Chapter 4

RTL SystemC Modeling

Zainalabedin Navabi

Slides prepared by Nooshin Nosrati

SystemC Modeling

✚SystemC Gate-Level Modeling

Utilities for HDL Orientation

✚Timing & SystemC RT-Level Modeling

✚Components for RTL Design

✚SystemC RTL Design - Example

SystemC Functional Modeling

✚SystemC Functional Design - Example

✚Taking Off From C++

Summary

SystemC Modeling

- Taking Off From C++
  - C++ modeling of 1-bit Adder
- SystemC Gate-level Modeling
  - SystemC modeling of 1-bit Adder

Utilities for HDL Orientation

- Timing & SystemC RT-Level Modeling
  - Hierarchical timed design for Serial Adder
- Components for RTL Design
  - Combinational
  - Sequential
- SystemC RTL Design - Examples

  Sequence Detector 11011

  A configurable Memory

  Exponential Circuit

SystemC Functional Modeling

- SystemC Functional Design - Example

  Multiplier

  Divider

  Exponential Circuit

Summary

SystemC Modeling

- Taking Off From C++
  - C++ modeling of 1-bit Adder
    - Components
    - Top-Level
    - Testbench

+ SystemC Gate-Level Modeling

Utilities for HDL Orientation

+ Timing & SystemC RT-Level Modeling

+ Components for RTL Design

+ SystemC RTL Design - Example

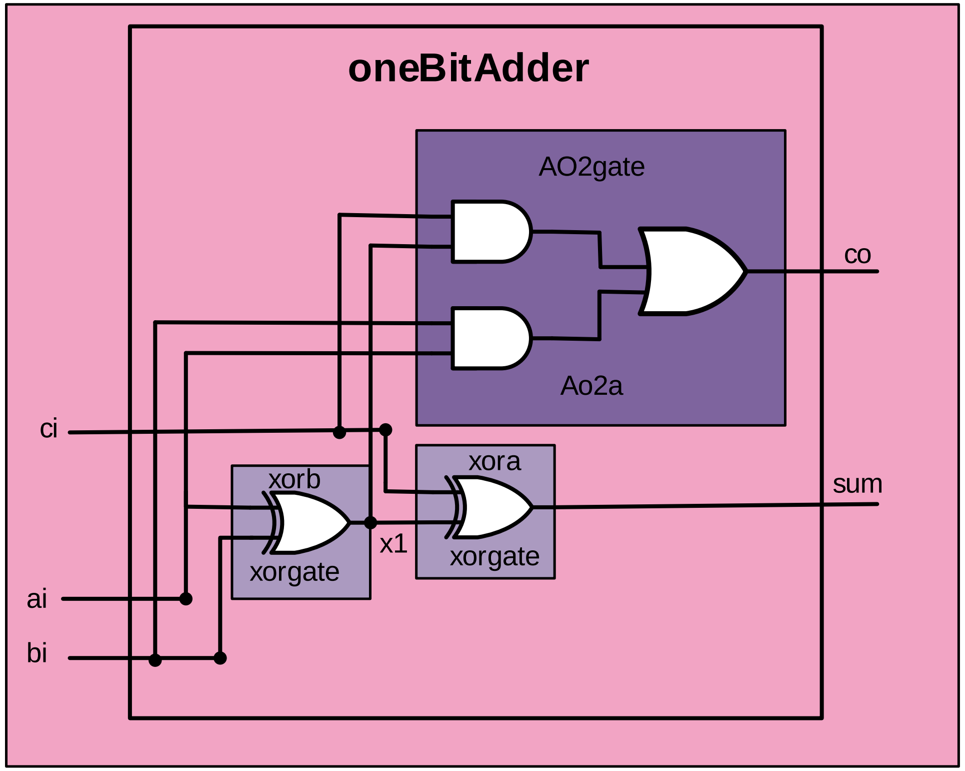
SystemC Functional Modeling

+ SystemC Functional Design - Example

Summary

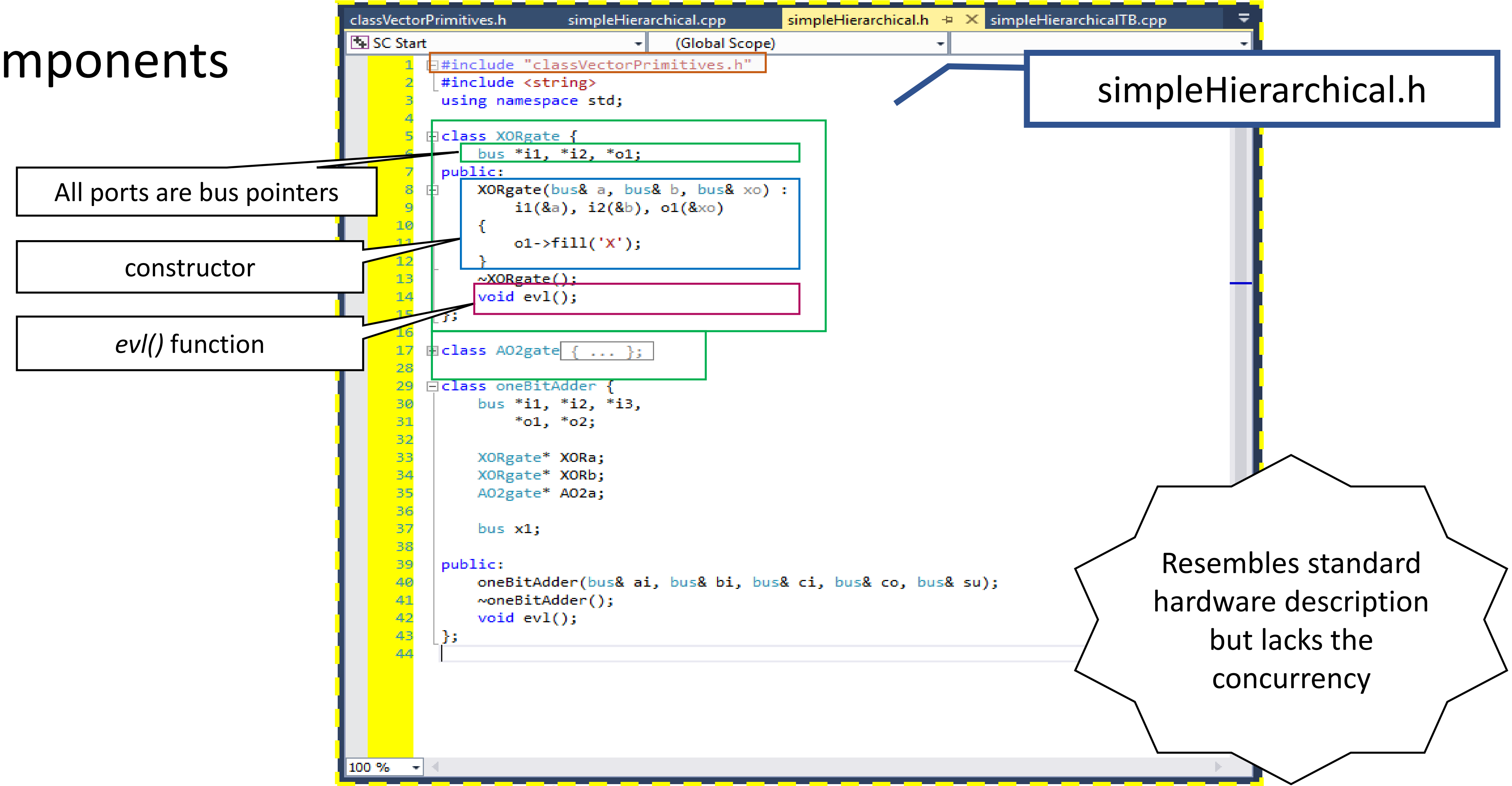
C++ modeling of 1-bit Adder

- Hierarchical Structure Hardware



C++ modeling of 1-bit Adder

- Components



```
#include "classVectorPrimitives.h"
#include <string>
using namespace std;

class XORgate {
    bus *i1, *i2, *o1;
public:
    XORgate(bus& a, bus& b, bus& xo) :
        i1(&a), i2(&b), o1(&xo)
    {
        o1->fill('X');
    }
    ~XORgate();
    void evl();
};

class AO2gate{ ... };

class oneBitAdder {
    bus *i1, *i2, *i3,
        *o1, *o2;

    XORgate* XORa;
    XORgate* XORb;
    AO2gate* AO2a;

    bus x1;

public:
    oneBitAdder(bus& ai, bus& bi, bus& ci, bus& co, bus& su);
    ~oneBitAdder();
    void evl();
};
```

