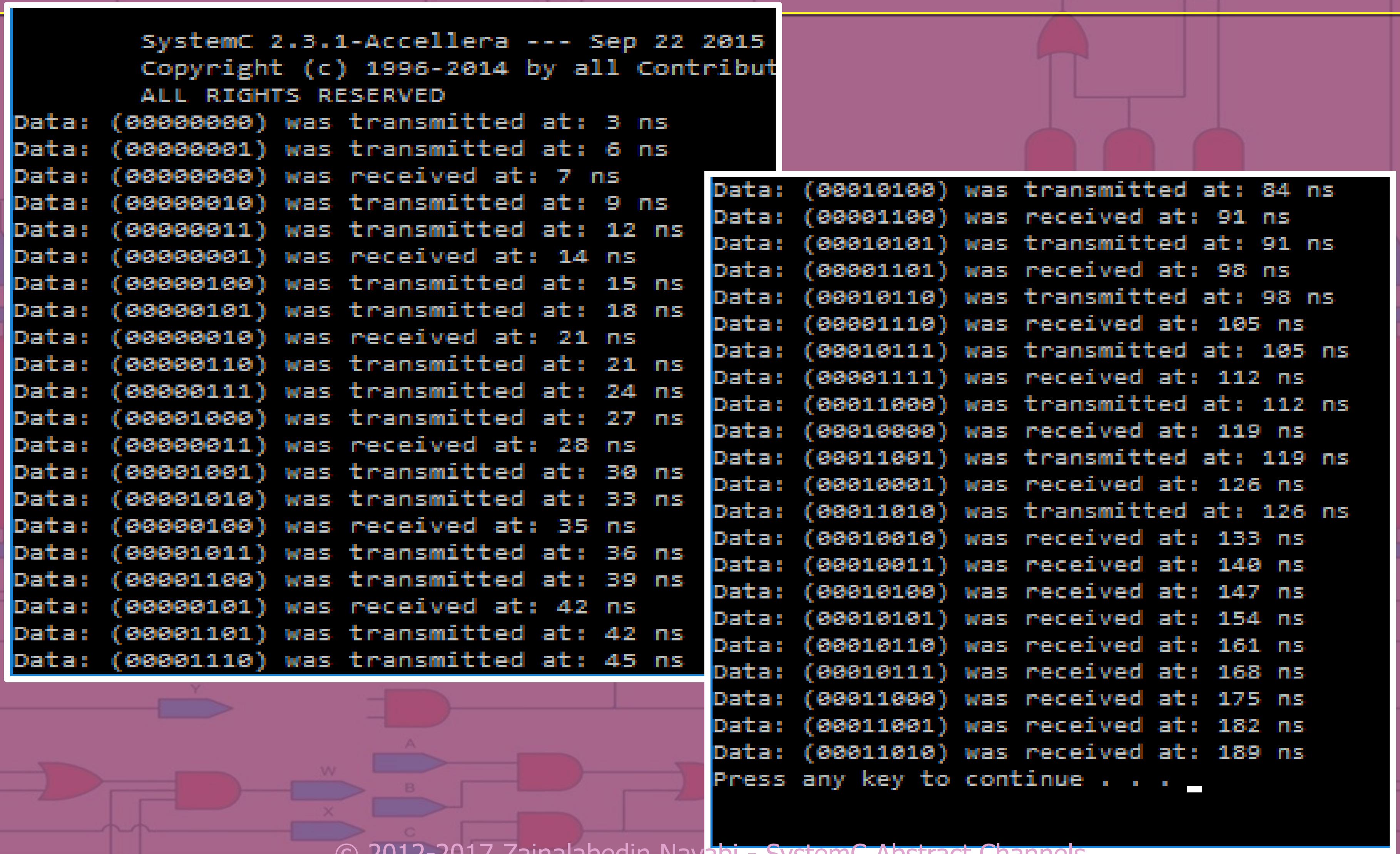
Example 2: FiFo Channel, *fifoPutGet_main.cpp*

fifoChannel_main.cpp



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

Example 2: FiFo Channel, output

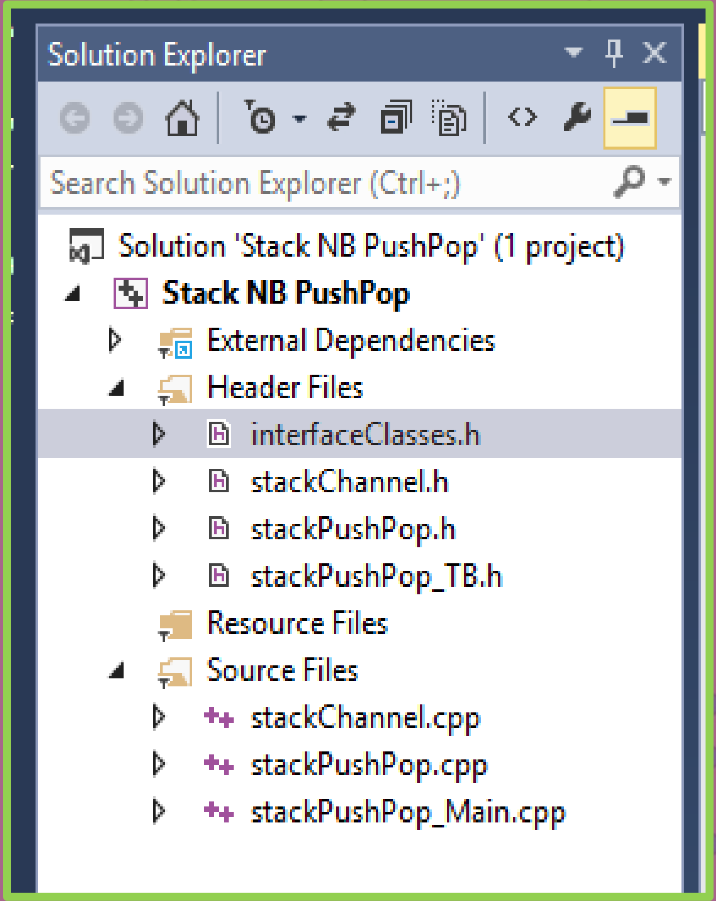


```
        SystemC 2.3.1-Accellera --- Sep 22 2015
        Copyright (c) 1996-2014 by all Contribut
        ALL RIGHTS RESERVED
Data: (00000000) was transmitted at: 3 ns
Data: (00000001) was transmitted at: 6 ns
Data: (00000000) was received at: 7 ns
Data: (00000010) was transmitted at: 9 ns
Data: (00000011) 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

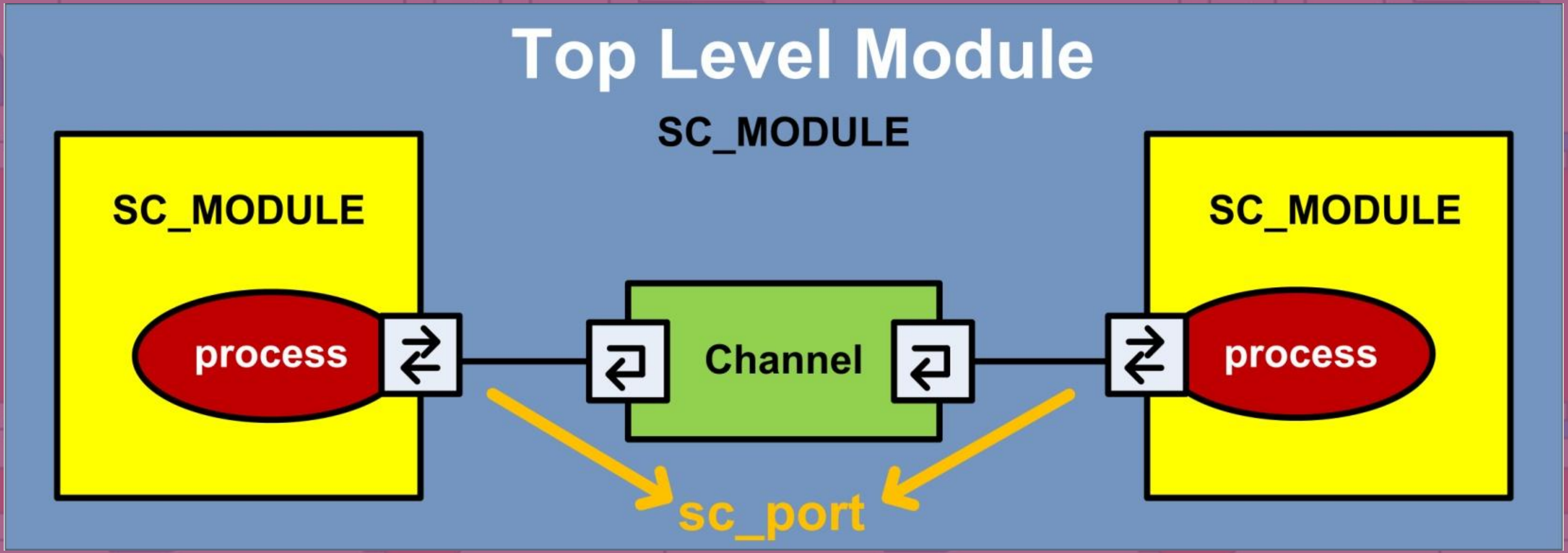
- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using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


Solution Explorer

Search Solution Explorer (Ctrl+;)

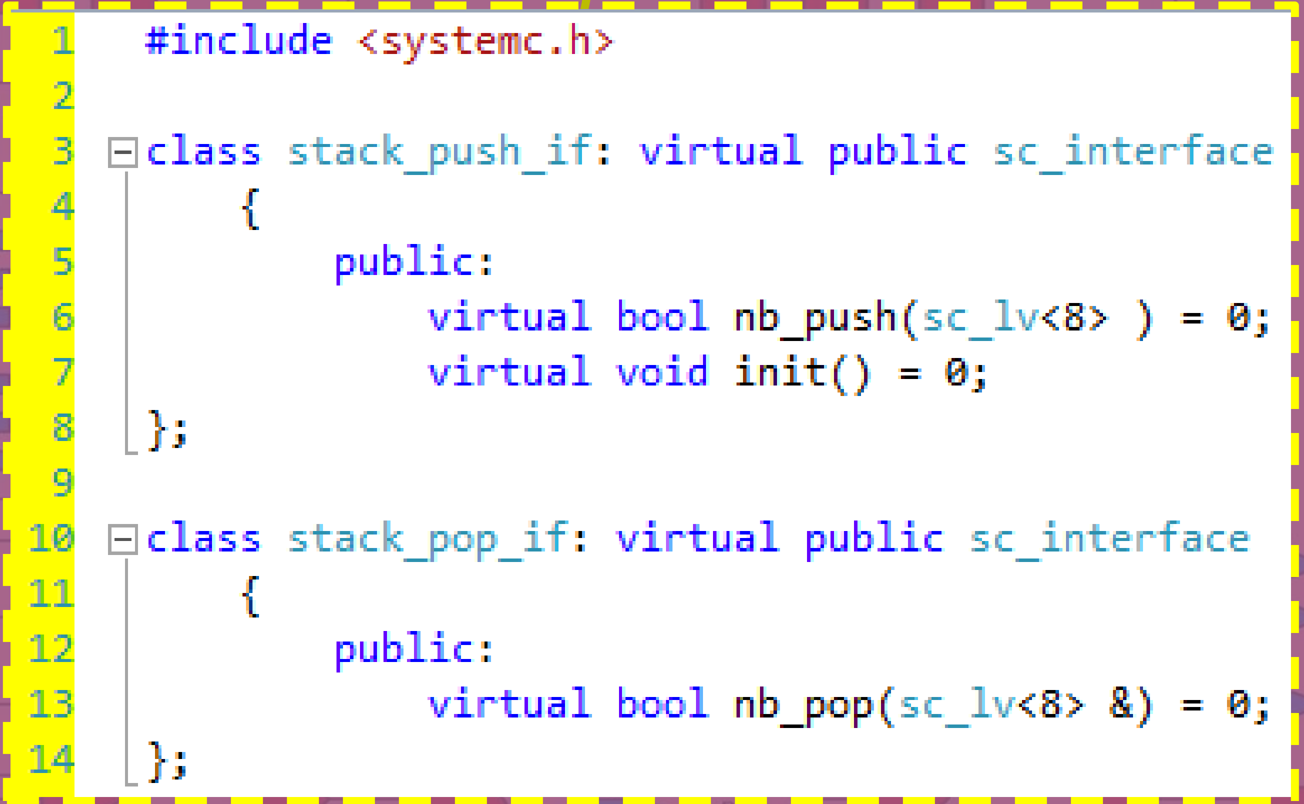
Solution 'Stack NB PushPop' (1 project)
Stack NB PushPop
External Dependencies
Header Files
interfaceClasses.h
stackChannel.h
stackPushPop.h
stackPushPop_TB.h
Resource Files
Source Files
stackChannel.cpp
stackPushPop.cpp
stackPushPop_Main.cpp

Example 3: Stack Channel

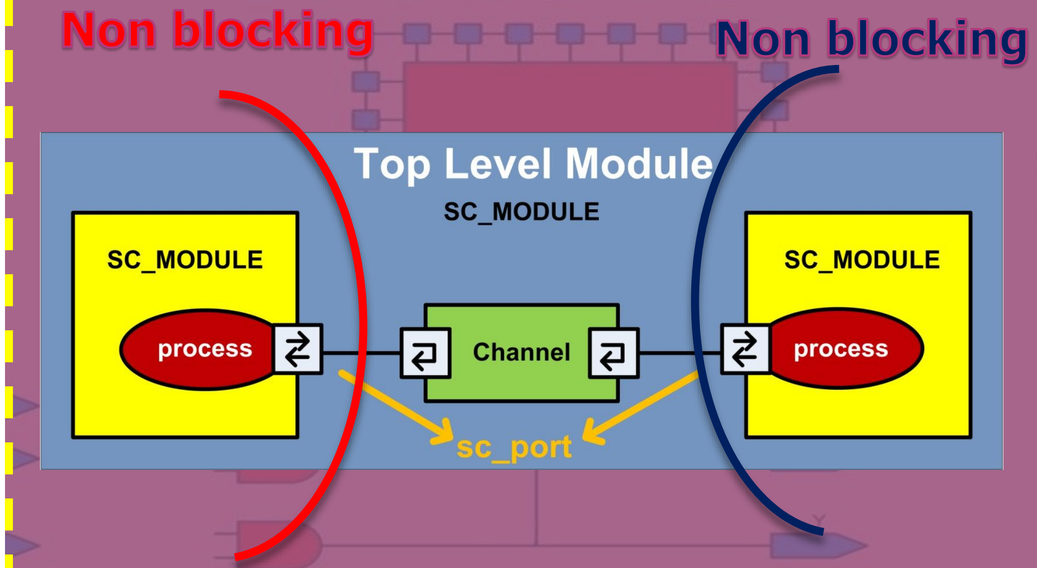


Example 3: Stack Channel, *interfaceClasses.h*

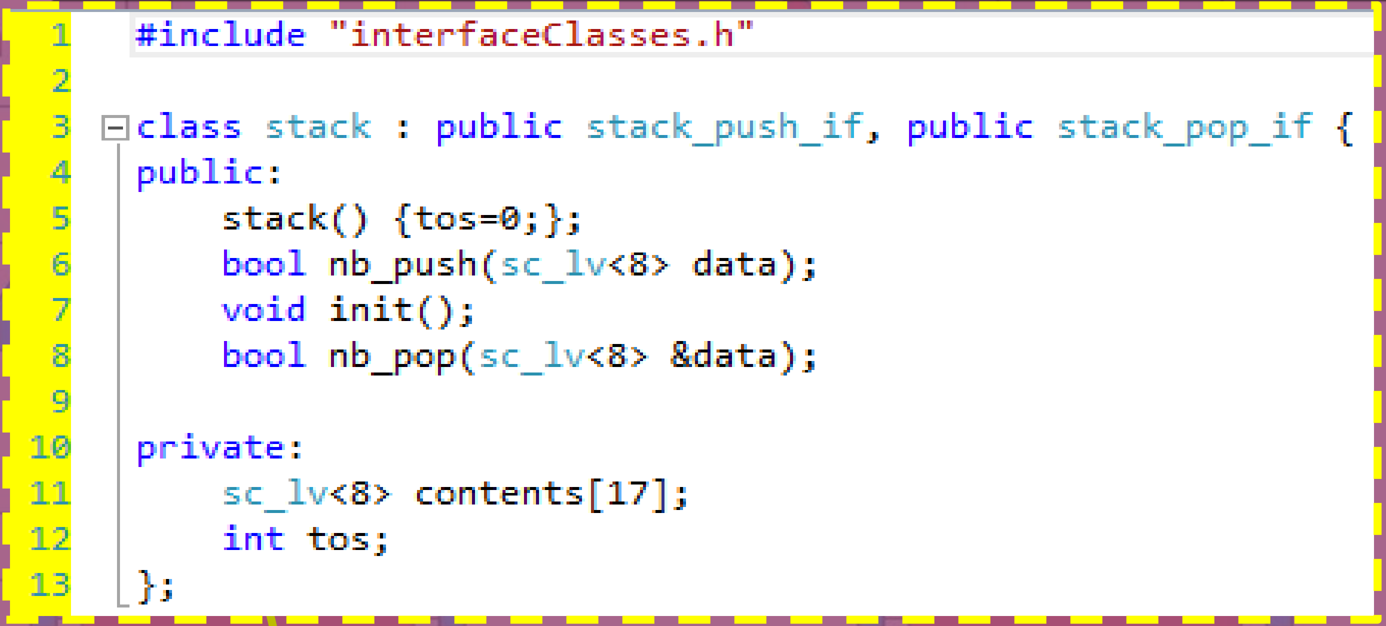
interfaceClasses.h



```
#include <systemc.h>

class stack_push_if: virtual public sc_interface
    {
        public:
            virtual bool nb_push(sc_lv<8> ) = 0;
            virtual void init() = 0;
};

class stack_pop_if: virtual public sc_interface
    {
        public:
            virtual bool nb_pop(sc_lv<8> &) = 0;
};
```

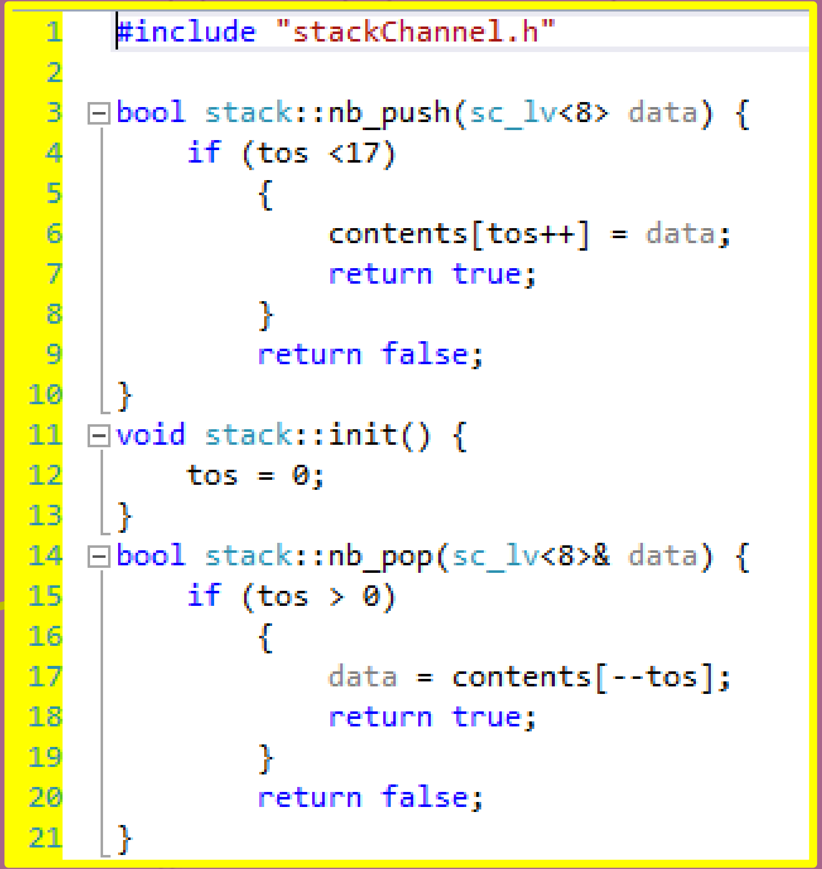


Example 3: Stack Channel, *stackChannel*



```
#include "interfaceClasses.h"

class stack : public stack_push_if, public stack_pop_if {
public:
    stack() {tos=0;};
    bool nb_push(sc_lv<8> data);
    void init();
    bool nb_pop(sc_lv<8> &data);

private:
    sc_lv<8> contents[17];
    int tos;
};
```

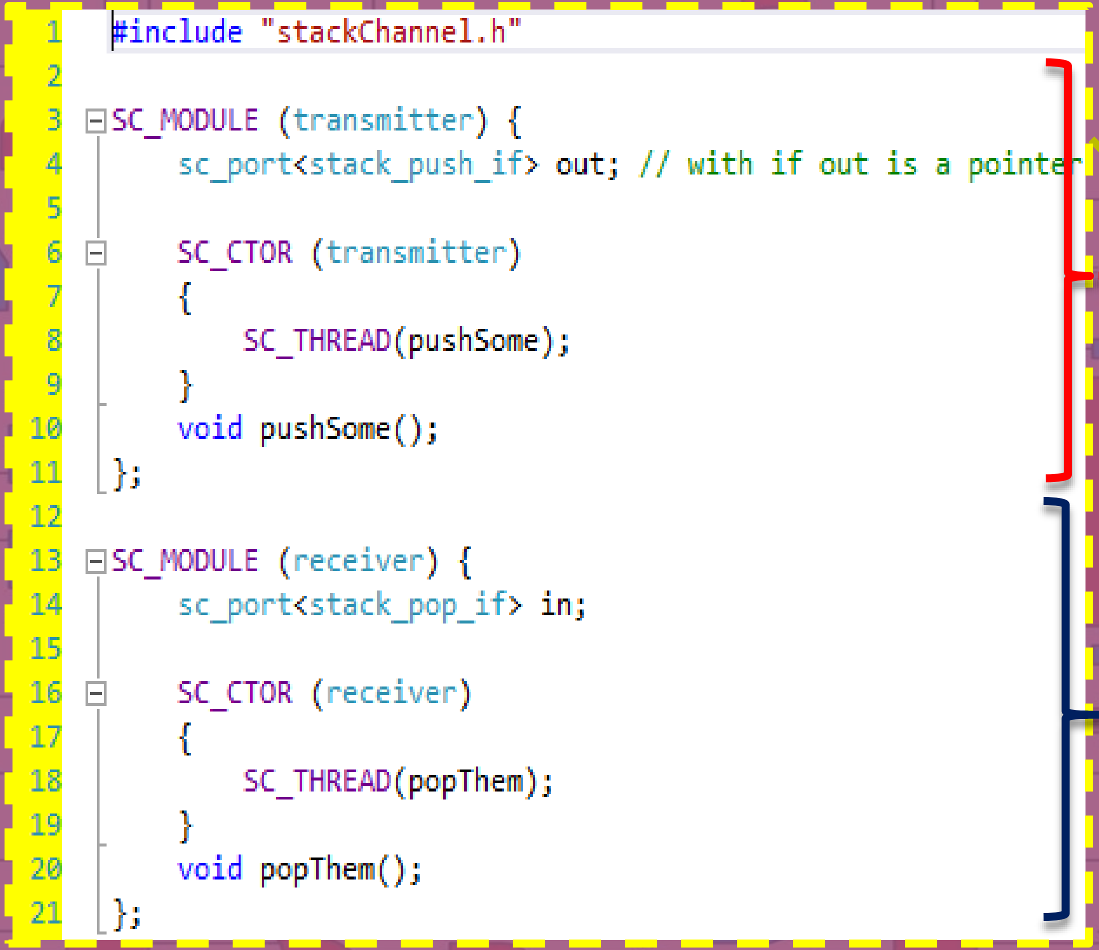
stackChannel.h



```
#include "stackChannel.h"

bool stack::nb_push(sc_lv<8> data) {
    if (tos <17)
        {
            contents[tos++] = data;
            return true;
        }
        return false;
}
void stack::init() {
    tos = 0;
}
bool stack::nb_pop(sc_lv<8>& data) {
    if (tos > 0)
        {
            data = contents[--tos];
            return true;
        }
        return false;
}
```

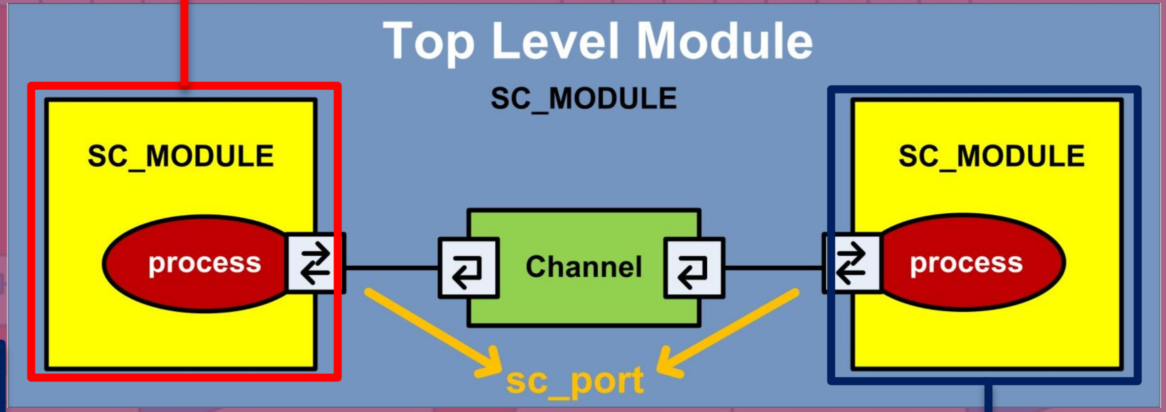
stackChannel.cpp

Example 3: Stack Channel, *stackPushPop*

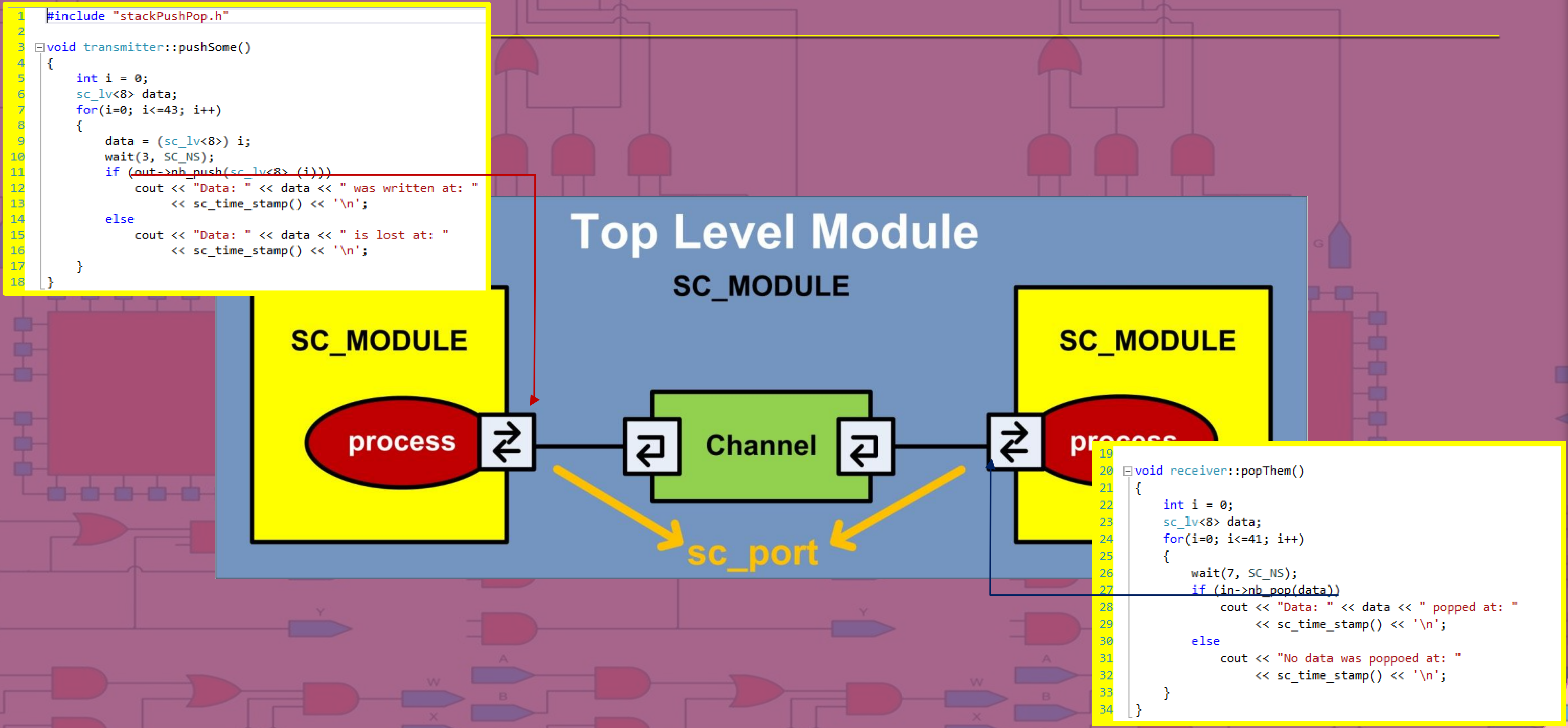


```
#include "stackChannel.h"

SC_MODULE (transmitter) {
    sc_port<stack_push_if> out; // with if out is a pointer

    SC_CTOR (transmitter)
    {
        SC_THREAD(pushSome);
    }
    void pushSome();
};

SC_MODULE (receiver) {
    sc_port<stack_pop_if> in;

    SC_CTOR (receiver)
    {
        SC_THREAD(popThem);
    }
    void popThem();
};
```

stackPushPop.h



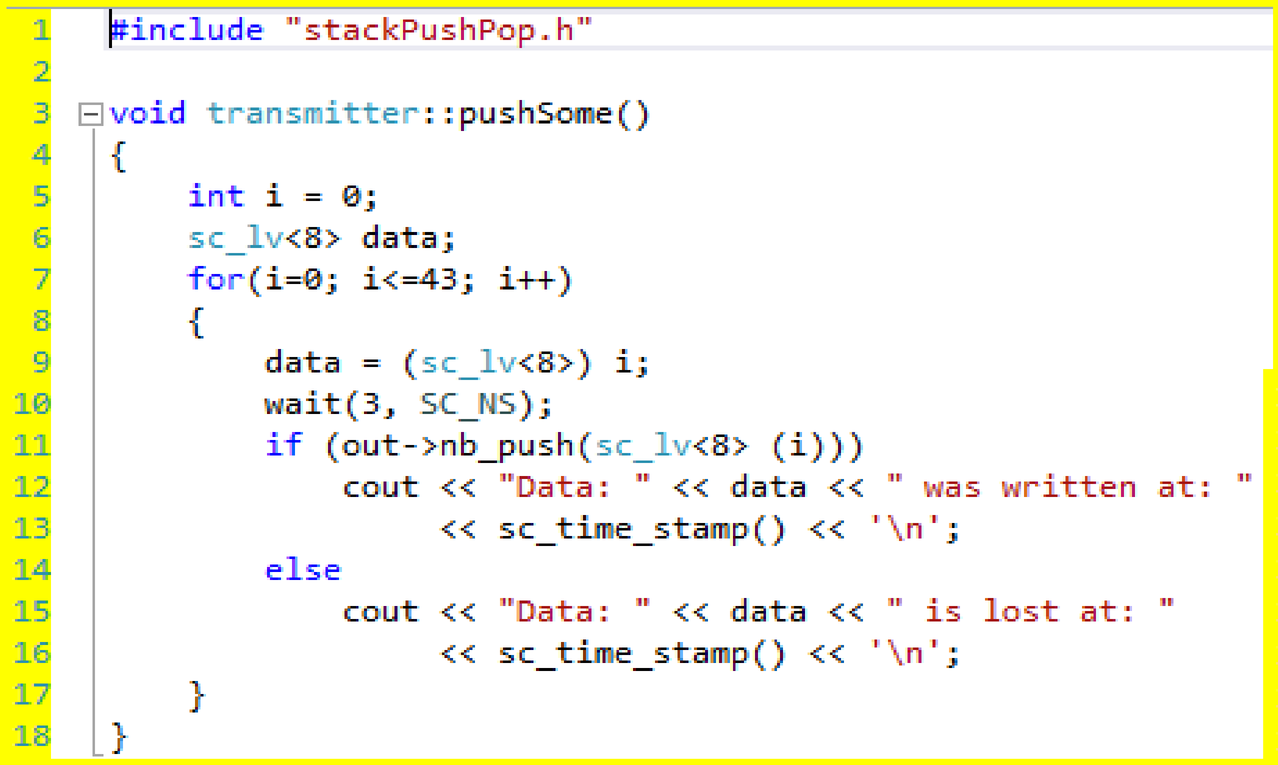
Example 3: Stack Channel, *stackPushPop*



```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';
    }
}
```

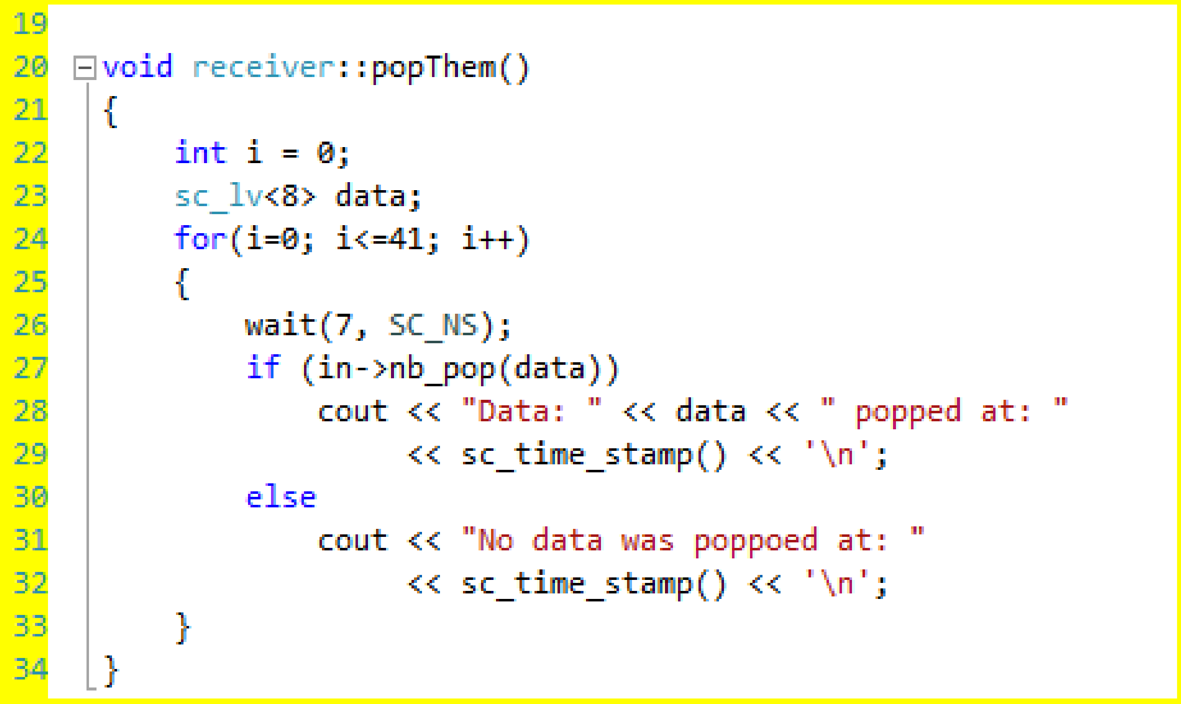
```cpp

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*



```
#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';
    }
}
```

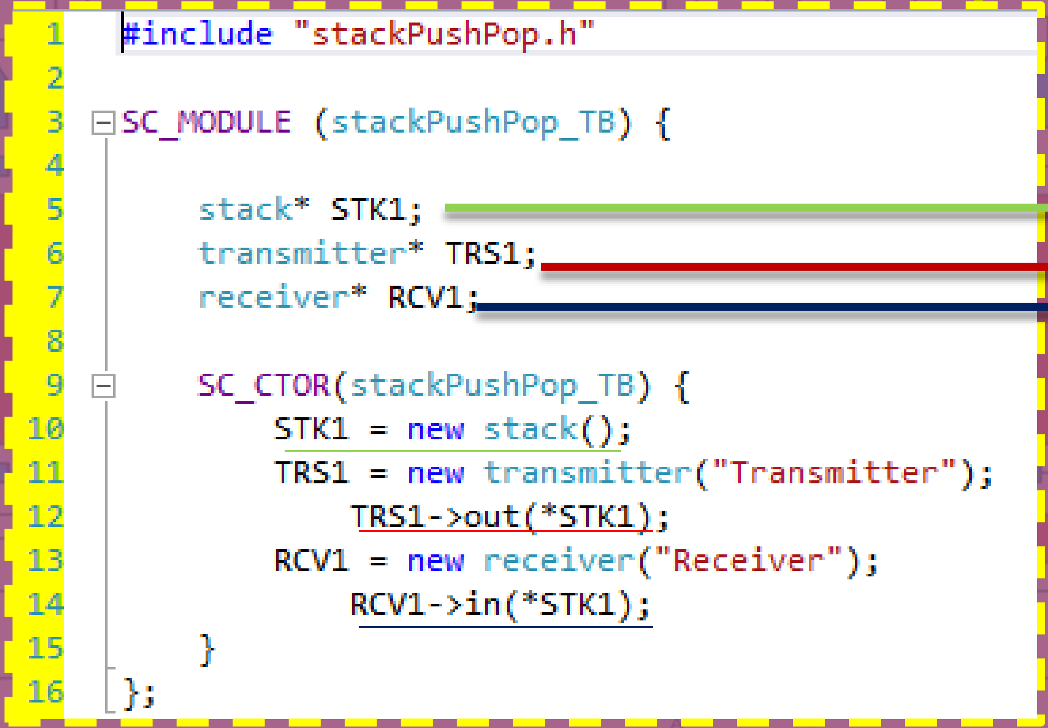
stackPushPop.cpp



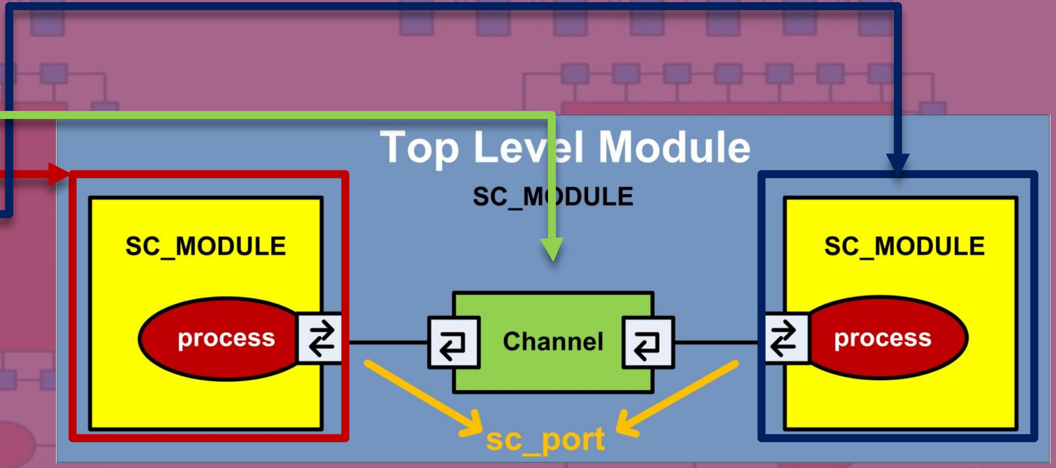
```
void receiver::popThem()
{
    int i = 0;
    sc_lv<8> data;
    for(i=0; i<=41; i++)
    {
        wait(7, SC_NS);
        if (in->nb_pop(data))
            cout << "Data: " << data << " popped at: "
                 << sc_time_stamp() << '\n';
        else
            cout << "No data was poppoed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

Example 3: Stack Channel, *stackPushPop_TB*

stackPushPop_TB.h

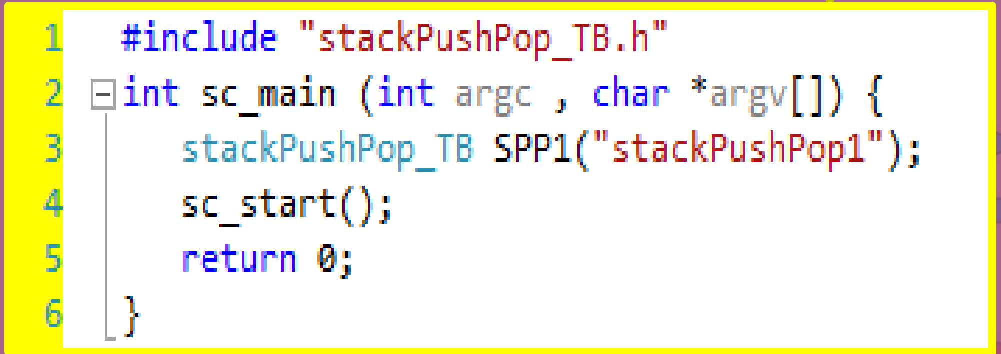


```
#include "stackPushPop.h"

SC_MODULE (stackPushPop_TB) {

    stack* STK1;
    transmitter* TRS1;
    receiver* RCV1;

    SC_CTOR(stackPushPop_TB) {
        STK1 = new stack();
        TRS1 = new transmitter("Transmitter");
            TRS1->out(*STK1);
        RCV1 = new receiver("Receiver");
            RCV1->in(*STK1);
    }
};
```



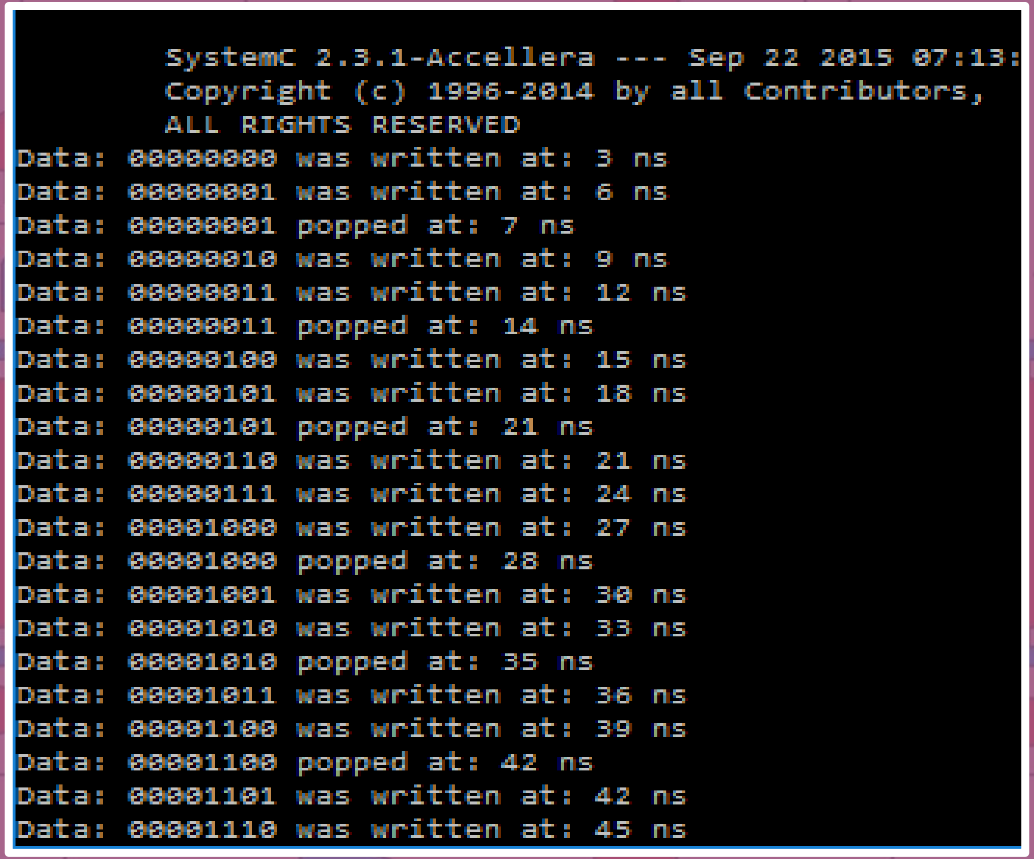
Example 3: Stack Channel, *stackPushPop_main*

stackPushPop_main.cpp

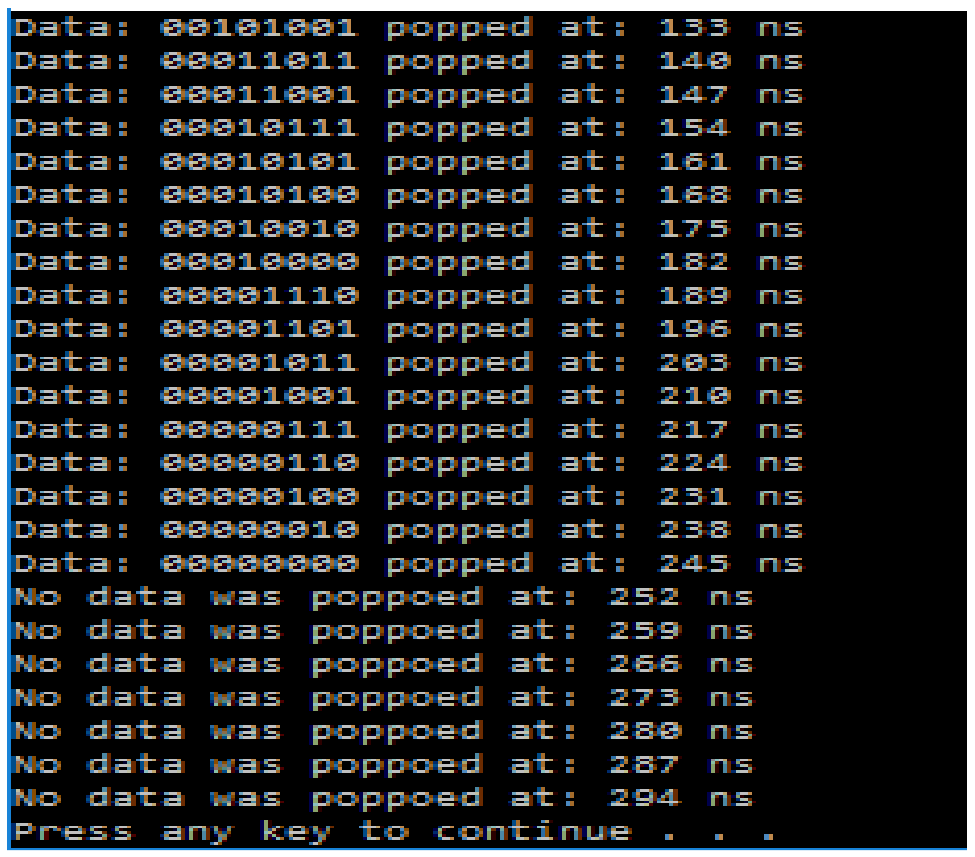


```cpp
#include "stackPushPop_TB.h"
int sc_main (int argc , char *argv[]) {
    stackPushPop_TB SPP1("stackPushPop1");
    sc_start();
    return 0;
}
```

Example 3: Stack Channel, *output*



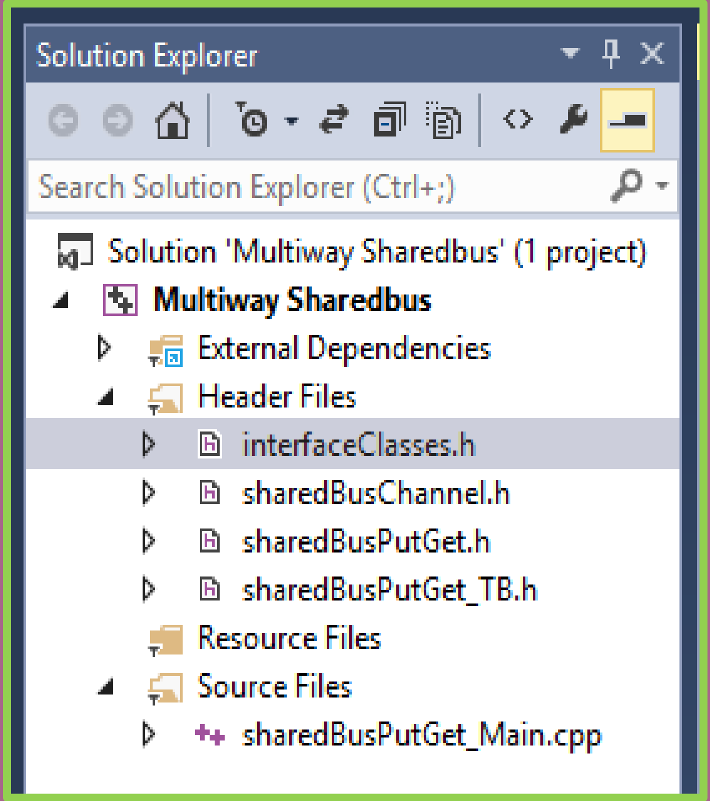
```
        SystemC 2.3.1-Accellera --- Sep 22 2015 07:13:
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED
Data: 00000000 was written at: 3 ns
Data: 00000001 was written at: 6 ns
Data: 00000001 popped at: 7 ns
Data: 00000010 was written at: 9 ns
Data: 00000011 was written 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

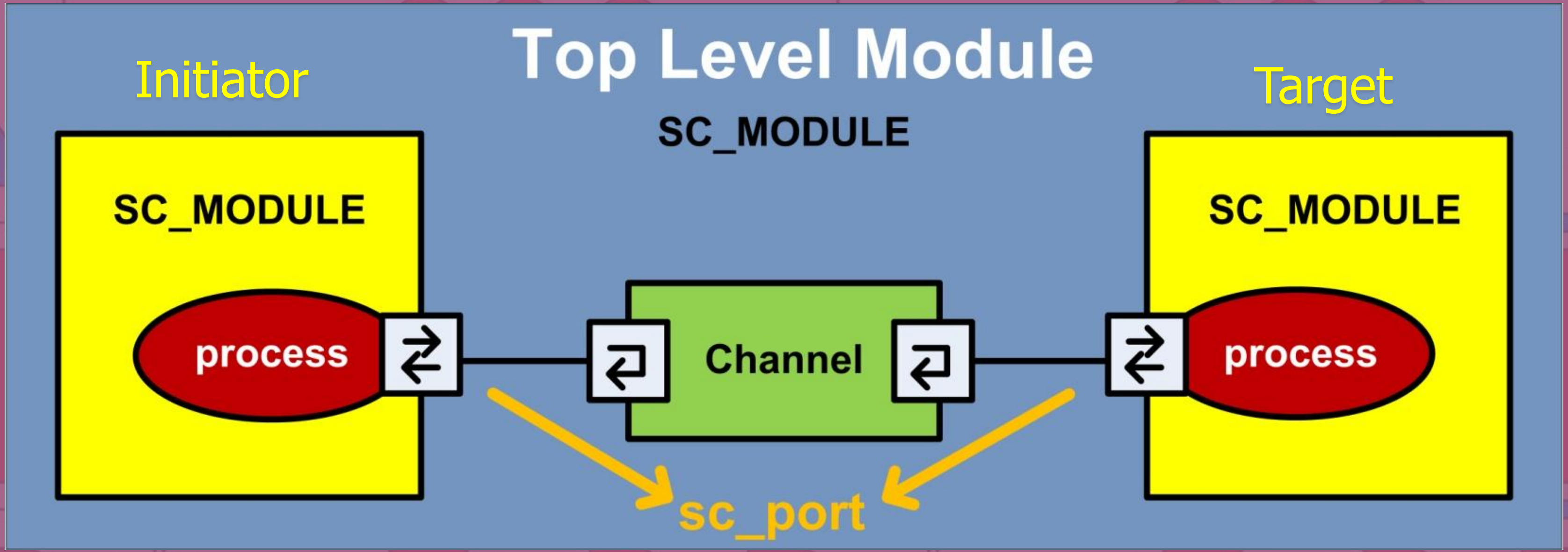
- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using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


Solution Explorer

Search Solution Explorer (Ctrl+;)

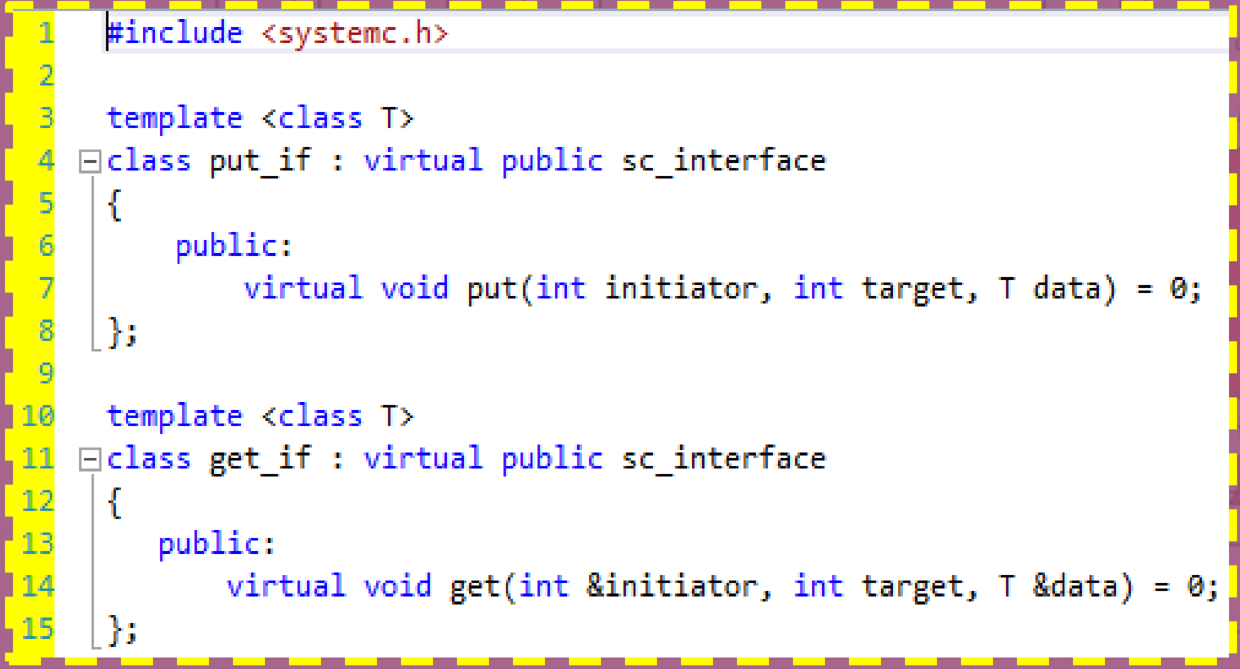
```
Solution 'Multiway Sharedbus' (1 project)
  Multiway Sharedbus
    External Dependencies
    Header Files
      interfaceClasses.h
      sharedBusChannel.h
      sharedBusPutGet.h
      sharedBusPutGet_TB.h
    Resource Files
    Source Files
      sharedBusPutGet_Main.cpp
```

Example 4: Multi-way shared bus



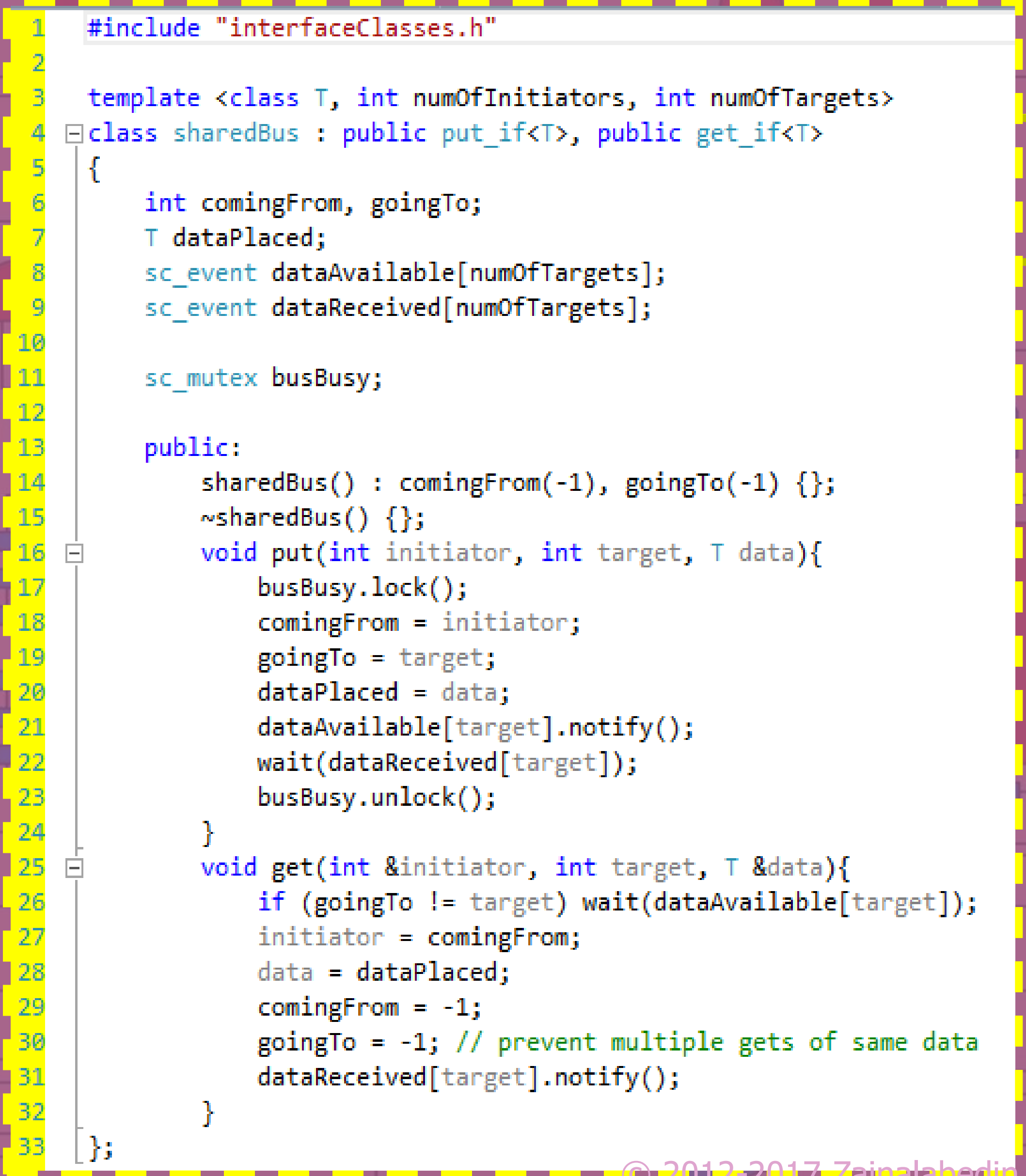
Example 4: Multi-way shared bus, *interfaceClasses.h*

interfaceClasses.h



```
#include <systemc.h>

template <class T>
class put_if : virtual public sc_interface
{
    public:
        virtual void put(int initiator, int target, T data) = 0;
};

template <class T>
class get_if : virtual public sc_interface
{
   public:
       virtual void get(int &initiator, int target, T &data) = 0;
};
```

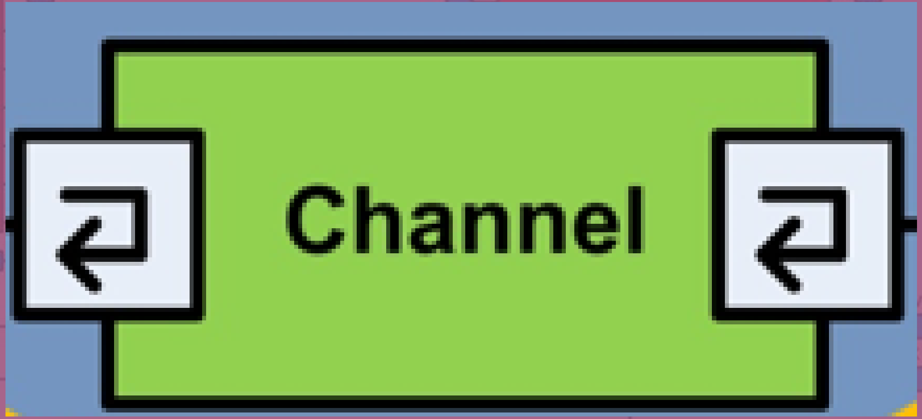
Example 4: Multi-way shared bus, *stackChannel*



```
#include "interfaceClasses.h"

template <class T, int numOfInitiators, int numOfTargets>
class sharedBus : public put_if<T>, public get_if<T>
{
    int comingFrom, goingTo;
    T dataPlaced;
    sc_event dataAvailable[numOfTargets];
    sc_event dataReceived[numOfTargets];

    sc_mutex busBusy;

    public:
        sharedBus() : comingFrom(-1), goingTo(-1) {};
        ~sharedBus() {};
        void put(int initiator, int target, T data){
            busBusy.lock();
            comingFrom = initiator;
            goingTo = target;
            dataPlaced = data;
            dataAvailable[target].notify();
            wait(dataReceived[target]);
            busBusy.unlock();
        }
        void get(int &initiator, int target, T &data){
            if (goingTo != target) wait(dataAvailable[target]);
            initiator = comingFrom;
            data = dataPlaced;
            comingFrom = -1;
            goingTo = -1; // prevent multiple gets of same data
            dataReceived[target].notify();
        }
};
```

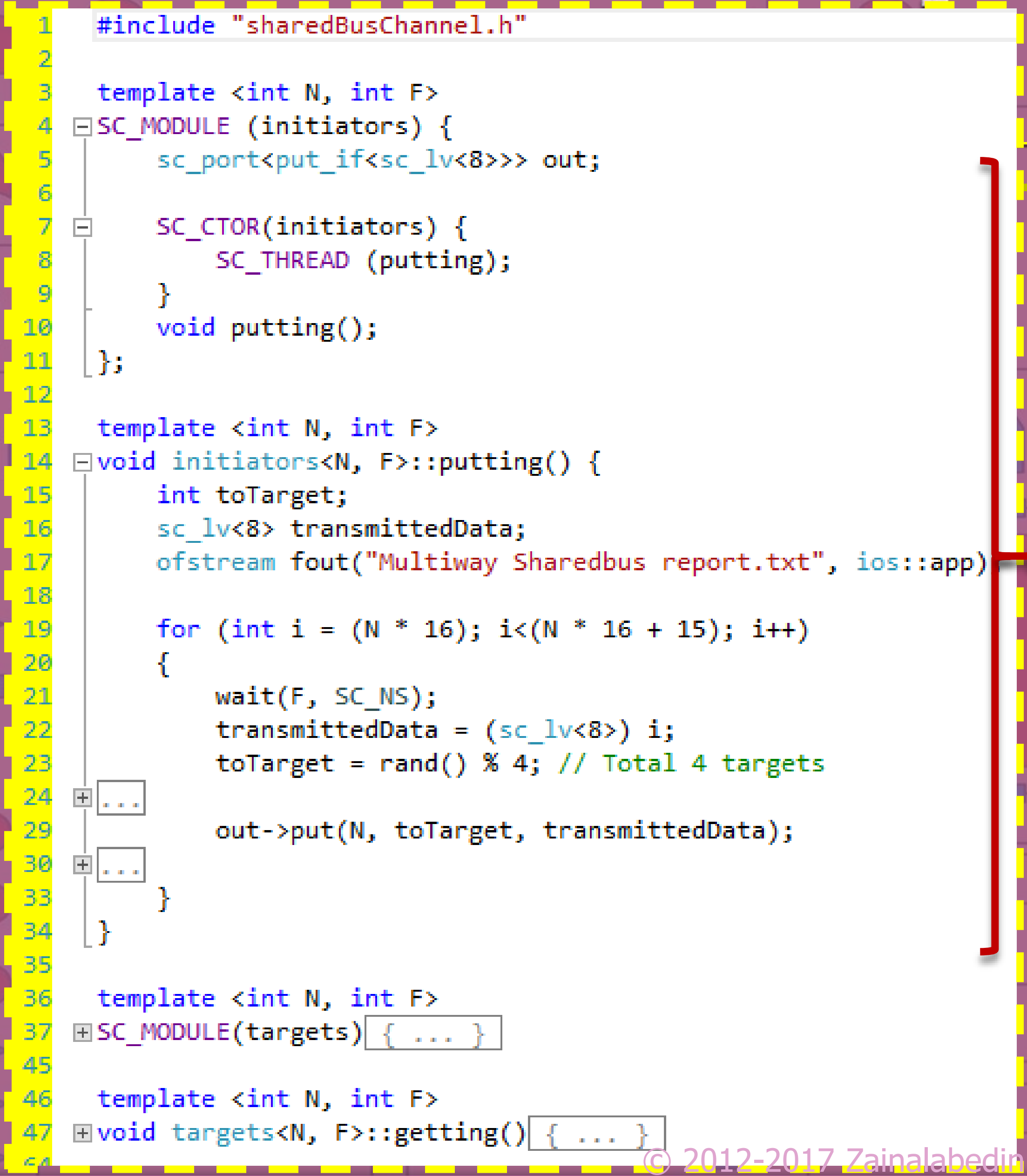
sharedBusChannel.h

Template form



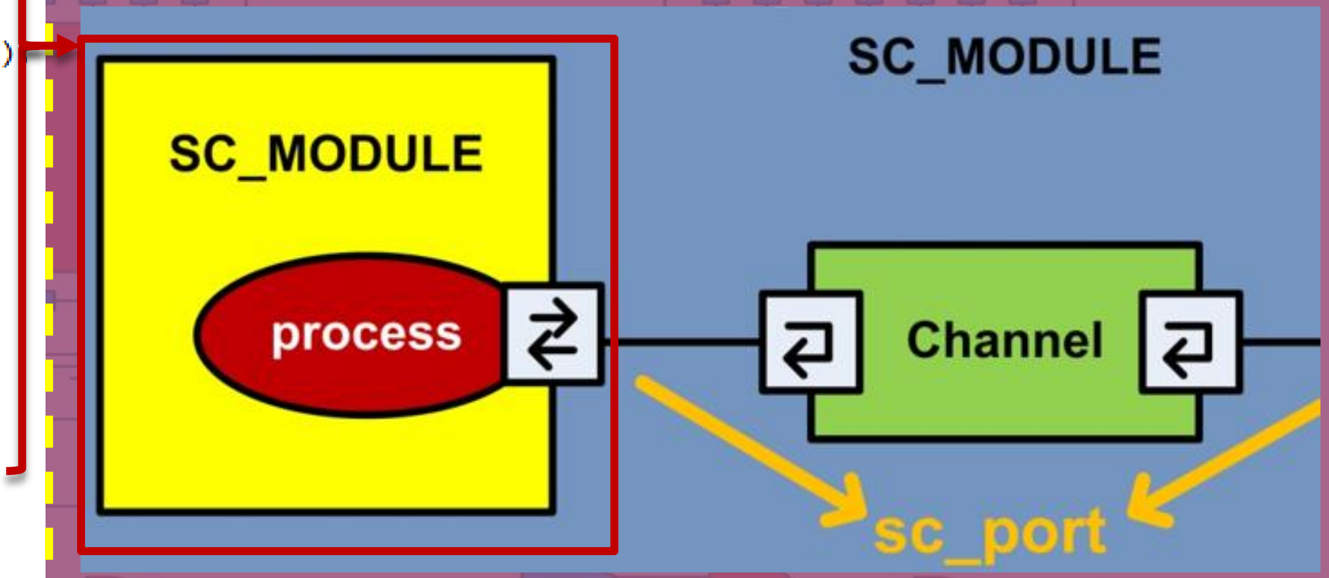
Example 4: Multi-way shared bus,

sharedBusPutGet

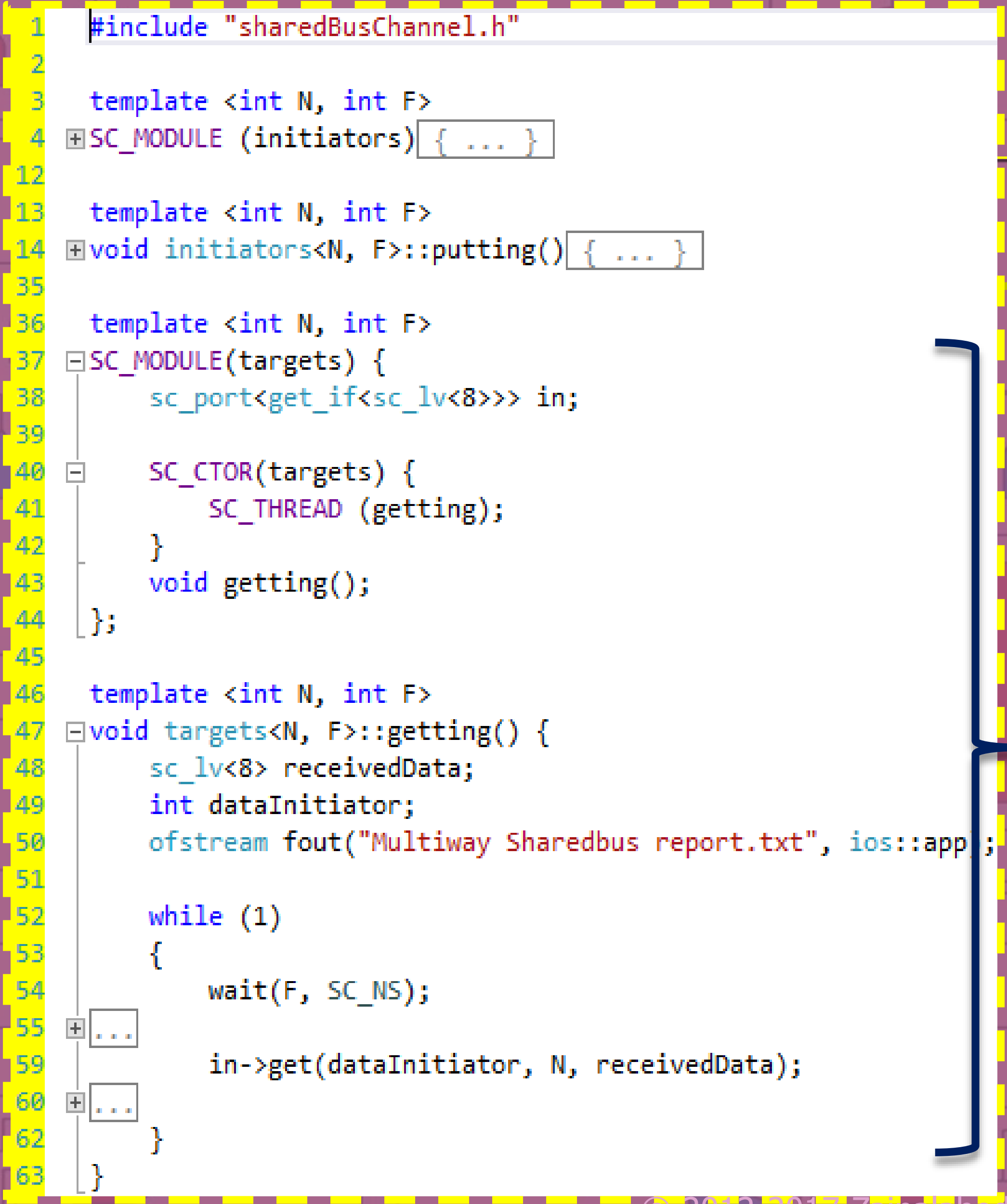


```
#include "sharedBusChannel.h"

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
...
        out->put(N, toTarget, transmittedData);
...
    }
}

template <int N, int F>
SC_MODULE(targets){ ... }

template <int N, int F>
void targets<N, F>::getting(){ ... }
```

sharedBusPutGet.h

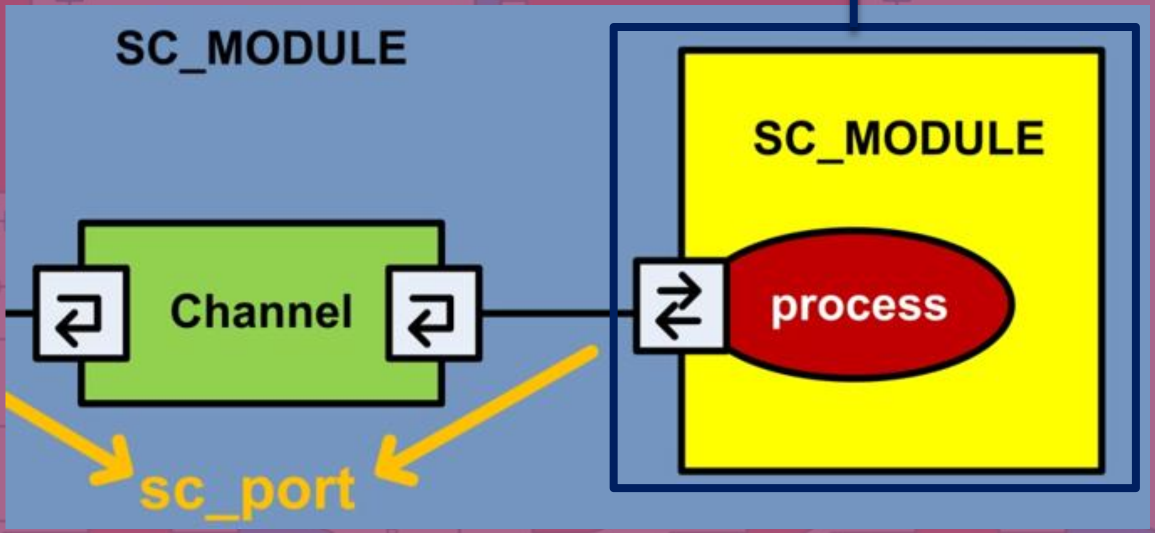


Example 4: Multi-way shared bus, *sharedBusPutGet*

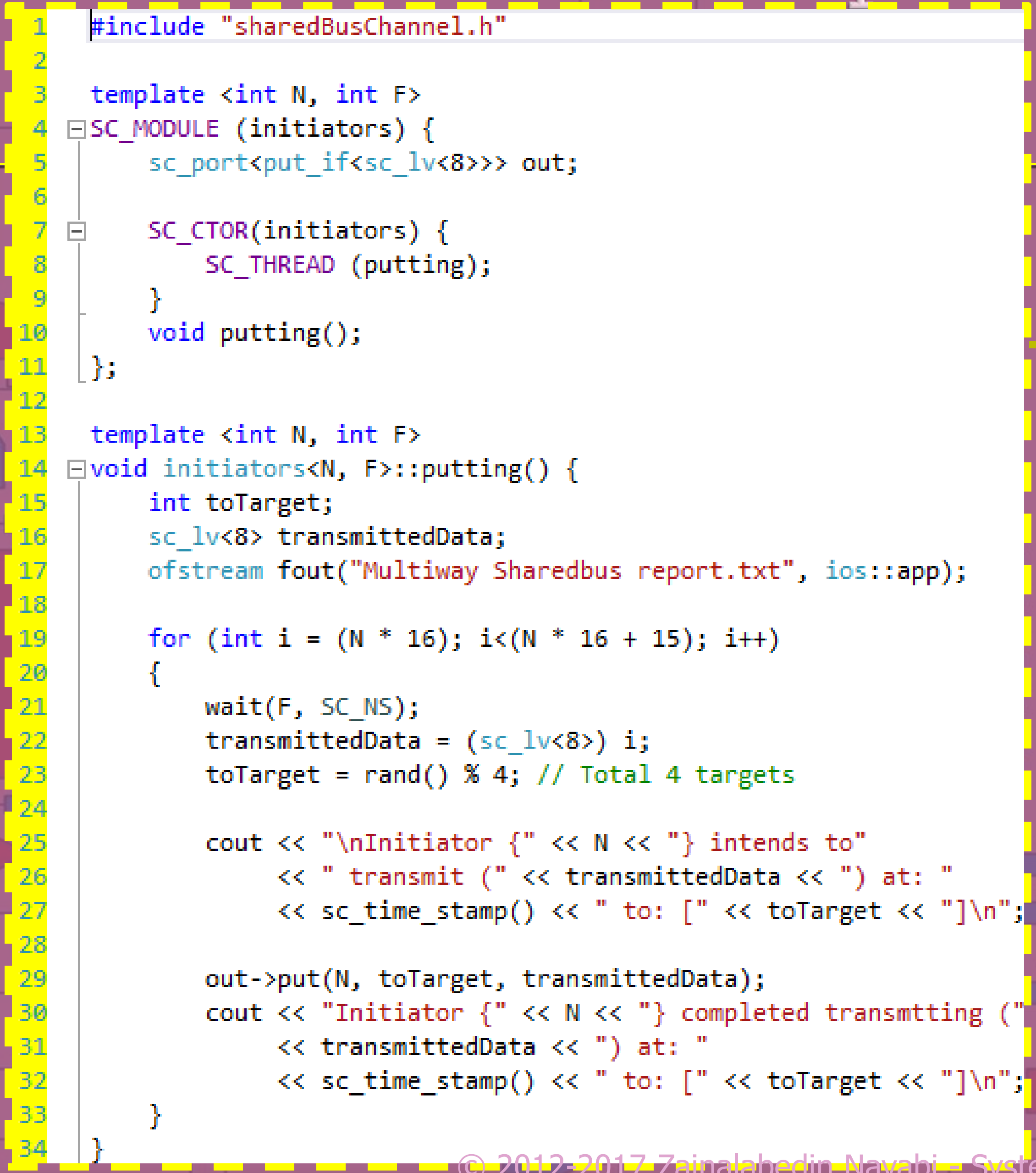


```cpp
#include "sharedBusChannel.h"

template <int N, int F>
SC_MODULE (initiators) { ... }

template <int N, int F>
void initiators<N, F>::putting() { ... }

template <int N, int F>
SC_MODULE(targets) {
    sc_port<get_if<sc_lv<8>>> in;

    SC_CTOR(targets) {
        SC_THREAD (getting);
    }
    void getting();
};

template <int N, int F>
void targets<N, F>::getting() {
    sc_lv<8> receivedData;
    int dataInitiator;
    ofstream fout("Multiway Sharedbus report.txt", ios::app);

    while (1)
    {
        wait(F, SC_NS);
...
        in->get(dataInitiator, N, receivedData);
...
    }
}
```

sharedBusPutGet.h



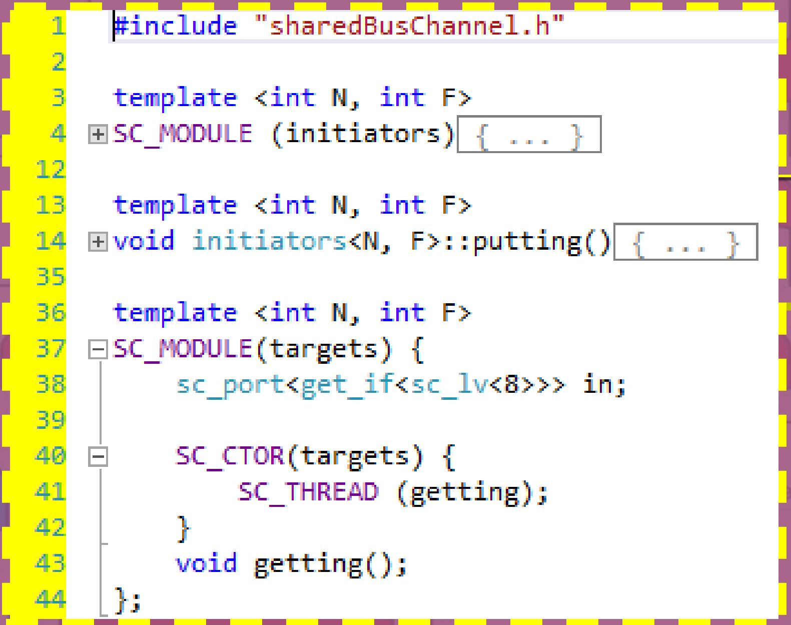
Example 4: Multi-way shared bus, *sharedBusPutGet*



```
#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";
    }
}
```

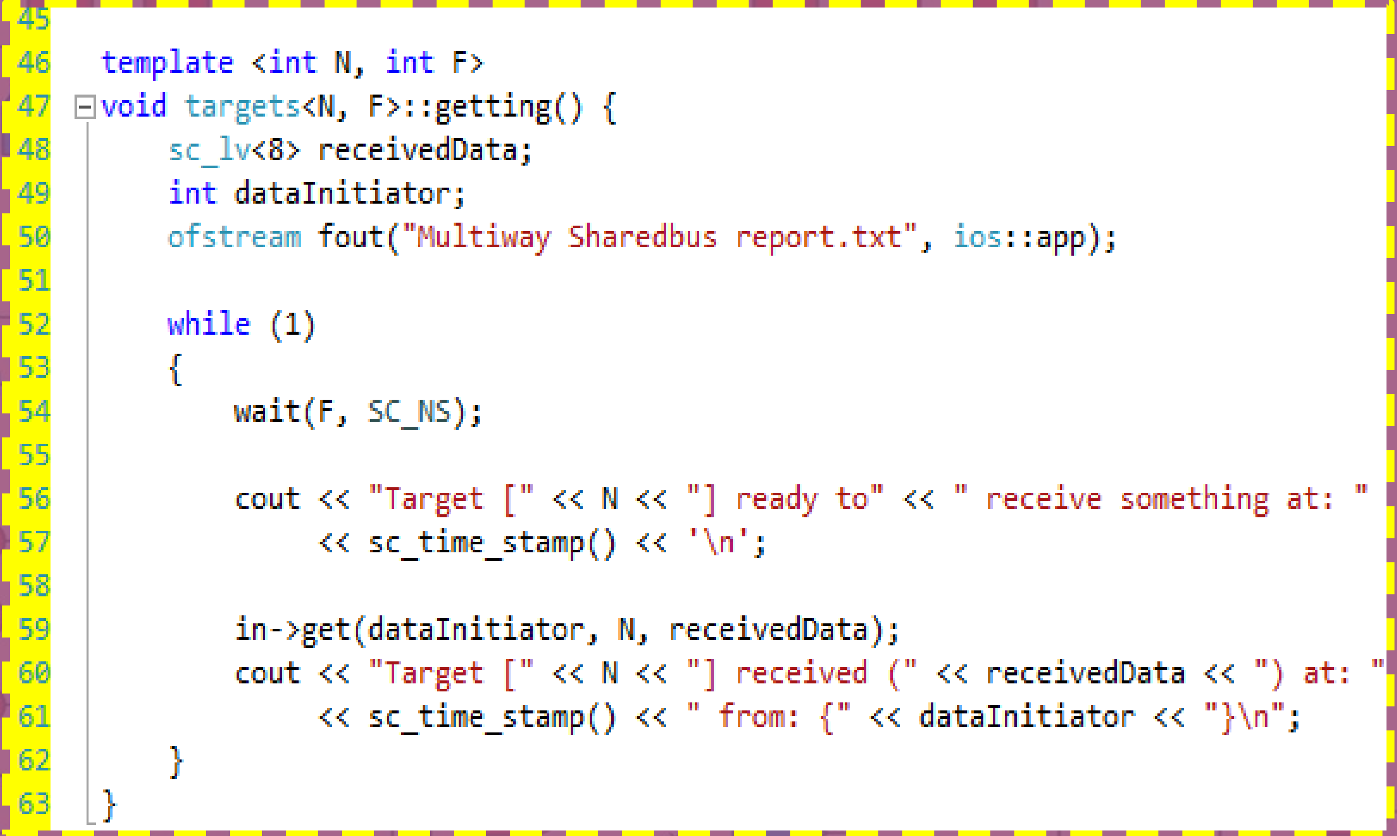
sharedBusPutGet.h

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



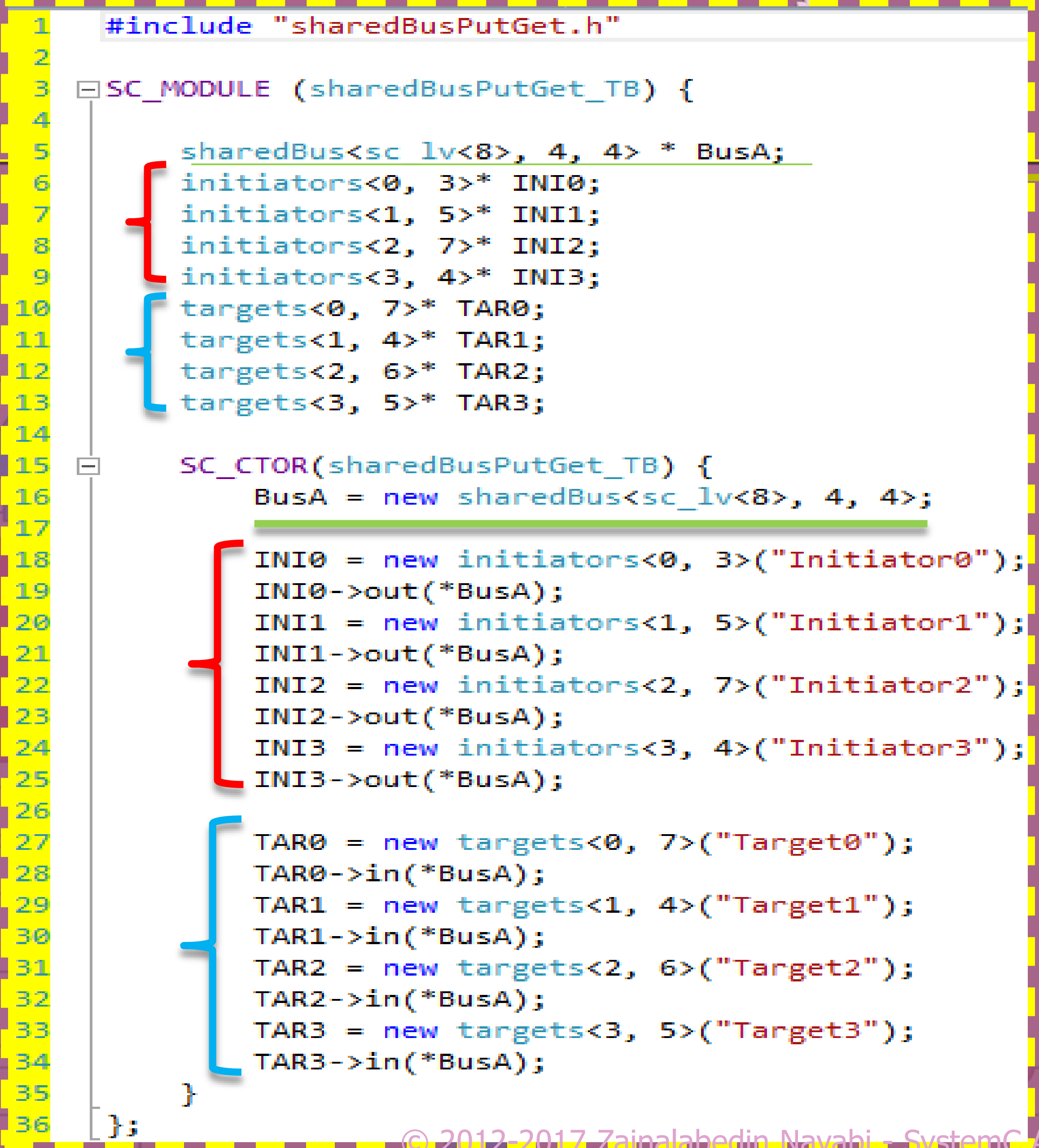
```
#include "sharedBusChannel.h"

template <int N, int F>
SC_MODULE (initiators) { ... }

template <int N, int F>
void initiators<N, F>::putting() { ... }

template <int N, int F>
SC_MODULE(targets) {
    sc_port<get_if<sc_lv<8>>> in;

    SC_CTOR(targets) {
        SC_THREAD (getting);
    }
    void getting();
};
```

sharedBusPutGet.h



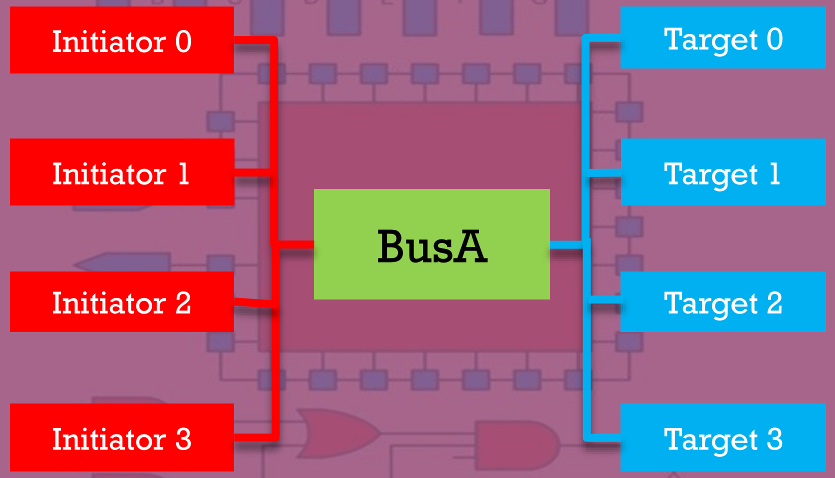
```

template <int N, int F>
void targets<N, F>::getting() {
    sc_lv<8> receivedData;
    int dataInitiator;
    ofstream fout("Multiway Sharedbus report.txt", ios::app);

    while (1)
    {
        wait(F, SC_NS);

        cout << "Target [" << N << "] ready to" << " receive something at: "
            << sc_time_stamp() << '\n';

        in->get(dataInitiator, N, receivedData);
        cout << "Target [" << N << "] received (" << receivedData << ") at: "
            << sc_time_stamp() << " from: {" << dataInitiator << "}\n";
    }
}
```