simpleHierarchical.h

All ports are bus pointers

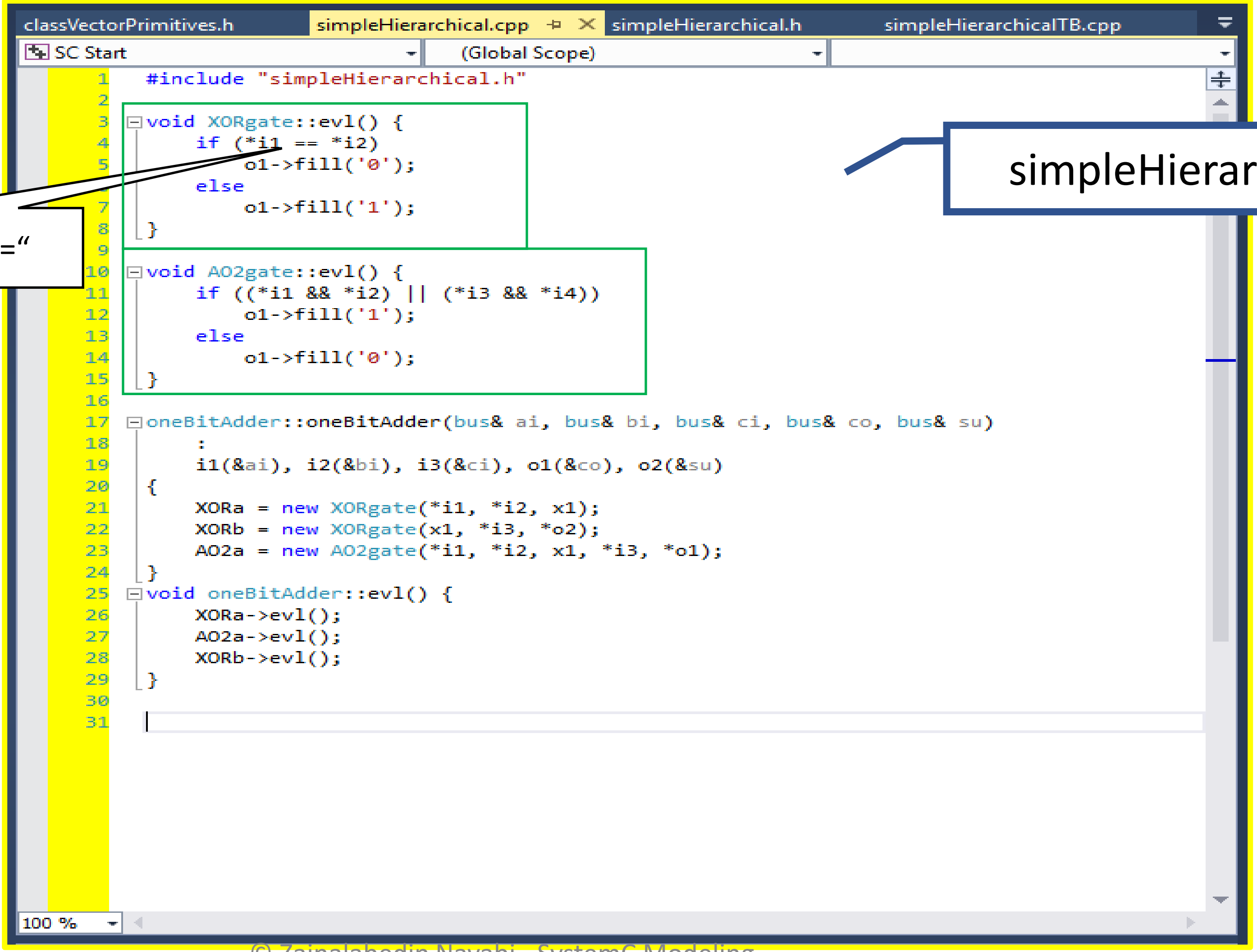
constructor

evl() function

Resembles standard hardware description but lacks the concurrency

C++ modeling of 1-bit Adder

- Components



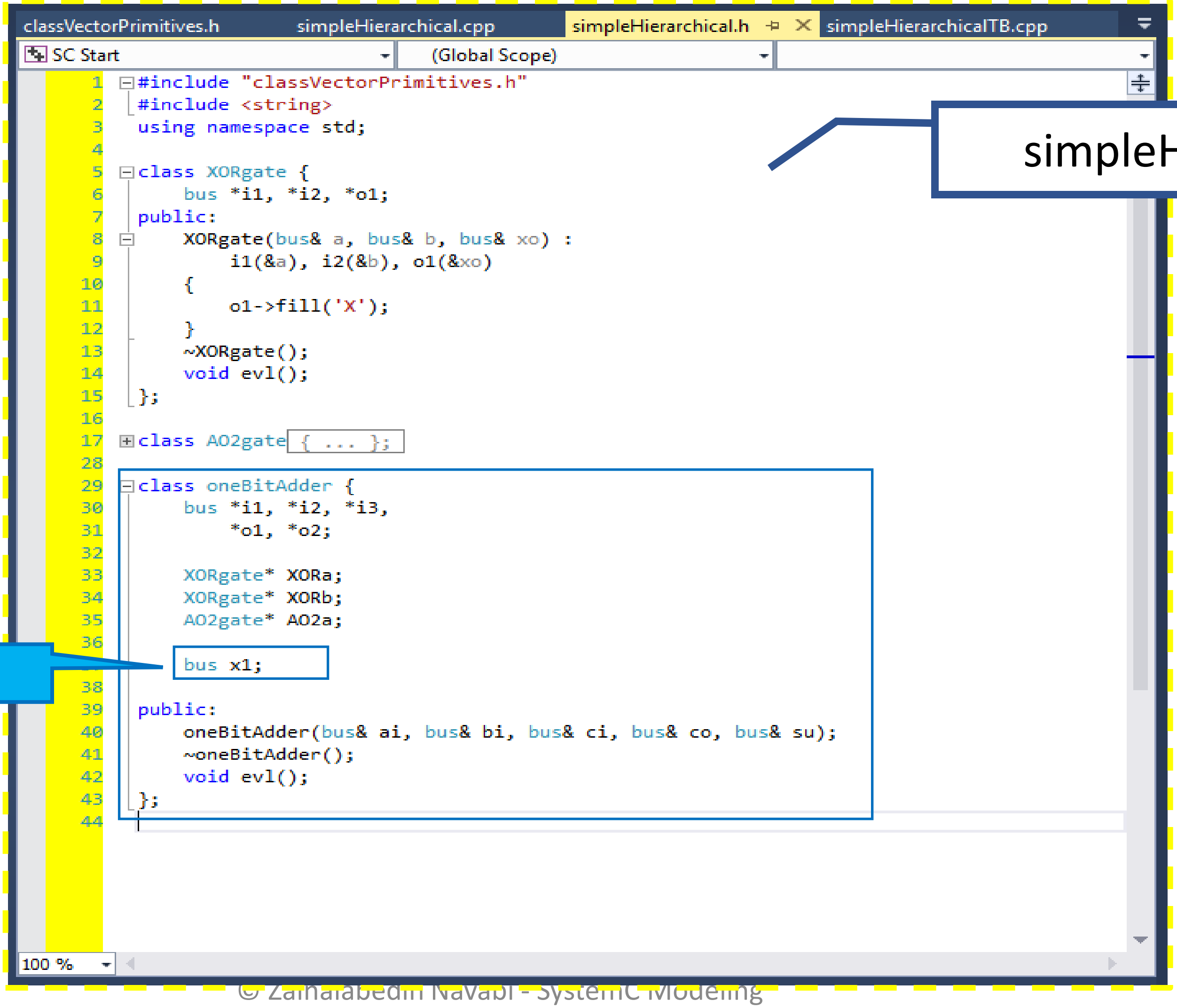
```
#include "simpleHierarchical.h"

void XORgate::evl() {
    if (*i1 == *i2)
        o1->fill('0');
    else
        o1->fill('1');
}

void AO2gate::evl() {
    if ((*i1 && *i2) || (*i3 && *i4))
        o1->fill('1');
    else
        o1->fill('0');
}

oneBitAdder::oneBitAdder(bus& ai, bus& bi, bus& ci, bus& co, bus& su)
    :
    i1(&ai), i2(&bi), i3(&ci), o1(&co), o2(&su)
{
    XORa = new XORgate(*i1, *i2, x1);
    XORb = new XORgate(x1, *i3, *o2);
    AO2a = new AO2gate(*i1, *i2, x1, *i3, *o1);
}
void oneBitAdder::evl() {
    XORa->evl();
    AO2a->evl();
    XORb->evl();
}

```

Use overloaded “==“

simpleHierarchical.cpp

C++ modeling of 1-bit Adder

- Top-Level



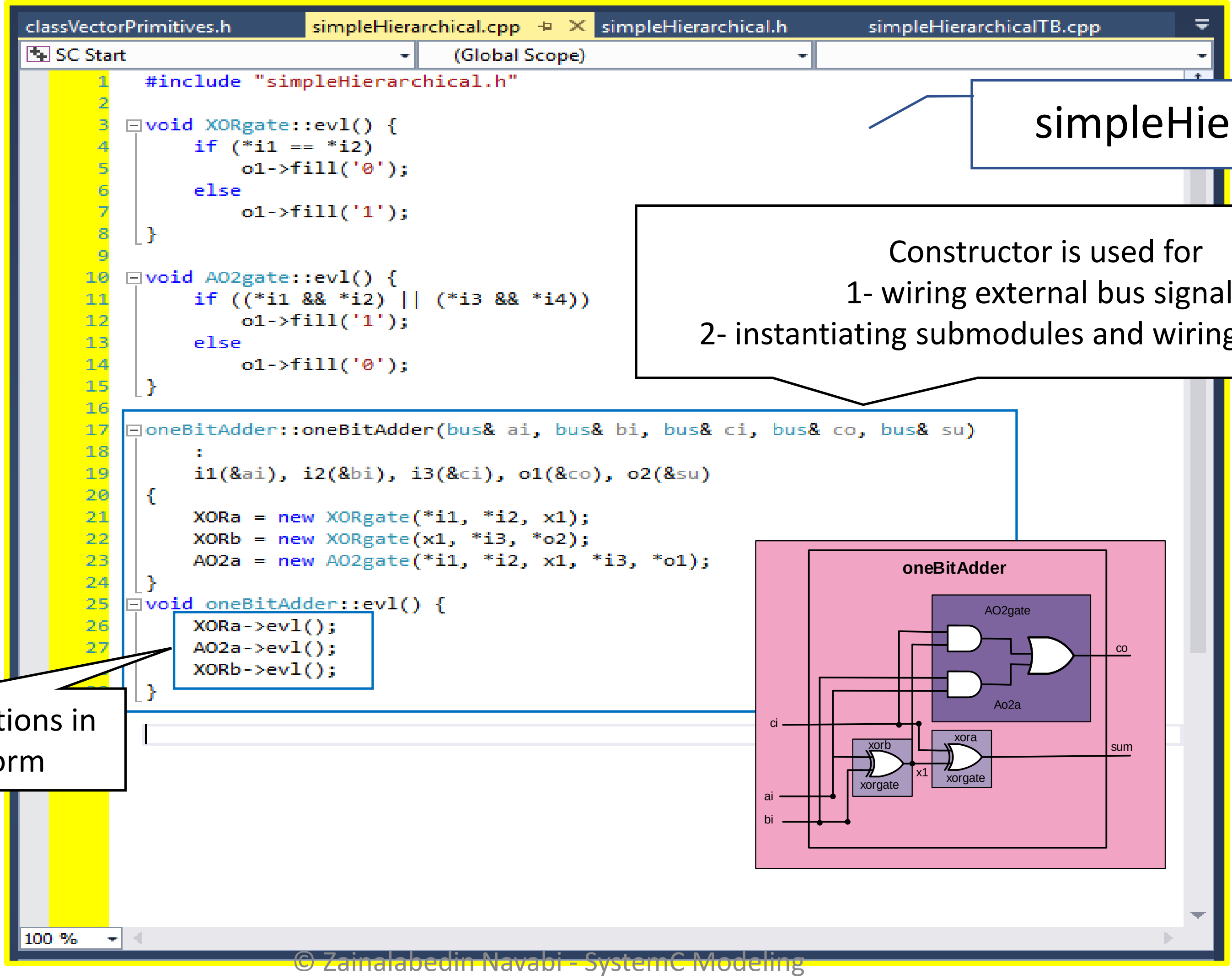
```
#include "classVectorPrimitives.h"
#include <string>
using namespace std;

class XORgate {
    bus *i1, *i2, *o1;
public:
    XORgate(bus& a, bus& b, bus& xo) :
        i1(&a), i2(&b), o1(&xo)
    {
        o1->fill('X');
    }
    ~XORgate();
    void evl();
};

class AO2gate { ... };

class oneBitAdder {
    bus *i1, *i2, *i3,
        *o1, *o2;

    XORgate* XORa;
    XORgate* XORb;
    AO2gate* AO2a;

    bus x1;

public:
    oneBitAdder(bus& ai, bus& bi, bus& ci, bus& co, bus& su);
    ~oneBitAdder();
    void evl();
};

```

simpleHierarchical.h

Internal signal

C++ modeling of 1-bit Adder

- Top-Level



```
#include "simpleHierarchical.h"

void XORgate::evl() {
    if (*i1 == *i2)
        o1->fill('0');
    else
        o1->fill('1');
}

void AO2gate::evl() {
    if ((*i1 && *i2) || (*i3 && *i4))
        o1->fill('1');
    else
        o1->fill('0');
}

oneBitAdder::oneBitAdder(bus& ai, bus& bi, bus& ci, bus& co, bus& su)
    :
    i1(&ai), i2(&bi), i3(&ci), o1(&co), o2(&su)
{
    XORa = new XORgate(*i1, *i2, x1);
    XORb = new XORgate(x1, *i3, *o2);
    AO2a = new AO2gate(*i1, *i2, x1, *i3, *o1);
}
void oneBitAdder::evl() {
    XORa->evl();
    AO2a->evl();
    XORb->evl();
}
```

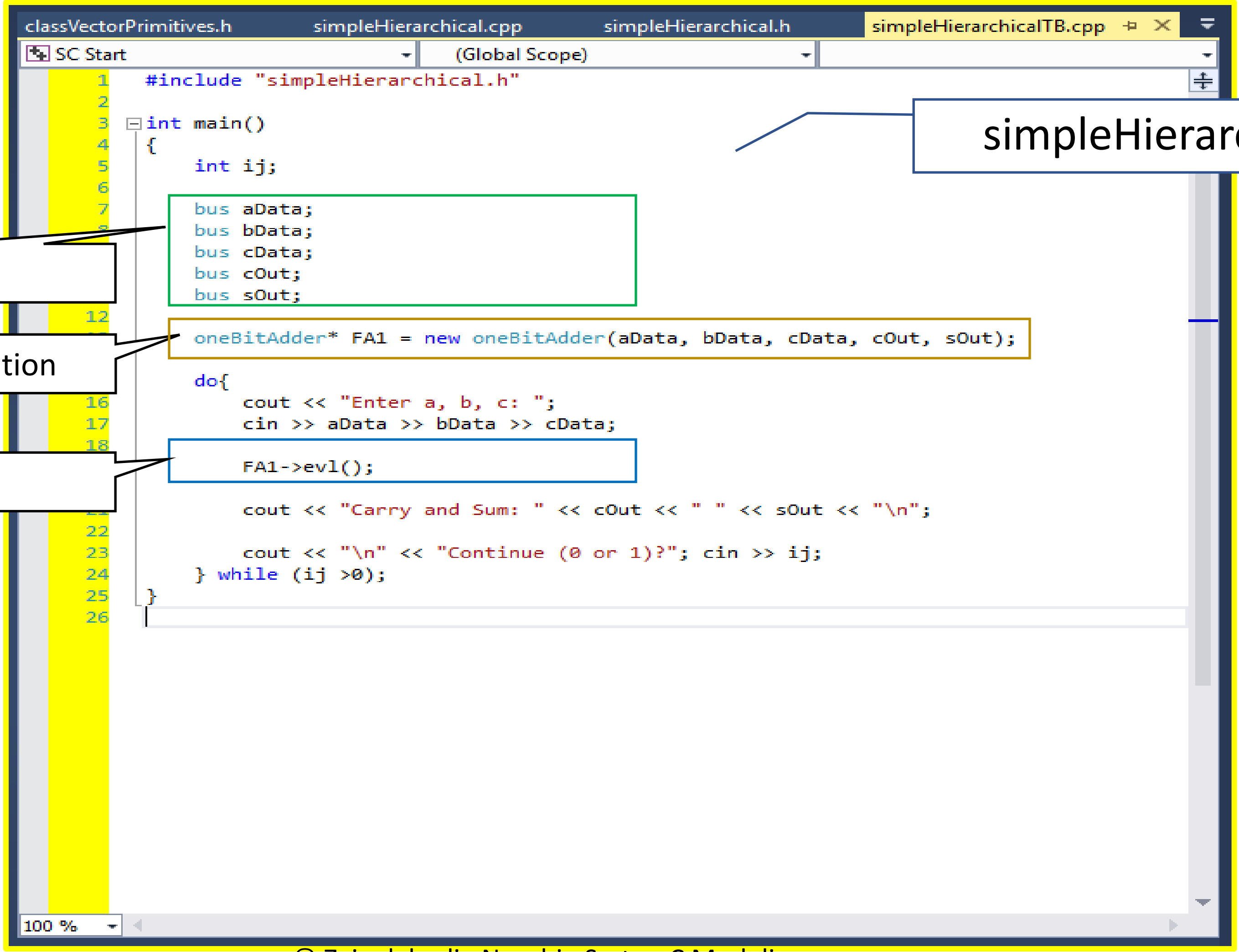
simpleHierarchical.cpp

Constructor is used for
1- wiring external bus signals
2- instantiating submodules and wiring their ports

Calling *evl()* functions in an ordered form

C++ modeling of 1-bit Adder

- Testbench



```
#include "simpleHierarchical.h"

int main()
{
    int ij;

    bus aData;
    bus bData;
    bus cData;
    bus cOut;
    bus sOut;

    oneBitAdder* FA1 = new oneBitAdder(aData, bData, cData, cOut, sOut);

    do{
        cout << "Enter a, b, c: ";
        cin >> aData >> bData >> cData;

        FA1->evl();

        cout << "Carry and Sum: " << cOut << " " << sOut << "\n";

        cout << "\n" << "Continue (0 or 1)?"; cin >> ij;
    } while (ij >0);
}
```