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



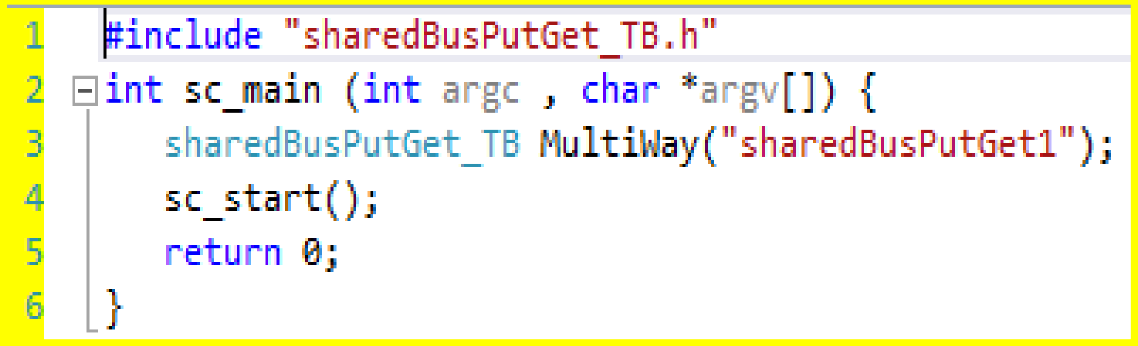
```
#include "sharedBusPutGet.h"

SC_MODULE (sharedBusPutGet_TB) {

    sharedBus<sc_lv<8>, 4, 4> * BusA;
    initiators<0, 3>* INI0;
    initiators<1, 5>* INI1;
    initiators<2, 7>* INI2;
    initiators<3, 4>* INI3;
    targets<0, 7>* TAR0;
    targets<1, 4>* TAR1;
    targets<2, 6>* TAR2;
    targets<3, 5>* TAR3;

    SC_CTOR(sharedBusPutGet_TB) {
        BusA = new sharedBus<sc_lv<8>, 4, 4>;

        INI0 = new initiators<0, 3>("Initiator0");
        INI0->out(*BusA);
        INI1 = new initiators<1, 5>("Initiator1");
        INI1->out(*BusA);
        INI2 = new initiators<2, 7>("Initiator2");
        INI2->out(*BusA);
        INI3 = new initiators<3, 4>("Initiator3");
        INI3->out(*BusA);

        TAR0 = new targets<0, 7>("Target0");
        TAR0->in(*BusA);
        TAR1 = new targets<1, 4>("Target1");
        TAR1->in(*BusA);
        TAR2 = new targets<2, 6>("Target2");
        TAR2->in(*BusA);
        TAR3 = new targets<3, 5>("Target3");
        TAR3->in(*BusA);
    }
};
```

sharedBusPutGet_TB.h



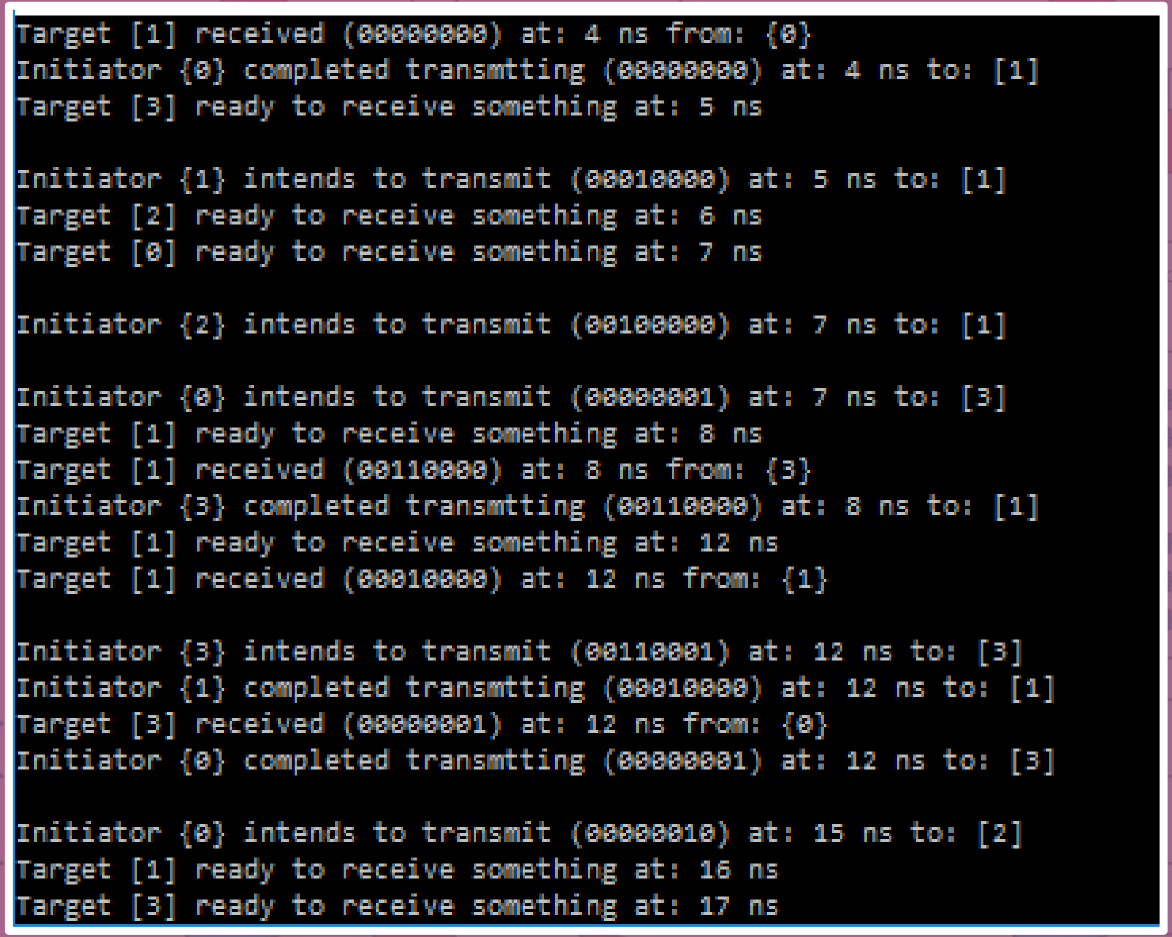
Example 4: Multi-way shared bus, *sharedBusPutGet_main*

sharedBusPutGet_main.cpp



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

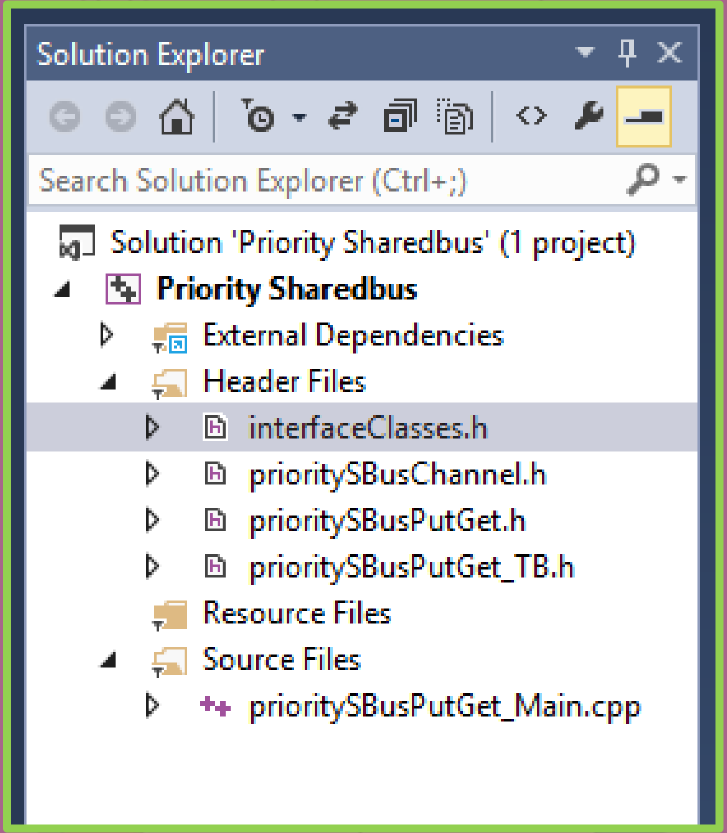
Example 4: Multi-way shared bus, *output*



```
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

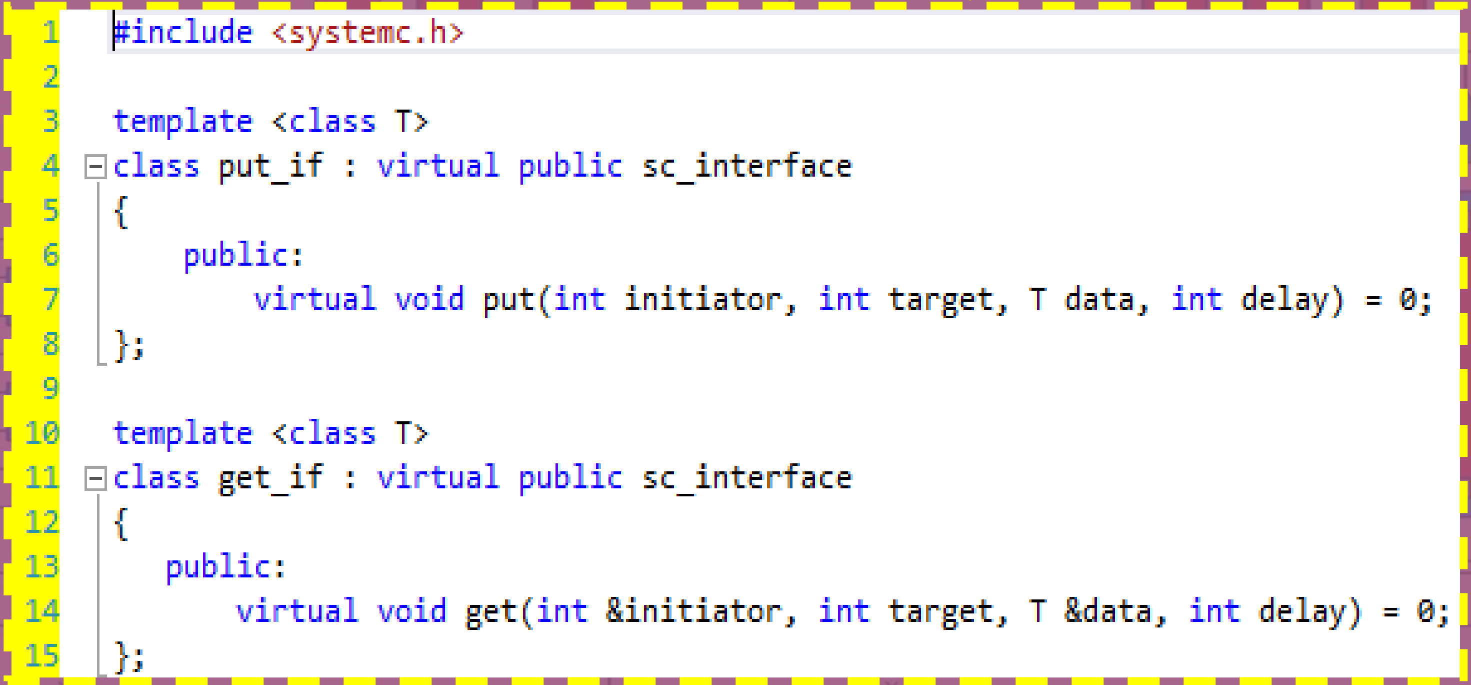
- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using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



```
Solution 'Priority Sharedbus' (1 project)
◢ Priority Sharedbus
   ▷ External Dependencies
   ◢ Header Files
      ▷ interfaceClasses.h
      ▷ prioritySBusChannel.h
      ▷ prioritySBusPutGet.h
      ▷ prioritySBusPutGet_TB.h
     Resource Files
   ◢ Source Files
      ▷ prioritySBusPutGet_Main.cpp
```

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

interfaceClasses.h

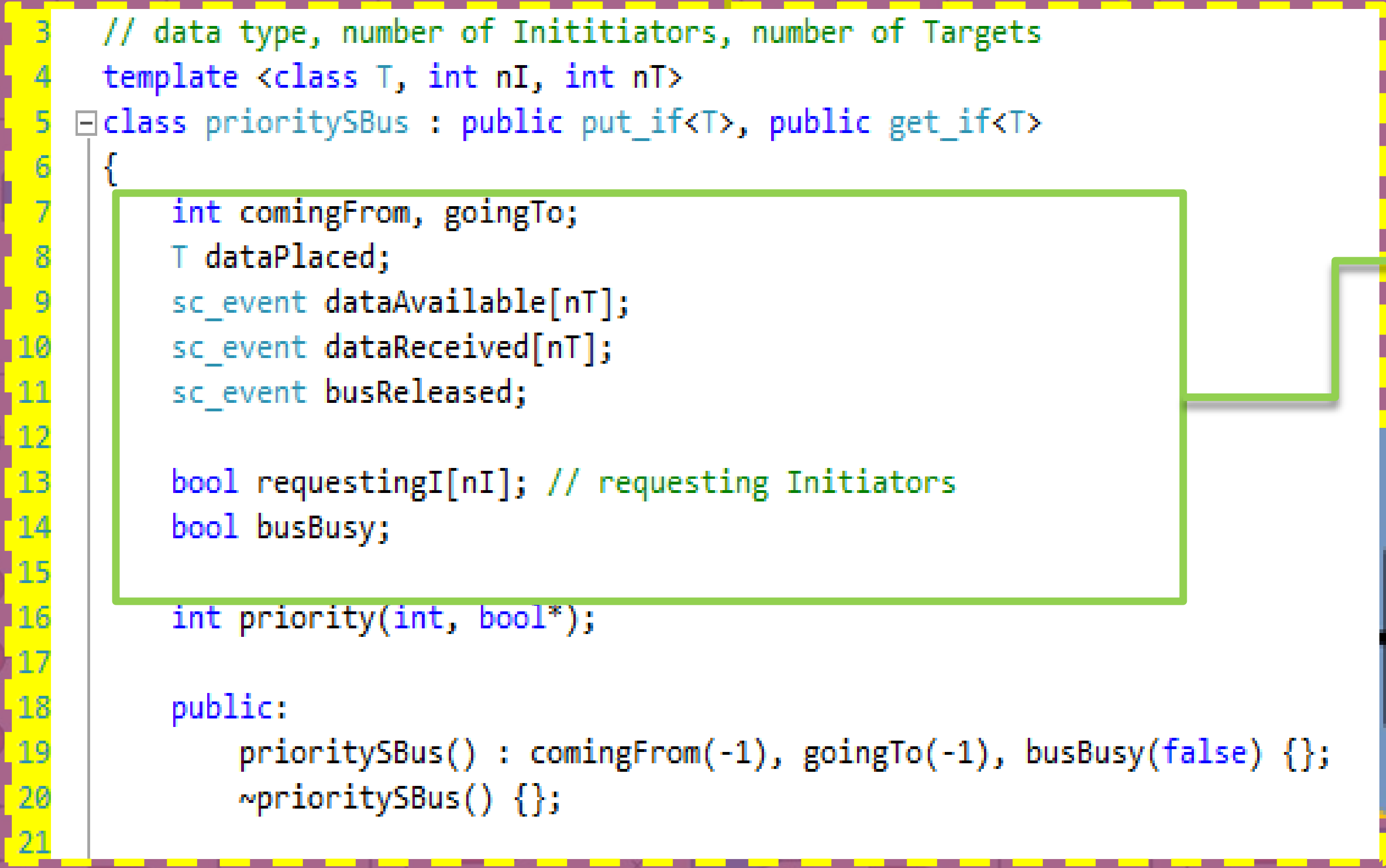


```
#include <systemc.h>

template <class T>
class put_if : virtual public sc_interface
{
    public:
        virtual void put(int initiator, int target, T data, int delay) = 0;
};

template <class T>
class get_if : virtual public sc_interface
{
   public:
       virtual void get(int &initiator, int target, T &data, int delay) = 0;
};
```

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

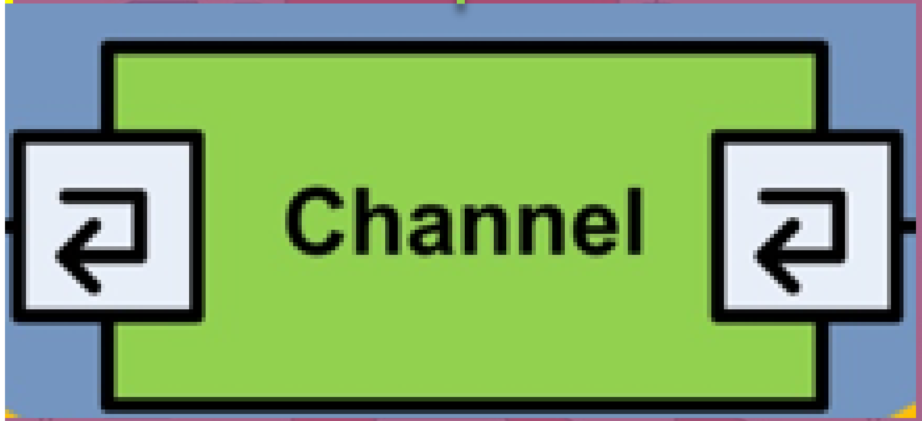
Template form

prioritySBusChannel.h



```
// data type, number of Inititiators, number of Targets
template <class T, int nI, int nT>
class prioritySBus : public put_if<T>, public get_if<T>
{
    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*);

    public:
        prioritySBus() : comingFrom(-1), goingTo(-1), busBusy(false) {};
        ~prioritySBus() {};

```

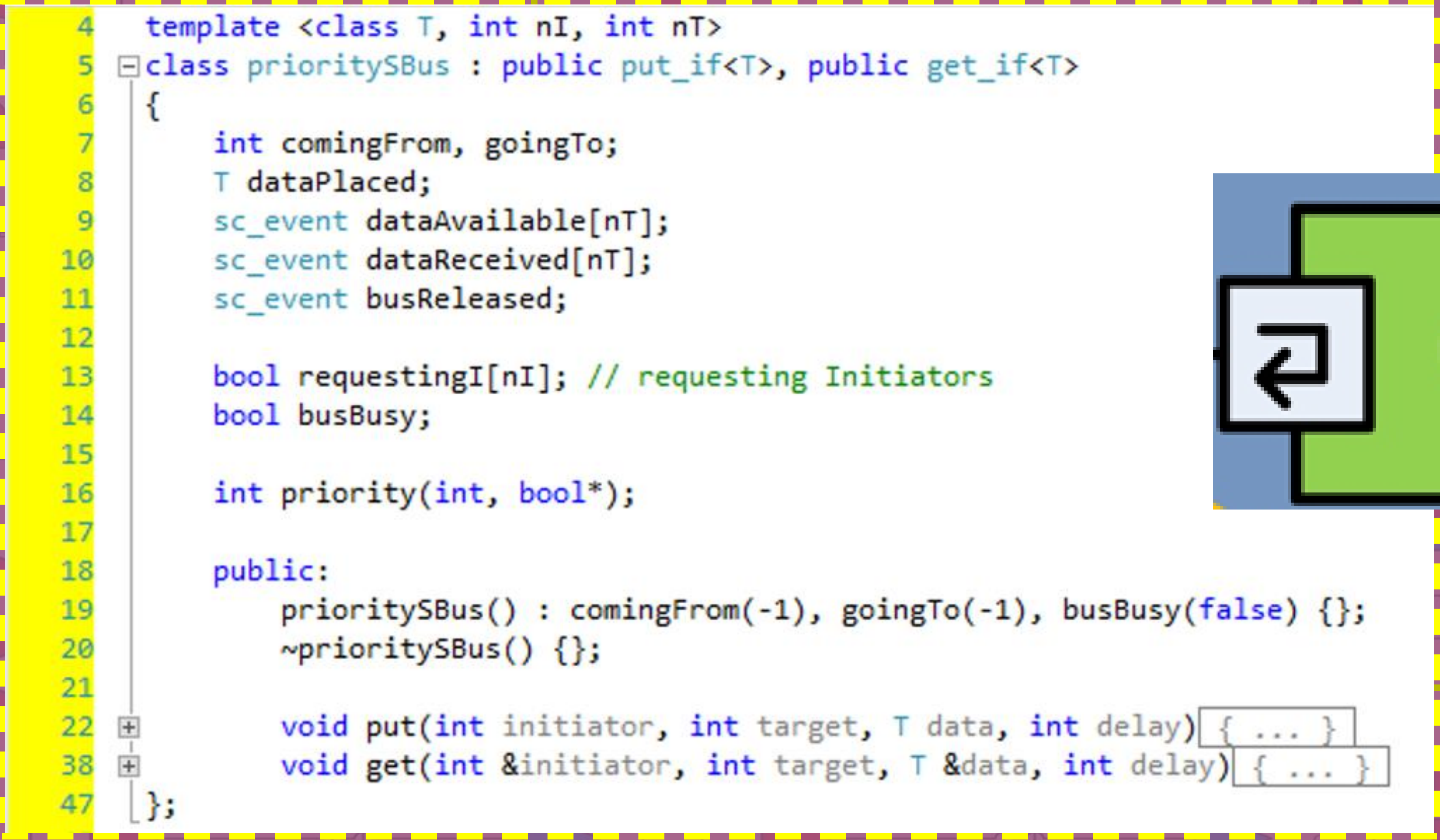
Handling Priorities



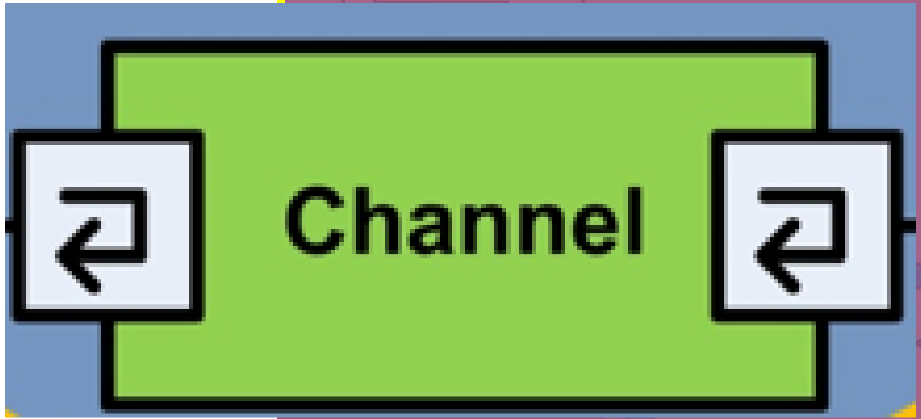
Example 5: *Priority shared bus,*

prioritySBusChannel.h

Template form

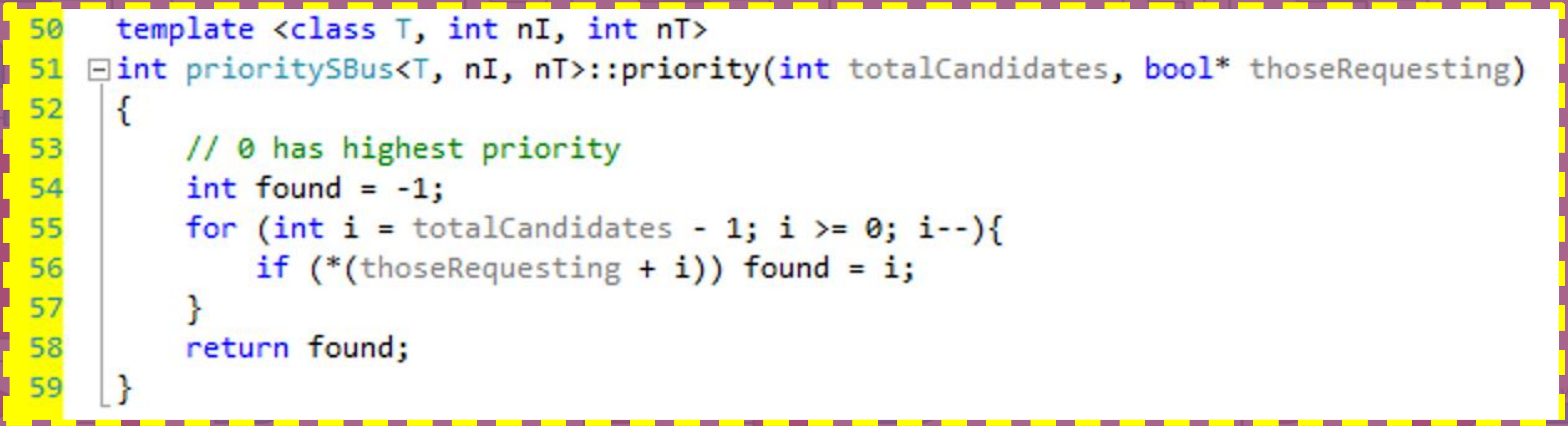


```
template <class T, int nI, int nT>
class prioritySBus : public put_if<T>, public get_if<T>
{
    int comingFrom, goingTo;
    T dataPlaced;
    sc_event dataAvailable[nT];
    sc_event dataReceived[nT];
    sc_event busReleased;

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

    int priority(int, bool*);

public:
    prioritySBus() : comingFrom(-1), goingTo(-1), busBusy(false) {};
    ~prioritySBus() {};

    void put(int initiator, int target, T data, int delay){ ... }
    void get(int &initiator, int target, T &data, int delay){ ... }
};
```



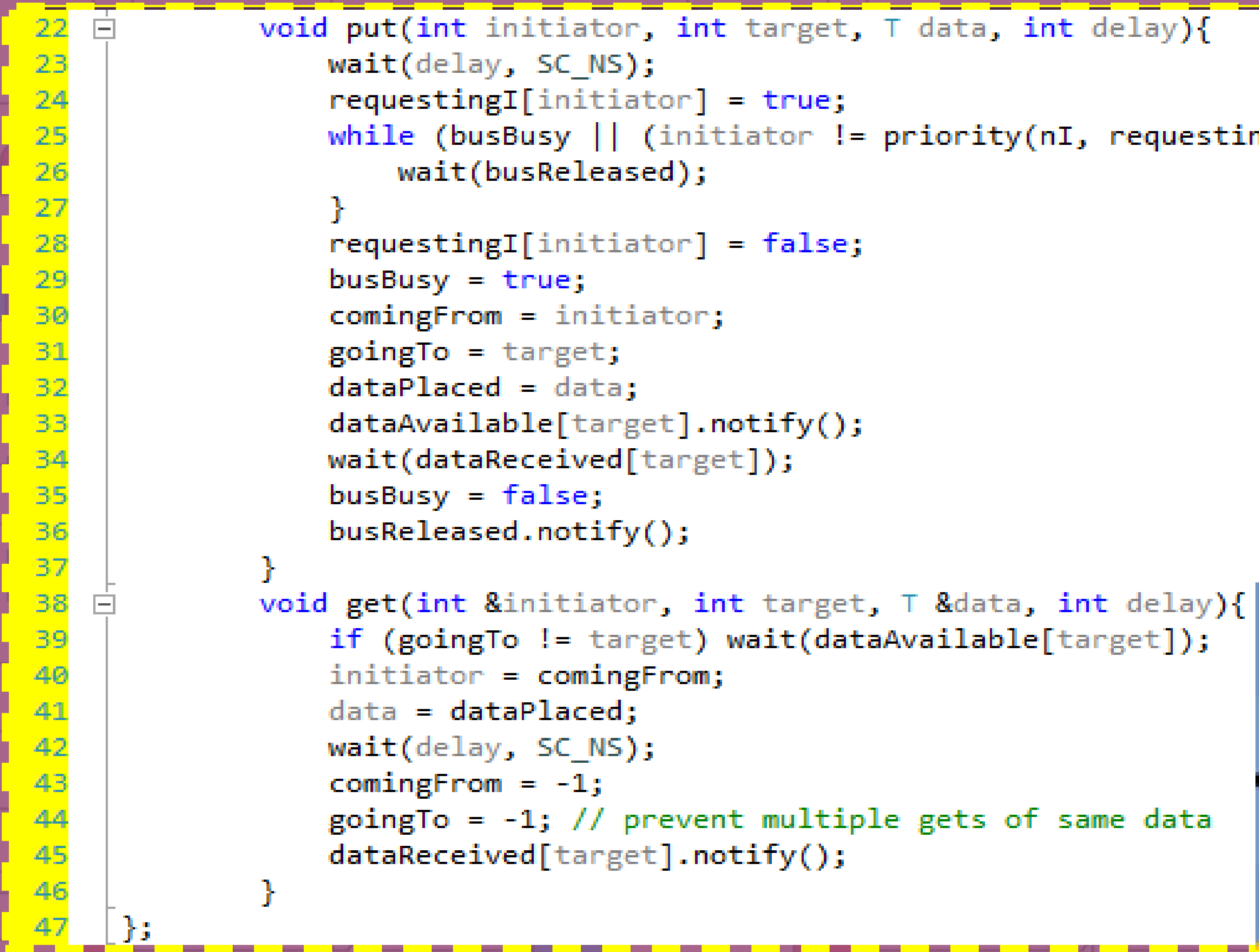
prioritySBusChannel.h

Example 5: Priority shared bus, prioritySBusChannel.h



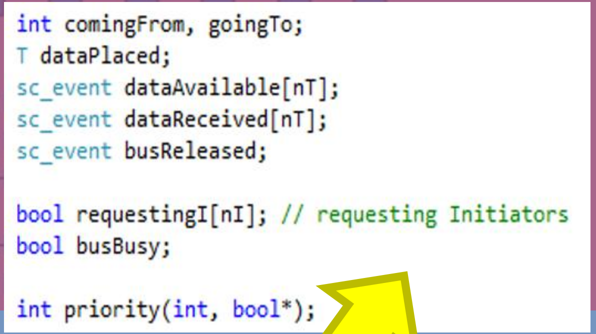
```
template <class T, int nI, int nT>
int prioritySBus<T, nI, nT>::priority(int totalCandidates, bool* thoseRequesting)
{
    // 0 has highest priority
    int found = -1;
    for (int i = totalCandidates - 1; i >= 0; i--){
        if (*(thoseRequesting + i)) found = i;
    }
    return found;
}
```

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



```
        void put(int initiator, int target, T data, int delay){
            wait(delay, SC_NS);
            requestingI[initiator] = true;
            while (busBusy || (initiator != priority(nI, requestingI))) {
                wait(busReleased);
            }
            requestingI[initiator] = false;
            busBusy = true;
            comingFrom = initiator;
            goingTo = target;
            dataPlaced = data;
            dataAvailable[target].notify();
            wait(dataReceived[target]);
            busBusy = false;
            busReleased.notify();
        }
        void get(int &initiator, int target, T &data, int delay){
            if (goingTo != target) wait(dataAvailable[target]);
            initiator = comingFrom;
            data = dataPlaced;
            wait(delay, SC_NS);
            comingFrom = -1;
            goingTo = -1; // prevent multiple gets of same data
            dataReceived[target].notify();
        }
};
```

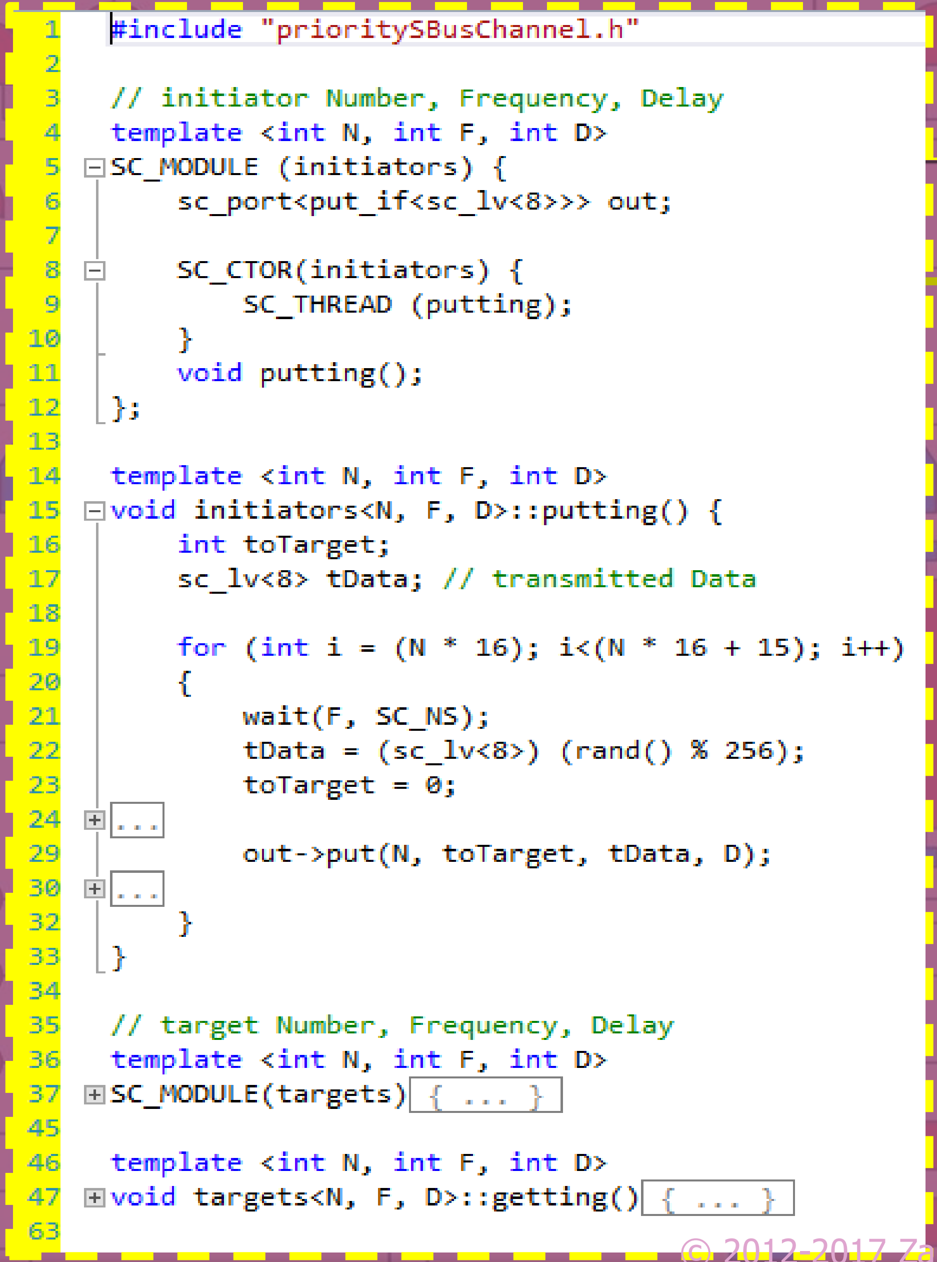
prioritySBusChannel.h



```
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*);
```

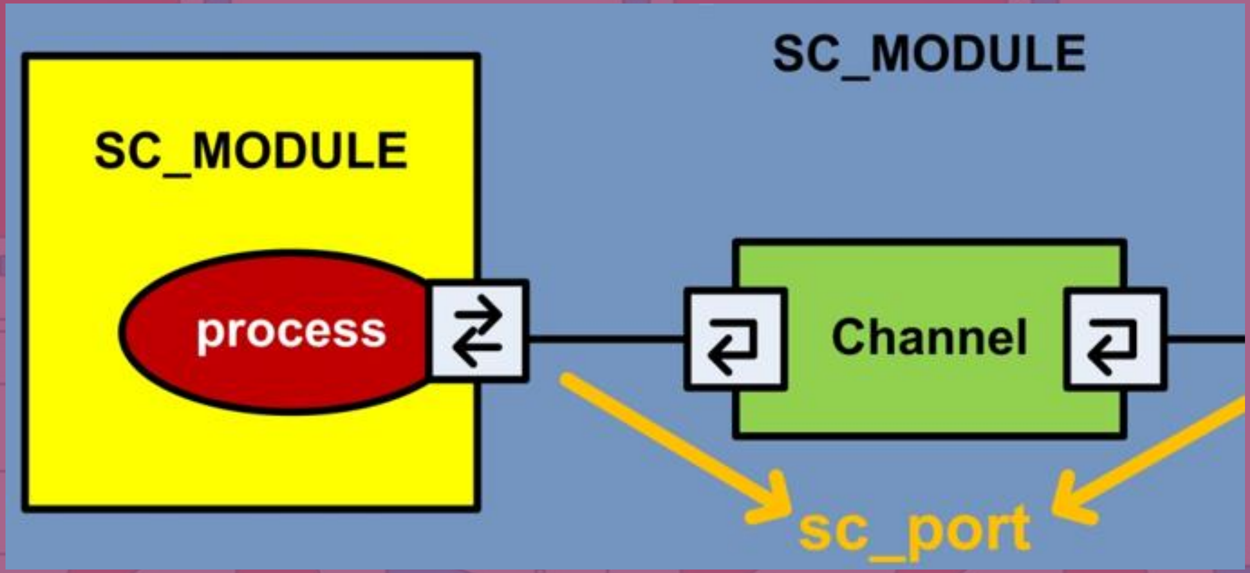
Example 5: *Priority shared bus,*

prioritySBusPutGet.h



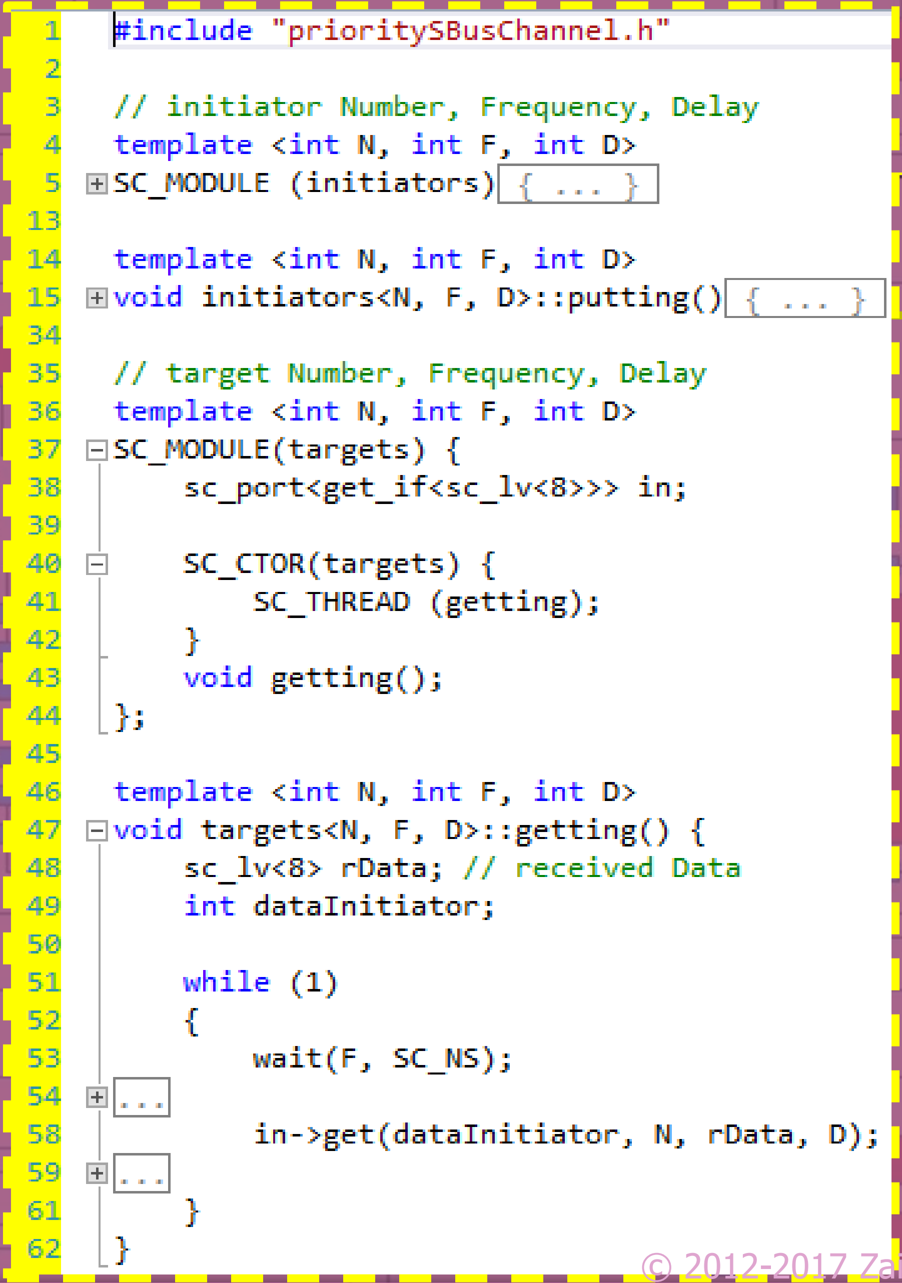
```
#include "prioritySBusChannel.h"

// initiator Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE (initiators) {
    sc_port<put_if<sc_lv<8>>> out;

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

template <int N, int F, int D>
void initiators<N, F, D>::putting() {
    int toTarget;
    sc_lv<8> tData; // transmitted Data

    for (int i = (N * 16); i<(N * 16 + 15); i++)
    {
        wait(F, SC_NS);
        tData = (sc_lv<8>) (rand() % 256);
        toTarget = 0;
...
        out->put(N, toTarget, tData, D);
...
    }
}