simpleHierarchicalTB.cpp

Local buses

Top-Level instantiation

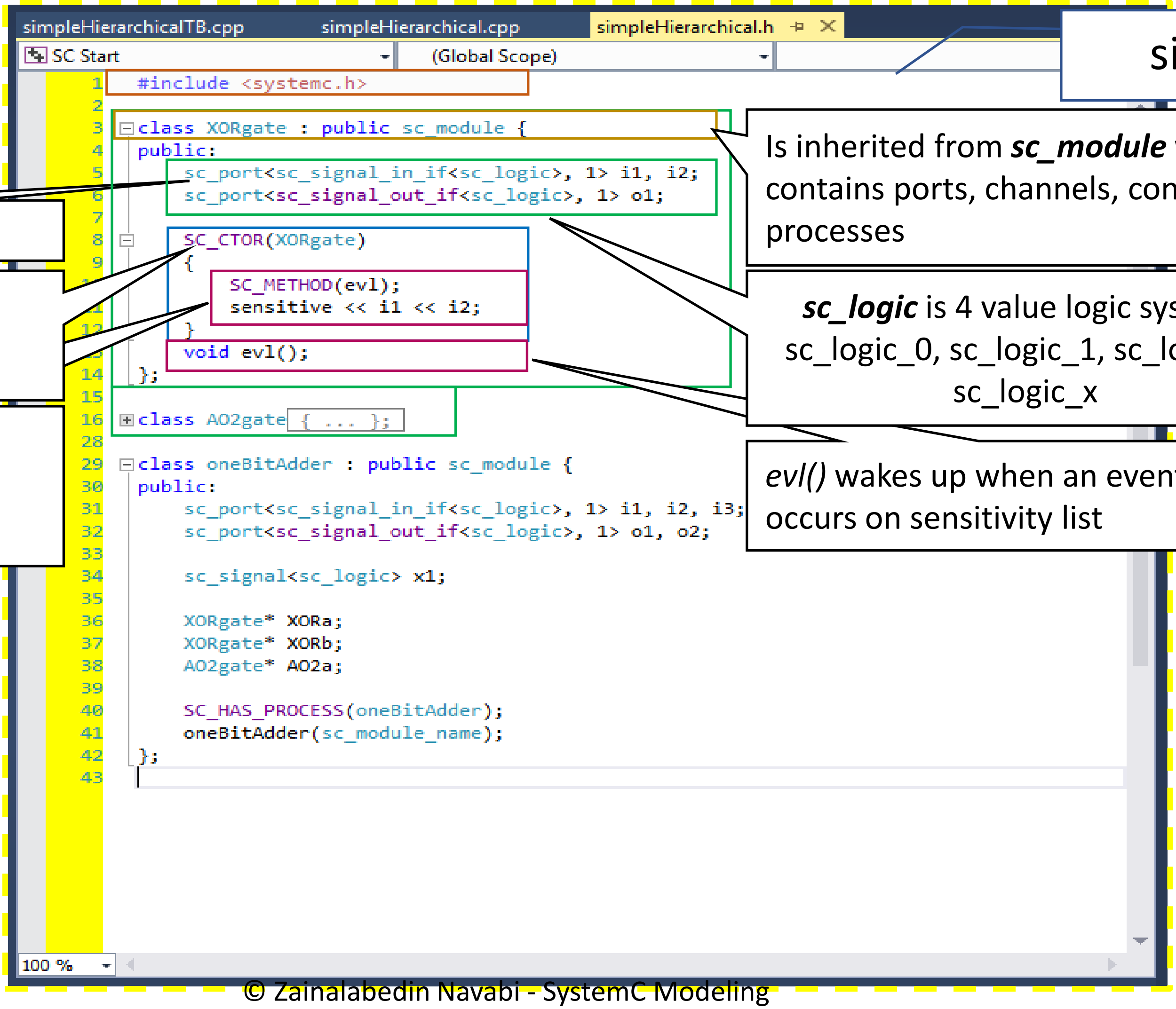
Calling *evl()* function

SystemC Modeling

- + Taking Off From C++
- – SystemC Gate-Level Modeling
    - – SystemC modeling of 1-bit Adder
        - Components
        - Top-Level
        - Testbench
- Utilities for HDL Orientation
- + Timing & SystemC RT-Level Modeling
- + Components for RTL Design
- + SystemC RTL Design - Example
- SystemC Functional Modeling
- + SystemC Functional Design - Example
- Summary

SystemC modeling of 1-bit Adder

- Components



```
#include <systemc.h>

class XORgate : public sc_module {
public:
    sc_port<sc_signal_in_if<sc_logic>, 1> i1, i2;
    sc_port<sc_signal_out_if<sc_logic>, 1> o1;

    SC_CTOR(XORgate)
    {
        SC_METHOD(evl);
        sensitive << i1 << i2;
    }
    void evl();
};

class AO2gate { ... };

class oneBitAdder : public sc_module {
public:
    sc_port<sc_signal_in_if<sc_logic>, 1> i1, i2, i3;
    sc_port<sc_signal_out_if<sc_logic>, 1> o1, o2;

    sc_signal<sc_logic> x1;

    XORgate* XORa;
    XORgate* XORb;
    AO2gate* AO2a;

    SC_HAS_PROCESS(oneBitAdder);
    oneBitAdder(sc_module_name);
};

```

simpleHierarchical.h

All ports are ***sc_port***

CS_CTOR macro allows inline definition of constructor

evl() function is registered as ***sc_method*** that also run once at the beginning of the simulation

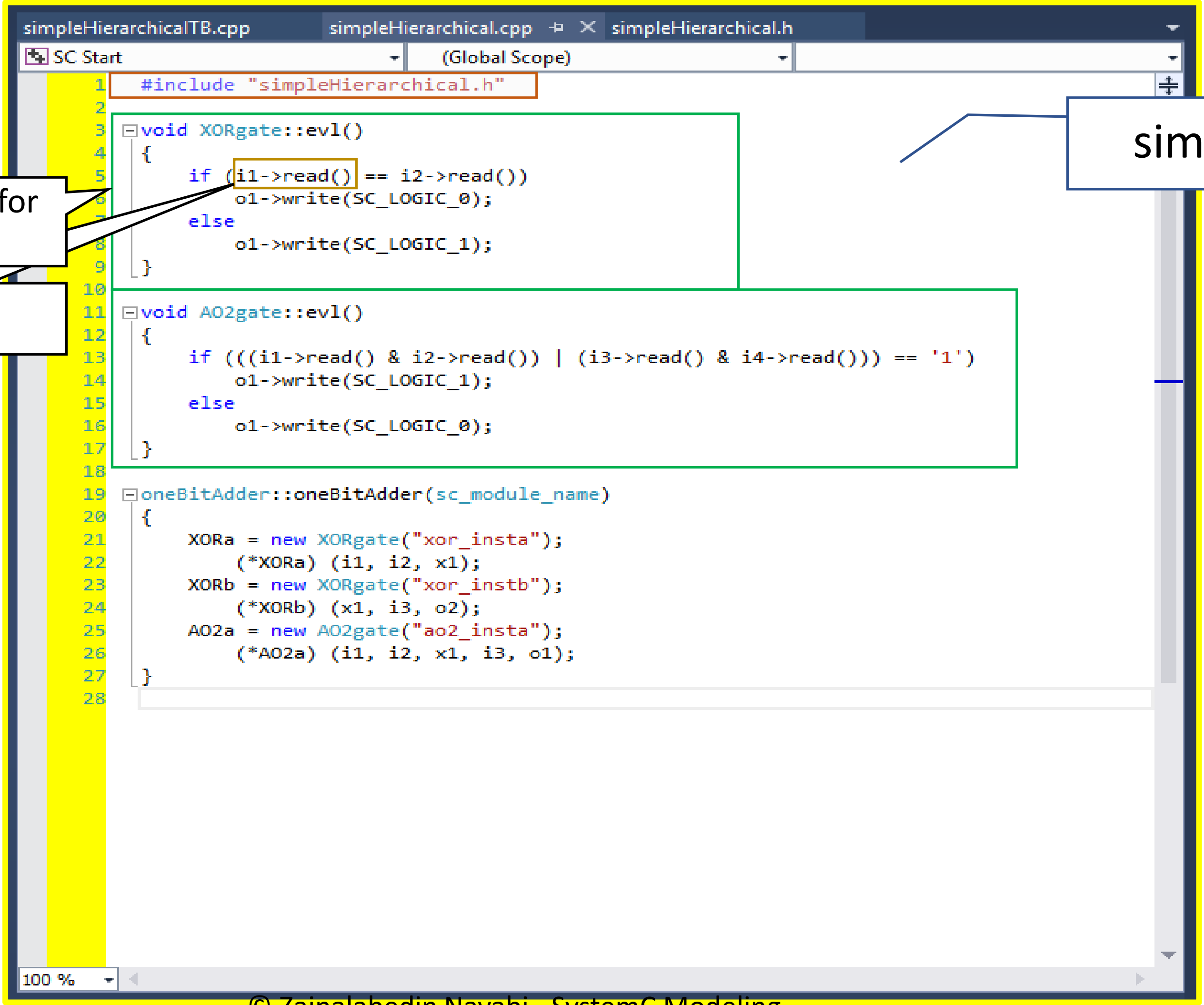
Is inherited from ***sc_module*** which contains ports, channels, concurrent processes

sc_logic is 4 value logic system: sc_logic_0, sc_logic_1, sc_logic_z, sc_logic_x

evl() wakes up when an event occurs on sensitivity list

SystemC modeling of 1-bit Adder

- Components



```
#include "simpleHierarchical.h"

void XORgate::evl()
{
    if (i1->read() == i2->read())
        o1->write(SC_LOGIC_0);
    else
        o1->write(SC_LOGIC_1);
}

void AO2gate::evl()
{
    if (((i1->read() & i2->read()) | (i3->read() & i4->read())) == '1')
        o1->write(SC_LOGIC_1);
    else
        o1->write(SC_LOGIC_0);
}

oneBitAdder::oneBitAdder(sc_module_name)
{
    XORa = new XORgate("xor_insta");
        (*XORa) (i1, i2, x1);
    XORb = new XORgate("xor_instb");
        (*XORb) (x1, i3, o2);
    AO2a = new AO2gate("ao2_insta");
        (*AO2a) (i1, i2, x1, i3, o1);
}
```

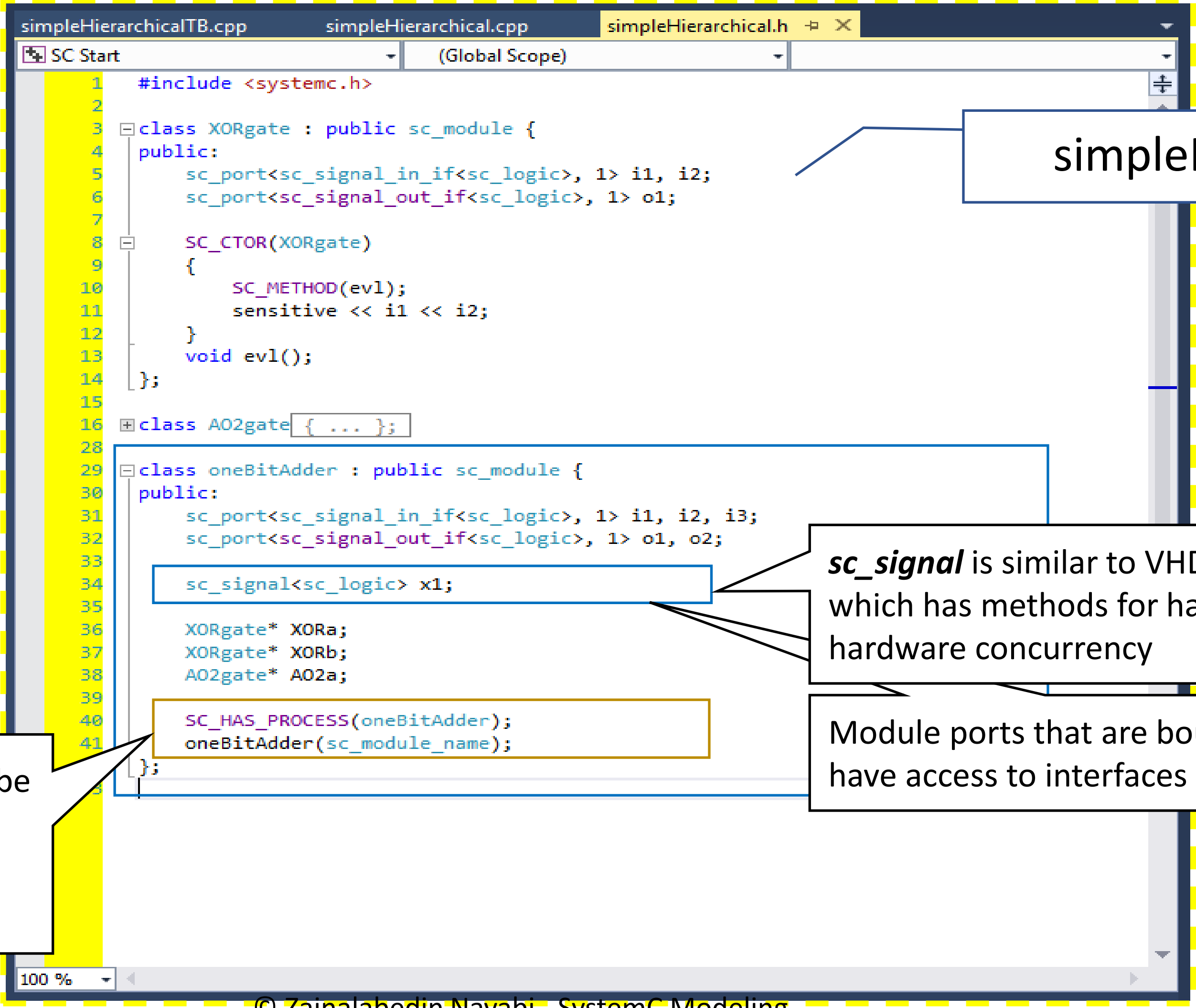
Operators are overloaded for ***sc_logic***

Other form: (*i1).read()

simpleHierarchical.cpp

SystemC modeling of 1-bit Adder

- Top-Level



```
#include <systemc.h>

class XORgate : public sc_module {
public:
    sc_port<sc_signal_in_if<sc_logic>, 1> i1, i2;
    sc_port<sc_signal_out_if<sc_logic>, 1> o1;

    SC_CTOR(XORgate)
    {
        SC_METHOD(evl);
        sensitive << i1 << i2;
    }
    void evl();
};

class AO2gate { ... };

class oneBitAdder : public sc_module {
public:
    sc_port<sc_signal_in_if<sc_logic>, 1> i1, i2, i3;
    sc_port<sc_signal_out_if<sc_logic>, 1> o1, o2;

    sc_signal<sc_logic> x1;

    XORgate* XORa;
    XORgate* XORb;
    AO2gate* AO2a;

    SC_HAS_PROCESS(oneBitAdder);
    oneBitAdder(sc_module_name);
};
```

simpleHierarchical.h

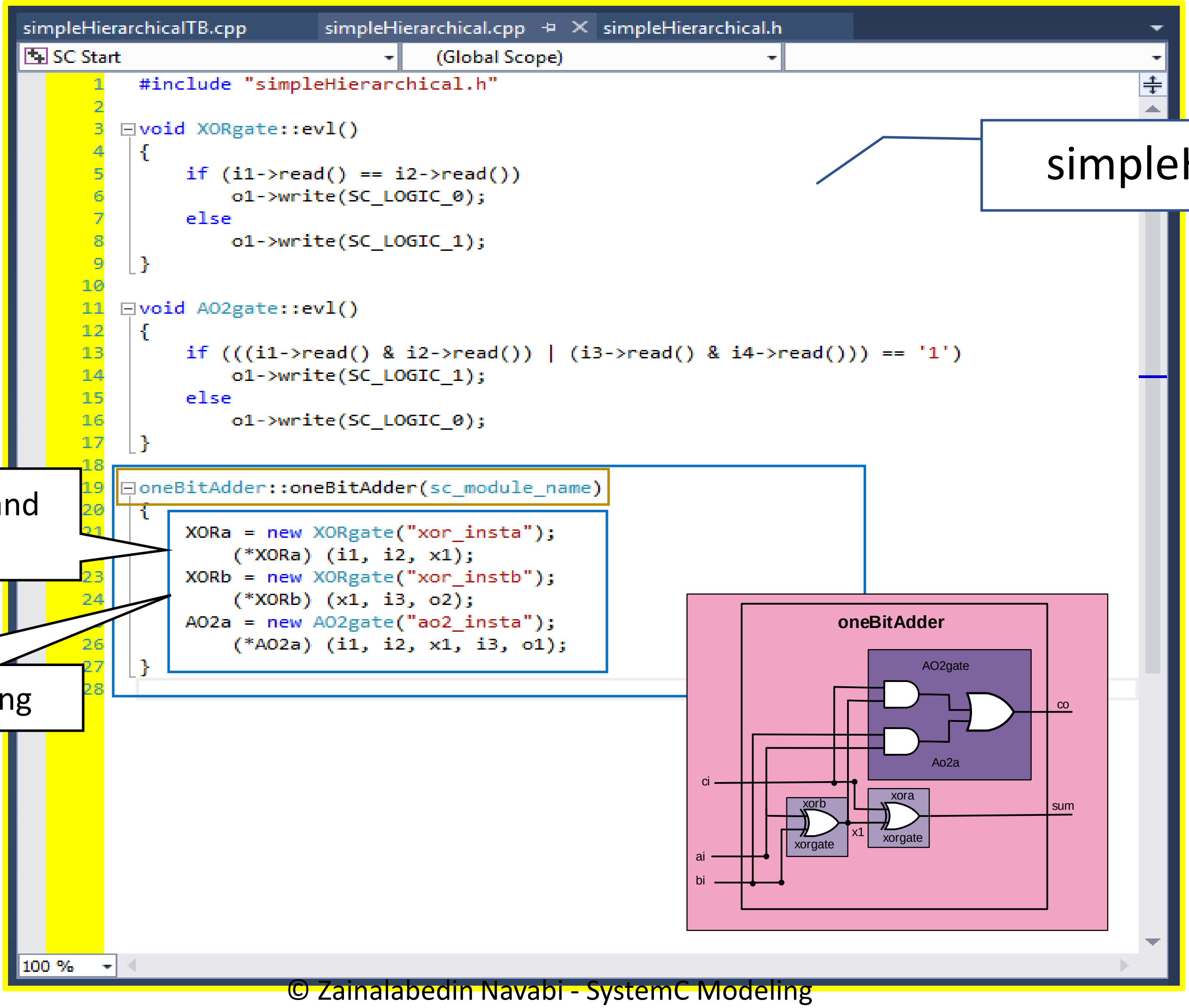
sc_signal is similar to VHDL signals which has methods for handling hardware concurrency

Module ports that are bound to x1 have access to interfaces

SC_HAS_PROCESS constructor can be separate from module declaration and other arguments also can be passed

SystemC modeling of 1-bit Adder

- Top-Level



```
#include "simpleHierarchical.h"

void XORgate::evl()
{
    if (i1->read() == i2->read())
        o1->write(SC_LOGIC_0);
    else
        o1->write(SC_LOGIC_1);
}

void AO2gate::evl()
{
    if (((i1->read() & i2->read()) | (i3->read() & i4->read())) == '1')
        o1->write(SC_LOGIC_1);
    else
        o1->write(SC_LOGIC_0);
}

oneBitAdder::oneBitAdder(sc_module_name)
{
    XORa = new XORgate("xor_insta");
        (*XORa) (i1, i2, x1);
    XORb = new XORgate("xor_instb");
        (*XORb) (x1, i3, o2);
    AO2a = new AO2gate("ao2_insta");
        (*AO2a) (i1, i2, x1, i3, o1);
}
```