// target Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE(targets) { ... }

template <int N, int F, int D>
void targets<N, F, D>::getting() { ... }
```

prioritySBusPutGet.h



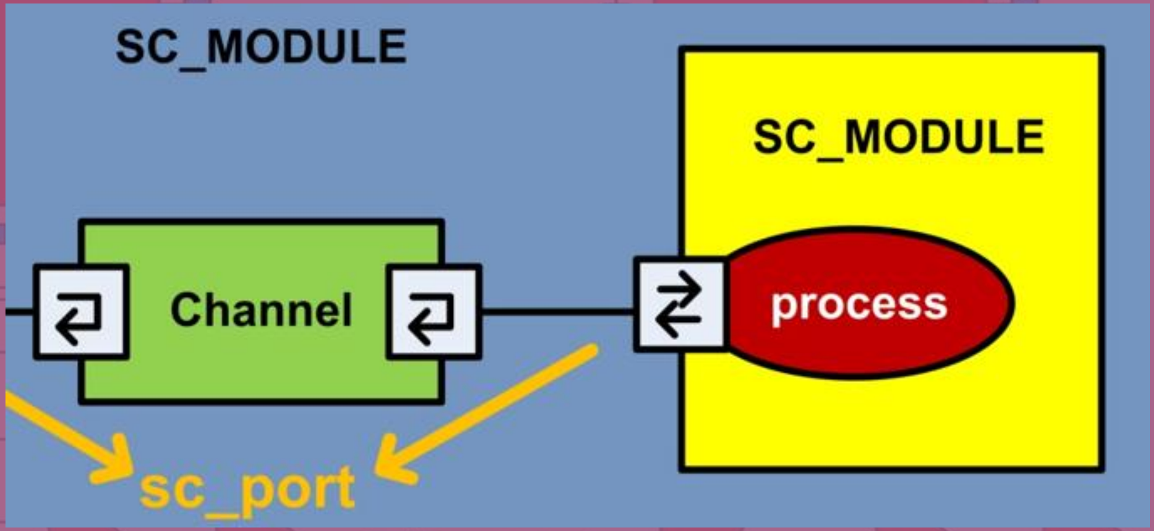
Example 5: *Priority shared bus,*

prioritySBusPutGet.h



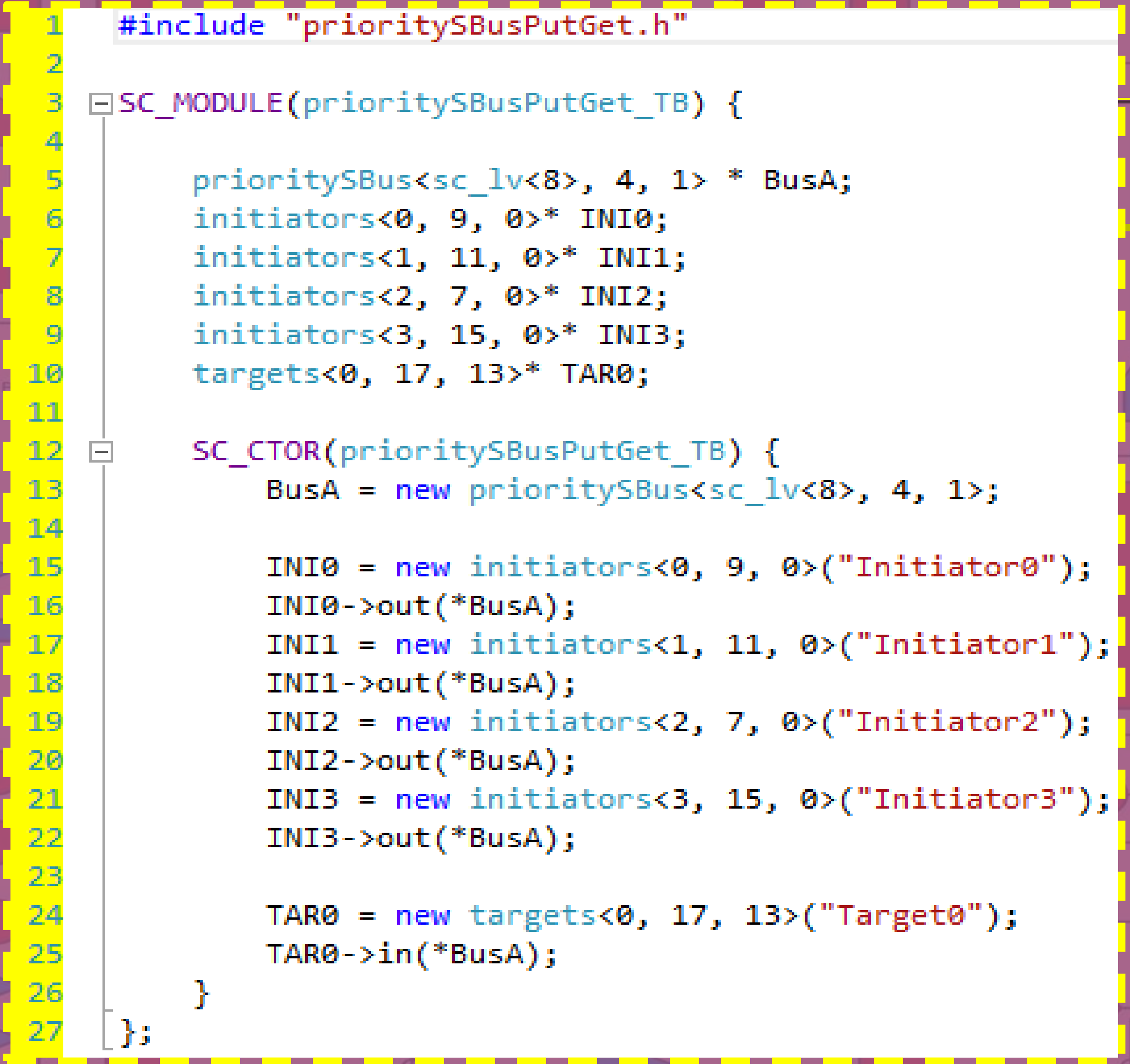
```
#include "prioritySBusChannel.h"

// initiator Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE (initiators) { ... }

template <int N, int F, int D>
void initiators<N, F, D>::putting() { ... }

// target Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE(targets) {
    sc_port<get_if<sc_lv<8>>> in;

    SC_CTOR(targets) {
        SC_THREAD (getting);
    }
    void getting();
};

template <int N, int F, int D>
void targets<N, F, D>::getting() {
    sc_lv<8> rData; // received Data
    int dataInitiator;

    while (1)
    {
        wait(F, SC_NS);
...
        in->get(dataInitiator, N, rData, D);
...
    }
}
```

prioritySBusPutGet.h



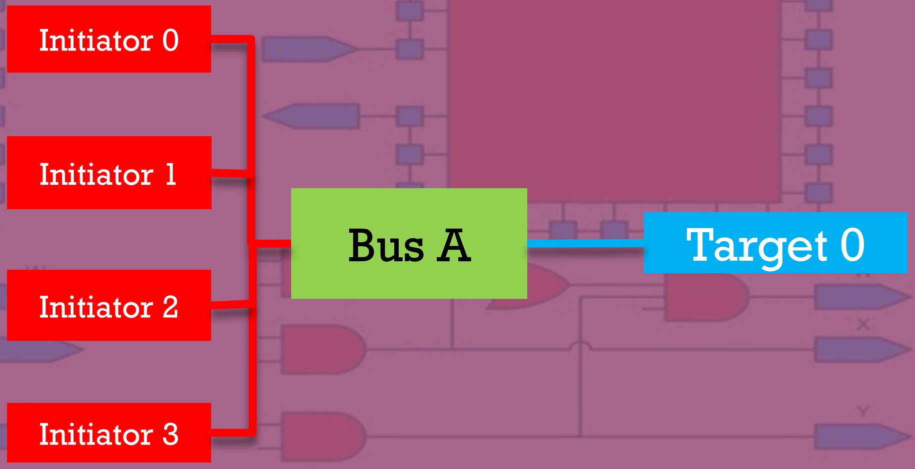
Example 5: *Priority shared bus,*

prioritySBusPutGet_TB.h



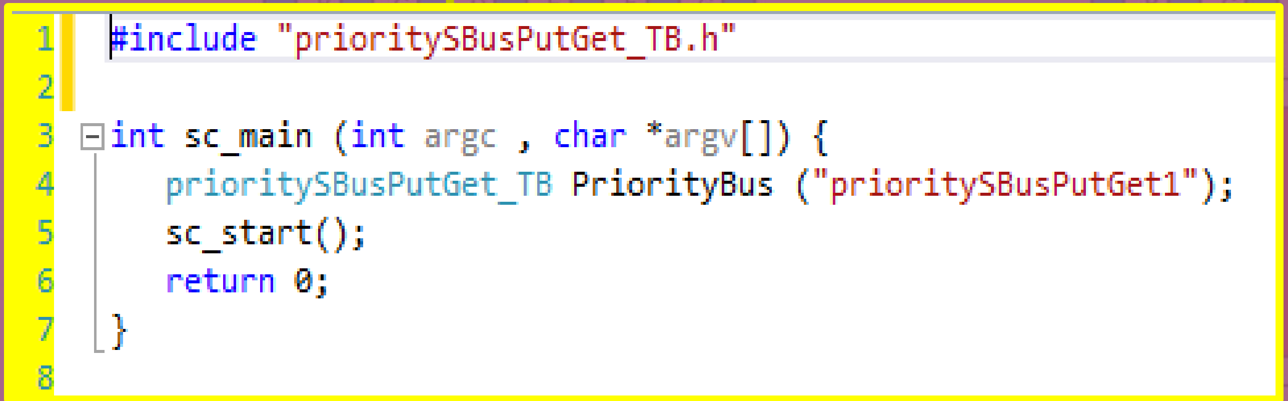
```
#include "prioritySBusPutGet.h"

SC_MODULE(prioritySBusPutGet_TB) {

    prioritySBus<sc_lv<8>, 4, 1> * BusA;
    initiators<0, 9, 0>* INI0;
    initiators<1, 11, 0>* INI1;
    initiators<2, 7, 0>* INI2;
    initiators<3, 15, 0>* INI3;
    targets<0, 17, 13>* TAR0;

    SC_CTOR(prioritySBusPutGet_TB) {
        BusA = new prioritySBus<sc_lv<8>, 4, 1>;

        INI0 = new initiators<0, 9, 0>("Initiator0");
        INI0->out(*BusA);
        INI1 = new initiators<1, 11, 0>("Initiator1");
        INI1->out(*BusA);
        INI2 = new initiators<2, 7, 0>("Initiator2");
        INI2->out(*BusA);
        INI3 = new initiators<3, 15, 0>("Initiator3");
        INI3->out(*BusA);

        TAR0 = new targets<0, 17, 13>("Target0");
        TAR0->in(*BusA);
    }
};
```

prioritySBusPutGet_TB.h



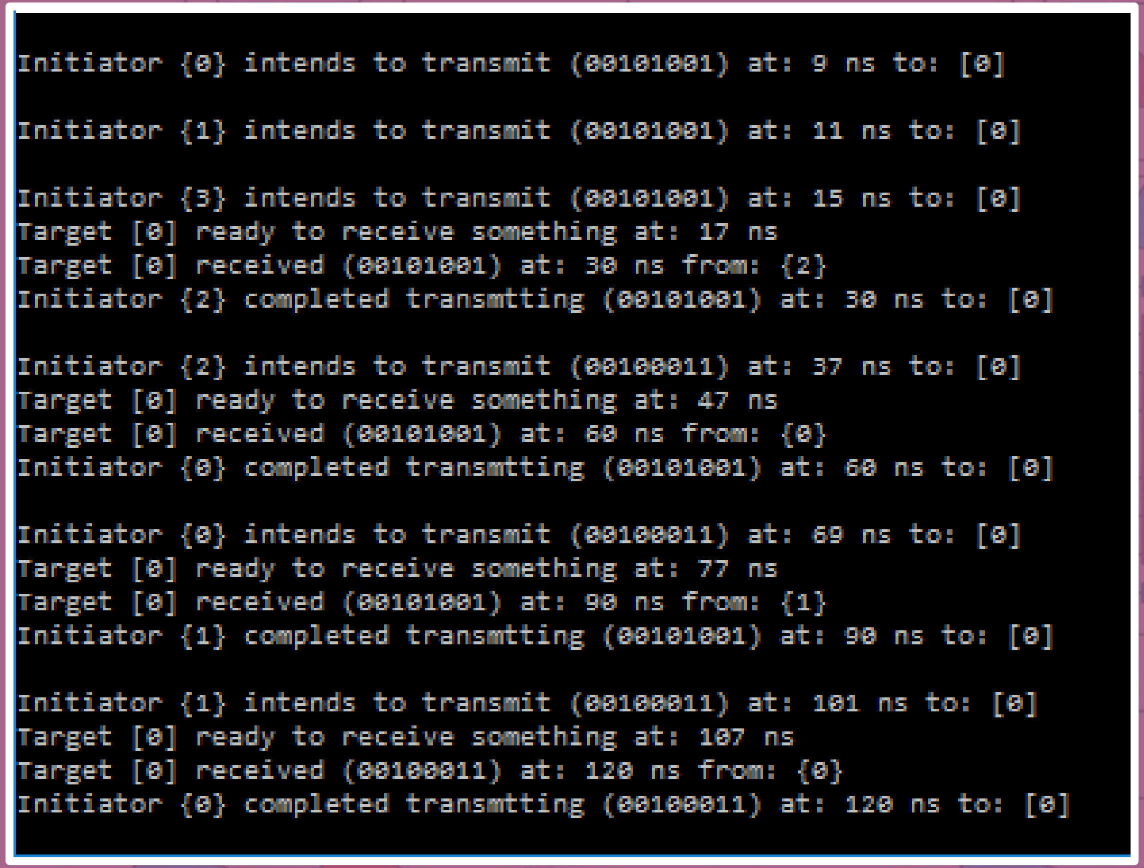
Example 5: *Priority shared bus,prioritySBusPutGet_main*

prioritySBusPutGet_main.cpp



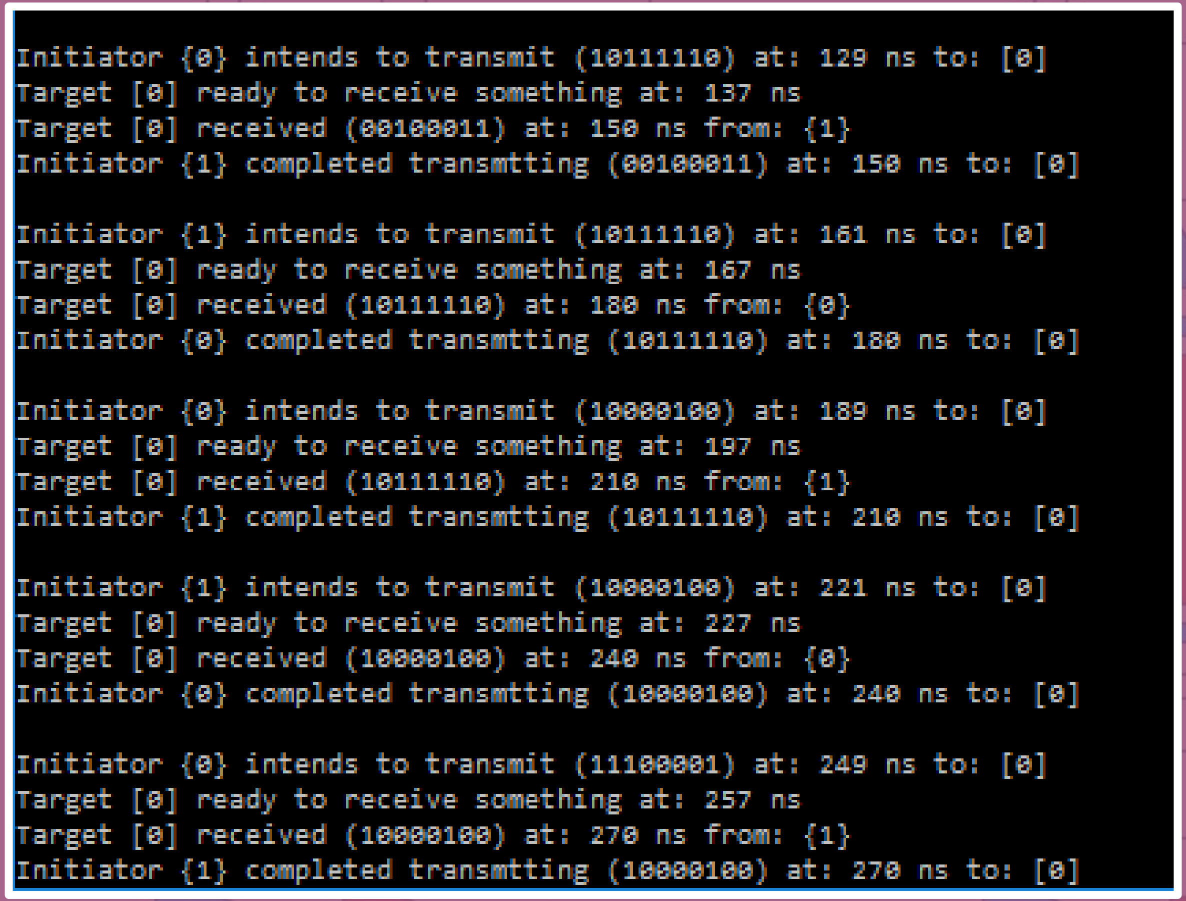
```
#include "prioritySBusPutGet_TB.h"

int sc_main (int argc , char *argv[]) {
    prioritySBusPutGet_TB PriorityBus ("prioritySBusPutGet1");
    sc_start();
    return 0;
}
```

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]
```

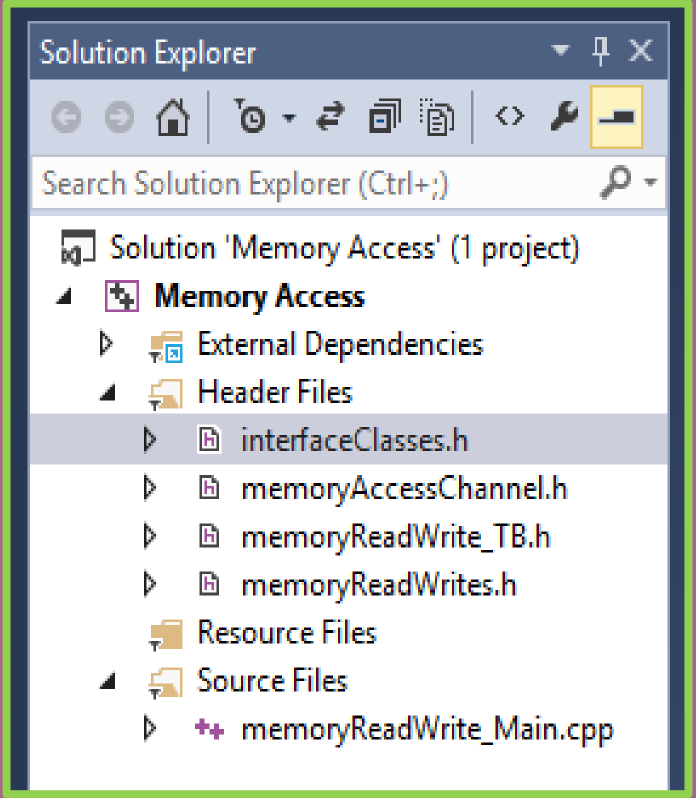
Example 5: *Priority shared bus*,*output*



```
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

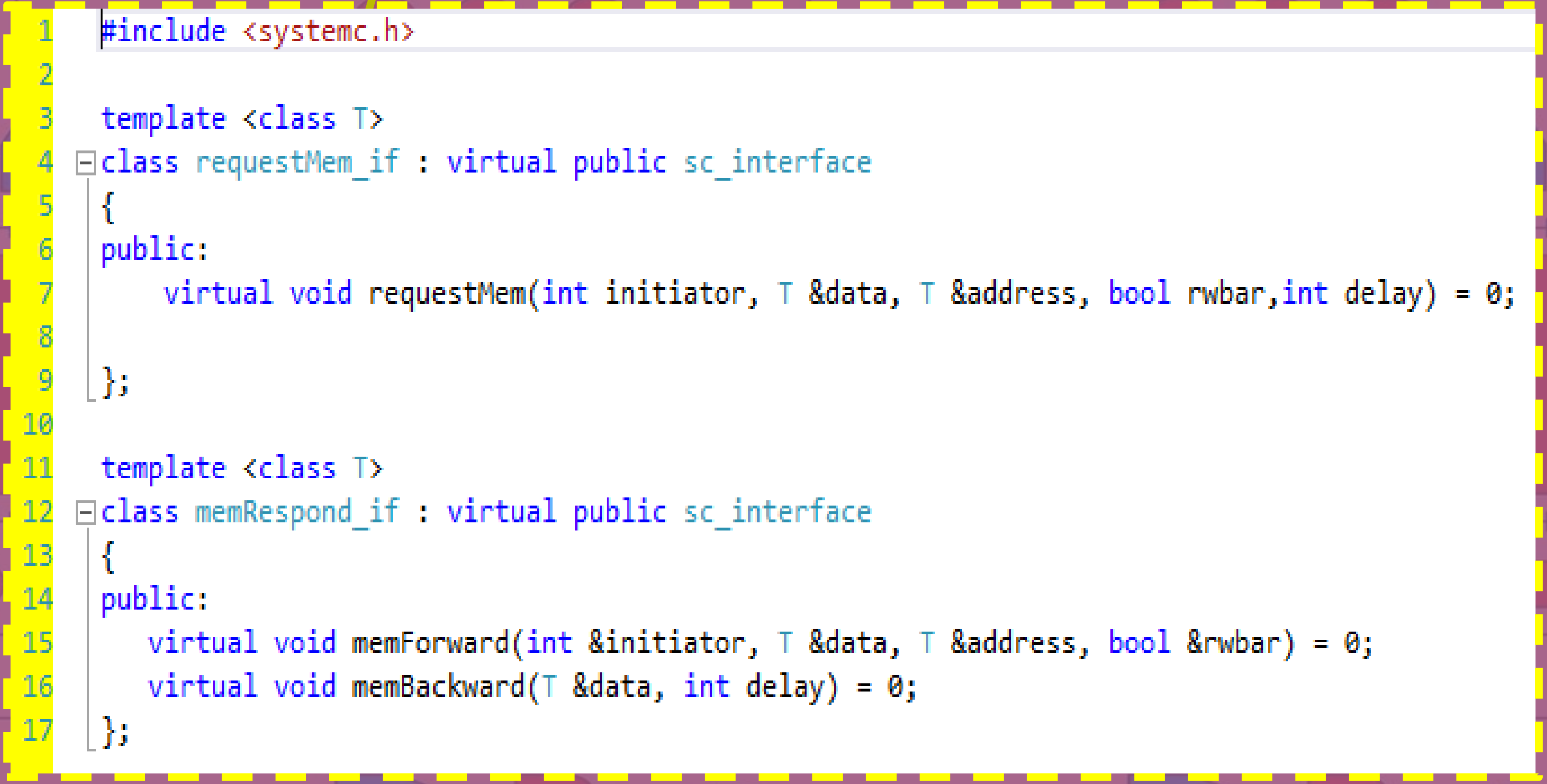

Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'Memory Access' (1 project)
◢ Memory Access
    ▷ External Dependencies
    ◢ Header Files
        ▷ interfaceClasses.h
        ▷ memoryAccessChannel.h
        ▷ memoryReadWrite_TB.h
        ▷ memoryReadWrites.h
      Resource Files
    ◢ Source Files
        ▷ memoryReadWrite_Main.cpp
```

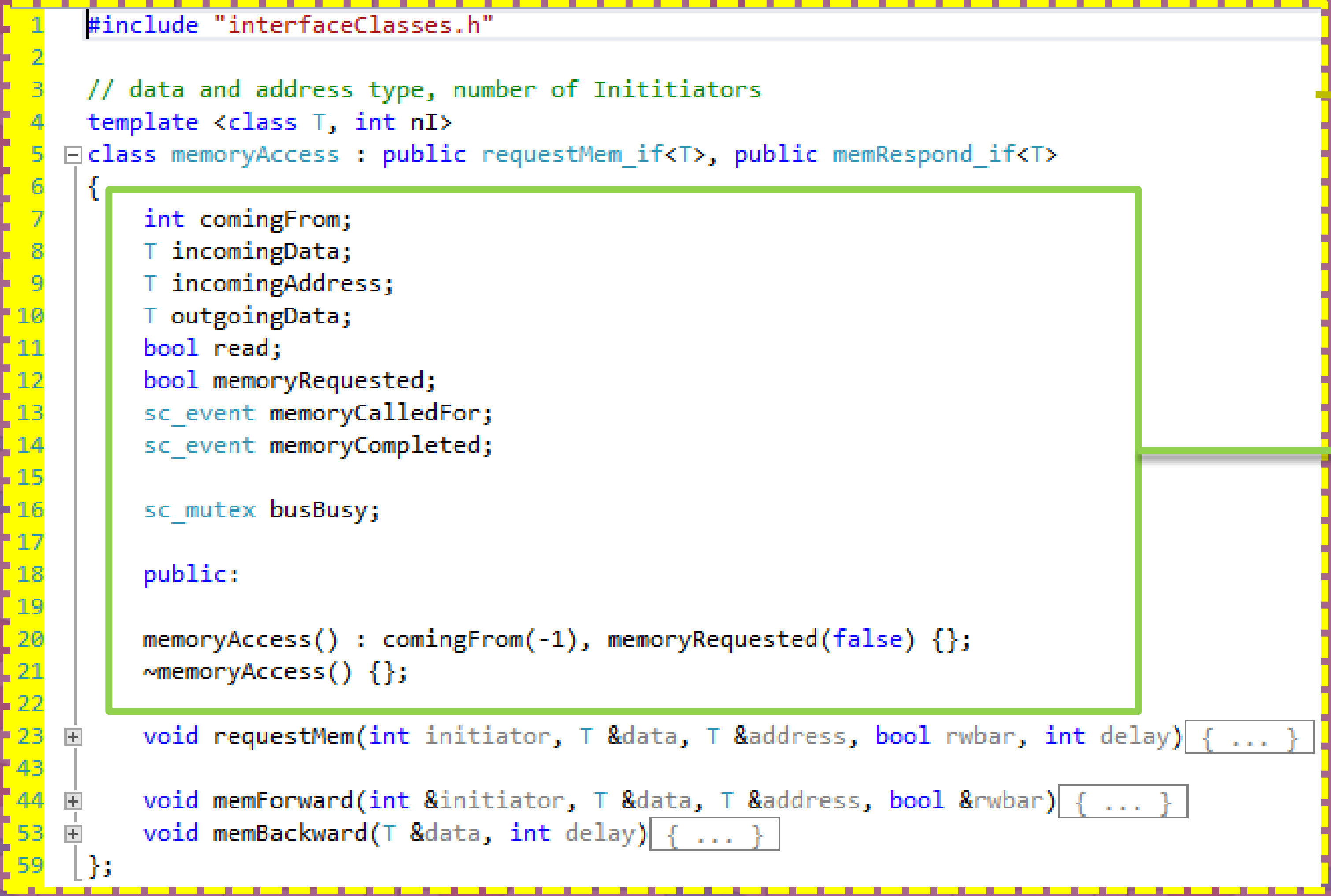
Example 6: *Memory Access*, *interfaceClasses.h*

interfaceClasses.h



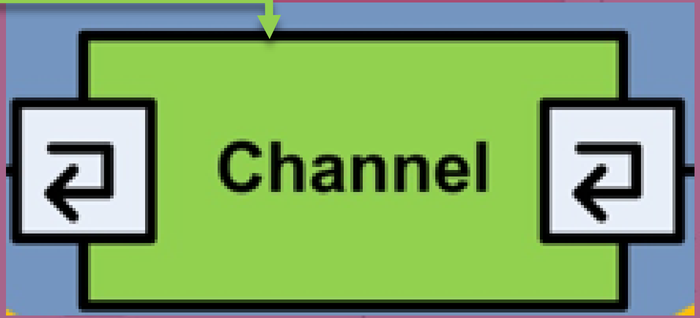
```
#include <systemc.h>

template <class T>
class requestMem_if : virtual public sc_interface
{
public:
    virtual void requestMem(int initiator, T &data, T &address, bool rwbar,int delay) = 0;

};

template <class T>
class memRespond_if : virtual public sc_interface
{
public:
   virtual void memForward(int &initiator, T &data, T &address, bool &rwbar) = 0;
   virtual void memBackward(T &data, int delay) = 0;
};
```

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

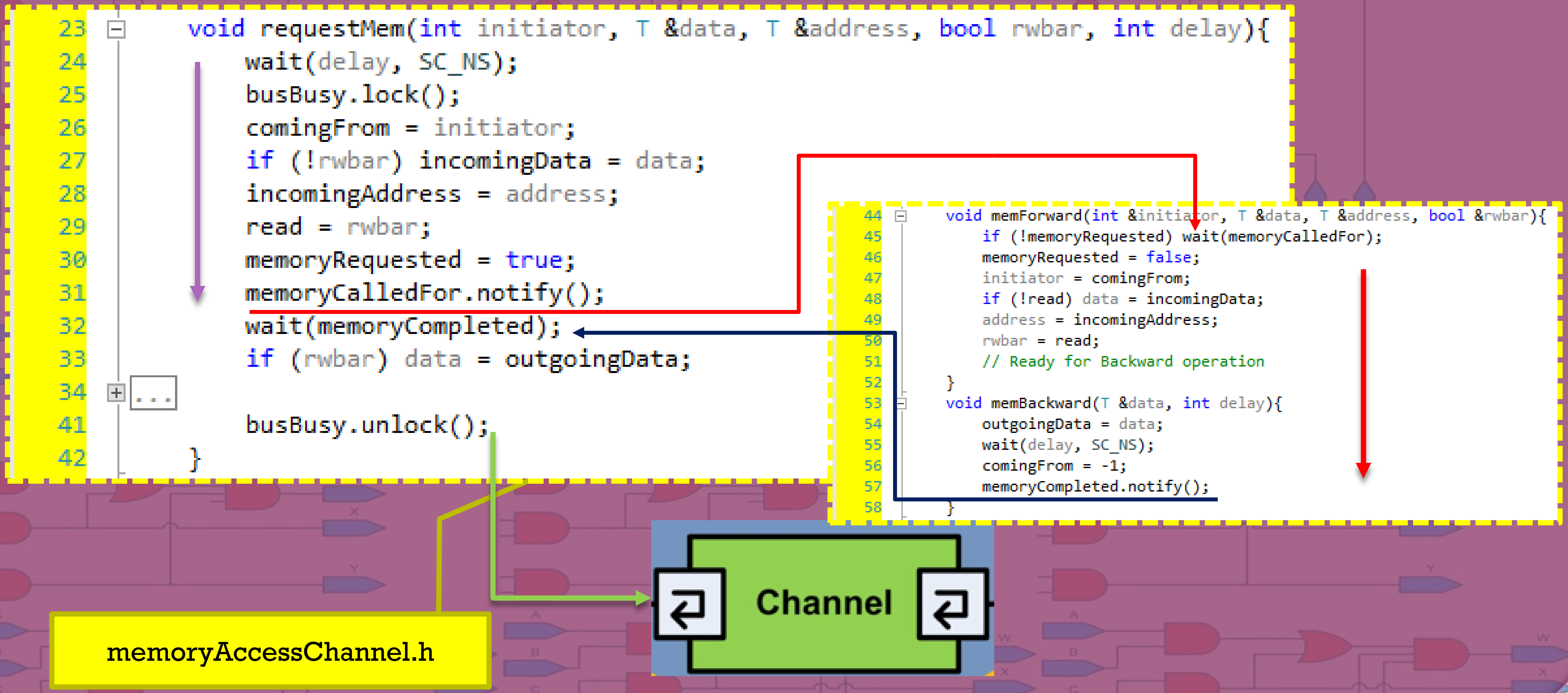


```
#include "interfaceClasses.h"

// data and address type, number of Inititiators
template <class T, int nI>
class memoryAccess : public requestMem_if<T>, public memRespond_if<T>
{
    int comingFrom;
    T incomingData;
    T incomingAddress;
    T outgoingData;
    bool read;
    bool memoryRequested;
    sc_event memoryCalledFor;
    sc_event memoryCompleted;

    sc_mutex busBusy;

    public:

    memoryAccess() : comingFrom(-1), memoryRequested(false) {};
    ~memoryAccess() {};

    void requestMem(int initiator, T &data, T &address, bool rwbar, int delay){ ... }

    void memForward(int &initiator, T &data, T &address, bool &rwbar){ ... }
    void memBackward(T &data, int delay){ ... }
};
```

memoryAccessChannel.h



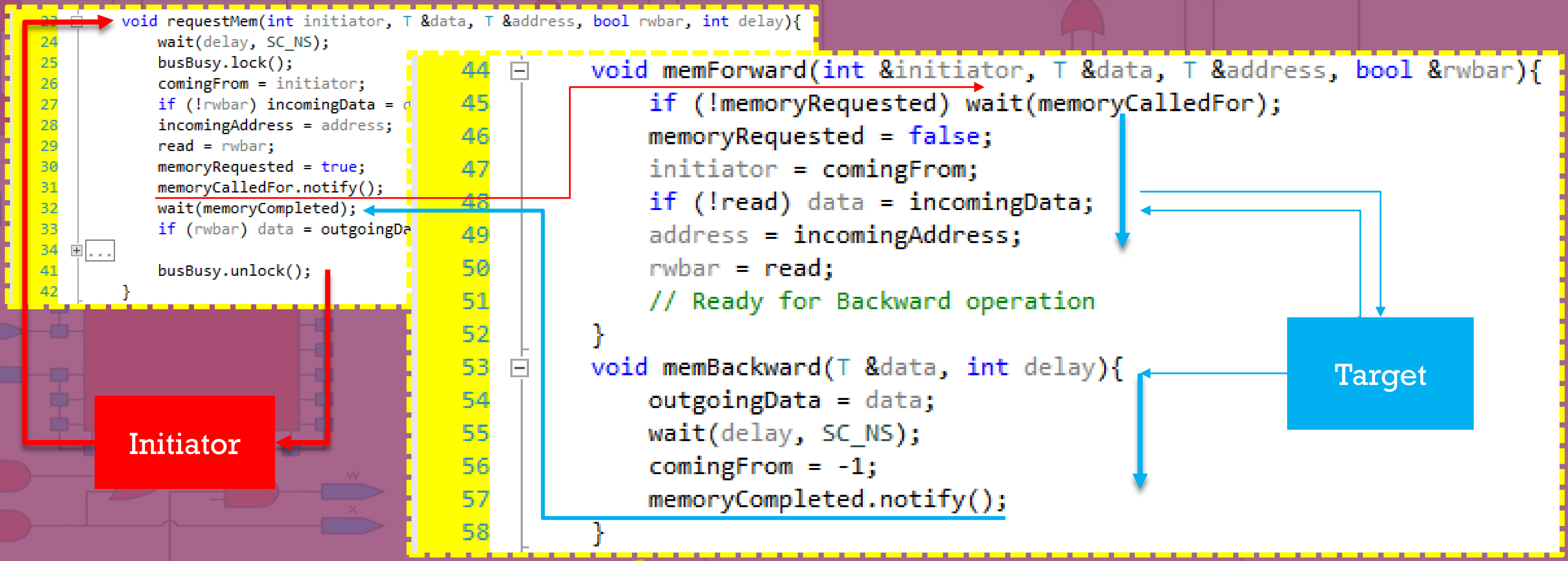
Example 6: *Memory Access*, *memoryAccessChannel.h*



```
void requestMem(int initiator, T &data, T &address, bool rwbar, int delay){
    wait(delay, SC_NS);
    busBusy.lock();
    comingFrom = initiator;
    if (!rwbar) incomingData = data;
    incomingAddress = address;
    read = rwbar;
    memoryRequested = true;
    memoryCalledFor.notify();
    wait(memoryCompleted);
    if (rwbar) data = outgoingData;
...
    busBusy.unlock();
}
```

```
void memForward(int &initiator, T &data, T &address, bool &rwbar){
    if (!memoryRequested) wait(memoryCalledFor);
    memoryRequested = false;
    initiator = comingFrom;
    if (!read) data = incomingData;
    address = incomingAddress;
    rwbar = read;
    // Ready for Backward operation
}
void memBackward(T &data, int delay){
    outgoingData = data;
    wait(delay, SC_NS);
    comingFrom = -1;
    memoryCompleted.notify();
}
```

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

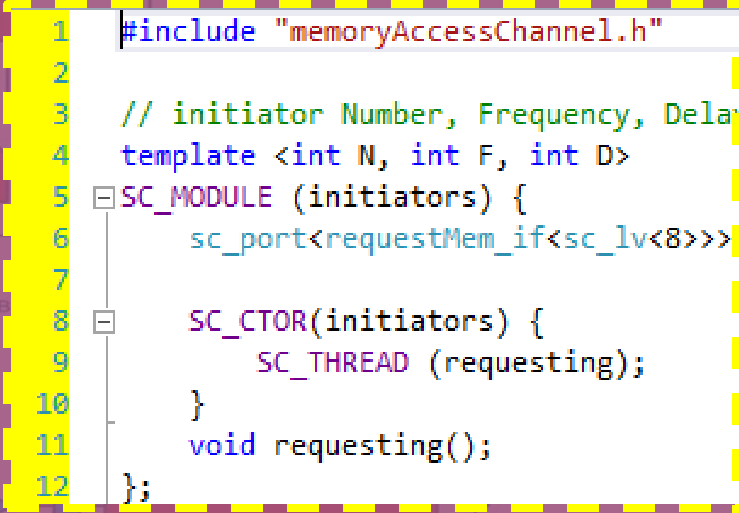


```
void requestMem(int initiator, T &data, T &address, bool rwbar, int delay){
    wait(delay, SC_NS);
    busBusy.lock();
    comingFrom = initiator;
    if (!rwbar) incomingData = d
    incomingAddress = address;
    read = rwbar;
    memoryRequested = true;
    memoryCalledFor.notify();
    wait(memoryCompleted);
    if (rwbar) data = outgoingDa
 ...
    busBusy.unlock();
}
```

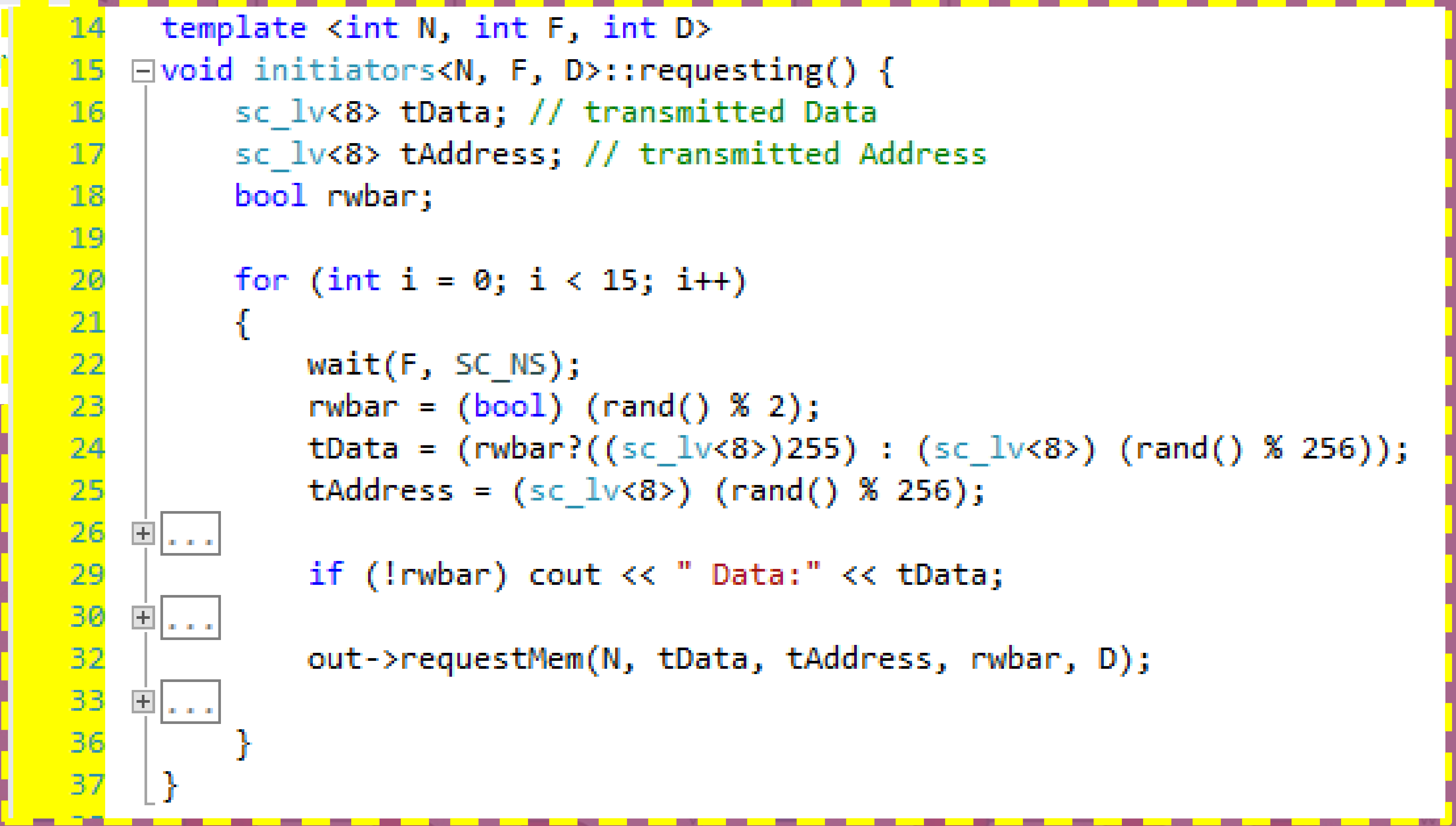
```
void memForward(int &initiator, T &data, T &address, bool &rwbar){
    if (!memoryRequested) wait(memoryCalledFor);
    memoryRequested = false;
    initiator = comingFrom;
    if (!read) data = incomingData;
    address = incomingAddress;
    rwbar = read;
    // Ready for Backward operation
}
void memBackward(T &data, int delay){
    outgoingData = data;
    wait(delay, SC_NS);
    comingFrom = -1;
    memoryCompleted.notify();
}
```

memoryAccessChannel.h

Example 6: *Memory Access,memoryReadWrites.h*



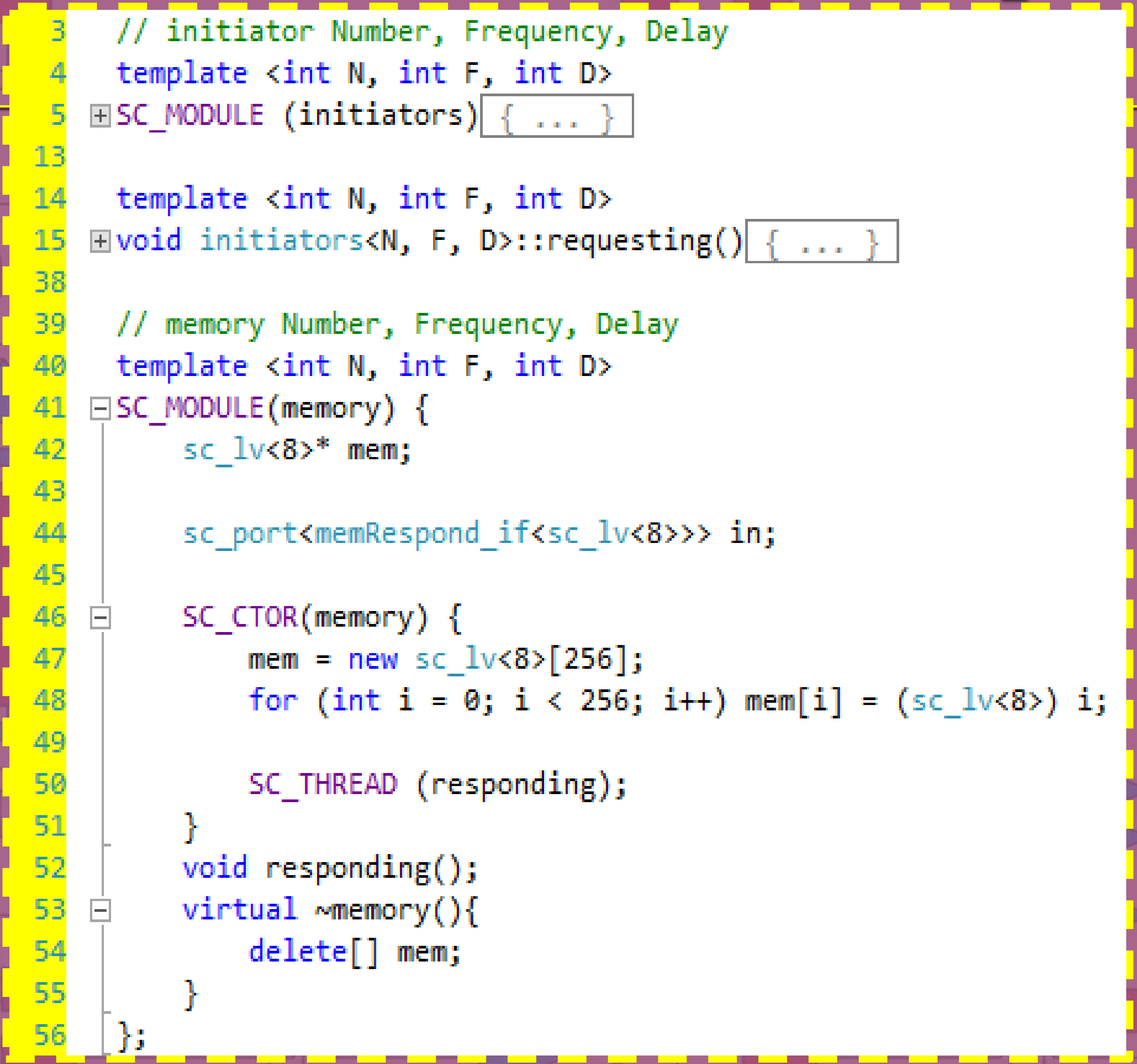
```
#include "memoryAccessChannel.h"

// initiator Number, Frequency, Dela
template <int N, int F, int D>
SC_MODULE (initiators) {
    sc_port<requestMem_if<sc_lv<8>>>

    SC_CTOR(initiators) {
        SC_THREAD (requesting);
    }
    void requesting();
};
```



```
template <int N, int F, int D>
void initiators<N, F, D>::requesting() {
    sc_lv<8> tData; // transmitted Data
    sc_lv<8> tAddress; // transmitted Address
    bool rwbar;

    for (int i = 0; i < 15; i++)
    {
        wait(F, SC_NS);
        rwbar = (bool) (rand() % 2);
        tData = (rwbar?((sc_lv<8>)255) : (sc_lv<8>) (rand() % 256));
        tAddress = (sc_lv<8>) (rand() % 256);
...
        if (!rwbar) cout << " Data:" << tData;
...
        out->requestMem(N, tData, tAddress, rwbar, D);
...
    }
}
```

memoryReadWrites.h

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

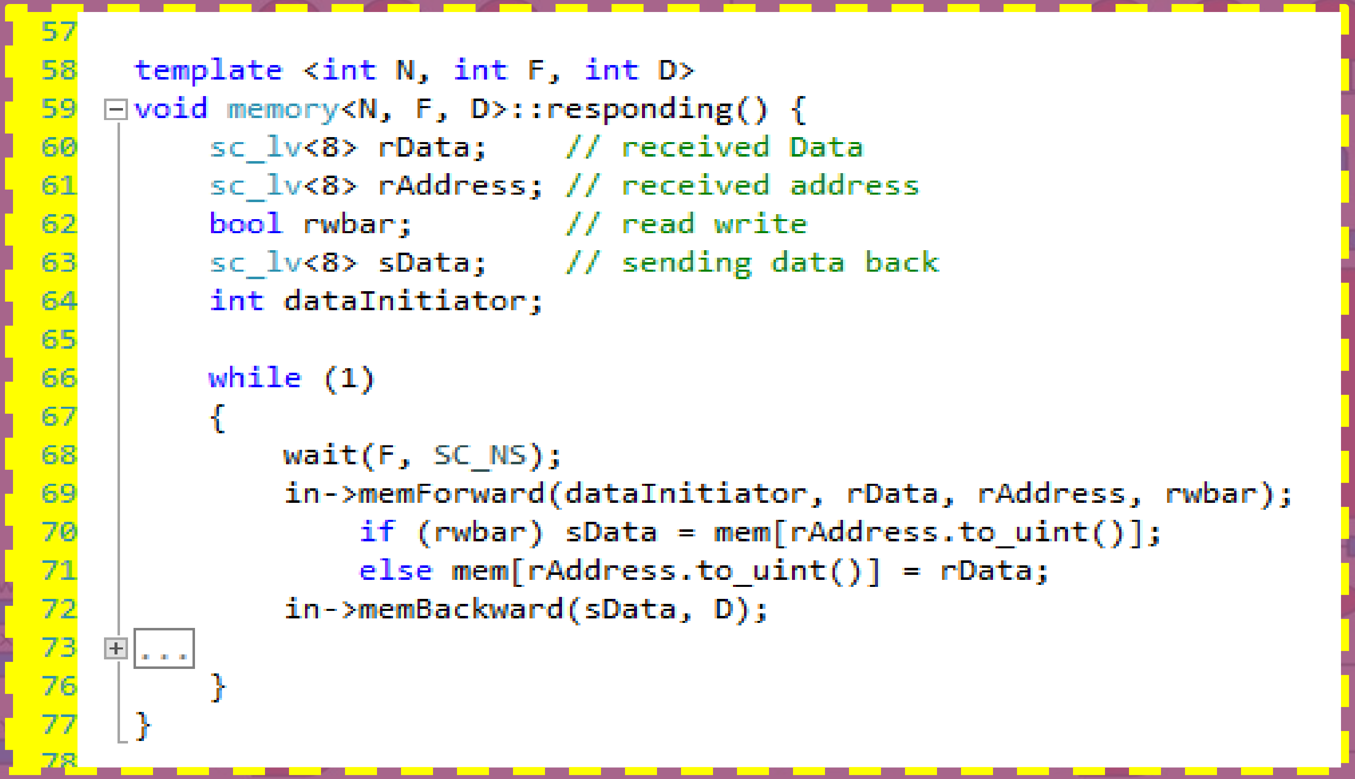


```
// initiator Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE (initiators) { ... }

template <int N, int F, int D>
void initiators<N, F, D>::requesting() { ... }

// memory Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE(memory) {
    sc_lv<8>* mem;

    sc_port<memRespond_if<sc_lv<8>>> in;

    SC_CTOR(memory) {
        mem = new sc_lv<8>[256];
        for (int i = 0; i < 256; i++) mem[i] = (sc_lv<8>) i;

        SC_THREAD (responding);
    }
    void responding();
    virtual ~memory(){
        delete[] mem;
    }
};
```

memoryReadWrites.h

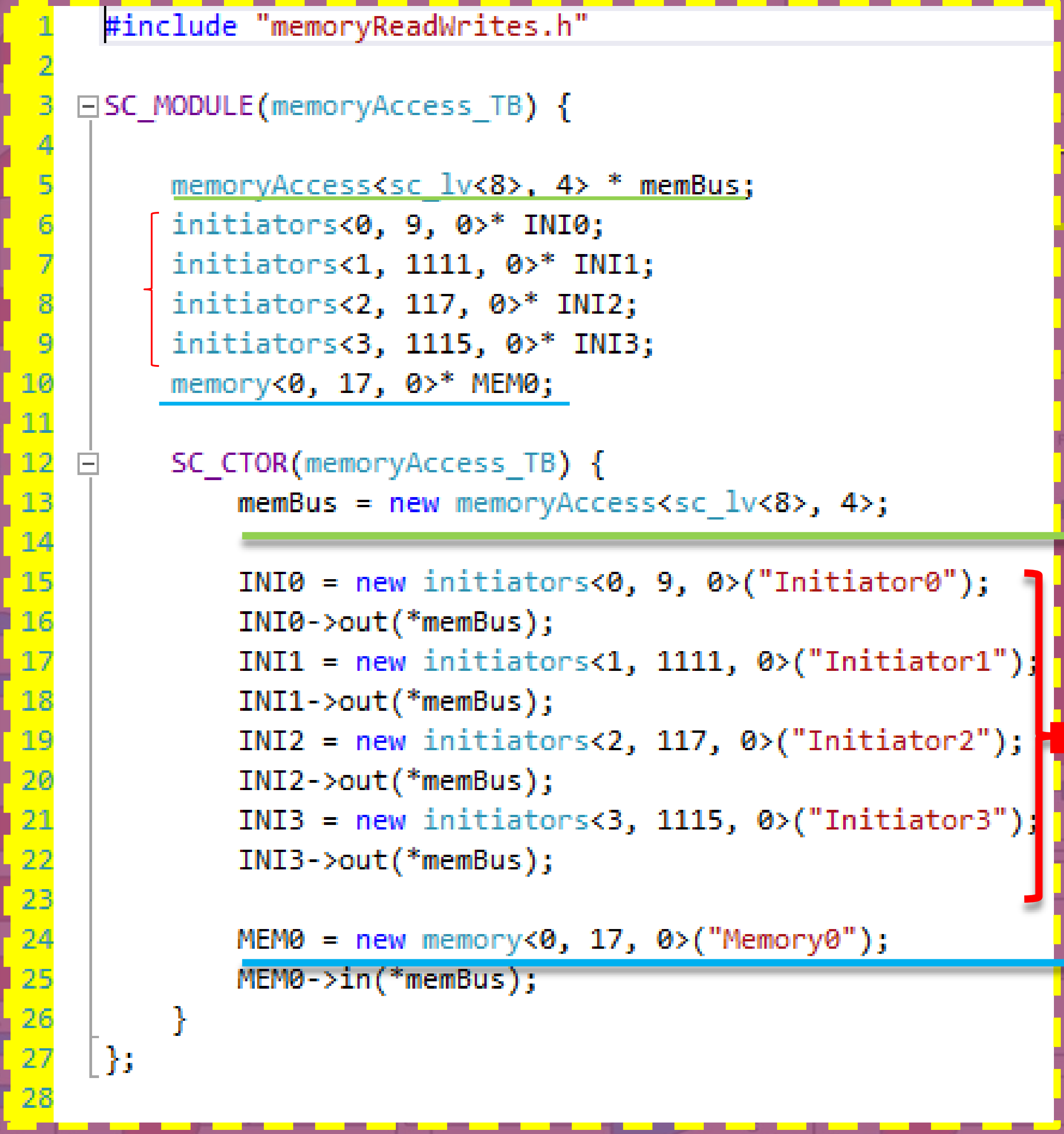
Example 6: *Memory Access*, *memoryReadWrites.h*

memoryReadWrites.h

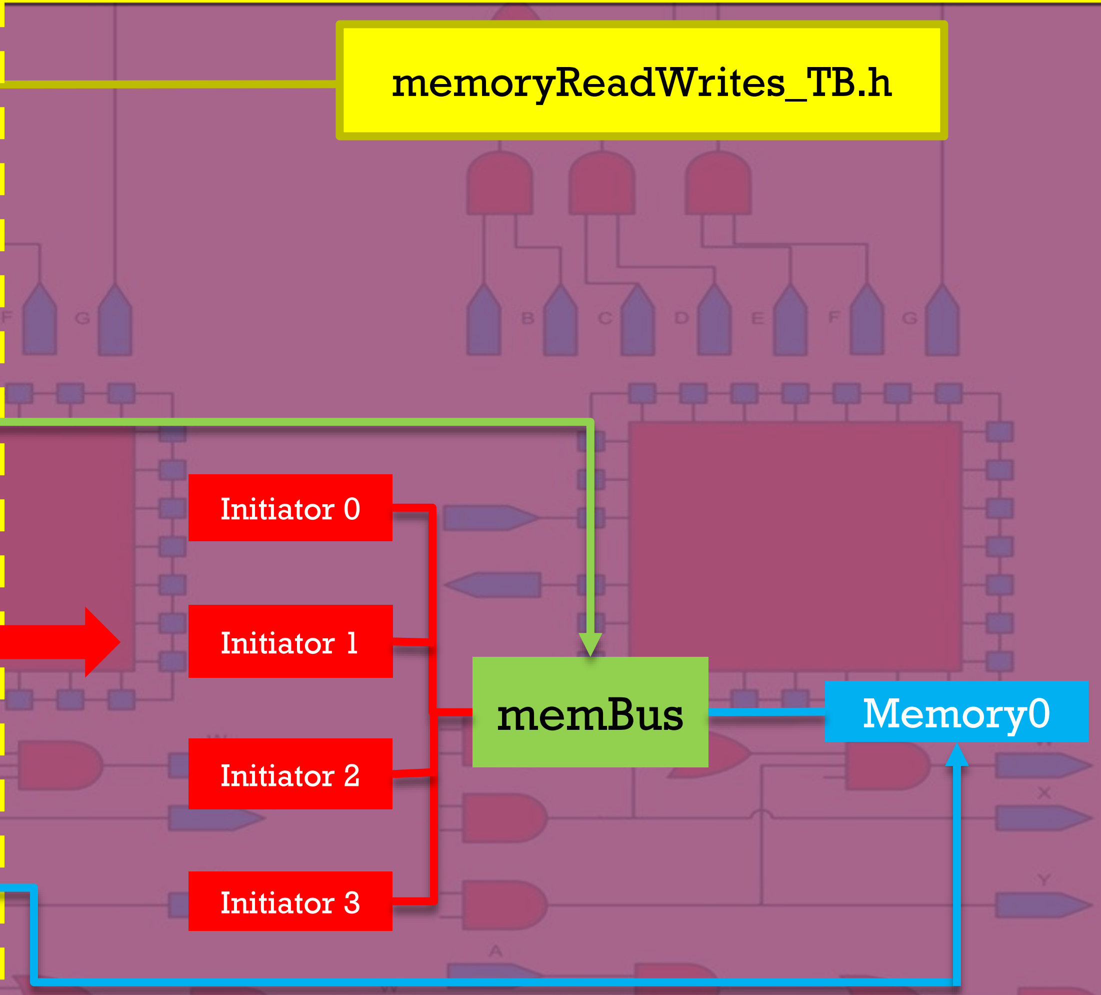


```
template <int N, int F, int D>
void memory<N, F, D>::responding() {
    sc_lv<8> rData;    // received Data
    sc_lv<8> rAddress; // received address
    bool rwbar;        // read write
    sc_lv<8> sData;    // sending data back
    int dataInitiator;

    while (1)
    {
        wait(F, SC_NS);
        in->memForward(dataInitiator, rData, rAddress, rwbar);
            if (rwbar) sData = mem[rAddress.to_uint()];
            else mem[rAddress.to_uint()] = rData;
        in->memBackward(sData, D);
...
    }
}
```

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

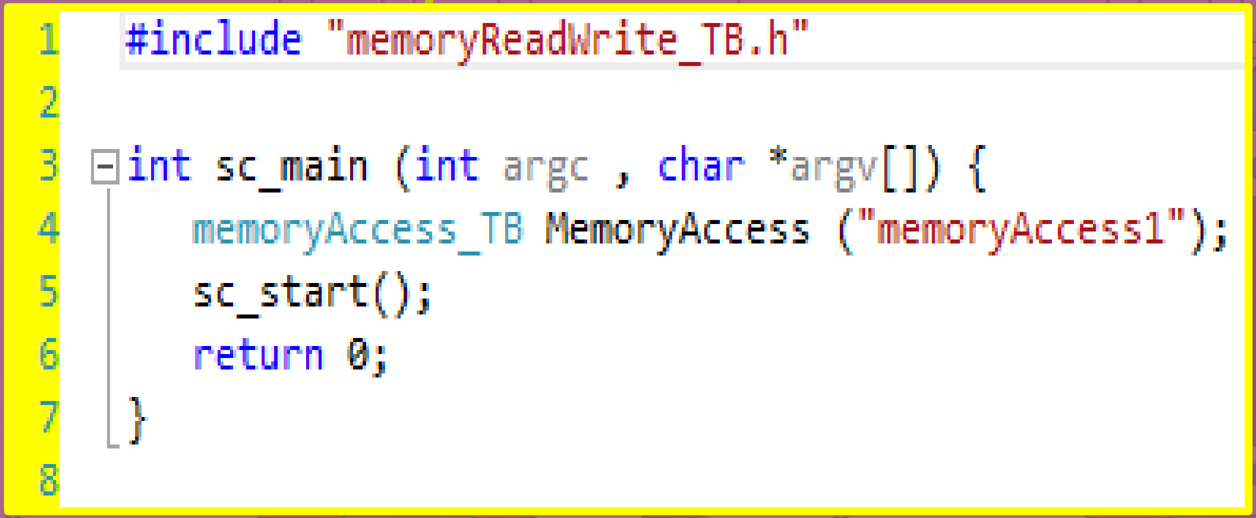


```
#include "memoryReadWrites.h"

SC_MODULE(memoryAccess_TB) {

    memoryAccess<sc_lv<8>, 4> * memBus;
    initiators<0, 9, 0>* INI0;
    initiators<1, 1111, 0>* INI1;
    initiators<2, 117, 0>* INI2;
    initiators<3, 1115, 0>* INI3;
    memory<0, 17, 0>* MEM0;

    SC_CTOR(memoryAccess_TB) {
        memBus = new memoryAccess<sc_lv<8>, 4>;

        INI0 = new initiators<0, 9, 0>("Initiator0");
        INI0->out(*memBus);
        INI1 = new initiators<1, 1111, 0>("Initiator1");
        INI1->out(*memBus);
        INI2 = new initiators<2, 117, 0>("Initiator2");
        INI2->out(*memBus);
        INI3 = new initiators<3, 1115, 0>("Initiator3");
        INI3->out(*memBus);

        MEM0 = new memory<0, 17, 0>("Memory0");
        MEM0->in(*memBus);
    }
};

```



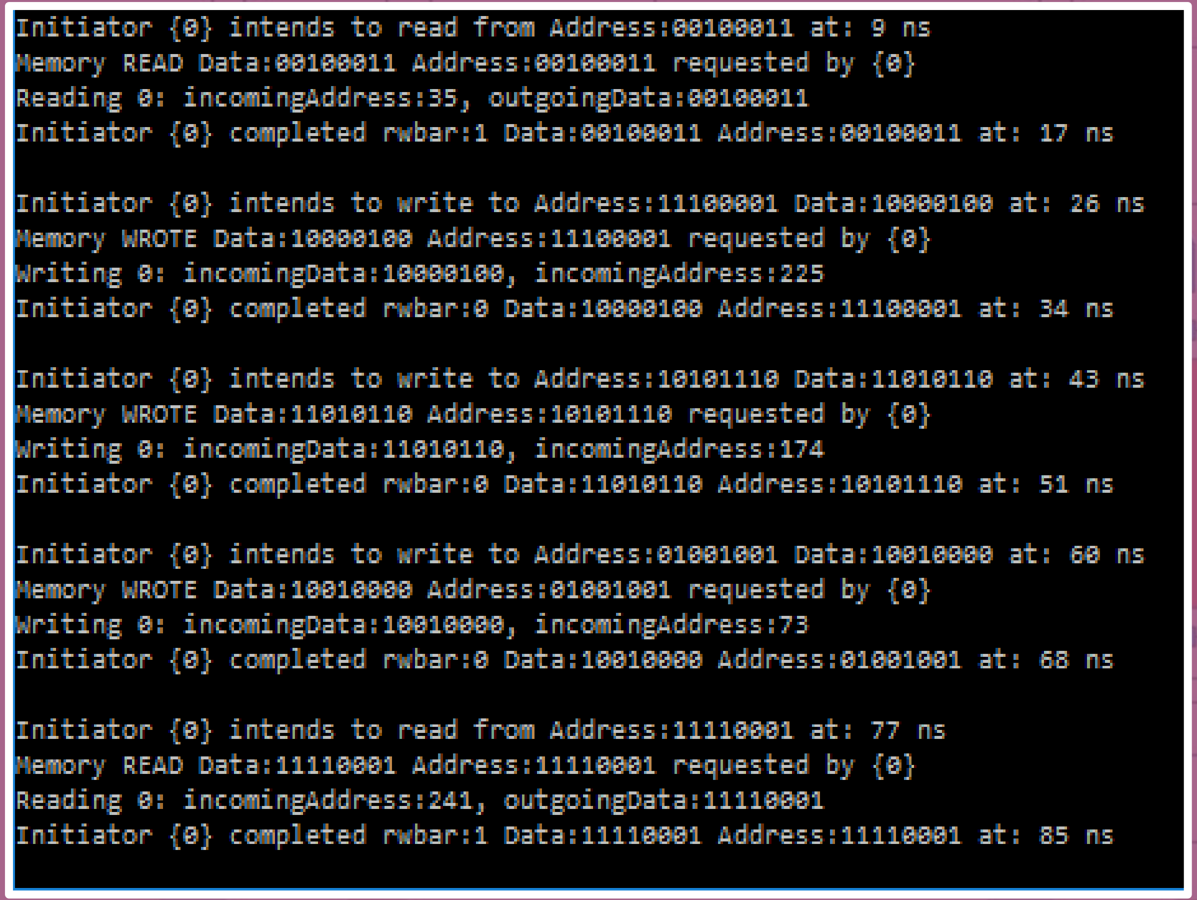
Example 6: *Memory Access*, *memoryReadWrites_main*

memoryReadWrites_main.cpp



```
#include "memoryReadWrite_TB.h"

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

```

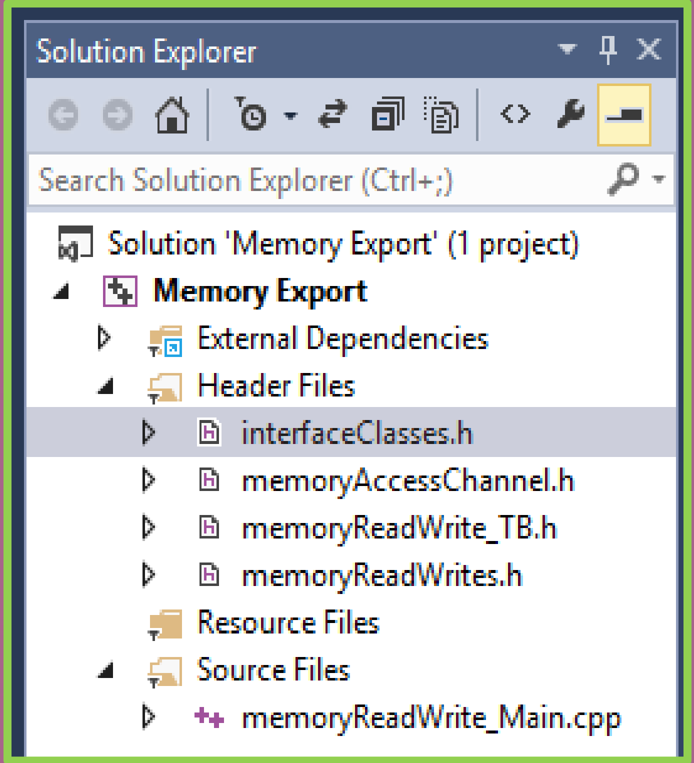
Example 6: *Memory Access*, *output*



```
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

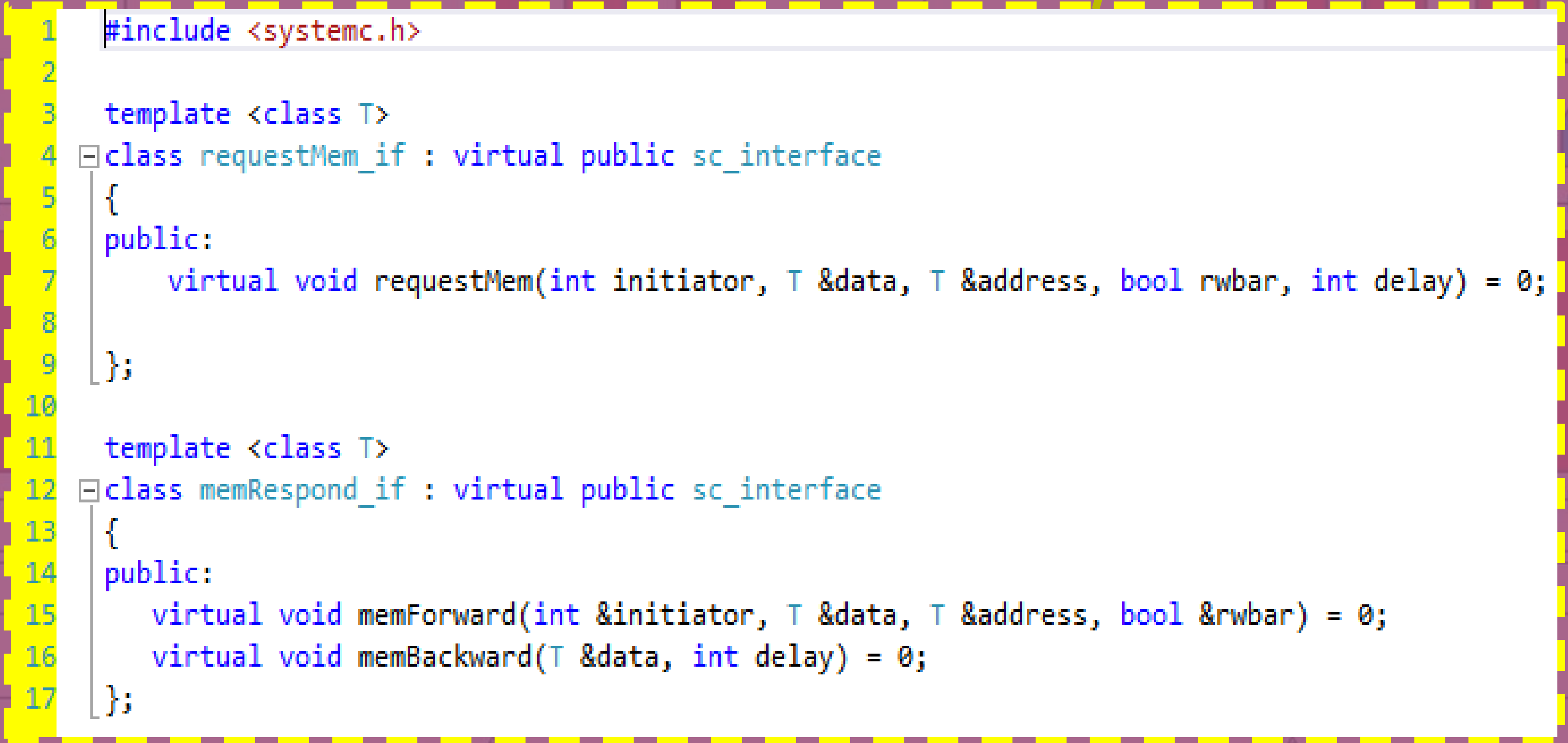
- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using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



Solution 'Memory Export' (1 project)
- **Memory Export**
  - External Dependencies
  - Header Files
    - interfaceClasses.h
    - memoryAccessChannel.h
    - memoryReadWrite_TB.h
    - memoryReadWrites.h
  - Resource Files
  - Source Files
    - memoryReadWrite_Main.cpp

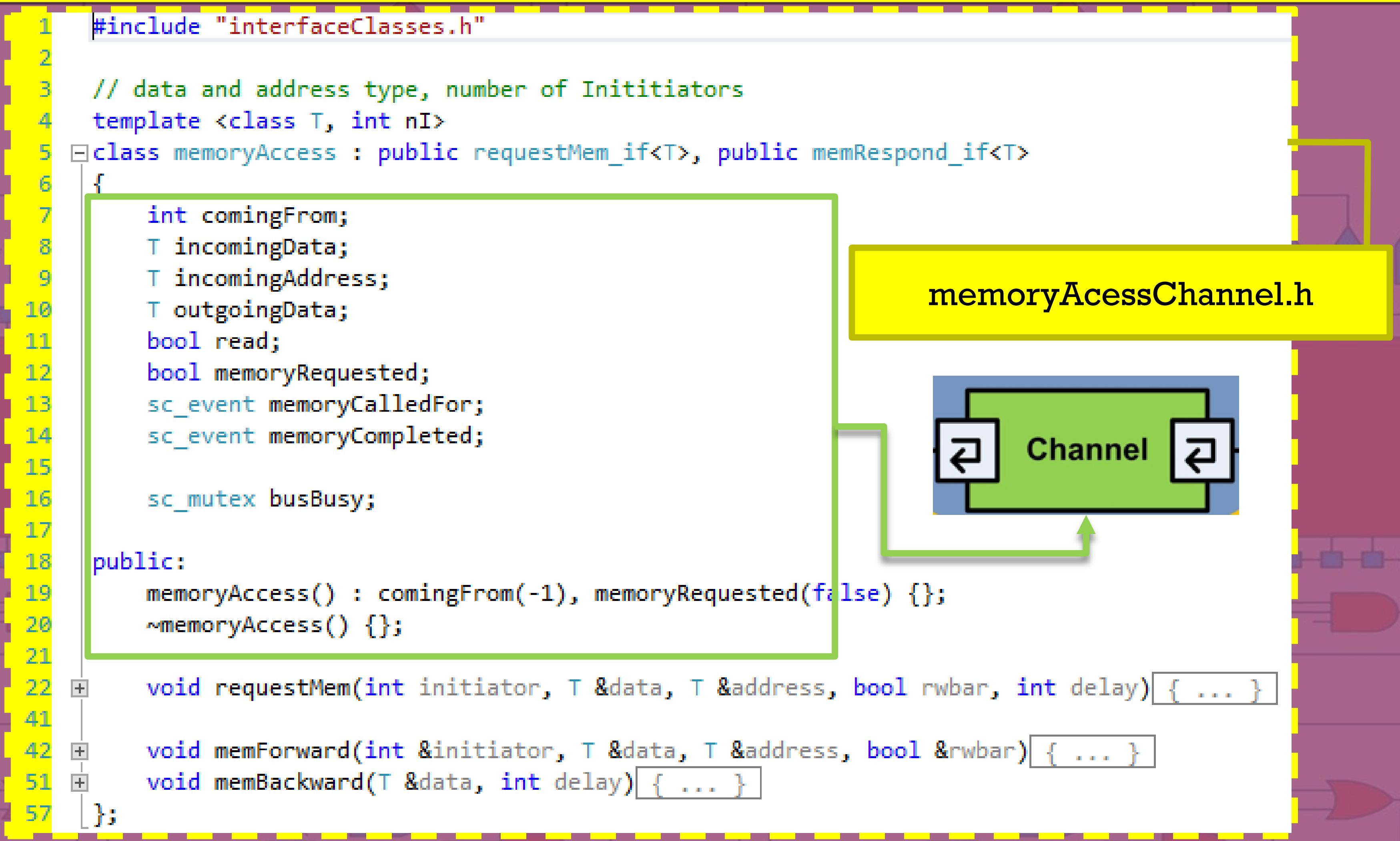
Example 7: *Memory Export*, *interfaceClasses.h*

interfaceClasses.h



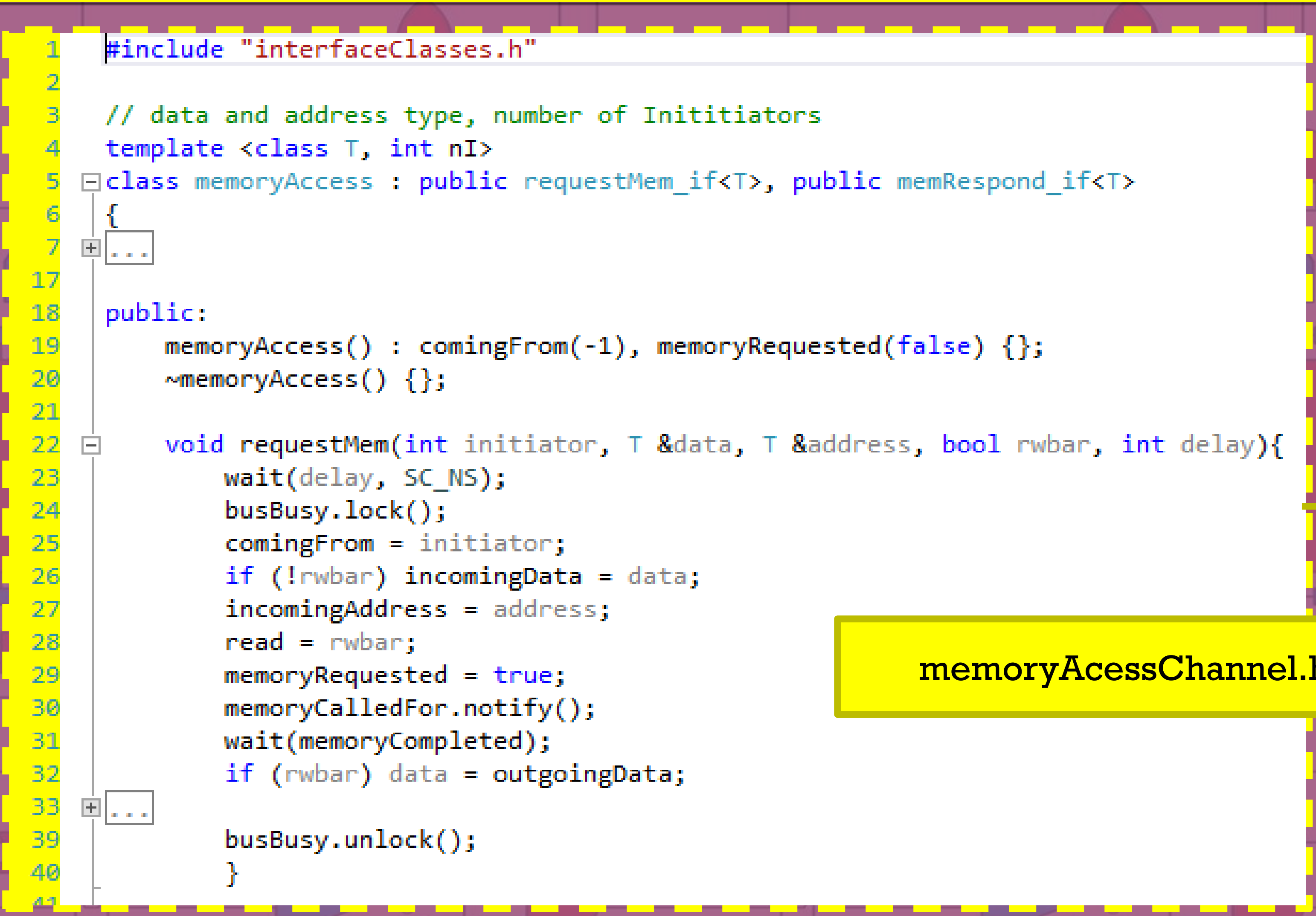
```
#include <systemc.h>

template <class T>
class requestMem_if : virtual public sc_interface
{
public:
    virtual void requestMem(int initiator, T &data, T &address, bool rwbar, int delay) = 0;

};

template <class T>
class memRespond_if : virtual public sc_interface
{
public:
   virtual void memForward(int &initiator, T &data, T &address, bool &rwbar) = 0;
   virtual void memBackward(T &data, int delay) = 0;
};
```

Example 7: *Memory Export*, memoryAcessChannel.h



```
#include "interfaceClasses.h"

// data and address type, number of Inititiators
template <class T, int nI>
class memoryAccess : public requestMem_if<T>, public memRespond_if<T>
{
    int comingFrom;
    T incomingData;
    T incomingAddress;
    T outgoingData;
    bool read;
    bool memoryRequested;
    sc_event memoryCalledFor;
    sc_event memoryCompleted;

    sc_mutex busBusy;

public:
    memoryAccess() : comingFrom(-1), memoryRequested(false) {};
    ~memoryAccess() {};

    void requestMem(int initiator, T &data, T &address, bool rwbar, int delay){ ... }

    void memForward(int &initiator, T &data, T &address, bool &rwbar){ ... }
    void memBackward(T &data, int delay){ ... }
};
```

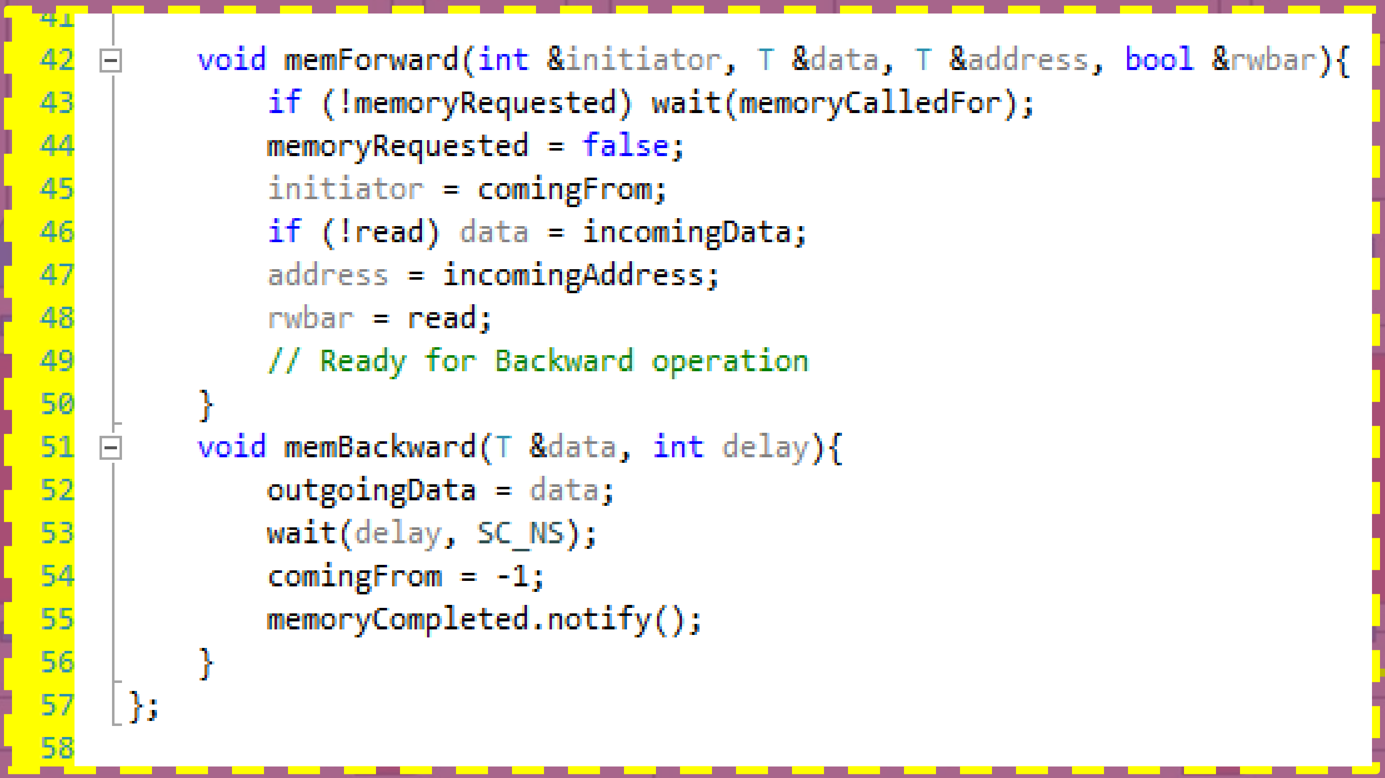
Example 7: *Memory Export*, *memoryAcessChannel.h*



```
#include "interfaceClasses.h"

// data and address type, number of Inititiators
template <class T, int nI>
class memoryAccess : public requestMem_if<T>, public memRespond_if<T>
{
...

public:
    memoryAccess() : comingFrom(-1), memoryRequested(false) {};
    ~memoryAccess() {};

    void requestMem(int initiator, T &data, T &address, bool rwbar, int delay){
        wait(delay, SC_NS);
        busBusy.lock();
        comingFrom = initiator;
        if (!rwbar) incomingData = data;
        incomingAddress = address;
        read = rwbar;
        memoryRequested = true;
        memoryCalledFor.notify();
        wait(memoryCompleted);
        if (rwbar) data = outgoingData;
...
        busBusy.unlock();
        }
```

memoryAcessChannel.h

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

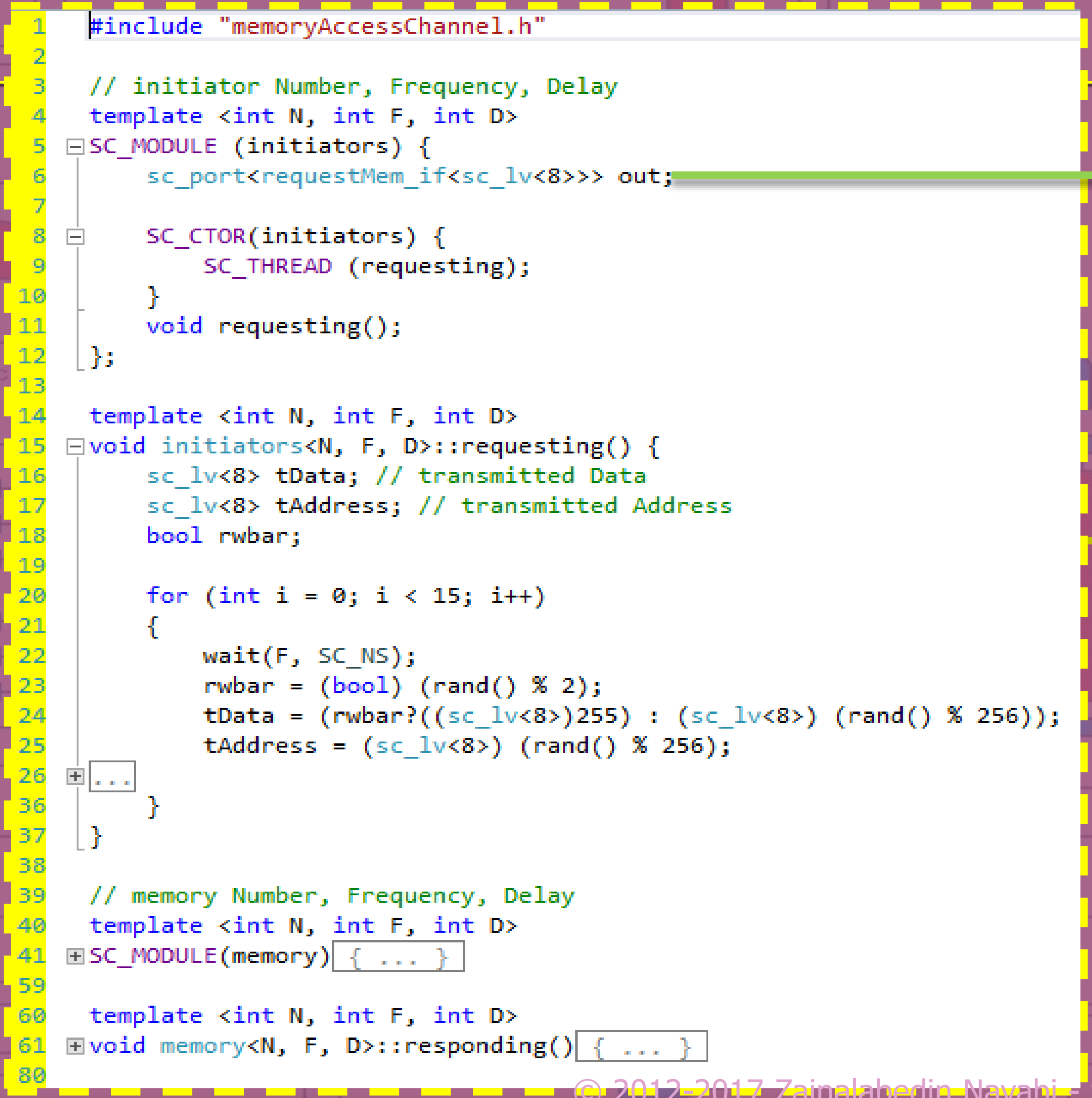


```
    void memForward(int &initiator, T &data, T &address, bool &rwbar){
        if (!memoryRequested) wait(memoryCalledFor);
        memoryRequested = false;
        initiator = comingFrom;
        if (!read) data = incomingData;
        address = incomingAddress;
        rwbar = read;
        // Ready for Backward operation
    }
    void memBackward(T &data, int delay){
        outgoingData = data;
        wait(delay, SC_NS);
        comingFrom = -1;
        memoryCompleted.notify();
    }
};
```

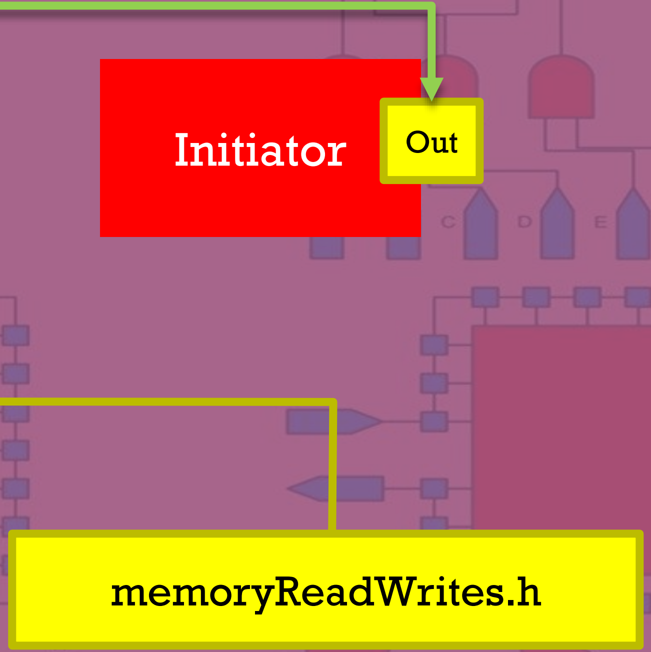
memoryAcessChannel.h

Example 7: *Memory Export,*

memoryReadWrites.h

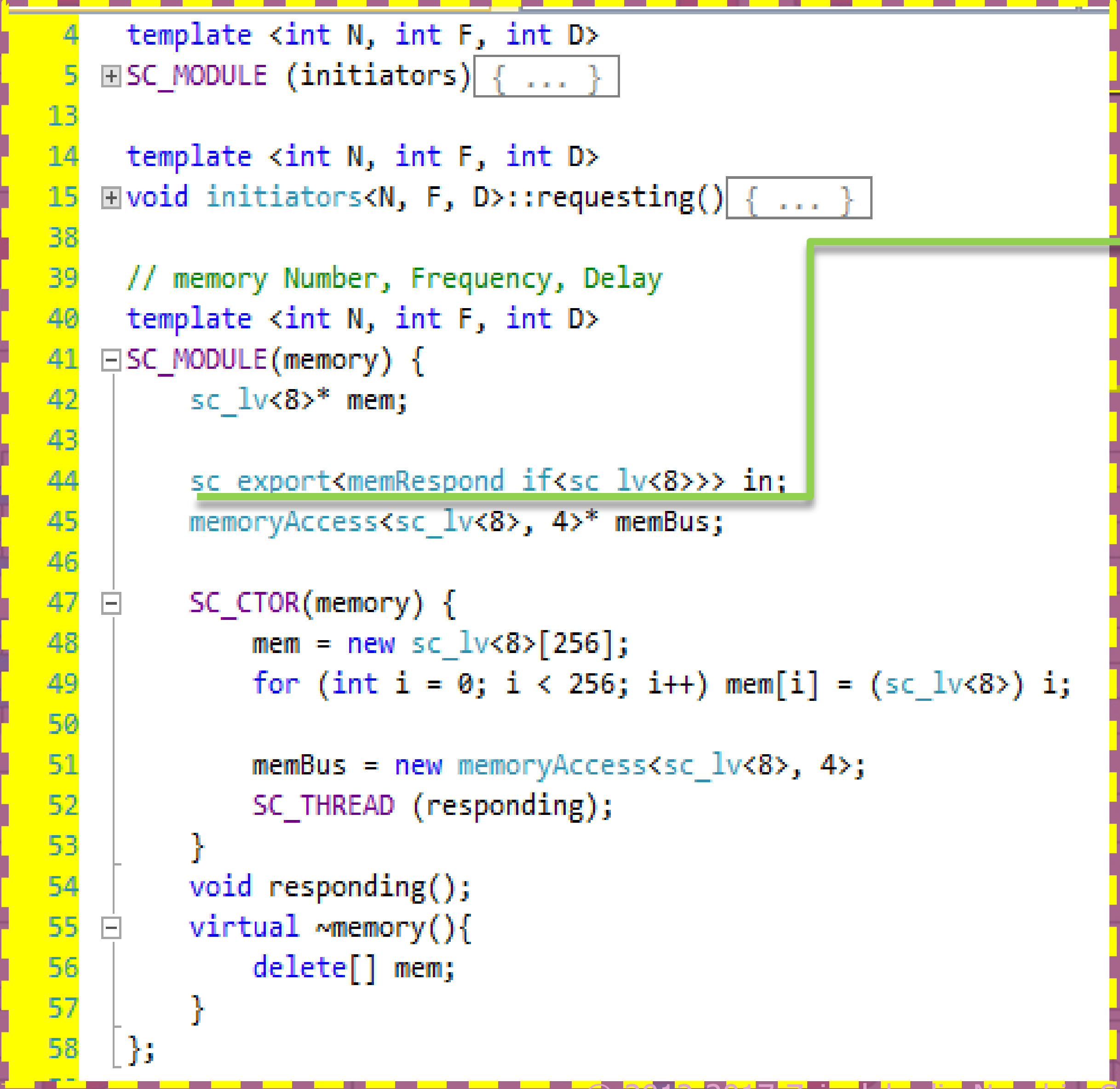