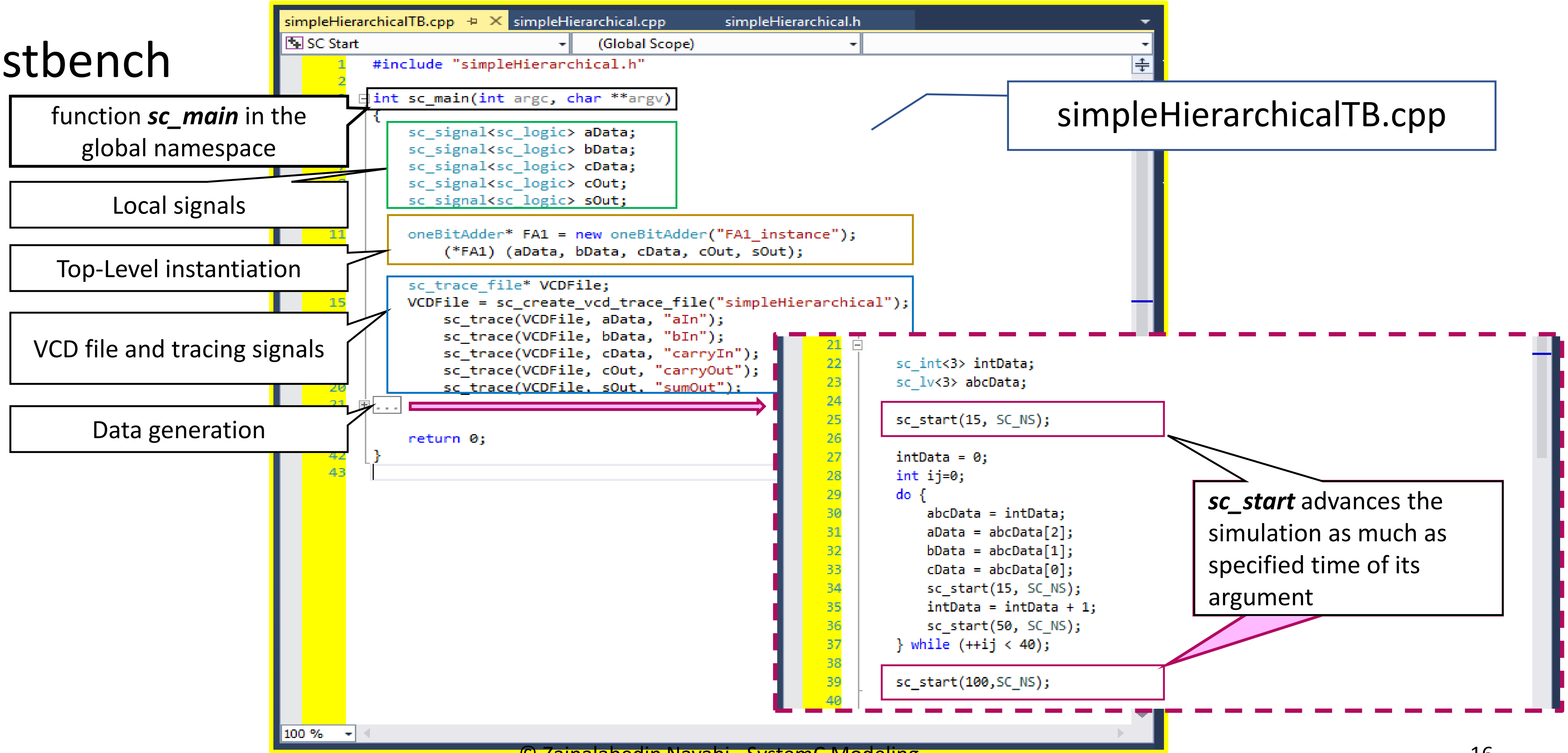
simpleHierarchical.cpp

Submodule instantiation and binding by position

No need to handle ordering

SystemC modeling of 1-bit Adder

- Testbench



simpleHierarchicalTB.cpp

function ***sc_main*** in the global namespace

Local signals

Top-Level instantiation

VCD file and tracing signals

Data generation

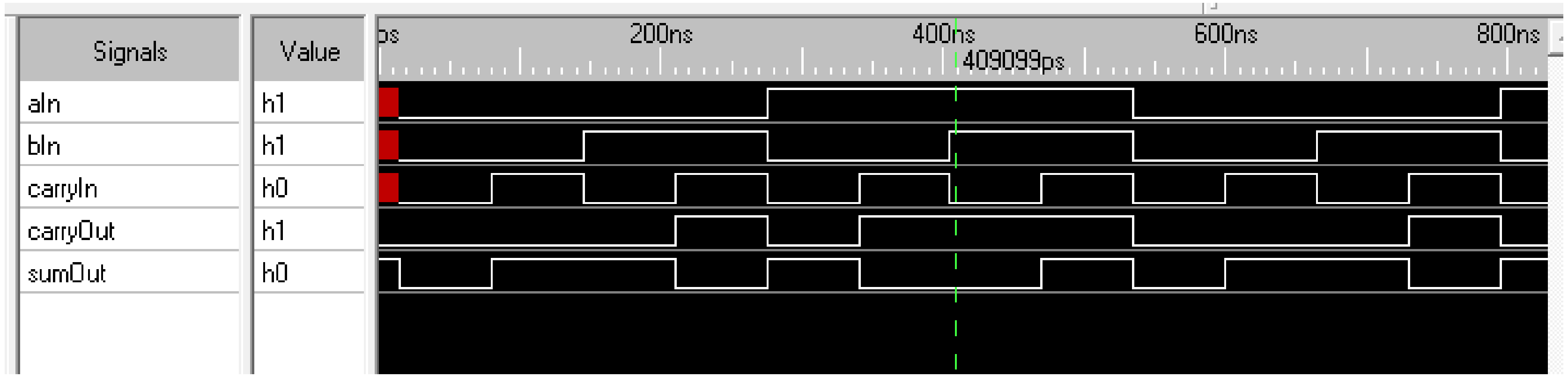
```
#include "simpleHierarchical.h"

int sc_main(int argc, char **argv)
{
    sc_signal<sc_logic> aData;
    sc_signal<sc_logic> bData;
    sc_signal<sc_logic> cData;
    sc_signal<sc_logic> cOut;
    sc_signal<sc_logic> sOut;

    oneBitAdder* FA1 = new oneBitAdder("FA1_instance");
        (*FA1) (aData, bData, cData, cOut, sOut);

    sc_trace_file* VCDFile;
    VCDFile = sc_create_vcd_trace_file("simpleHierarchical");
        sc_trace(VCDFile, aData, "aIn");
        sc_trace(VCDFile, bData, "bIn");
        sc_trace(VCDFile, cData, "carryIn");
        sc_trace(VCDFile, cOut, "carryOut");
        sc_trace(VCDFile, sOut, "sumOut");
    ...

    return 0;
}
```

```
    sc_int<3> intData;
    sc_lv<3> abcData;

    sc_start(15, SC_NS);

    intData = 0;
    int ij=0;
    do {
        abcData = intData;
        aData = abcData[2];
        bData = abcData[1];
        cData = abcData[0];
        sc_start(15, SC_NS);
        intData = intData + 1;
        sc_start(50, SC_NS);
    } while (++ij < 40);

    sc_start(100,SC_NS);
```

sc_start advances the simulation as much as specified time of its argument

SystemC modeling of 1-bit Adder

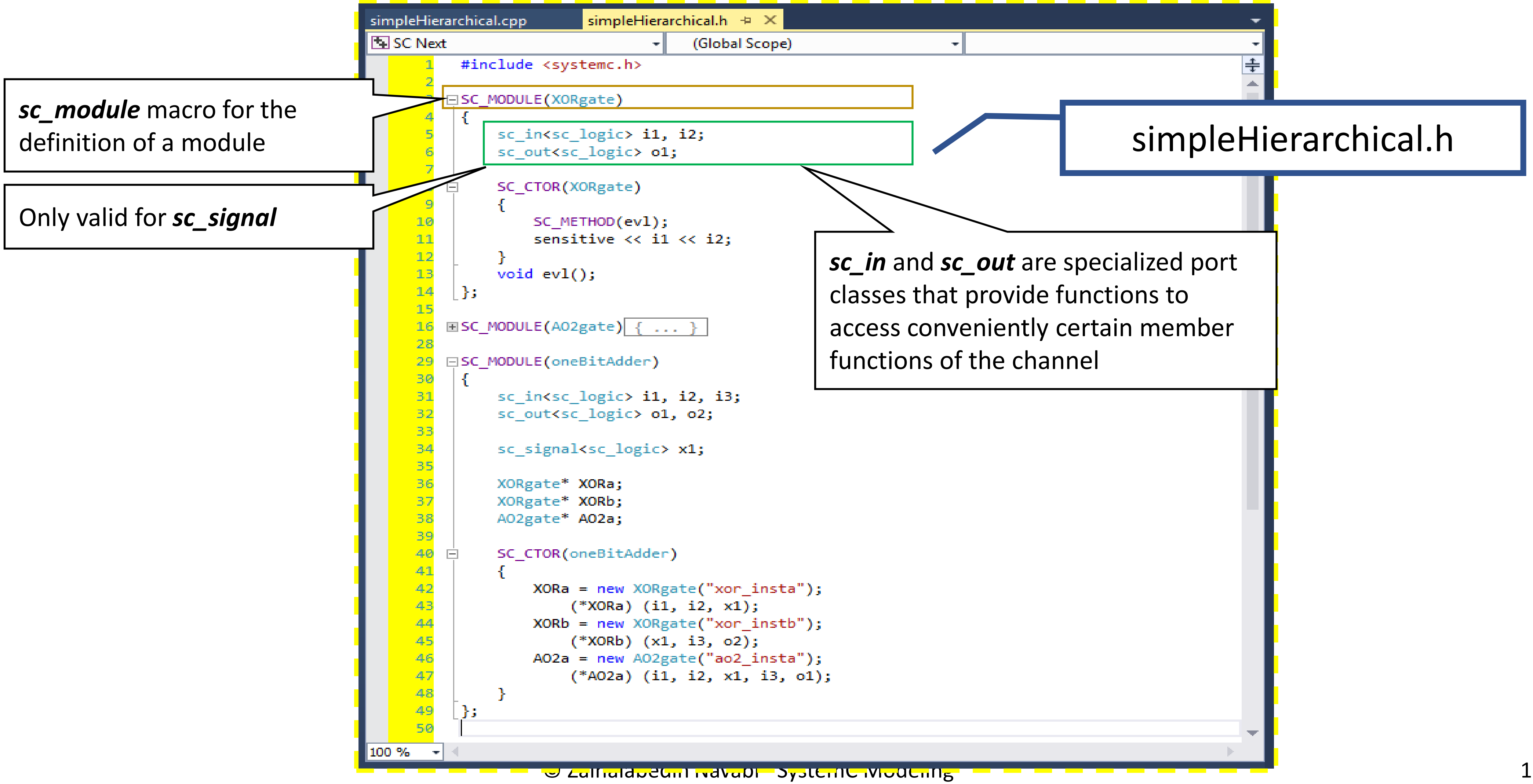
- Waveform in a VCD viewer
  - Generated VCD file can been visualized using a VCD viewer
  - Some of VCD viewer simulators:
    - Wave Editor
    - GTKWave



SystemC Modeling

- ✚ Taking Off From C++
- ✚ SystemC Gate-Level Modeling
- Utilities for HDL Orientation
- ✚ Timing & SystemC RT-Level Modeling
- ✚ Components for RTL Design
- ✚ SystemC RTL Design - Example
- SystemC Functional Modeling
- ✚ SystemC Functional Design - Example
- Summary

Utilities for HDL Orientation



```
#include <systemc.h>

SC_MODULE(XORgate)
{
    sc_in<sc_logic> i1, i2;
    sc_out<sc_logic> o1;

    SC_CTOR(XORgate)
    {
        SC_METHOD(evl);
        sensitive << i1 << i2;
    }
    void evl();
};

SC_MODULE(AO2gate){ ... }

SC_MODULE(oneBitAdder)
{
    sc_in<sc_logic> i1, i2, i3;
    sc_out<sc_logic> o1, o2;

    sc_signal<sc_logic> x1;

    XORgate* XORa;
    XORgate* XORb;
    AO2gate* AO2a;

    SC_CTOR(oneBitAdder)
    {
        XORa = new XORgate("xor_insta");
            (*XORa) (i1, i2, x1);
        XORb = new XORgate("xor_instb");
            (*XORb) (x1, i3, o2);
        AO2a = new AO2gate("ao2_insta");
            (*AO2a) (i1, i2, x1, i3, o1);
    }
};
```

sc_module macro for the definition of a module

Only valid for ***sc_signal***

simpleHierarchical.h

sc_in and ***sc_out*** are specialized port classes that provide functions to access conveniently certain member functions of the channel