```
#include "memoryAccessChannel.h"

// initiator Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE (initiators) {
    sc_port<requestMem_if<sc_lv<8>>> out;

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

template <int N, int F, int D>
void initiators<N, F, D>::requesting() {
    sc_lv<8> tData; // transmitted Data
    sc_lv<8> tAddress; // transmitted Address
    bool rwbar;

    for (int i = 0; i < 15; i++)
    {
        wait(F, SC_NS);
        rwbar = (bool) (rand() % 2);
        tData = (rwbar?((sc_lv<8>)255) : (sc_lv<8>) (rand() % 256));
        tAddress = (sc_lv<8>) (rand() % 256);
...
    }
}

// memory Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE(memory) { ... }

template <int N, int F, int D>
void memory<N, F, D>::responding() { ... }
```

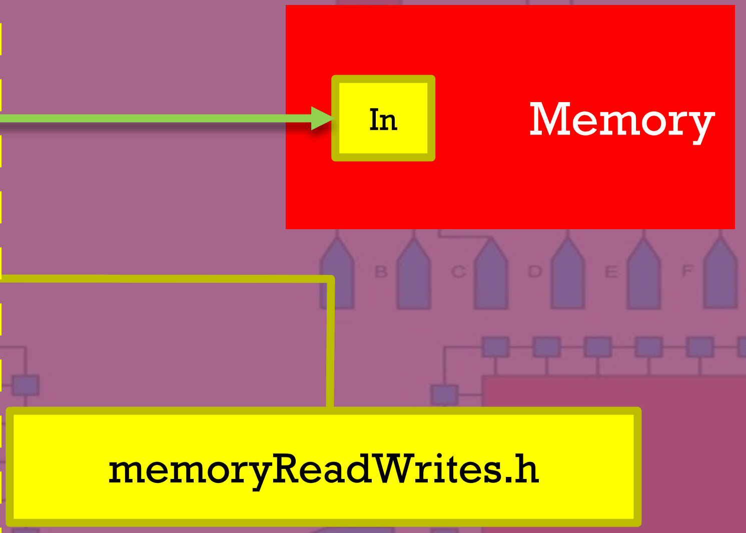


Example 7: *Memory Export,*

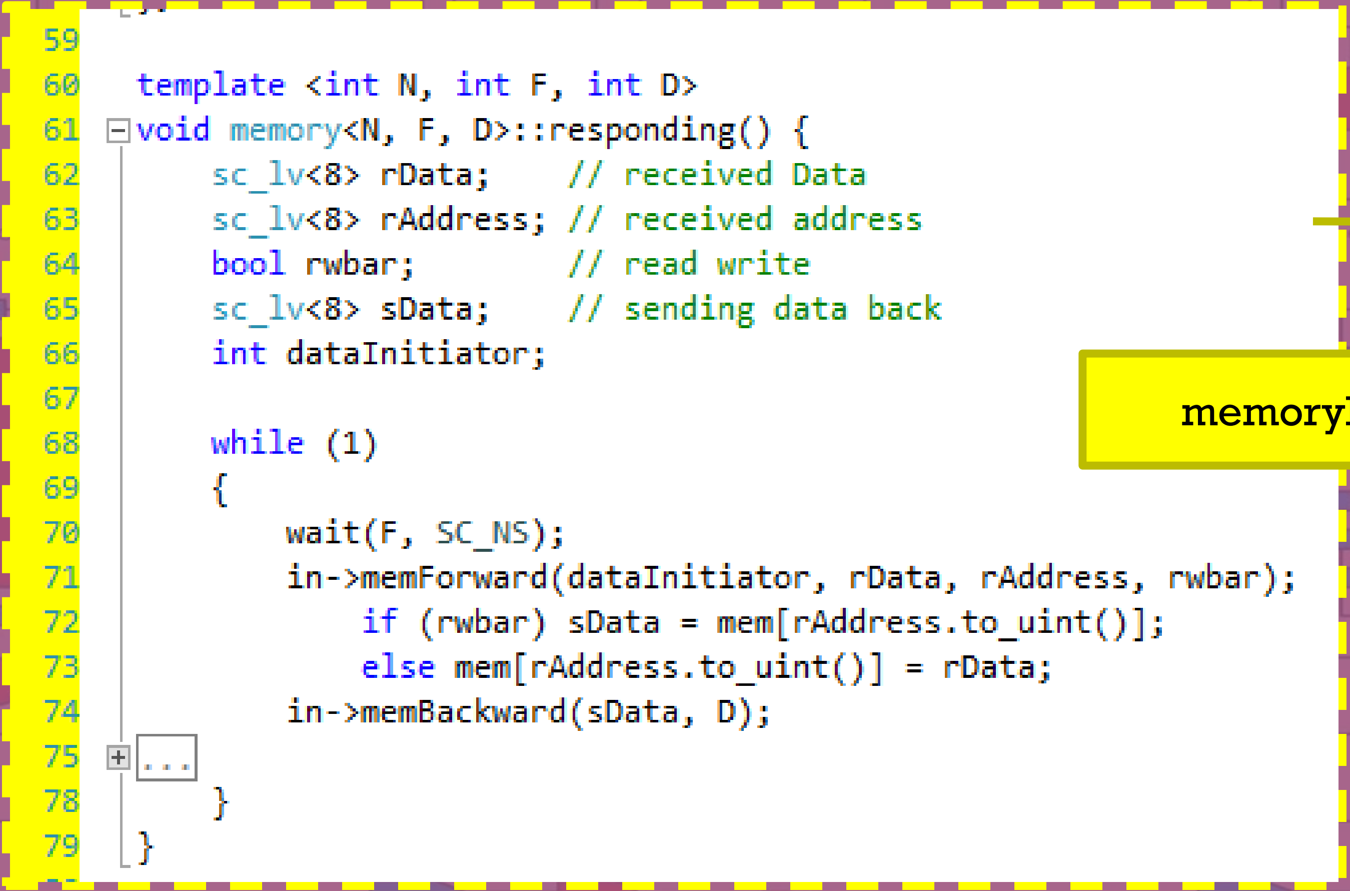
memoryReadWrites.h



```
template <int N, int F, int D>
SC_MODULE (initiators) { ... }

template <int N, int F, int D>
void initiators<N, F, D>::requesting() { ... }

// memory Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE(memory) {
    sc_lv<8>* mem;

    sc_export<memRespond_if<sc_lv<8>>> in;
    memoryAccess<sc_lv<8>, 4>* memBus;

    SC_CTOR(memory) {
        mem = new sc_lv<8>[256];
        for (int i = 0; i < 256; i++) mem[i] = (sc_lv<8>) i;

        memBus = new memoryAccess<sc_lv<8>, 4>;
        SC_THREAD (responding);
    }
    void responding();
    virtual ~memory(){
        delete[] mem;
    }
};
```



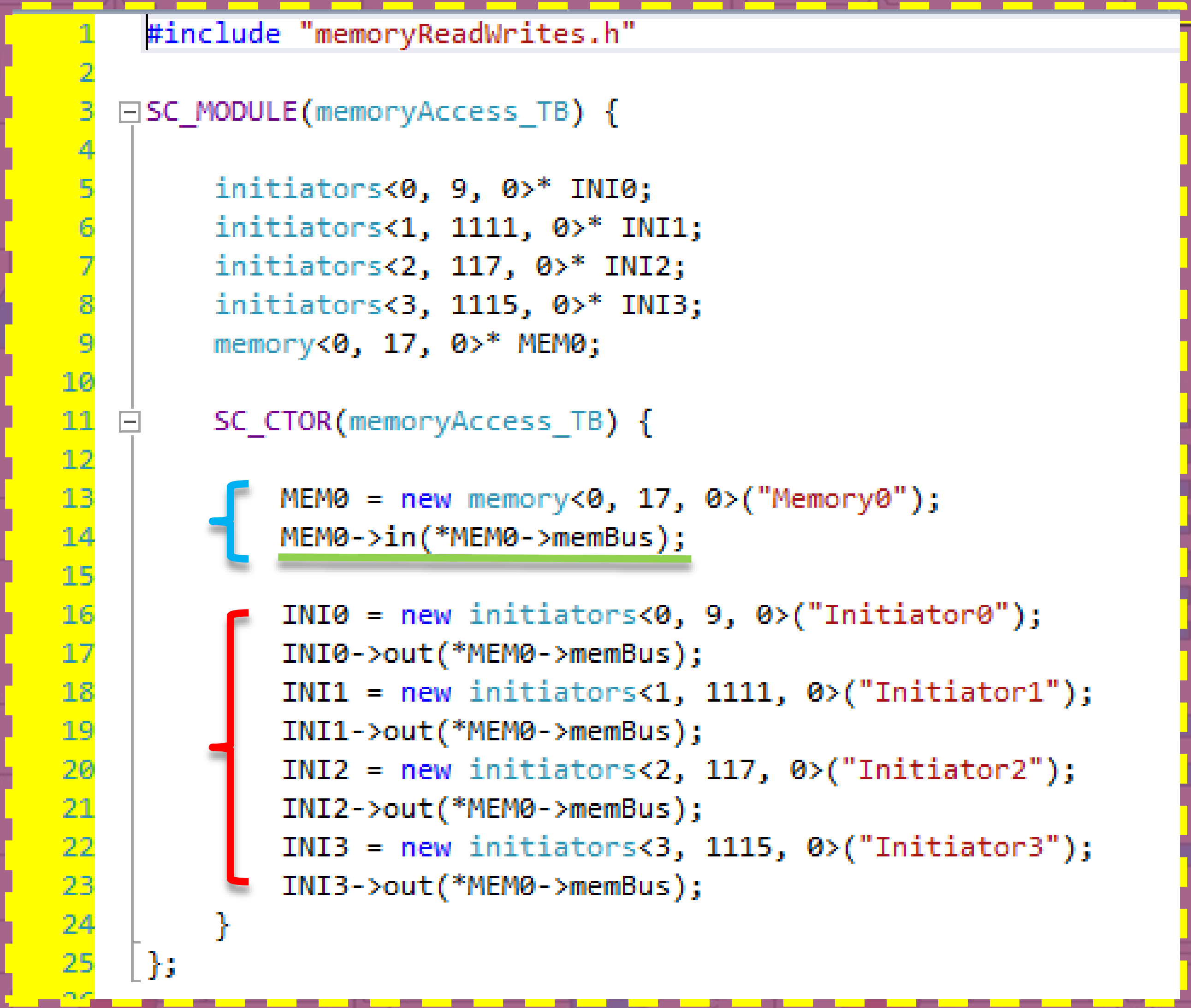
Example 7: *Memory Export*, *memoryReadWrites.h*



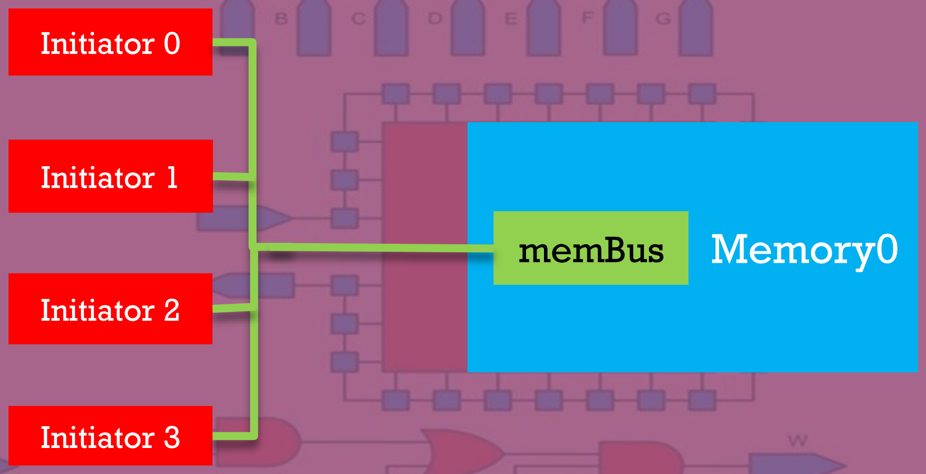
```
template <int N, int F, int D>
void memory<N, F, D>::responding() {
    sc_lv<8> rData;    // received Data
    sc_lv<8> rAddress; // received address
    bool rwbar;        // read write
    sc_lv<8> sData;    // sending data back
    int dataInitiator;

    while (1)
    {
        wait(F, SC_NS);
        in->memForward(dataInitiator, rData, rAddress, rwbar);
            if (rwbar) sData = mem[rAddress.to_uint()];
            else mem[rAddress.to_uint()] = rData;
        in->memBackward(sData, D);
...
    }
}
```

memoryReadWrites.h

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

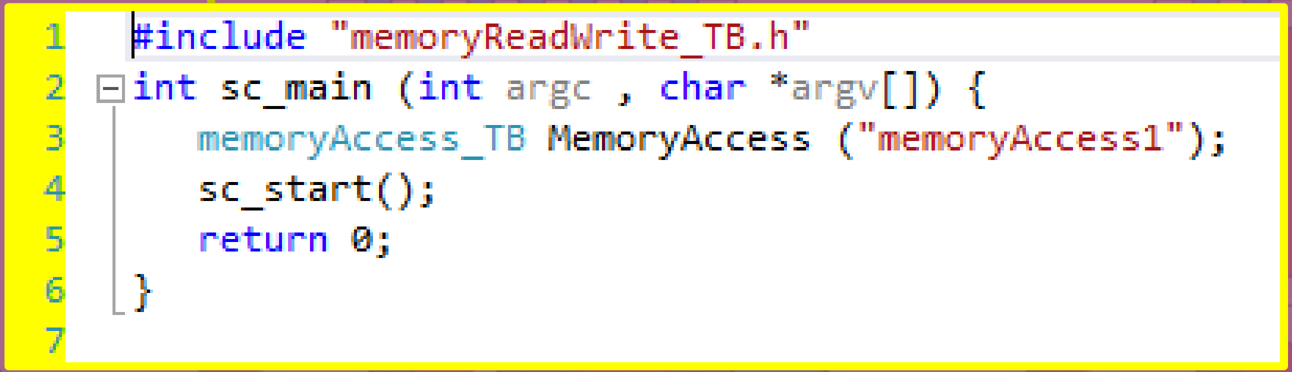


```
#include "memoryReadWrites.h"

SC_MODULE(memoryAccess_TB) {

    initiators<0, 9, 0>* INI0;
    initiators<1, 1111, 0>* INI1;
    initiators<2, 117, 0>* INI2;
    initiators<3, 1115, 0>* INI3;
    memory<0, 17, 0>* MEM0;

    SC_CTOR(memoryAccess_TB) {

        MEM0 = new memory<0, 17, 0>("Memory0");
        MEM0->in(*MEM0->memBus);

        INI0 = new initiators<0, 9, 0>("Initiator0");
        INI0->out(*MEM0->memBus);
        INI1 = new initiators<1, 1111, 0>("Initiator1");
        INI1->out(*MEM0->memBus);
        INI2 = new initiators<2, 117, 0>("Initiator2");
        INI2->out(*MEM0->memBus);
        INI3 = new initiators<3, 1115, 0>("Initiator3");
        INI3->out(*MEM0->memBus);
    }
};
```



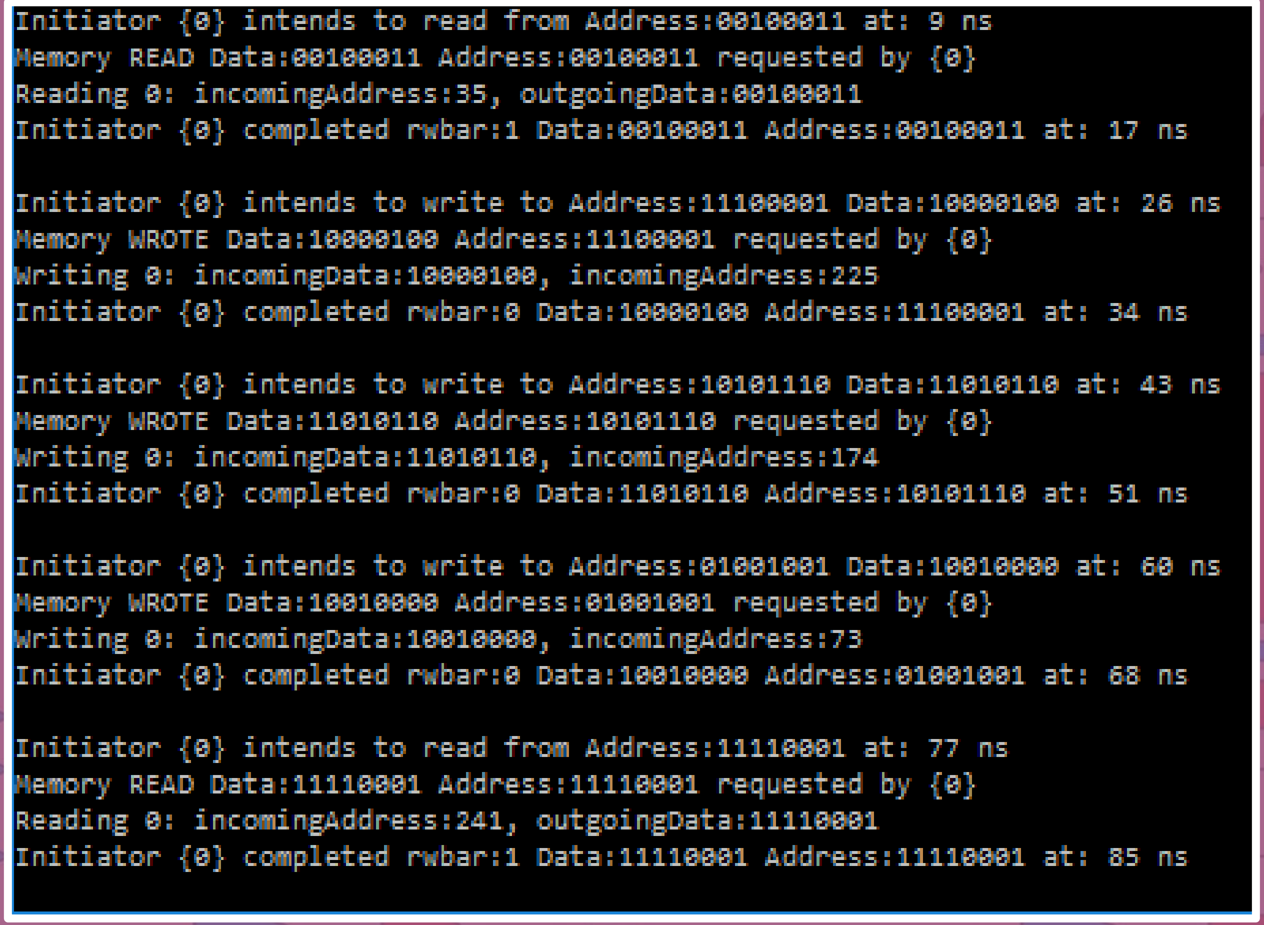
Example 7: *Memory Export*, *memoryReadWrites_main*

memoryReadWrites_main.cpp



```
#include "memoryReadWrite_TB.h"
int sc_main (int argc , char *argv[]) {
    memoryAccess_TB MemoryAccess ("memoryAccess1");
    sc_start();
    return 0;
}
```

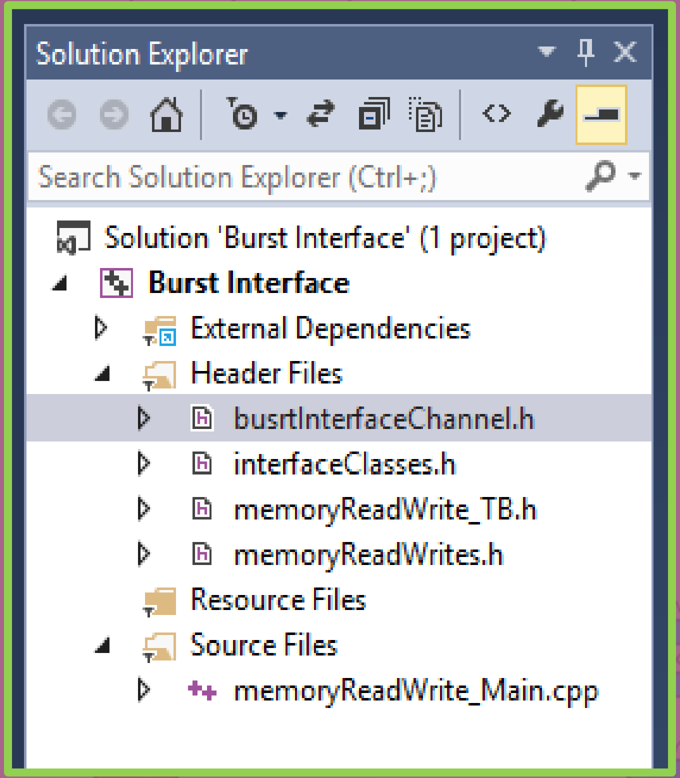
Example 7: *Memory Export*, *output*



```
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

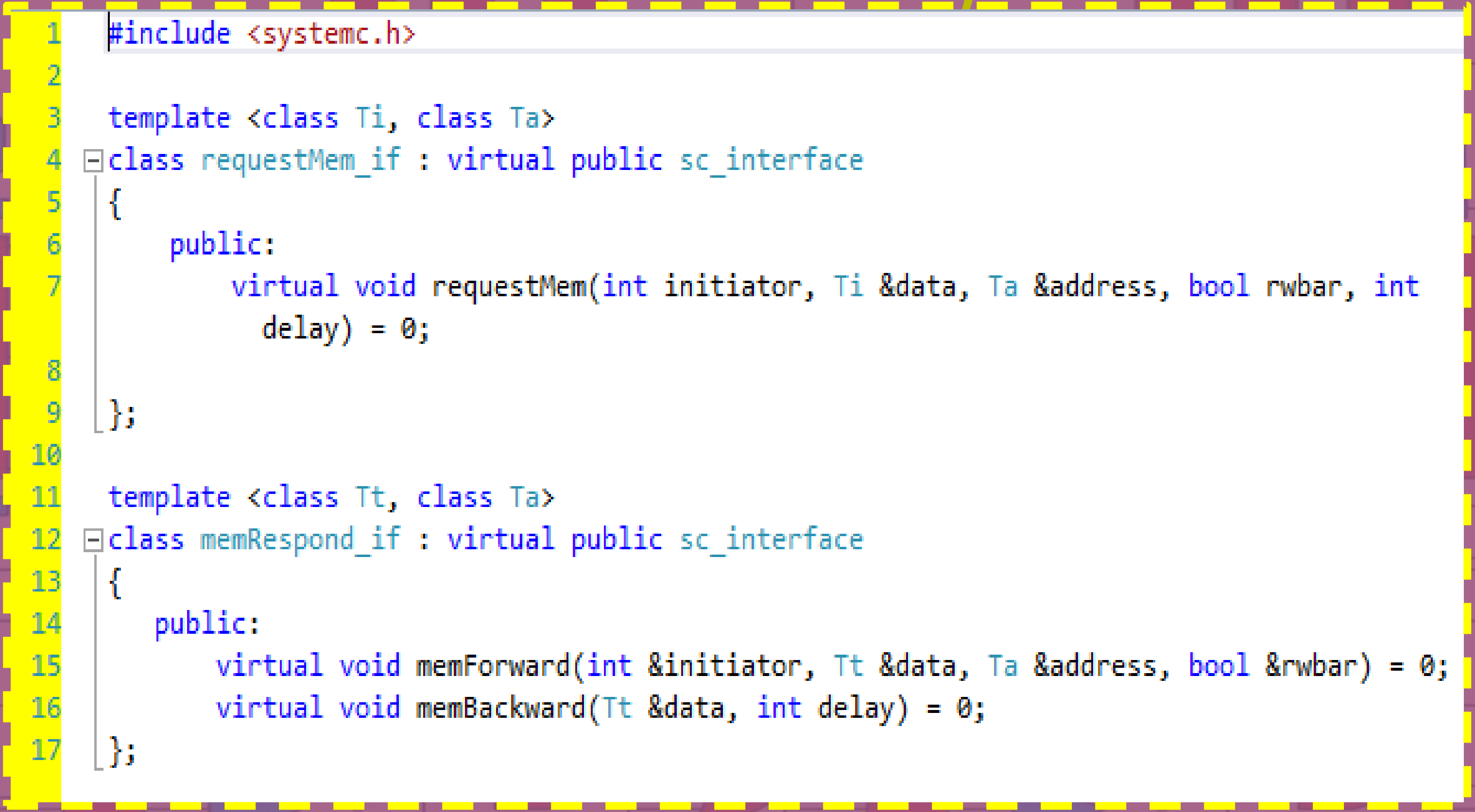
- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using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



Solution 'Burst Interface' (1 project)
- Burst Interface
  - External Dependencies
  - Header Files
    - busrtInterfaceChannel.h
    - interfaceClasses.h
    - memoryReadWrite_TB.h
    - memoryReadWrites.h
  - Resource Files
  - Source Files
    - memoryReadWrite_Main.cpp

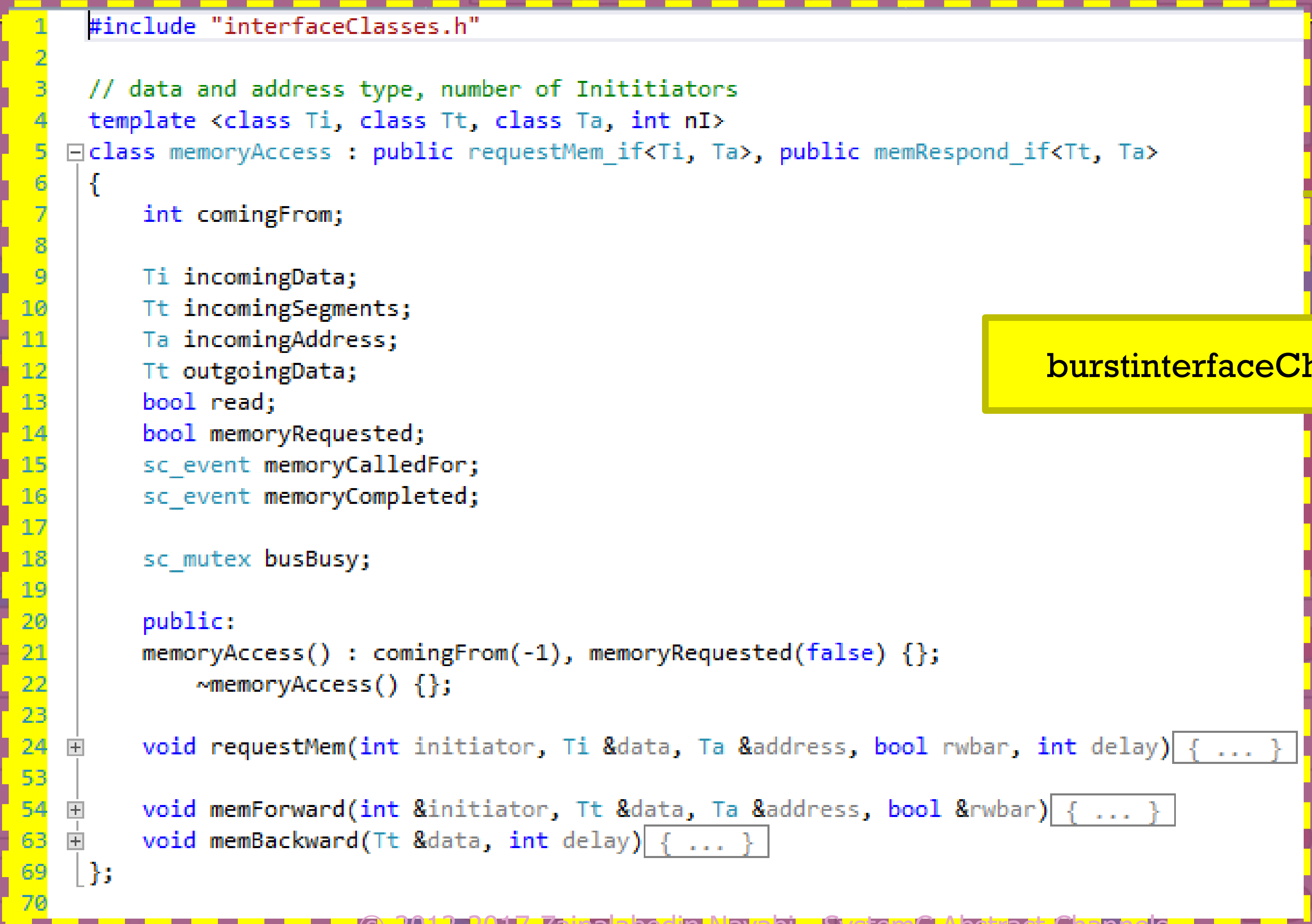
Example 8: *Burst interface*, *interfaceClasses.h*

interfaceClasses.h



```
#include <systemc.h>

template <class Ti, class Ta>
class requestMem_if : virtual public sc_interface
{
    public:
        virtual void requestMem(int initiator, Ti &data, Ta &address, bool rwbar, int delay) = 0;

};

template <class Tt, class Ta>
class memRespond_if : virtual public sc_interface
{
   public:
       virtual void memForward(int &initiator, Tt &data, Ta &address, bool &rwbar) = 0;
       virtual void memBackward(Tt &data, int delay) = 0;
};
```

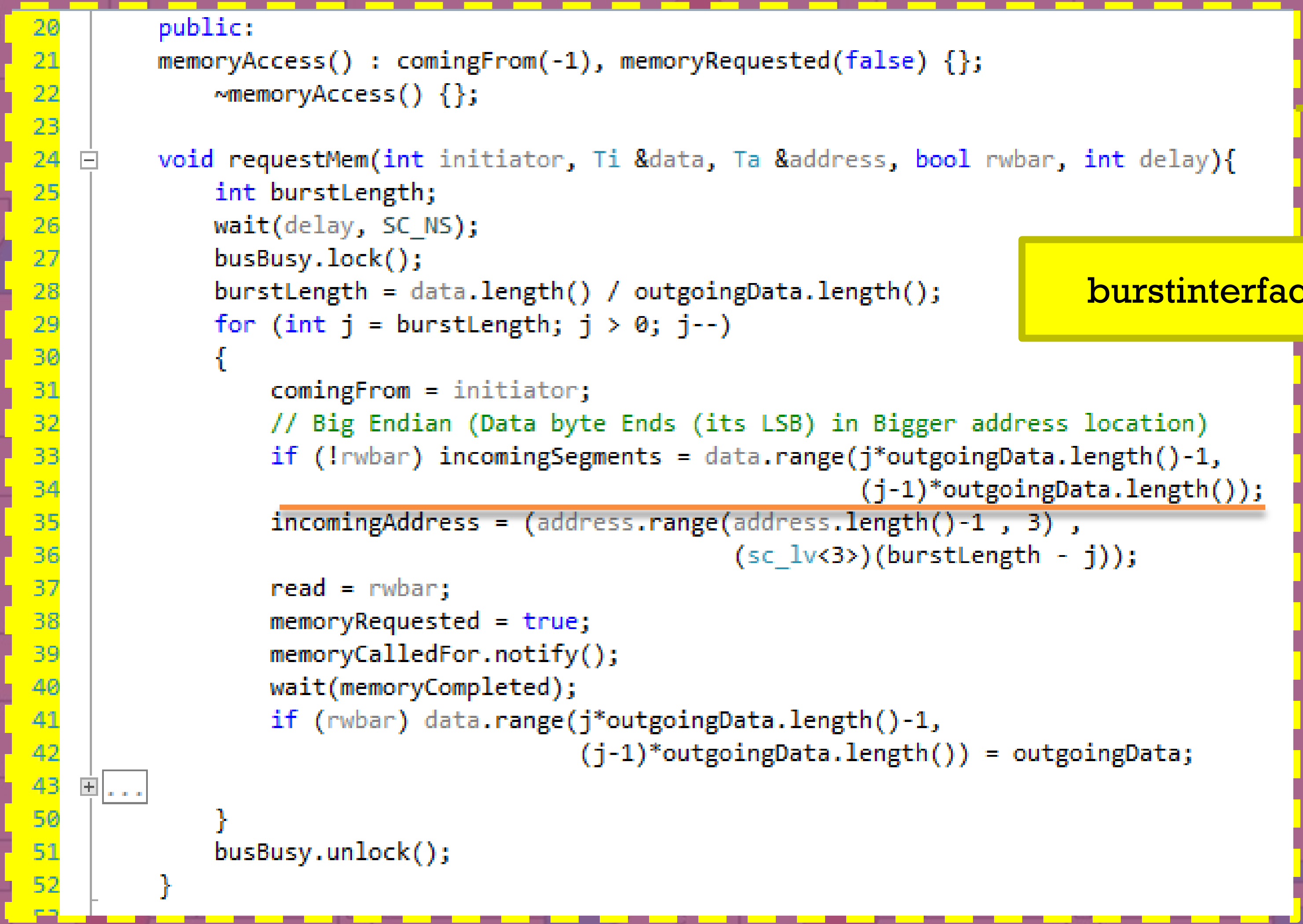
Example 8: *Burst interface*, *burstinterfaceChannel.h*



```
#include "interfaceClasses.h"

// data and address type, number of Inititiators
template <class Ti, class Tt, class Ta, int nI>
class memoryAccess : public requestMem_if<Ti, Ta>, public memRespond_if<Tt, Ta>
{
    int comingFrom;

    Ti incomingData;
    Tt incomingSegments;
    Ta incomingAddress;
    Tt outgoingData;
    bool read;
    bool memoryRequested;
    sc_event memoryCalledFor;
    sc_event memoryCompleted;

    sc_mutex busBusy;

    public:
    memoryAccess() : comingFrom(-1), memoryRequested(false) {};
        ~memoryAccess() {};

    void requestMem(int initiator, Ti &data, Ta &address, bool rwbar, int delay){ ... }

    void memForward(int &initiator, Tt &data, Ta &address, bool &rwbar){ ... }
    void memBackward(Tt &data, int delay){ ... }
};

```

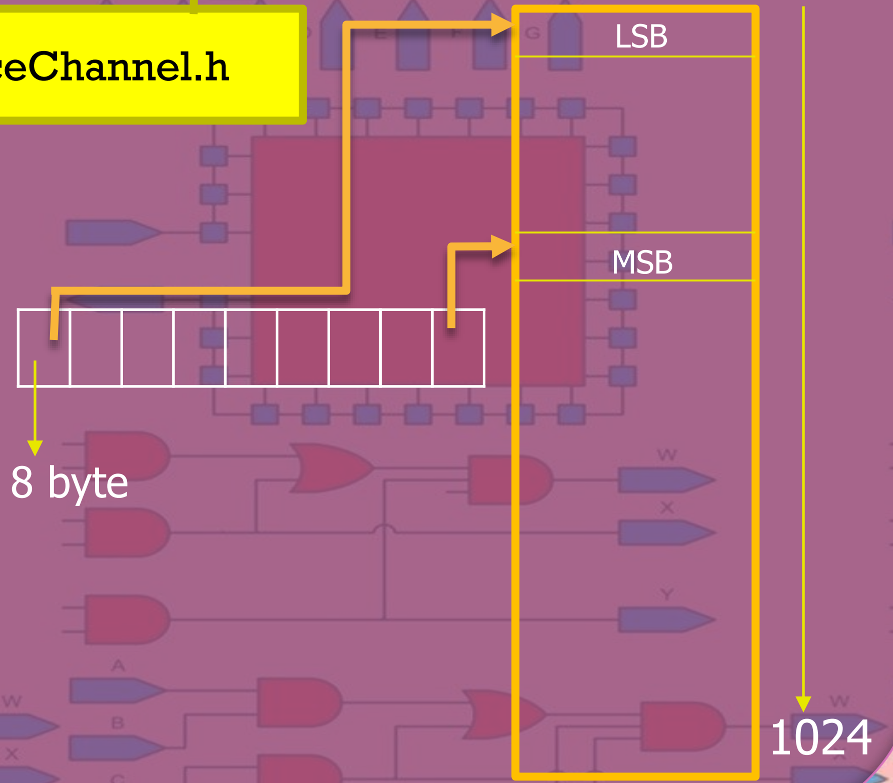
burstinterfaceChannel.h

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

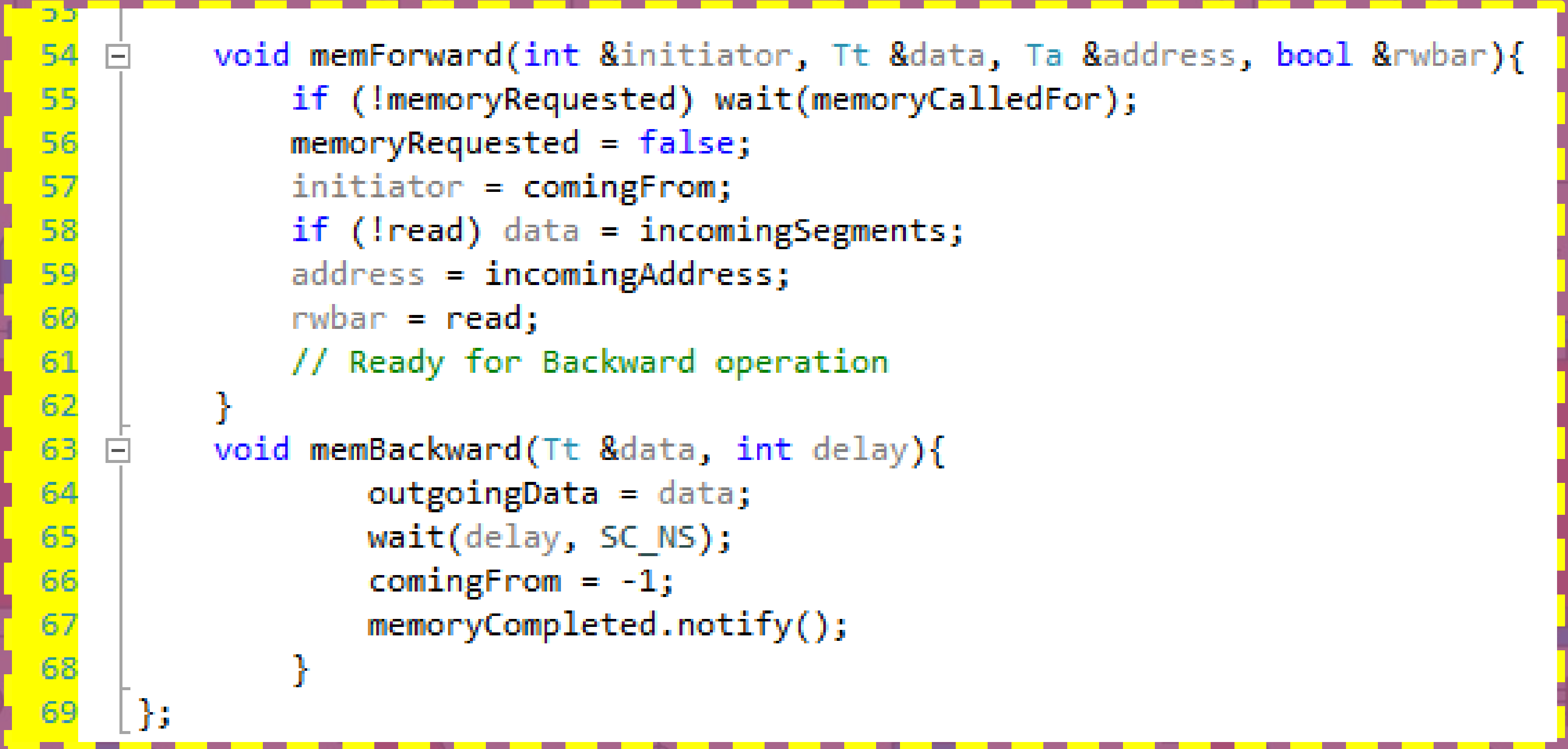


```
public:
memoryAccess() : comingFrom(-1), memoryRequested(false) {};
    ~memoryAccess() {};

void requestMem(int initiator, Ti &data, Ta &address, bool rwbar, int delay){
    int burstLength;
    wait(delay, SC_NS);
    busBusy.lock();
    burstLength = data.length() / outgoingData.length();
    for (int j = burstLength; j > 0; j--)
    {
        comingFrom = initiator;
        // Big Endian (Data byte Ends (its LSB) in Bigger address location)
        if (!rwbar) incomingSegments = data.range(j*outgoingData.length()-1,
                                          (j-1)*outgoingData.length());
        incomingAddress = (address.range(address.length()-1 , 3) ,
                                 (sc_lv<3>)(burstLength - j));
        read = rwbar;
        memoryRequested = true;
        memoryCalledFor.notify();
        wait(memoryCompleted);
        if (rwbar) data.range(j*outgoingData.length()-1,
                           (j-1)*outgoingData.length()) = outgoingData;
...
    }
    busBusy.unlock();
}
```

burstinterfaceChannel.h



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

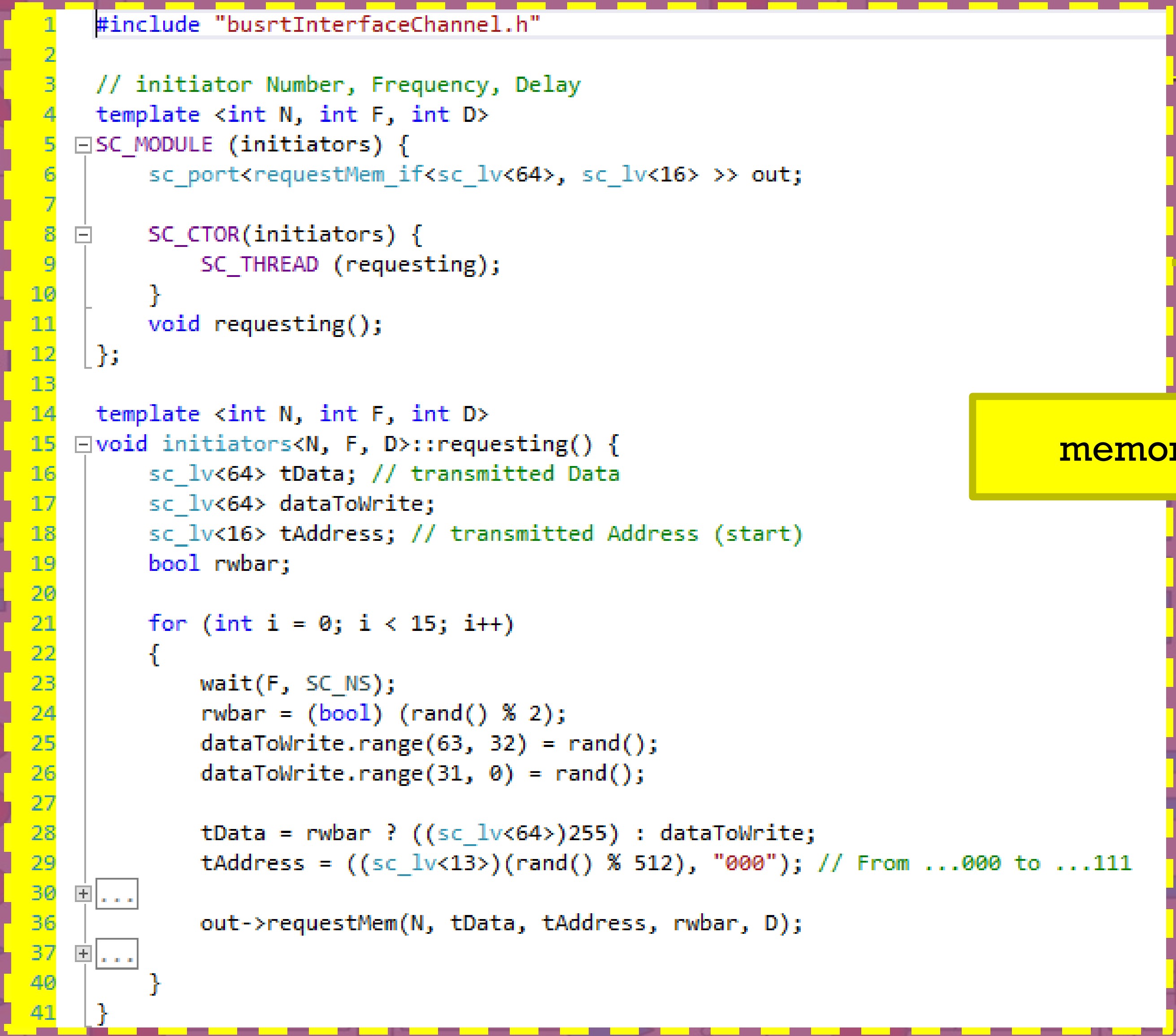


```
    void memForward(int &initiator, Tt &data, Ta &address, bool &rwbar){
        if (!memoryRequested) wait(memoryCalledFor);
        memoryRequested = false;
        initiator = comingFrom;
        if (!read) data = incomingSegments;
        address = incomingAddress;
        rwbar = read;
        // Ready for Backward operation
    }
    void memBackward(Tt &data, int delay){
            outgoingData = data;
            wait(delay, SC_NS);
            comingFrom = -1;
            memoryCompleted.notify();
        }
};
```