SystemC Modeling

+ Taking Off From C++
+ SystemC Gate-Level Modeling

Utilities for HDL Orientation

– Timing & SystemC RT-Level Modeling
  – Hierarchical timed design for Serial Adder
    - Timed Components
    - Top-Level
    - Testbench

+ Components for RTL Design
+ SystemC RTL Design - Example

SystemC Functional Modeling

+ SystemC Functional Design - Example

Summary

Hierarchical timed design for Serial Adder

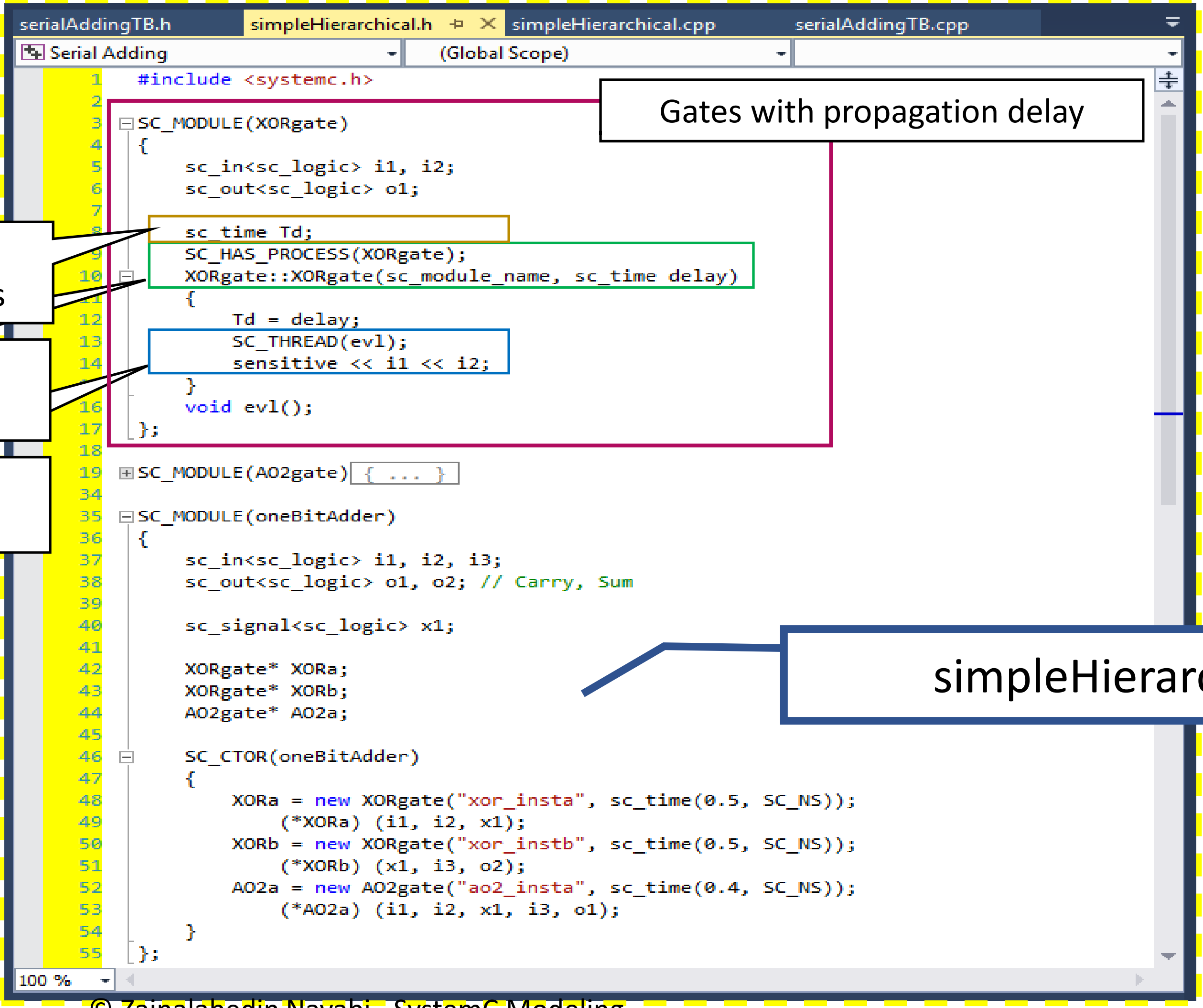
- Timed Components
  - XOR Gate

sc_time class is used to represent simulation time and time intervals

Use ***SC_HAS_PROCESS*** for passing delay argument

In order to use delay in the functionality ***SC_THREAD*** is used

SC_METHOD cannot be suspended temporarily by use of delay to wait



```
#include <systemc.h>

SC_MODULE(XORgate)
{
    sc_in<sc_logic> i1, i2;
    sc_out<sc_logic> o1;

    sc_time Td;
    SC_HAS_PROCESS(XORgate);
    XORgate::XORgate(sc_module_name, sc_time delay)
    {
        Td = delay;
        SC_THREAD(evl);
        sensitive << i1 << i2;
    }
    void evl();
};

SC_MODULE(AO2gate){ ... }

SC_MODULE(oneBitAdder)
{
    sc_in<sc_logic> i1, i2, i3;
    sc_out<sc_logic> o1, o2; // Carry, Sum

    sc_signal<sc_logic> x1;

    XORgate* XORa;
    XORgate* XORb;
    AO2gate* AO2a;

    SC_CTOR(oneBitAdder)
    {
        XORa = new XORgate("xor_insta", sc_time(0.5, SC_NS));
            (*XORa) (i1, i2, x1);
        XORb = new XORgate("xor_instb", sc_time(0.5, SC_NS));
            (*XORb) (x1, i3, o2);
        AO2a = new AO2gate("ao2_insta", sc_time(0.4, SC_NS));
            (*AO2a) (i1, i2, x1, i3, o1);
    }
};
```

Gates with propagation delay

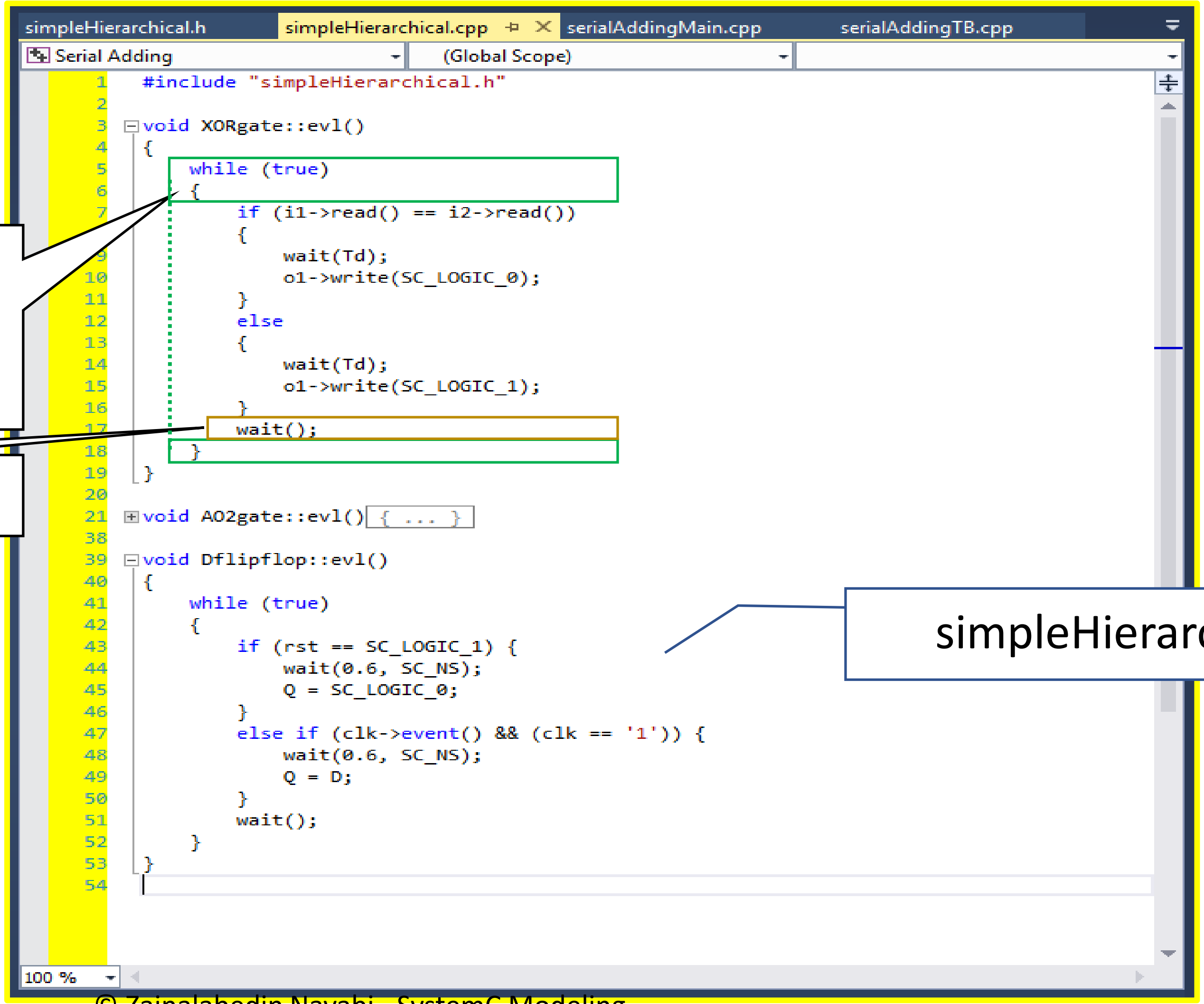
simpleHierarchical.h

Hierarchical timed design for Serial Adder

- Timed Components
  - XOR Gate

SC_THREAD is mean to be executed only once throughout the simulation. It is able to execute for the whole duration of the simulation using an infinite loop