burstinterfaceChannel.h

Example 8: *Burst interface*,

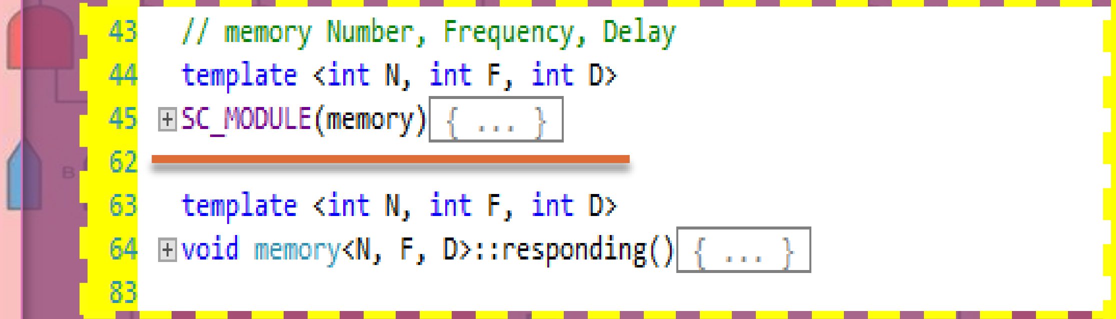
memoryReadWrites.h



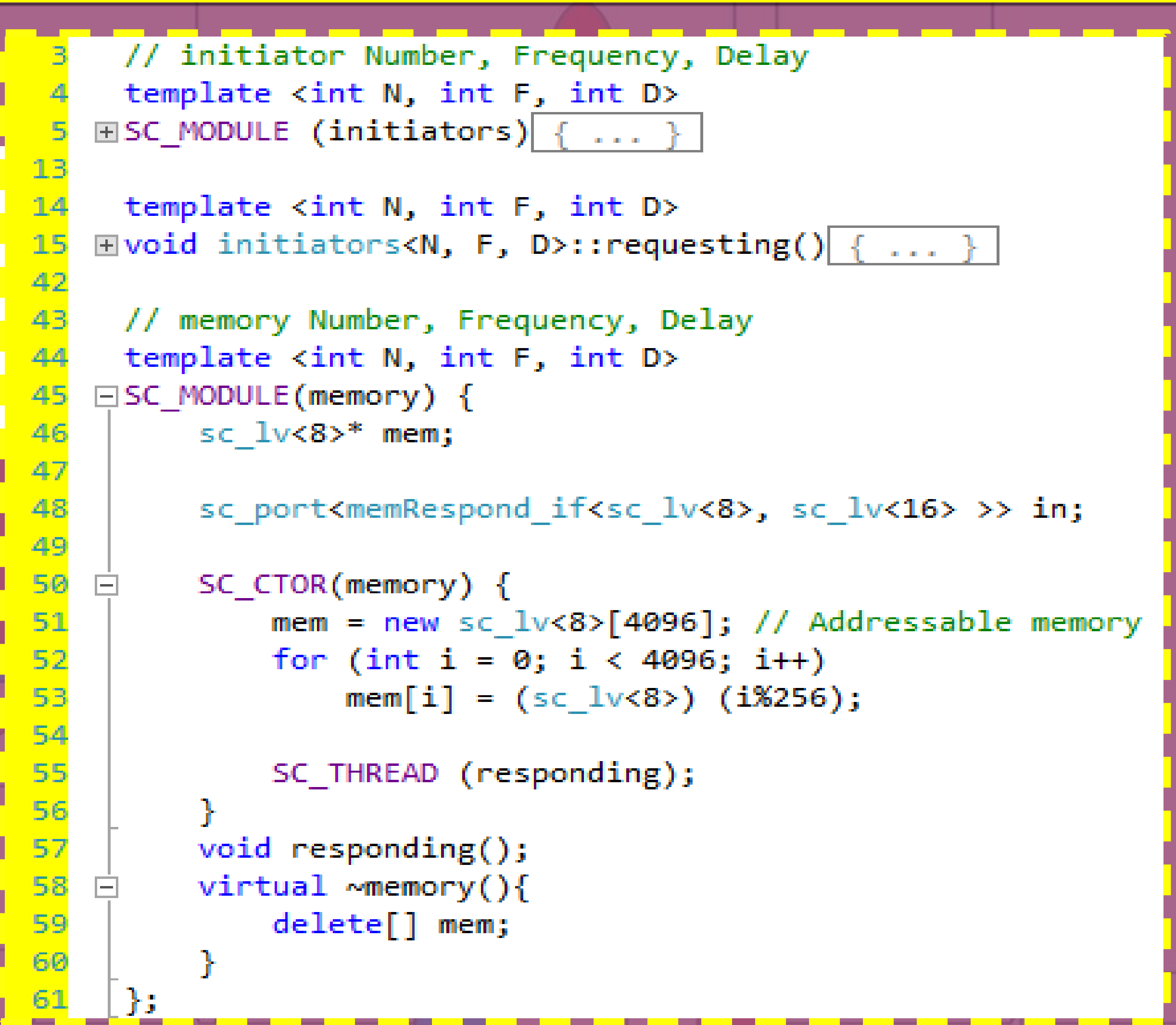
```
#include "busrtInterfaceChannel.h"

// initiator Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE (initiators) {
    sc_port<requestMem_if<sc_lv<64>, sc_lv<16> >> out;

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

template <int N, int F, int D>
void initiators<N, F, D>::requesting() {
    sc_lv<64> tData; // transmitted Data
    sc_lv<64> dataToWrite;
    sc_lv<16> tAddress; // transmitted Address (start)
    bool rwbar;

    for (int i = 0; i < 15; i++)
    {
        wait(F, SC_NS);
        rwbar = (bool) (rand() % 2);
        dataToWrite.range(63, 32) = rand();
        dataToWrite.range(31, 0) = rand();

        tData = rwbar ? ((sc_lv<64>)255) : dataToWrite;
        tAddress = ((sc_lv<13>)(rand() % 512), "000"); // From ...000 to ...111
...
        out->requestMem(N, tData, tAddress, rwbar, D);
...
    }
}
```

memoryReadWrites.h

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

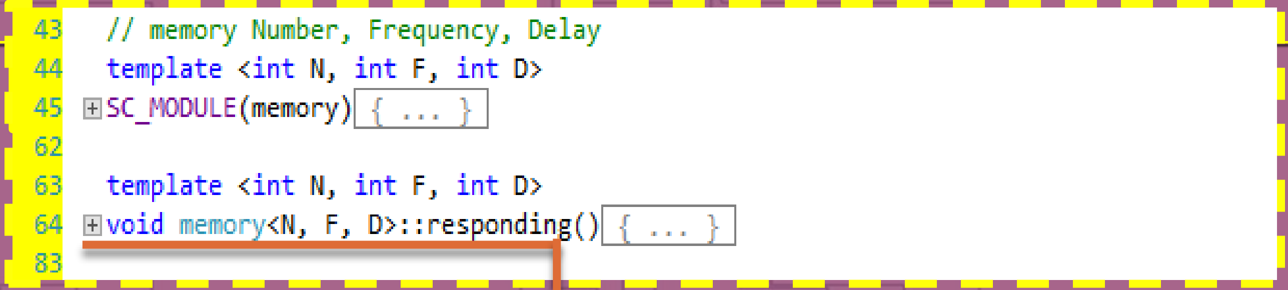


```
// memory Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE(memory){ ... }

template <int N, int F, int D>
void memory<N, F, D>::responding(){ ... }
```

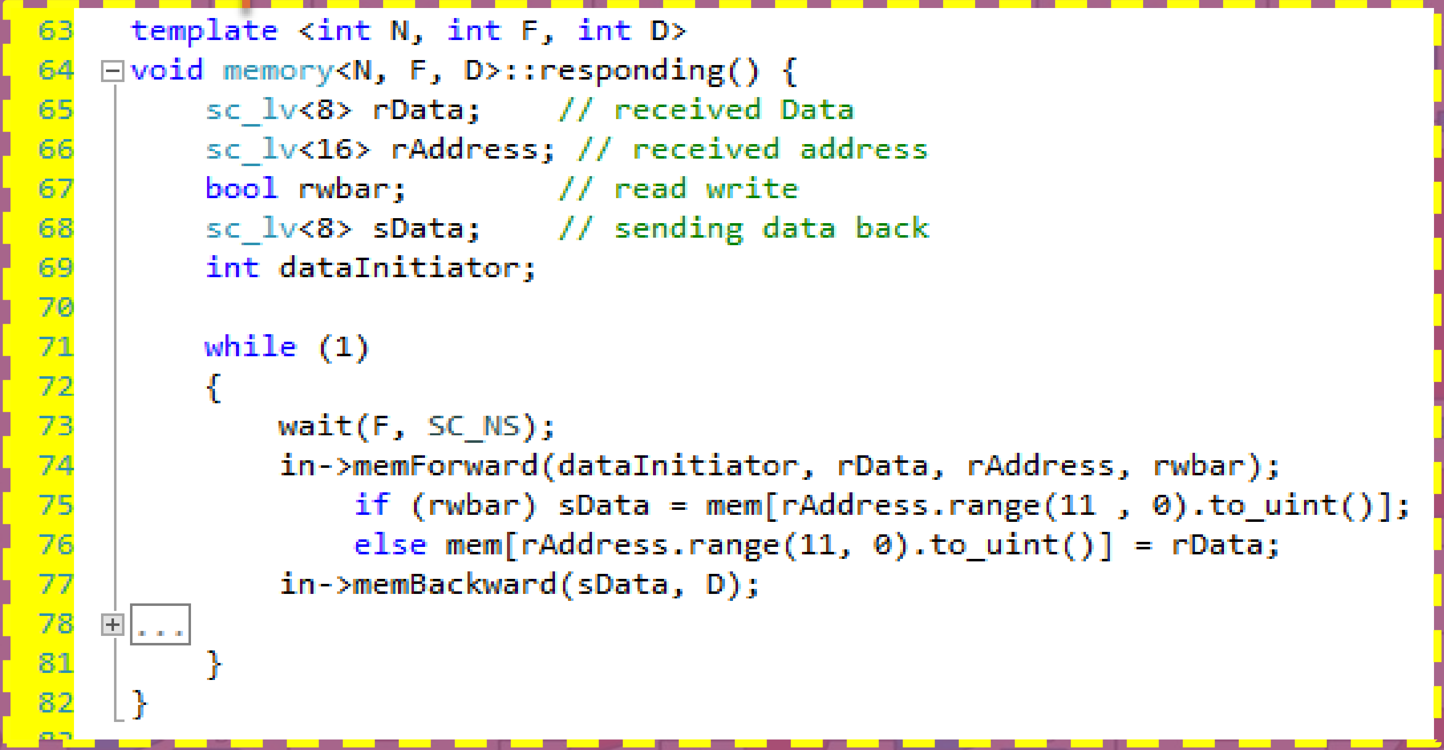


```
// initiator Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE (initiators){ ... }

template <int N, int F, int D>
void initiators<N, F, D>::requesting(){ ... }

// memory Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE(memory) {
    sc_lv<8>* mem;

    sc_port<memRespond_if<sc_lv<8>, sc_lv<16> >> in;

    SC_CTOR(memory) {
        mem = new sc_lv<8>[4096]; // Addressable memory
        for (int i = 0; i < 4096; i++)
            mem[i] = (sc_lv<8>) (i%256);

        SC_THREAD (responding);
    }
    void responding();
    virtual ~memory(){
        delete[] mem;
    }
};
```

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

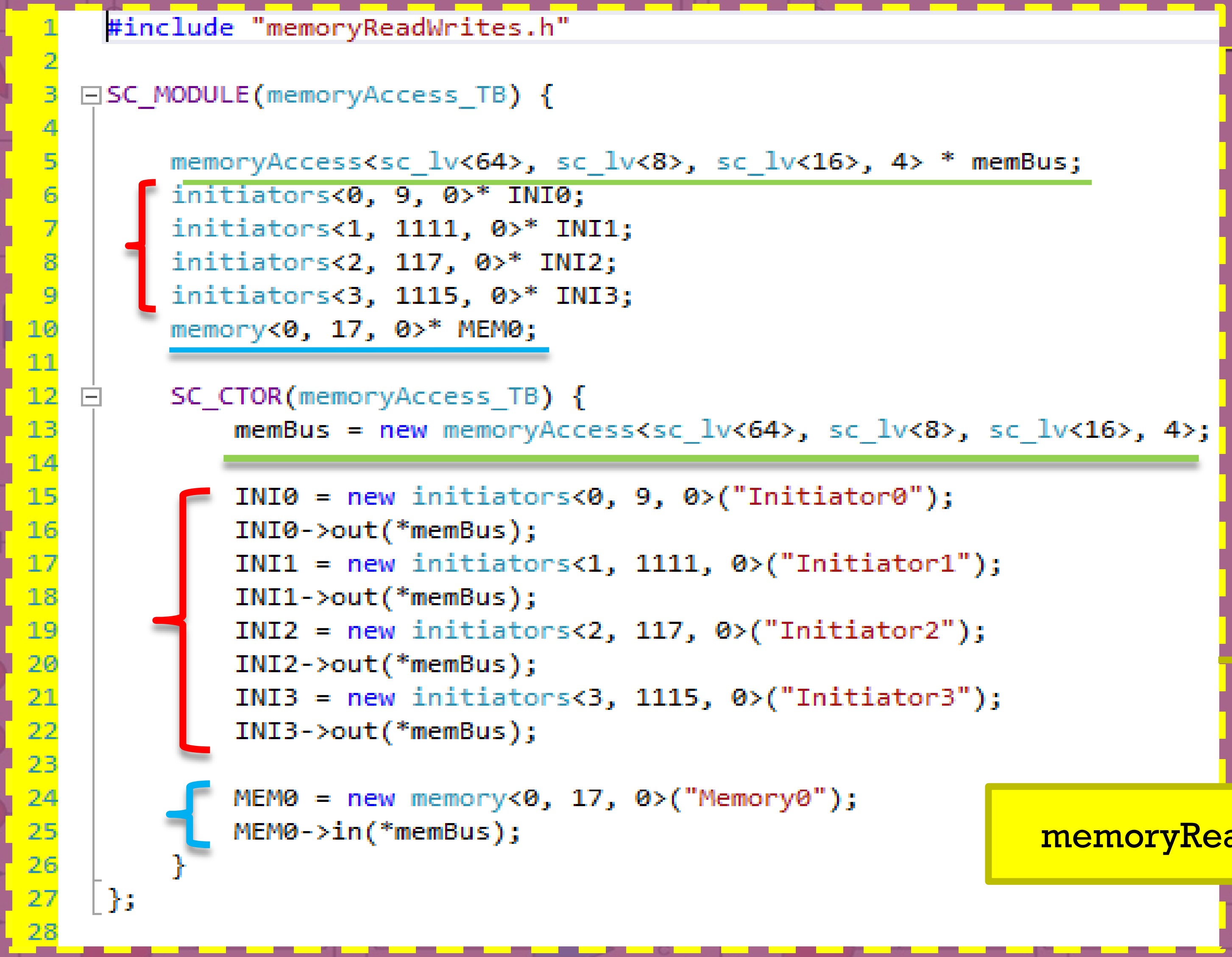


```
// memory Number, Frequency, Delay
template <int N, int F, int D>
SC_MODULE(memory) { ... }

template <int N, int F, int D>
void memory<N, F, D>::responding() { ... }
```

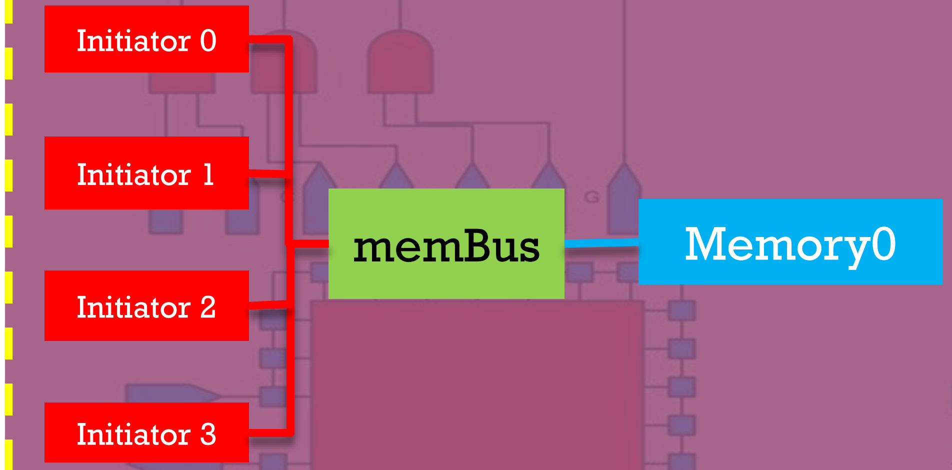


```
template <int N, int F, int D>
void memory<N, F, D>::responding() {
    sc_lv<8> rData;    // received Data
    sc_lv<16> rAddress; // received address
    bool rwbar;        // read write
    sc_lv<8> sData;    // sending data back
    int dataInitiator;

    while (1)
    {
        wait(F, SC_NS);
        in->memForward(dataInitiator, rData, rAddress, rwbar);
            if (rwbar) sData = mem[rAddress.range(11 , 0).to_uint()];
            else mem[rAddress.range(11, 0).to_uint()] = rData;
        in->memBackward(sData, D);
...
    }
}
```

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

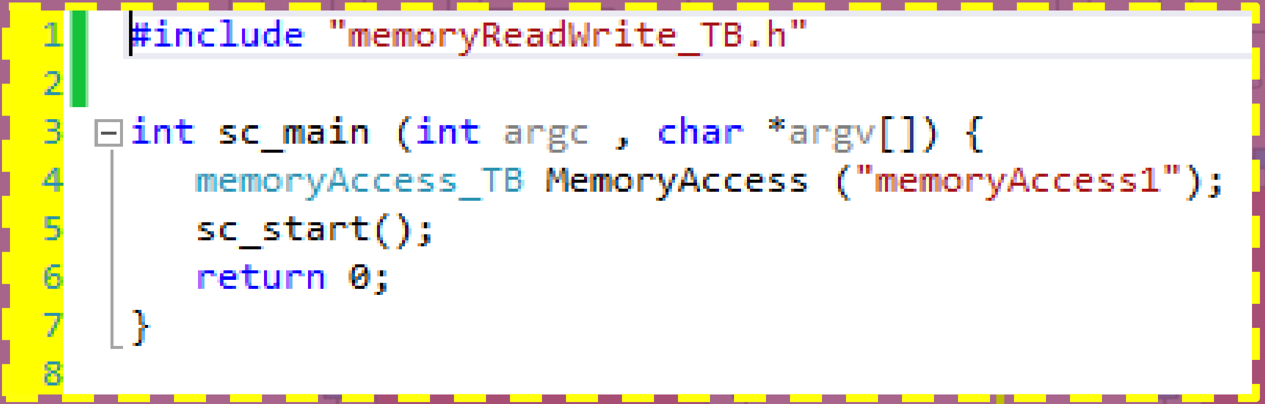


```
#include "memoryReadWrites.h"

SC_MODULE(memoryAccess_TB) {

    memoryAccess<sc_lv<64>, sc_lv<8>, sc_lv<16>, 4> * memBus;
    initiators<0, 9, 0>* INI0;
    initiators<1, 1111, 0>* INI1;
    initiators<2, 117, 0>* INI2;
    initiators<3, 1115, 0>* INI3;
    memory<0, 17, 0>* MEM0;

    SC_CTOR(memoryAccess_TB) {
        memBus = new memoryAccess<sc_lv<64>, sc_lv<8>, sc_lv<16>, 4>;

        INI0 = new initiators<0, 9, 0>("Initiator0");
        INI0->out(*memBus);
        INI1 = new initiators<1, 1111, 0>("Initiator1");
        INI1->out(*memBus);
        INI2 = new initiators<2, 117, 0>("Initiator2");
        INI2->out(*memBus);
        INI3 = new initiators<3, 1115, 0>("Initiator3");
        INI3->out(*memBus);

        MEM0 = new memory<0, 17, 0>("Memory0");
        MEM0->in(*memBus);
    }
};

```



memoryReadWrites_TB.h

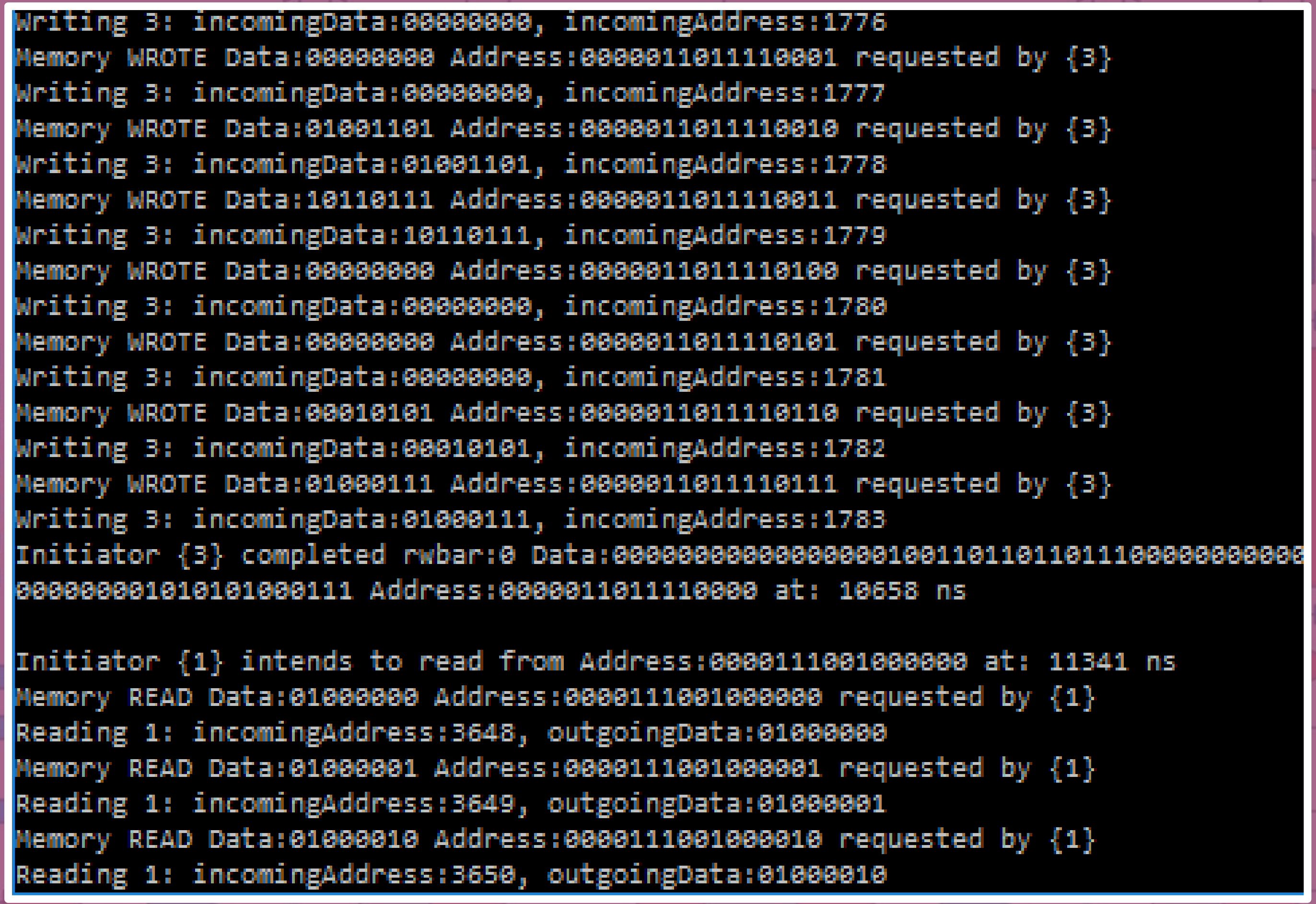
Example 8: *Burst interface*, *memoryReadWrites_main*



```
#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:0000000000000000010011011011011100000000000
000000001010101000111 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

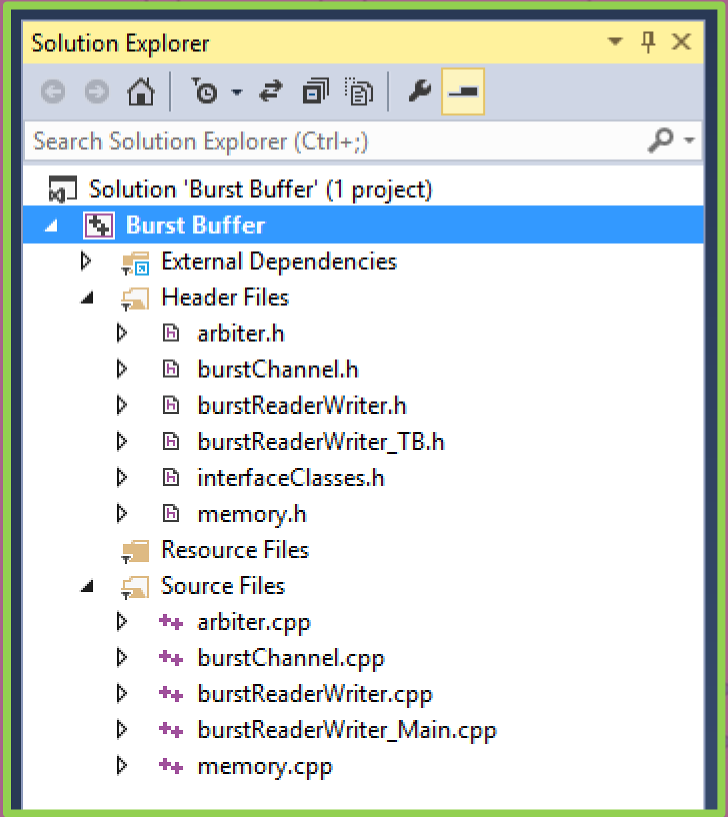
- Handshaking
  - Serial to Parallel Stack Writer
- Channels
  - Basics of Channels
    - sc_signal
    - sc_mutex
  - Primitive Channels
    - Simple put-get buffer channel
    - FIFO channel
    - Stack non-blocking channel
    - Multi-way shared bus
    - Priority shared bus
    - Memory access, using 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

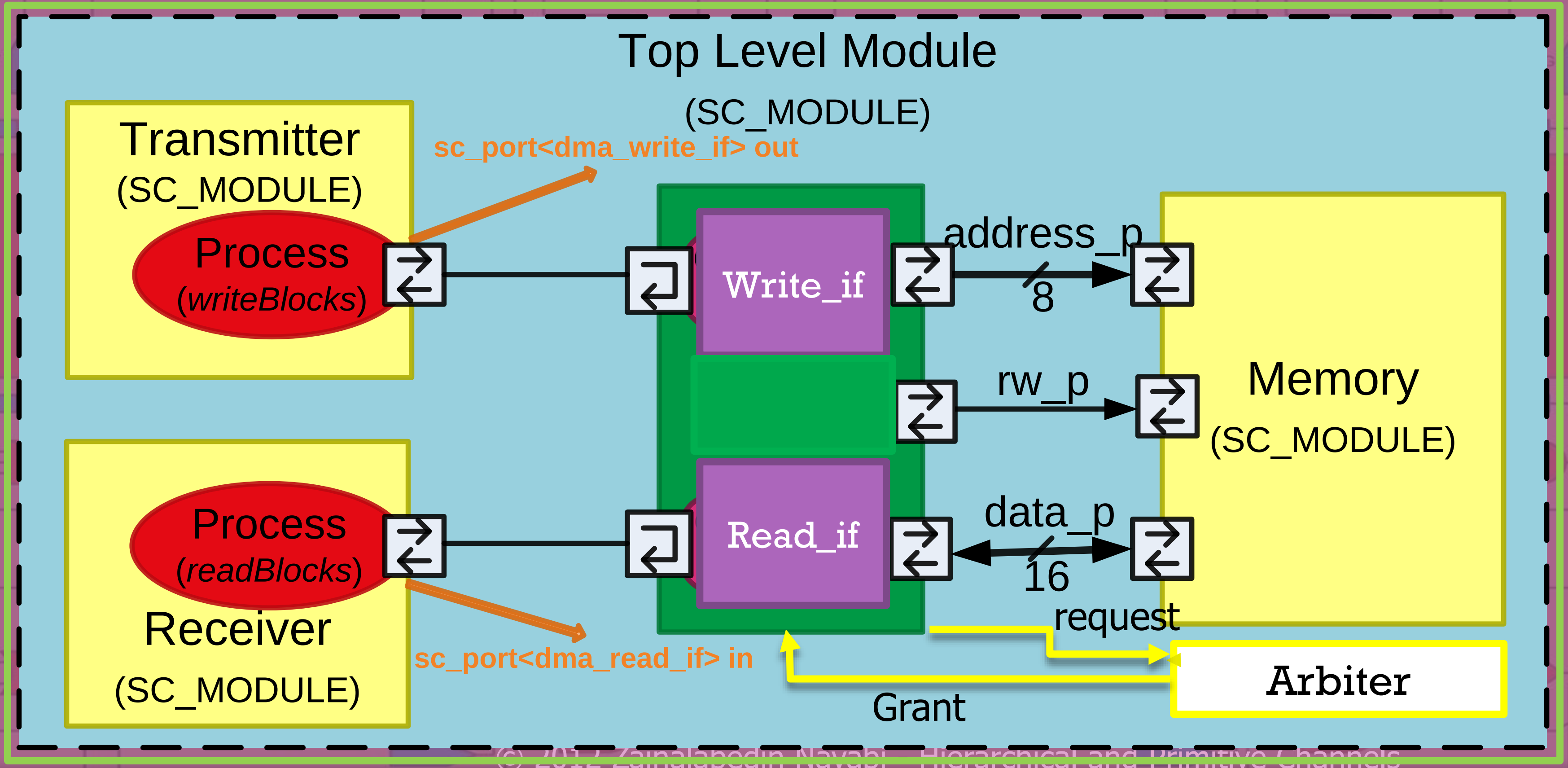

Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'Burst Buffer' (1 project)
  Burst Buffer
    External Dependencies
    Header Files
      arbiter.h
      burstChannel.h
      burstReaderWriter.h
      burstReaderWriter_TB.h
      interfaceClasses.h
      memory.h
    Resource Files
    Source Files
      arbiter.cpp
      burstChannel.cpp
      burstReaderWriter.cpp
      burstReaderWriter_Main.cpp
      memory.cpp
```

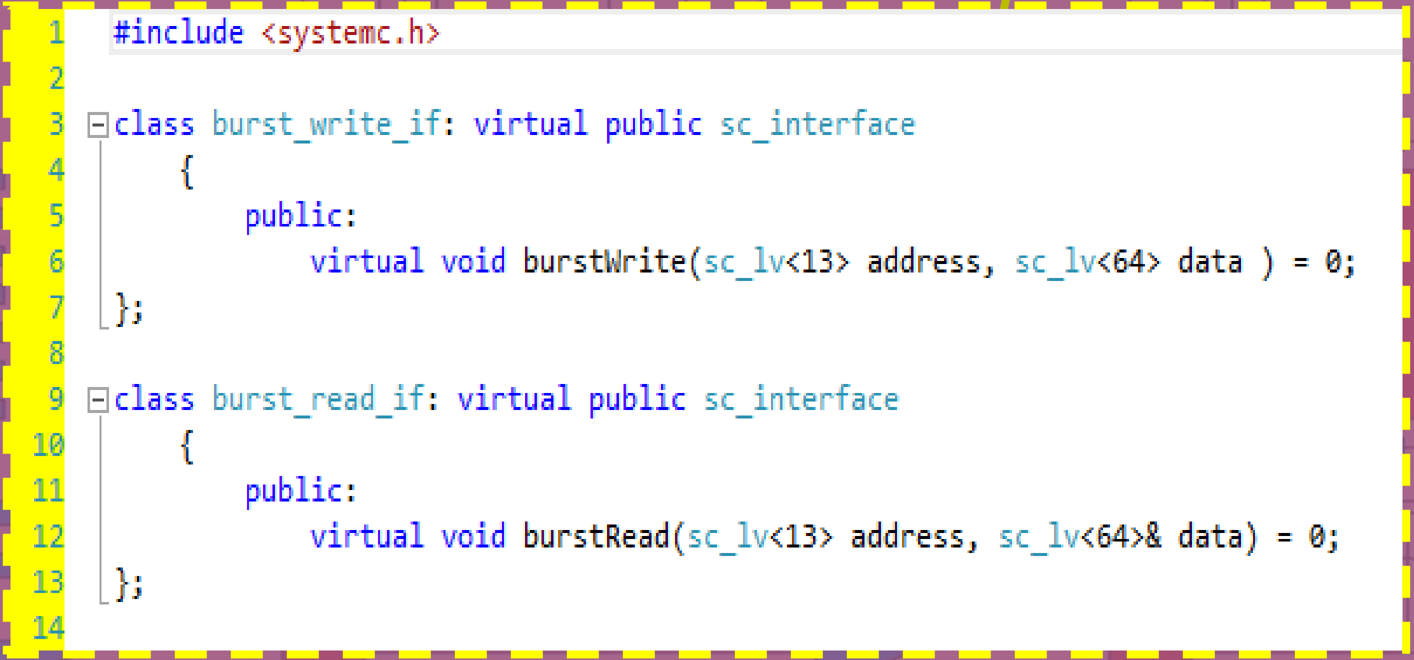
Burst Buffer

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



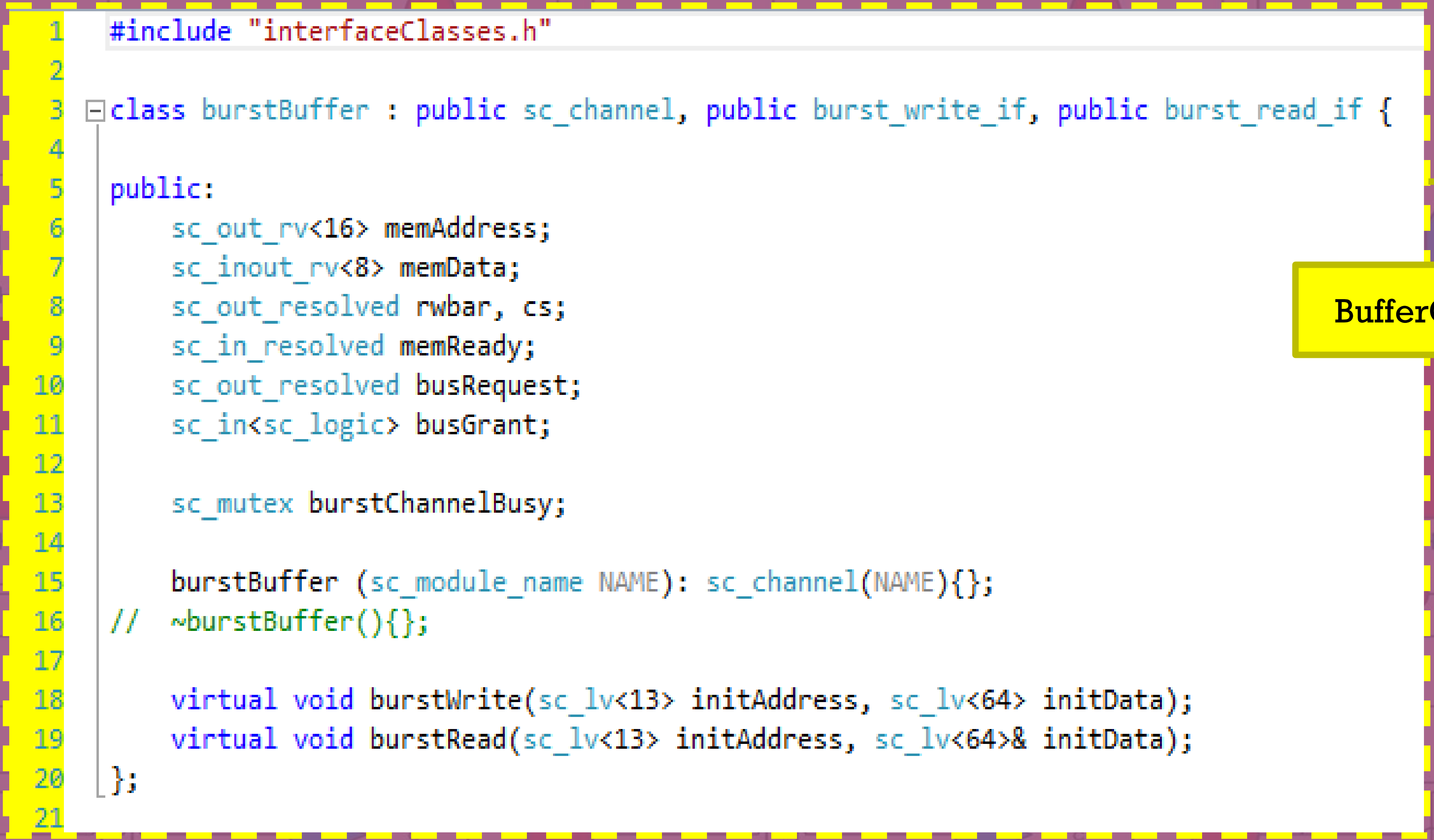
Example :Burst Buffer, *interfaceClasses*

interfaceClasses.h



```
#include <systemc.h>

class burst_write_if: virtual public sc_interface
    {
        public:
            virtual void burstWrite(sc_lv<13> address, sc_lv<64> data ) = 0;
};

class burst_read_if: virtual public sc_interface
    {
        public:
            virtual void burstRead(sc_lv<13> address, sc_lv<64>& data) = 0;
};

```

Example :Burst Buffer, *Buffer Channel.h*

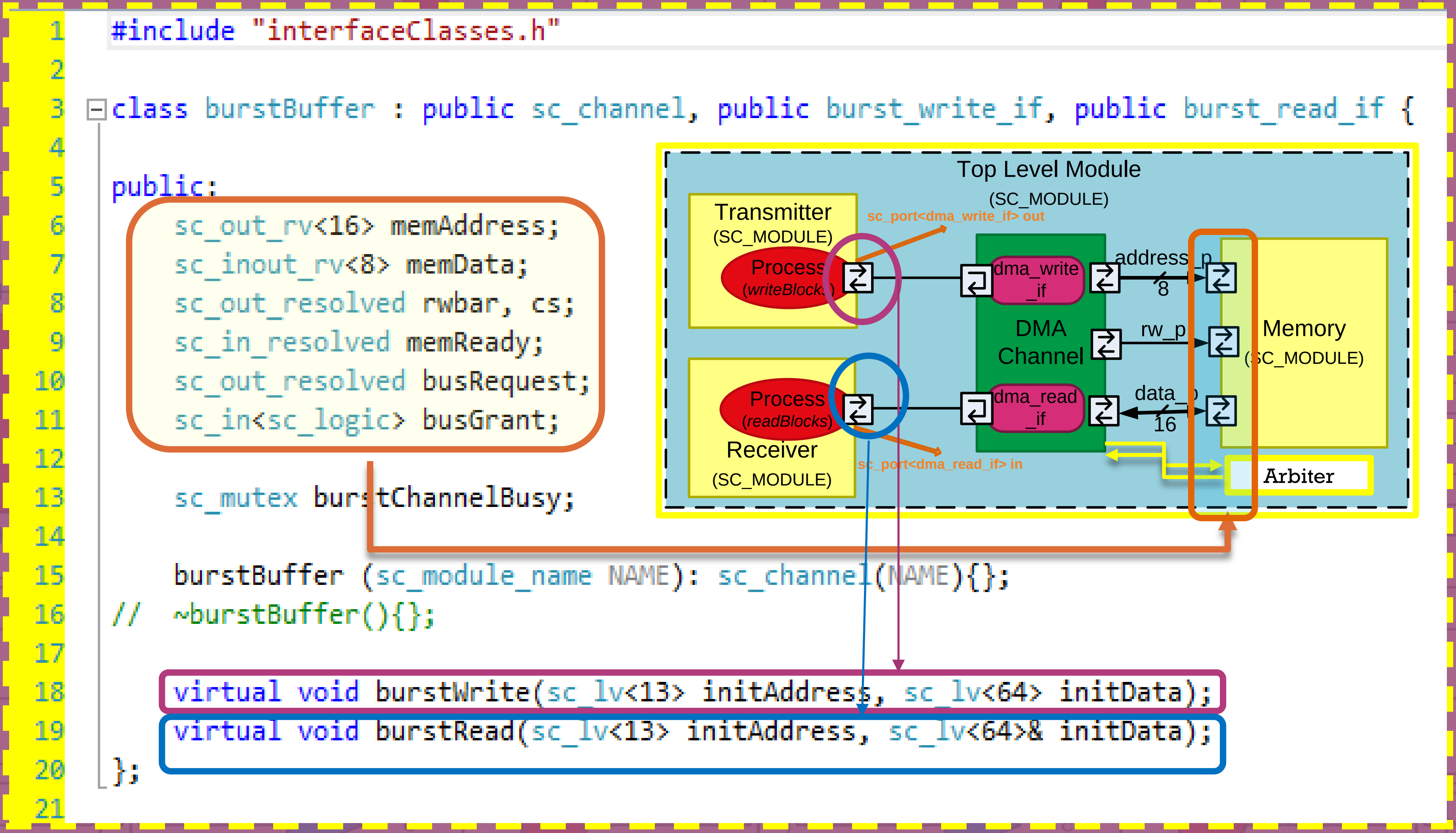


```
#include "interfaceClasses.h"

class burstBuffer : public sc_channel, public burst_write_if, public burst_read_if {

public:
    sc_out_rv<16> memAddress;
    sc_inout_rv<8> memData;
    sc_out_resolved rwbar, cs;
    sc_in_resolved memReady;
    sc_out_resolved busRequest;
    sc_in<sc_logic> busGrant;

    sc_mutex burstChannelBusy;

    burstBuffer (sc_module_name NAME): sc_channel(NAME){};
//  ~burstBuffer(){};

    virtual void burstWrite(sc_lv<13> initAddress, sc_lv<64> initData);
    virtual void burstRead(sc_lv<13> initAddress, sc_lv<64>& initData);
};

```

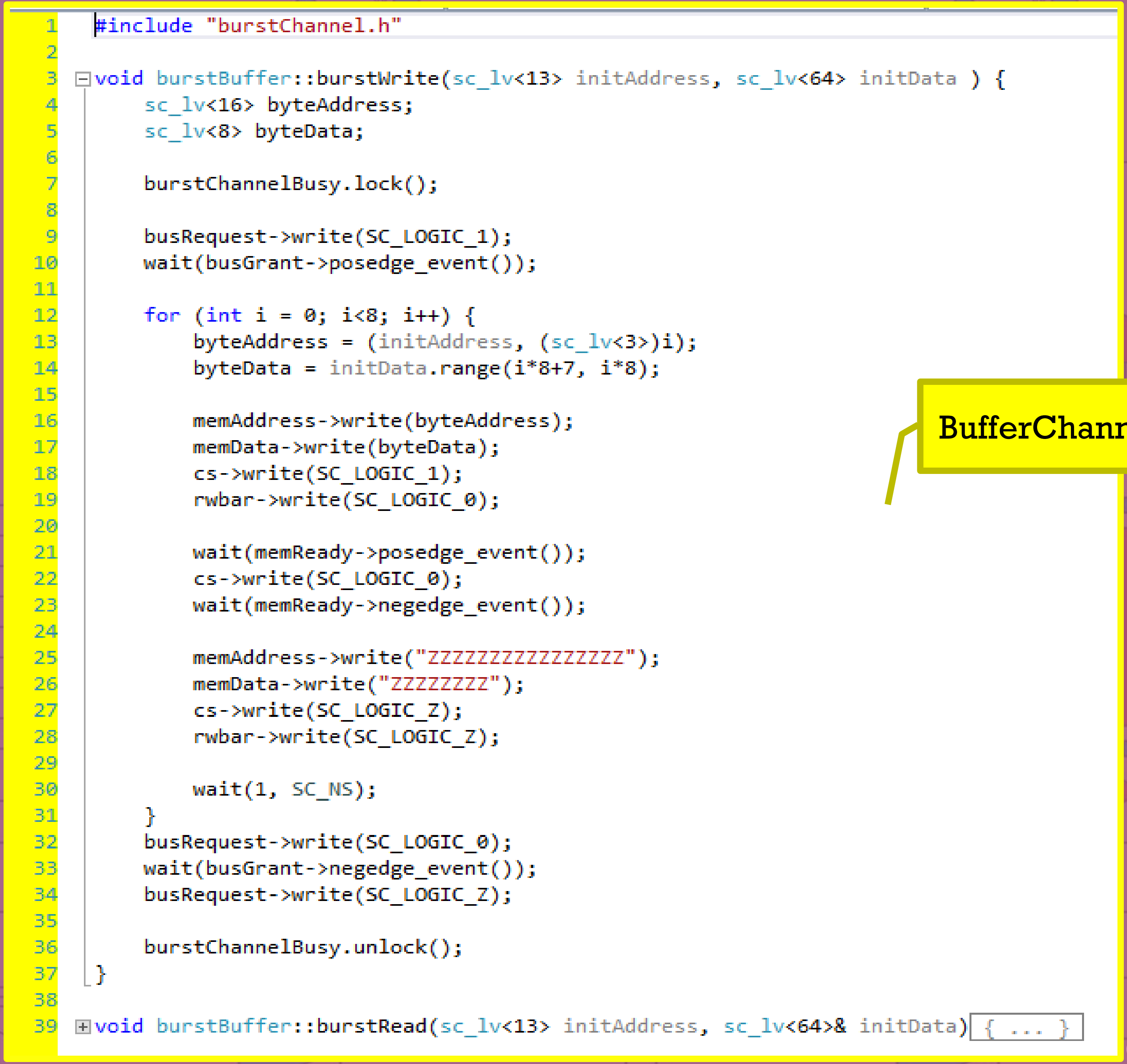
BufferChannel.h

Example :Burst Buffer, *Buffer Channel.h*

BufferChannel.h

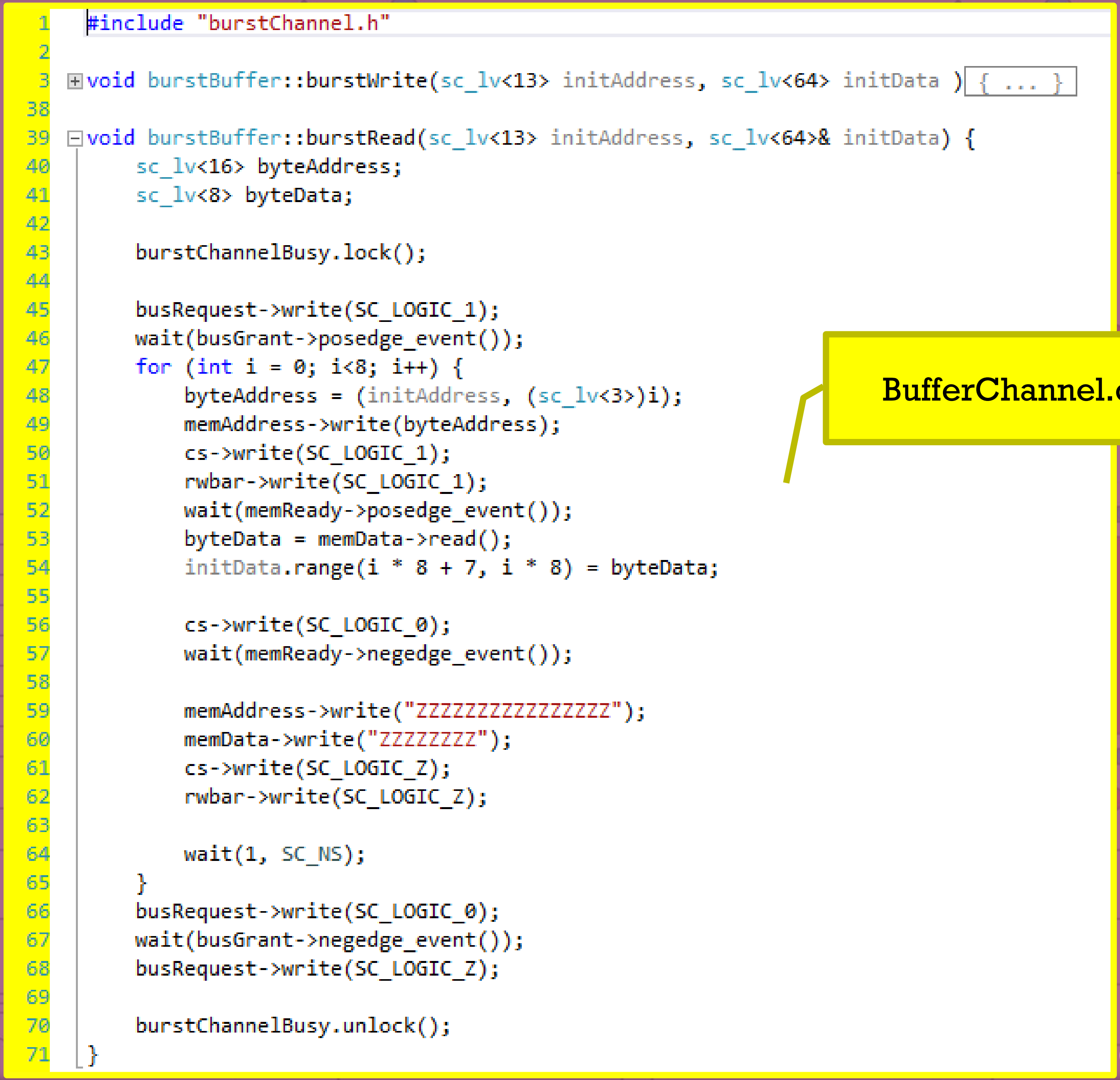


```
#include "interfaceClasses.h"

class burstBuffer : public sc_channel, public burst_write_if, public burst_read_if {

public:
    sc_out_rv<16> memAddress;
    sc_inout_rv<8> memData;
    sc_out_resolved rwbar, cs;
    sc_in_resolved memReady;
    sc_out_resolved busRequest;
    sc_in<sc_logic> busGrant;

    sc_mutex burstChannelBusy;

    burstBuffer (sc_module_name NAME): sc_channel(NAME){};
//  ~burstBuffer(){};

    virtual void burstWrite(sc_lv<13> initAddress, sc_lv<64> initData);
    virtual void burstRead(sc_lv<13> initAddress, sc_lv<64>& initData);
};
```



```
#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

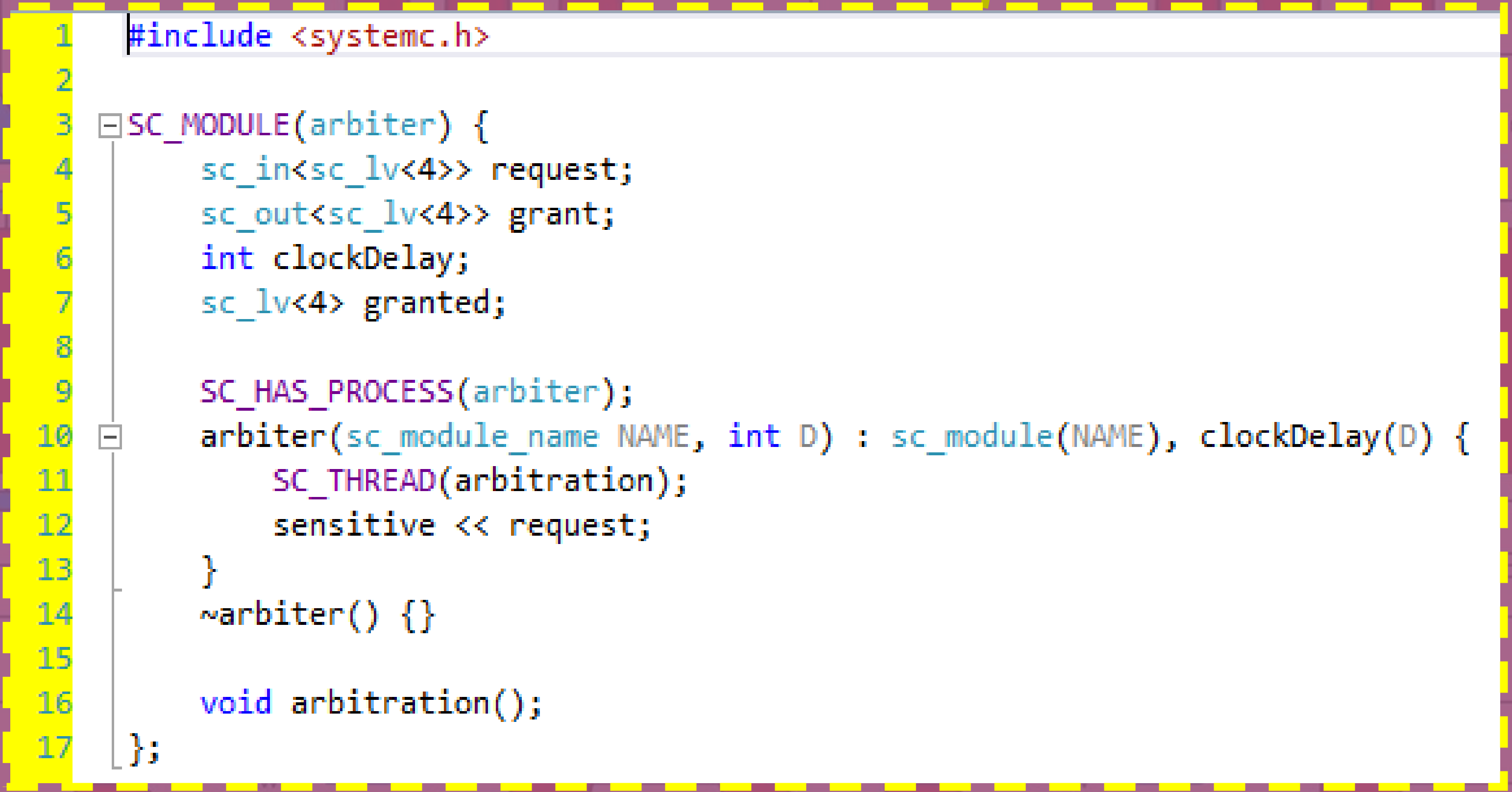


```
#include "burstChannel.h"

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

void burstBuffer::burstRead(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);
        memAddress->write(byteAddress);
        cs->write(SC_LOGIC_1);
        rwbar->write(SC_LOGIC_1);
        wait(memReady->posedge_event());
        byteData = memData->read();
        initData.range(i * 8 + 7, i * 8) = byteData;

        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();
}
```

BufferChannel.cpp

Example :Burst Buffer, *Arbiter.h*

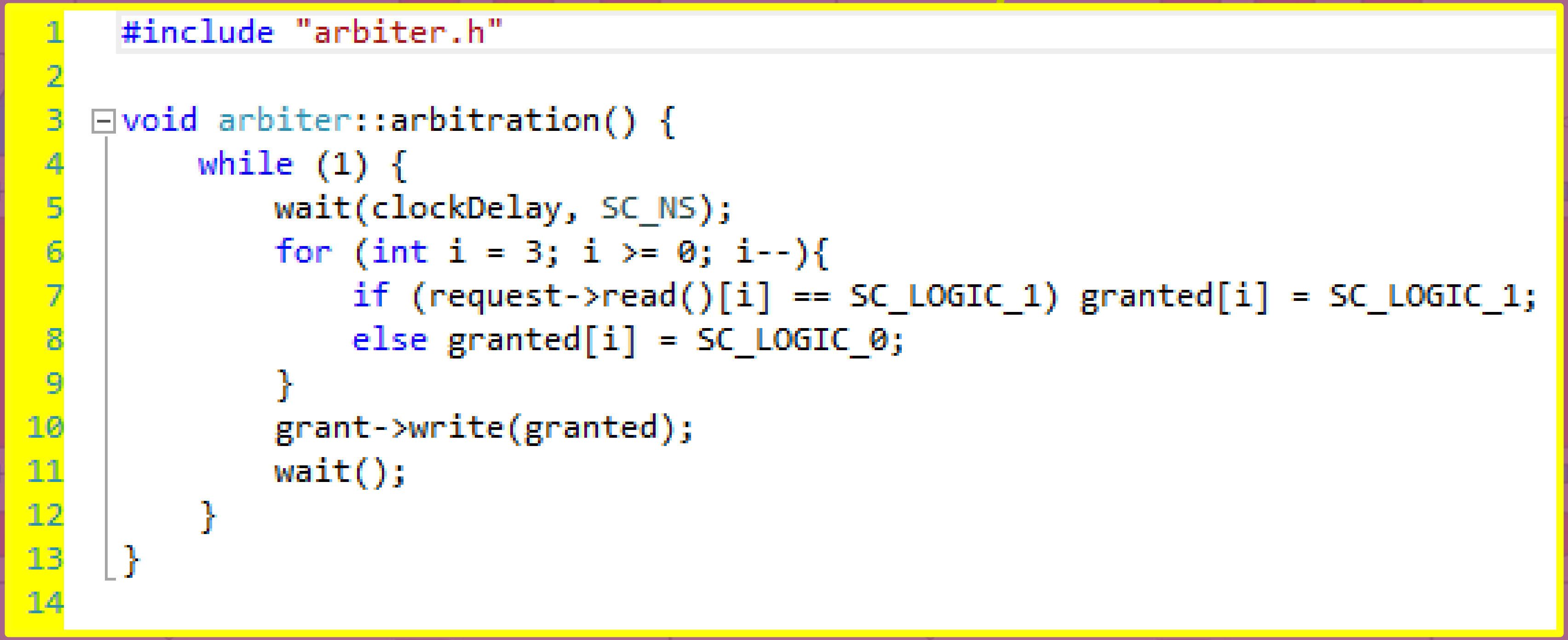
Arbiter.h



```
#include <systemc.h>

SC_MODULE(arbiter) {
    sc_in<sc_lv<4>> request;
    sc_out<sc_lv<4>> grant;
    int clockDelay;
    sc_lv<4> granted;

    SC_HAS_PROCESS(arbiter);
    arbiter(sc_module_name NAME, int D) : sc_module(NAME), clockDelay(D) {
        SC_THREAD(arbitration);
        sensitive << request;
    }
    ~arbiter() {}

    void arbitration();
};
```

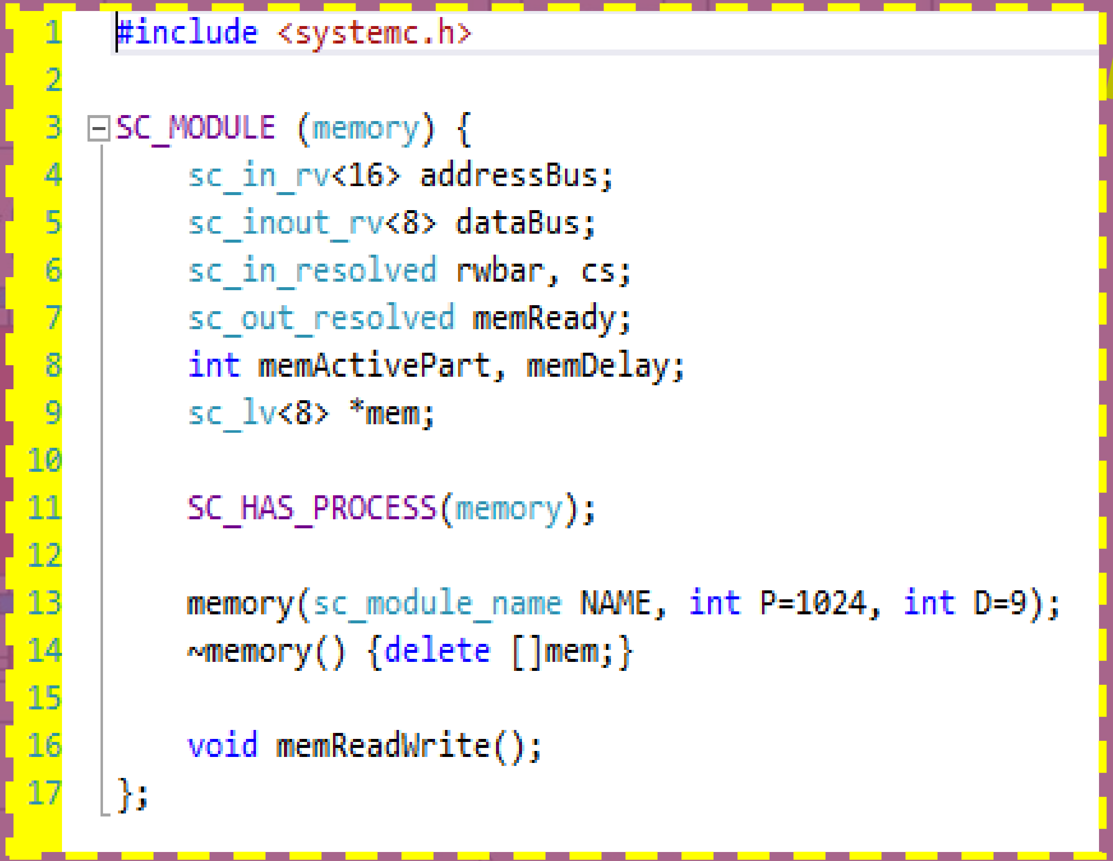
Example :Burst Buffer, *Arbiter.cpp*

Arbiter.cpp



```cpp
#include "arbiter.h"

void arbiter::arbitration() {
    while (1) {
        wait(clockDelay, SC_NS);
        for (int i = 3; i >= 0; i--){
            if (request->read()[i] == SC_LOGIC_1) granted[i] = SC_LOGIC_1;
            else granted[i] = SC_LOGIC_0;
        }
        grant->write(granted);
        wait();
    }
}

```

Example :Burst Buffer, *Memory.h*

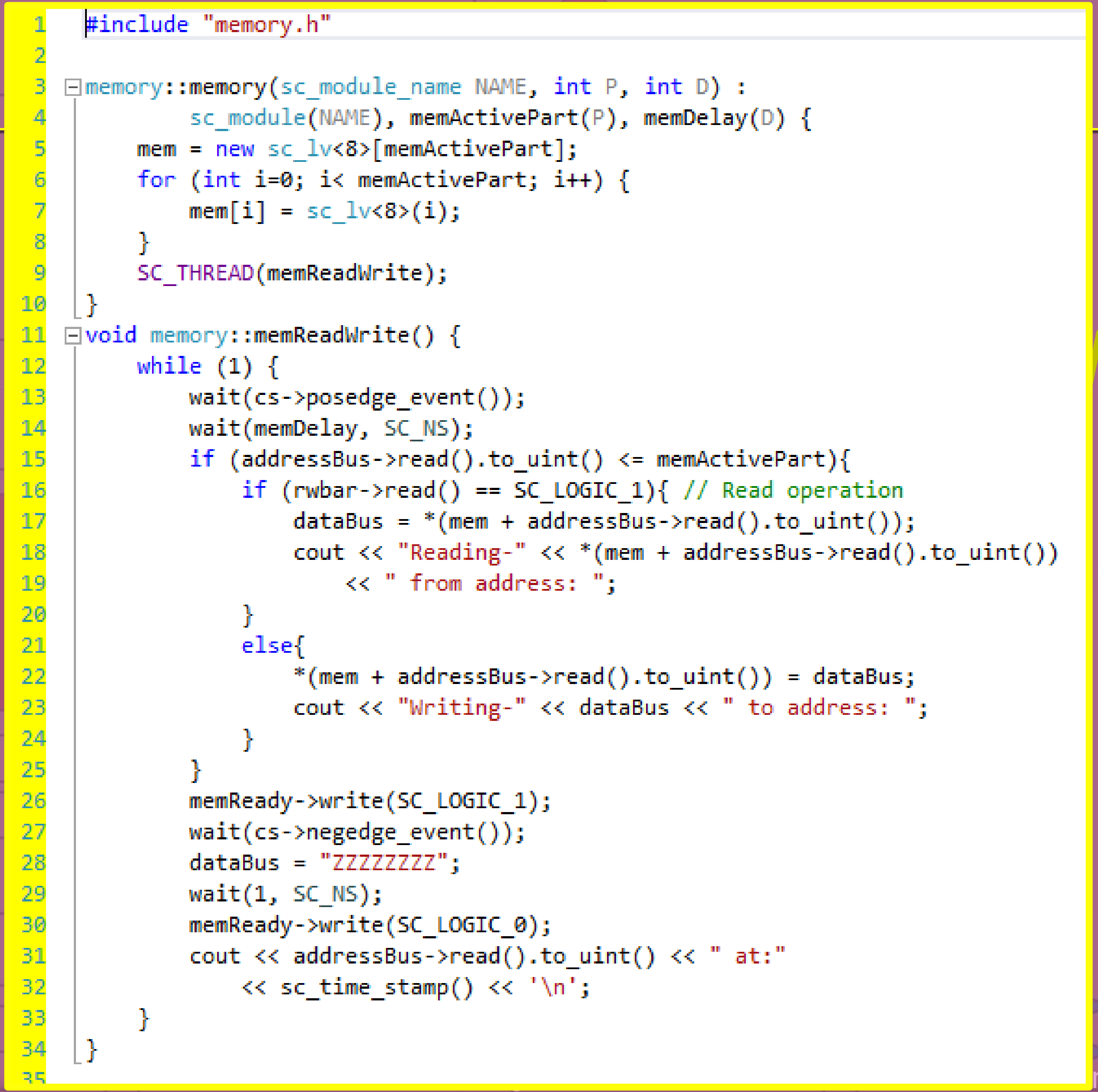


```
#include <systemc.h>

SC_MODULE (memory) {
    sc_in_rv<16> addressBus;
    sc_inout_rv<8> dataBus;
    sc_in_resolved rwbar, cs;
    sc_out_resolved memReady;
    int memActivePart, memDelay;
    sc_lv<8> *mem;

    SC_HAS_PROCESS(memory);

    memory(sc_module_name NAME, int P=1024, int D=9);
    ~memory() {delete []mem;}

    void memReadWrite();
};
```

Memory.h

Example :Burst Buffer,

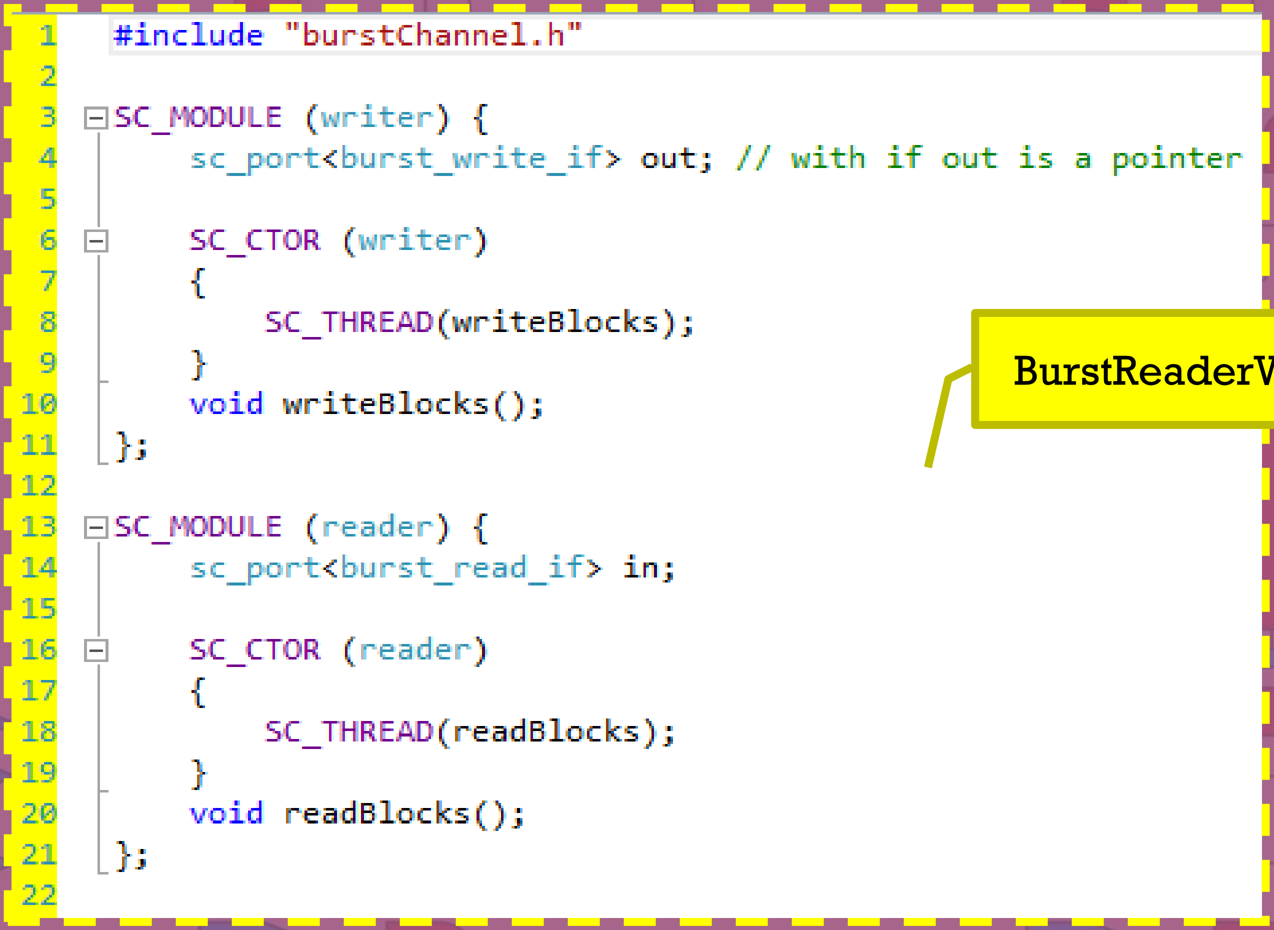
Memory.cpp



```
#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*

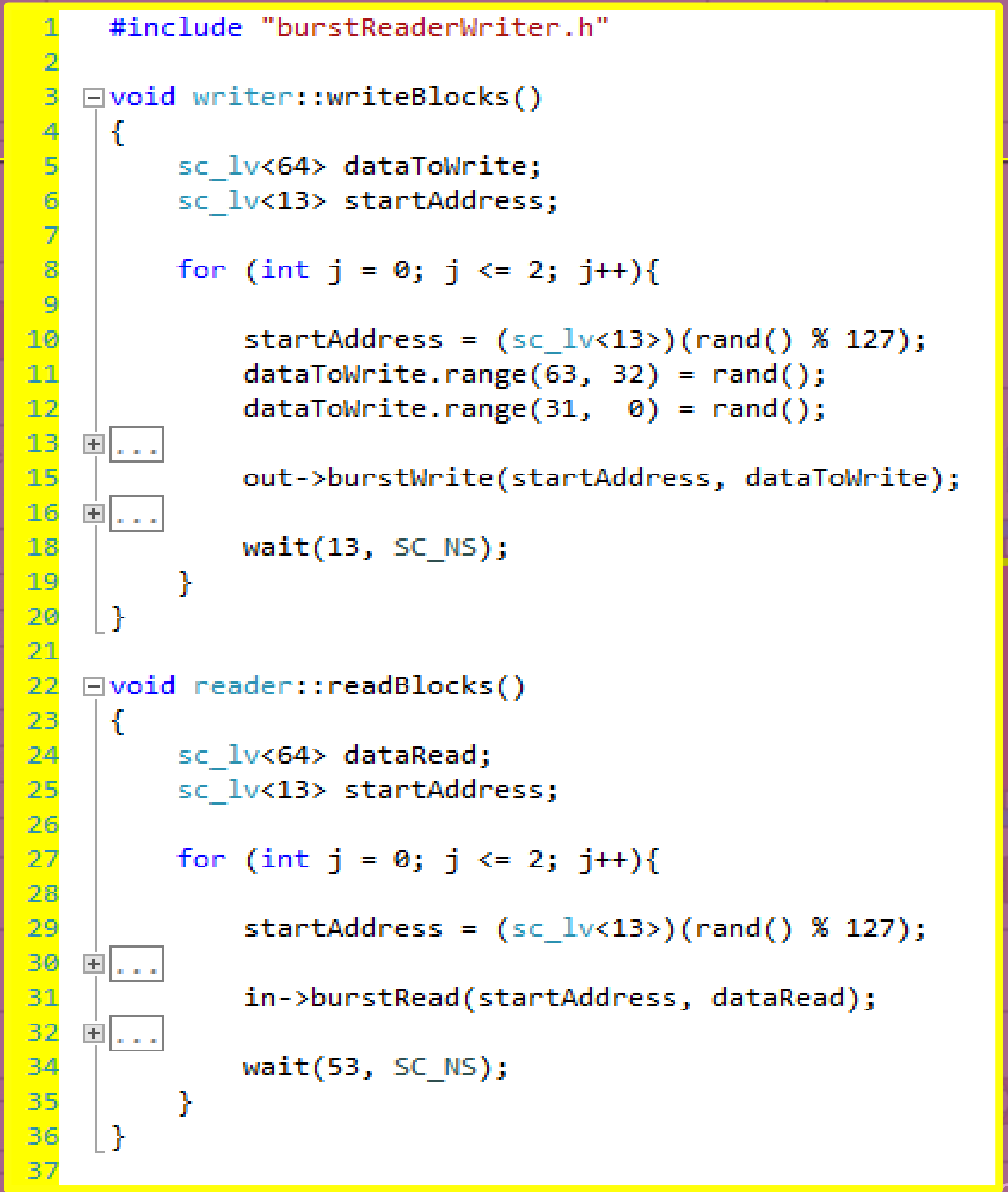


```
#include "burstChannel.h"

SC_MODULE (writer) {
    sc_port<burst_write_if> out; // with if out is a pointer

    SC_CTOR (writer)
    {
        SC_THREAD(writeBlocks);
    }
    void writeBlocks();
};

SC_MODULE (reader) {
    sc_port<burst_read_if> in;

    SC_CTOR (reader)
    {
        SC_THREAD(readBlocks);
    }
    void readBlocks();
};
```

BurstReaderWriter.h

Example :Burst Buffer,

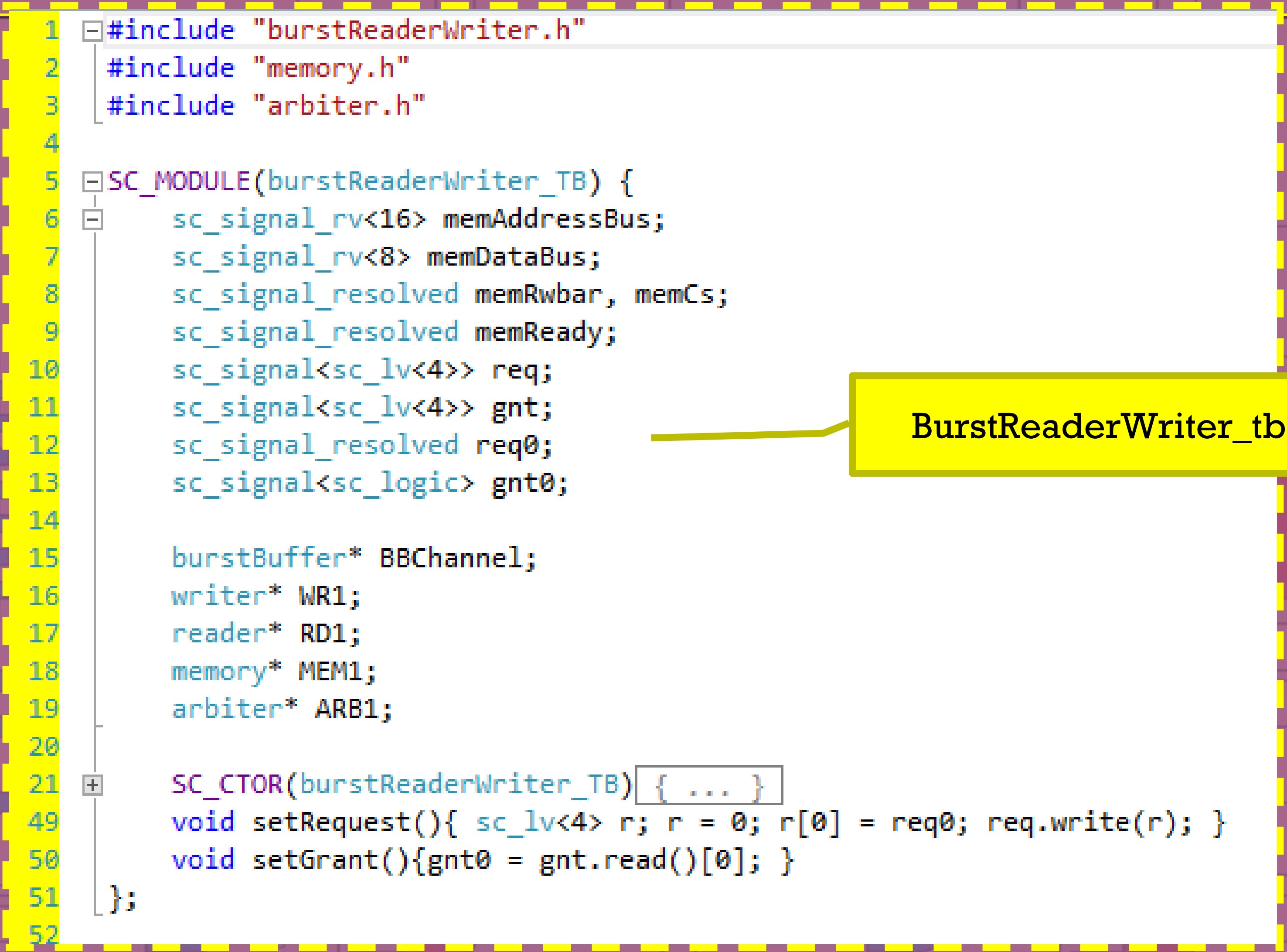
Burst Reader Writer.cpp



```
#include "burstReaderWriter.h"

void writer::writeBlocks()
{
    sc_lv<64> dataToWrite;
    sc_lv<13> startAddress;

    for (int j = 0; j <= 2; j++){

        startAddress = (sc_lv<13>)(rand() % 127);
        dataToWrite.range(63, 32) = rand();
        dataToWrite.range(31,  0) = rand();
...
        out->burstWrite(startAddress, dataToWrite);
...
        wait(13, SC_NS);
    }
}

void reader::readBlocks()
{
    sc_lv<64> dataRead;
    sc_lv<13> startAddress;

    for (int j = 0; j <= 2; j++){

        startAddress = (sc_lv<13>)(rand() % 127);
...
        in->burstRead(startAddress, dataRead);
...
        wait(53, SC_NS);
    }
}

```

BurstReaderWriter.cpp

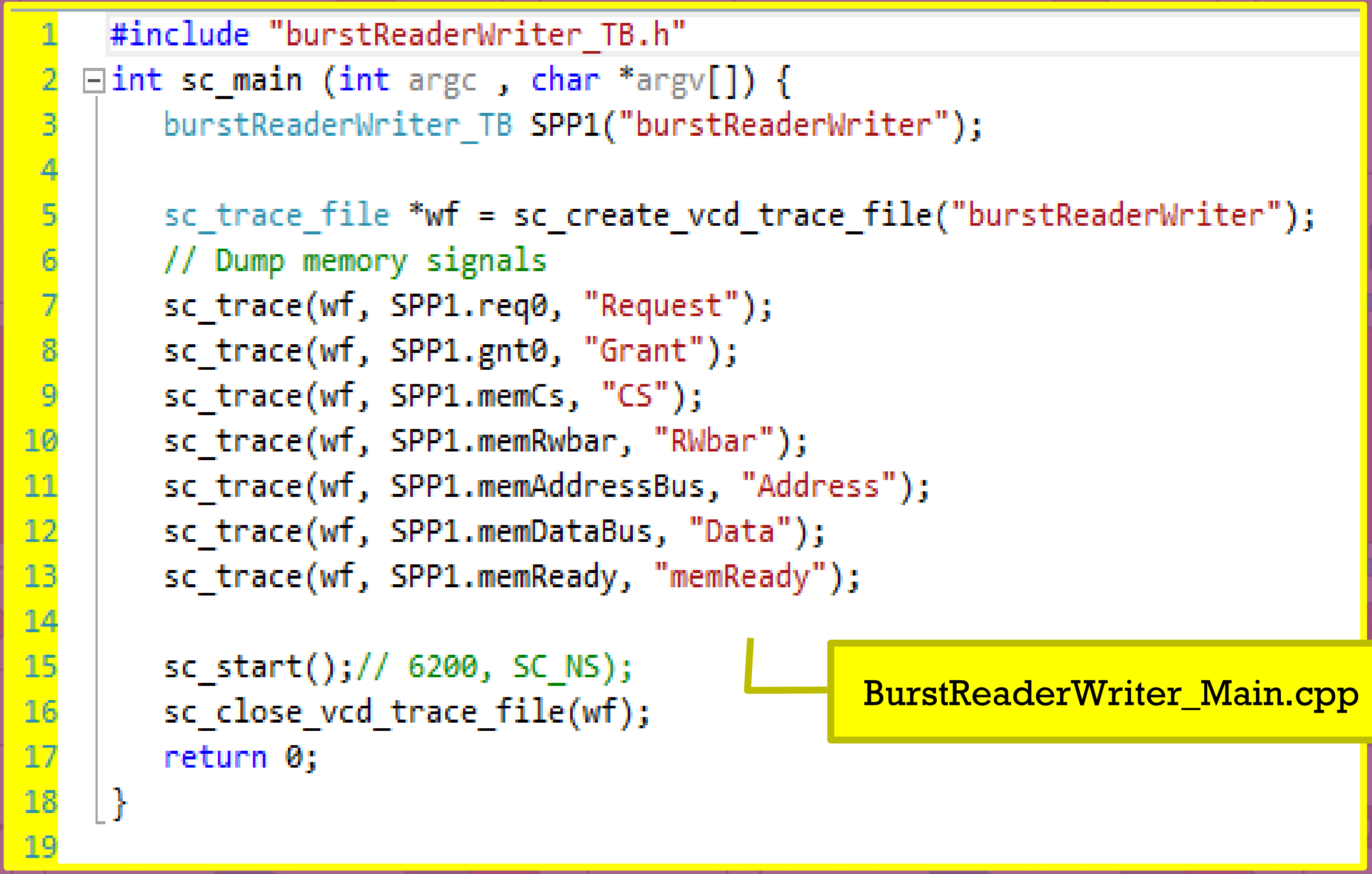
Example :Burst Buffer, *Burst Reader Writer TestBench*



```
#include "burstReaderWriter.h"
#include "memory.h"
#include "arbiter.h"

SC_MODULE(burstReaderWriter_TB) {
    sc_signal_rv<16> memAddressBus;
    sc_signal_rv<8> memDataBus;
    sc_signal_resolved memRwbar, memCs;
    sc_signal_resolved memReady;
    sc_signal<sc_lv<4>> req;
    sc_signal<sc_lv<4>> gnt;
    sc_signal_resolved req0;
    sc_signal<sc_logic> gnt0;

    burstBuffer* BBChannel;
    writer* WR1;
    reader* RD1;
    memory* MEM1;
    arbiter* ARB1;

    SC_CTOR(burstReaderWriter_TB) { ... }
    void setRequest(){ sc_lv<4> r; r = 0; r[0] = req0; req.write(r); }
    void setGrant(){gnt0 = gnt.read()[0]; }
};

```

BurstReaderWriter_tb.h

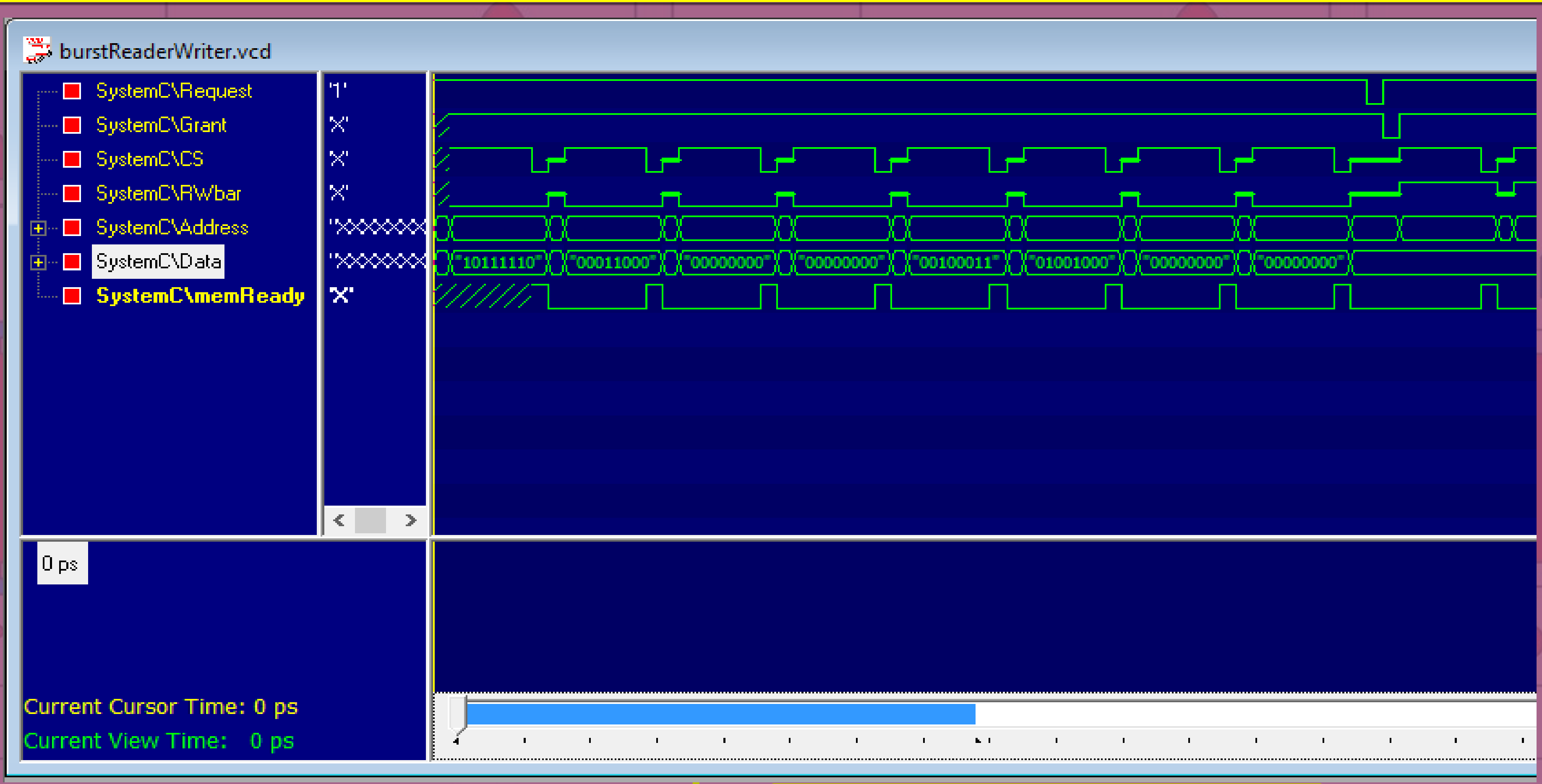
Example :Burst Buffer, *Burst Reader Writer Main.cpp*



```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



Current Cursor Time: 0 ps
Current View Time: 0 ps

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