Suspend until the next event on i1 or i2

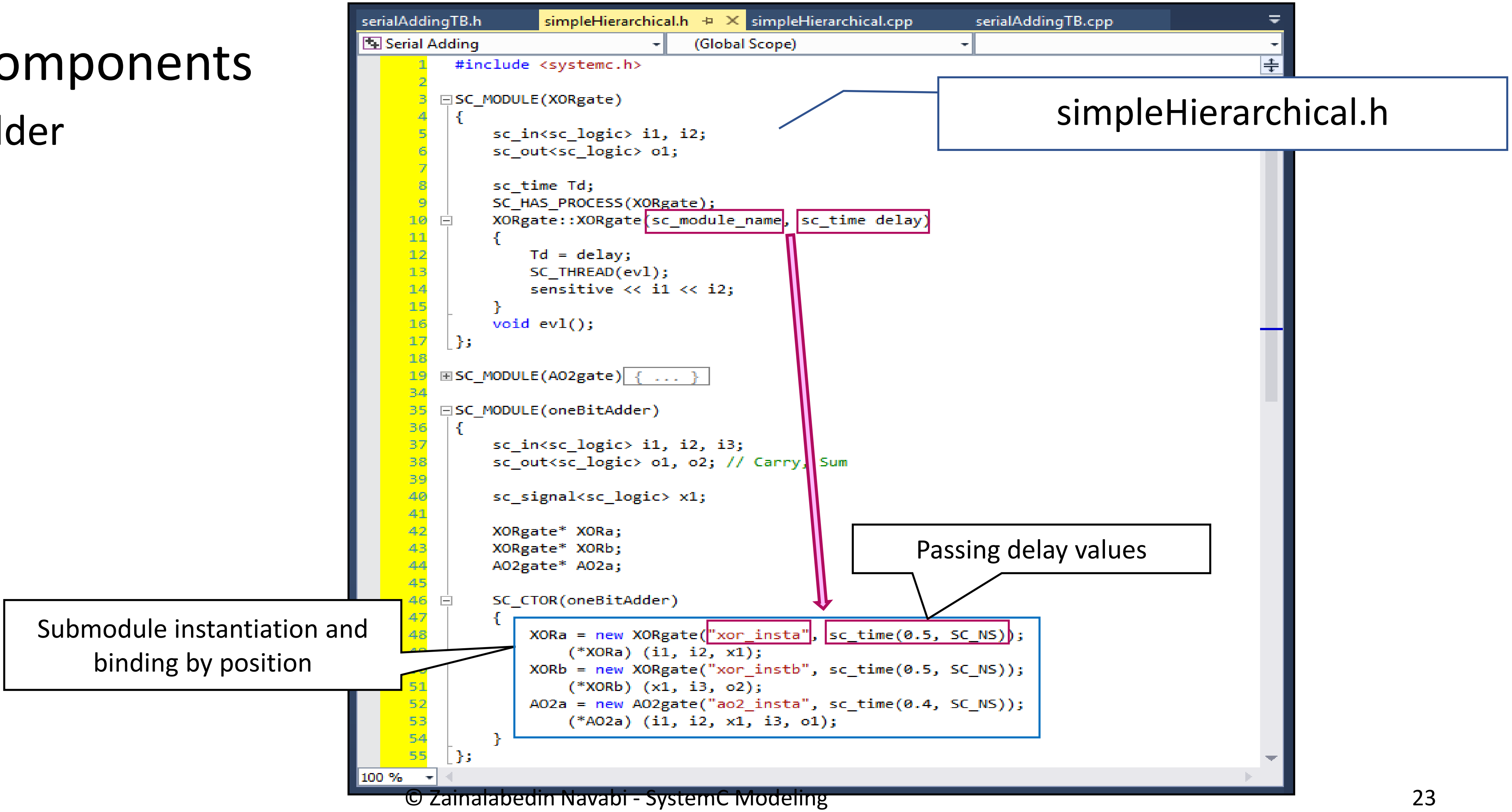


```
#include "simpleHierarchical.h"

void XORgate::evl()
{
    while (true)
    {
        if (i1->read() == i2->read())
        {
            wait(Td);
            o1->write(SC_LOGIC_0);
        }
        else
        {
            wait(Td);
            o1->write(SC_LOGIC_1);
        }
        wait();
    }
}

void AO2gate::evl(){ ... }

void Dflipflop::evl()
{
    while (true)
    {
        if (rst == SC_LOGIC_1) {
            wait(0.6, SC_NS);
            Q = SC_LOGIC_0;
        }
        else if (clk->event() && (clk == '1')) {
            wait(0.6, SC_NS);
            Q = D;
        }
        wait();
    }
}
```

simpleHierarchical.cpp

Hierarchical timed design for Serial Adder

- Timed Components
  - 1-bit Adder



```
#include <systemc.h>

SC_MODULE(XORgate)
{
    sc_in<sc_logic> i1, i2;
    sc_out<sc_logic> o1;

    sc_time Td;
    SC_HAS_PROCESS(XORgate);
    XORgate::XORgate(sc_module_name, sc_time delay)
    {
        Td = delay;
        SC_THREAD(evl);
        sensitive << i1 << i2;
    }
    void evl();
};

SC_MODULE(AO2gate){ ... }

SC_MODULE(oneBitAdder)
{
    sc_in<sc_logic> i1, i2, i3;
    sc_out<sc_logic> o1, o2; // Carry, Sum

    sc_signal<sc_logic> x1;

    XORgate* XORa;
    XORgate* XORb;
    AO2gate* AO2a;

    SC_CTOR(oneBitAdder)
    {
        XORa = new XORgate("xor_insta", sc_time(0.5, SC_NS));
            (*XORa) (i1, i2, x1);
        XORb = new XORgate("xor_instb", sc_time(0.5, SC_NS));
            (*XORb) (x1, i3, o2);
        AO2a = new AO2gate("ao2_insta", sc_time(0.4, SC_NS));
            (*AO2a) (i1, i2, x1, i3, o1);
    }
};
```

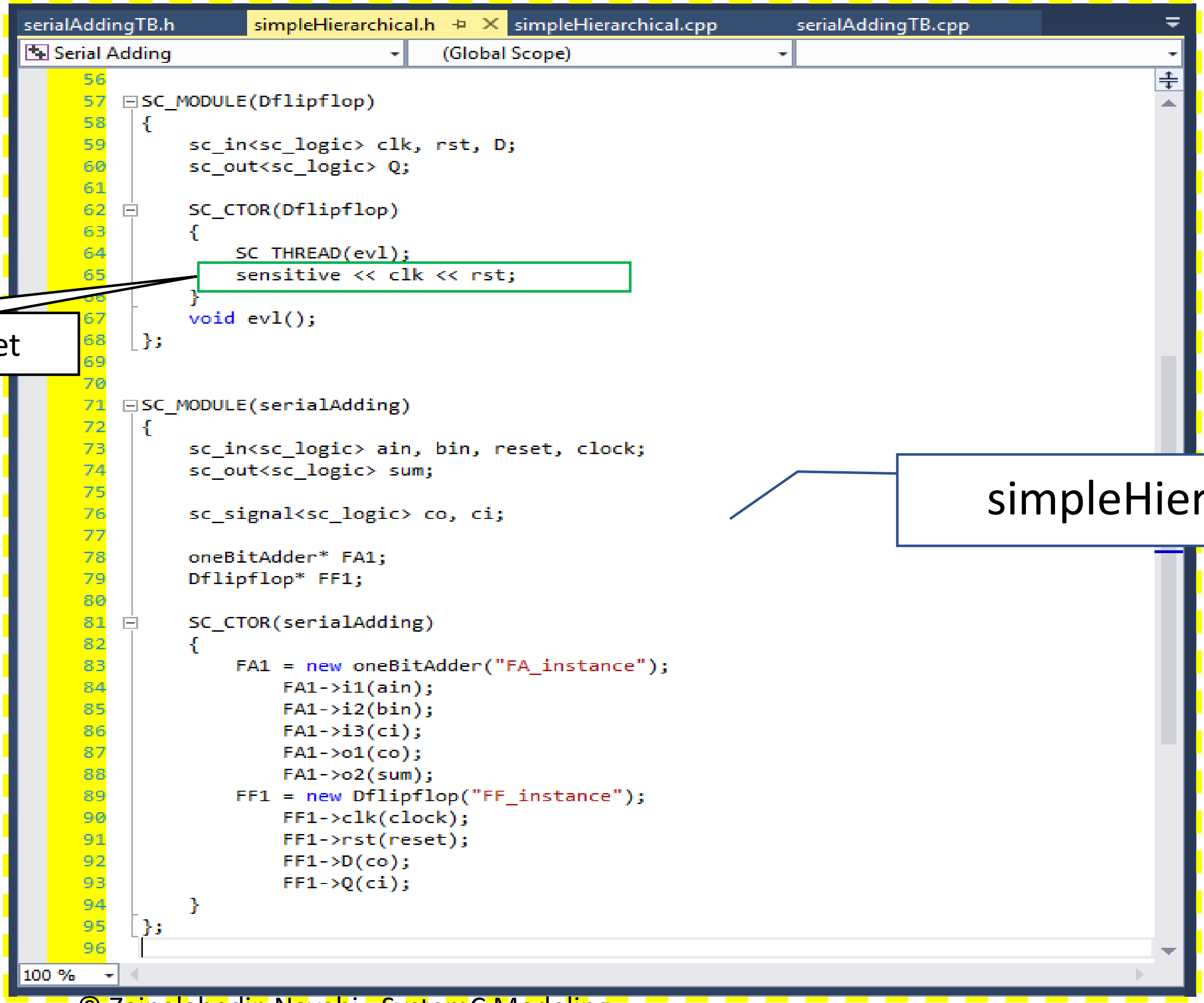
simpleHierarchical.h

Passing delay values

Submodule instantiation and binding by position

Hierarchical timed design for Serial Adder

- Timed Components
  - D-Flip Flop



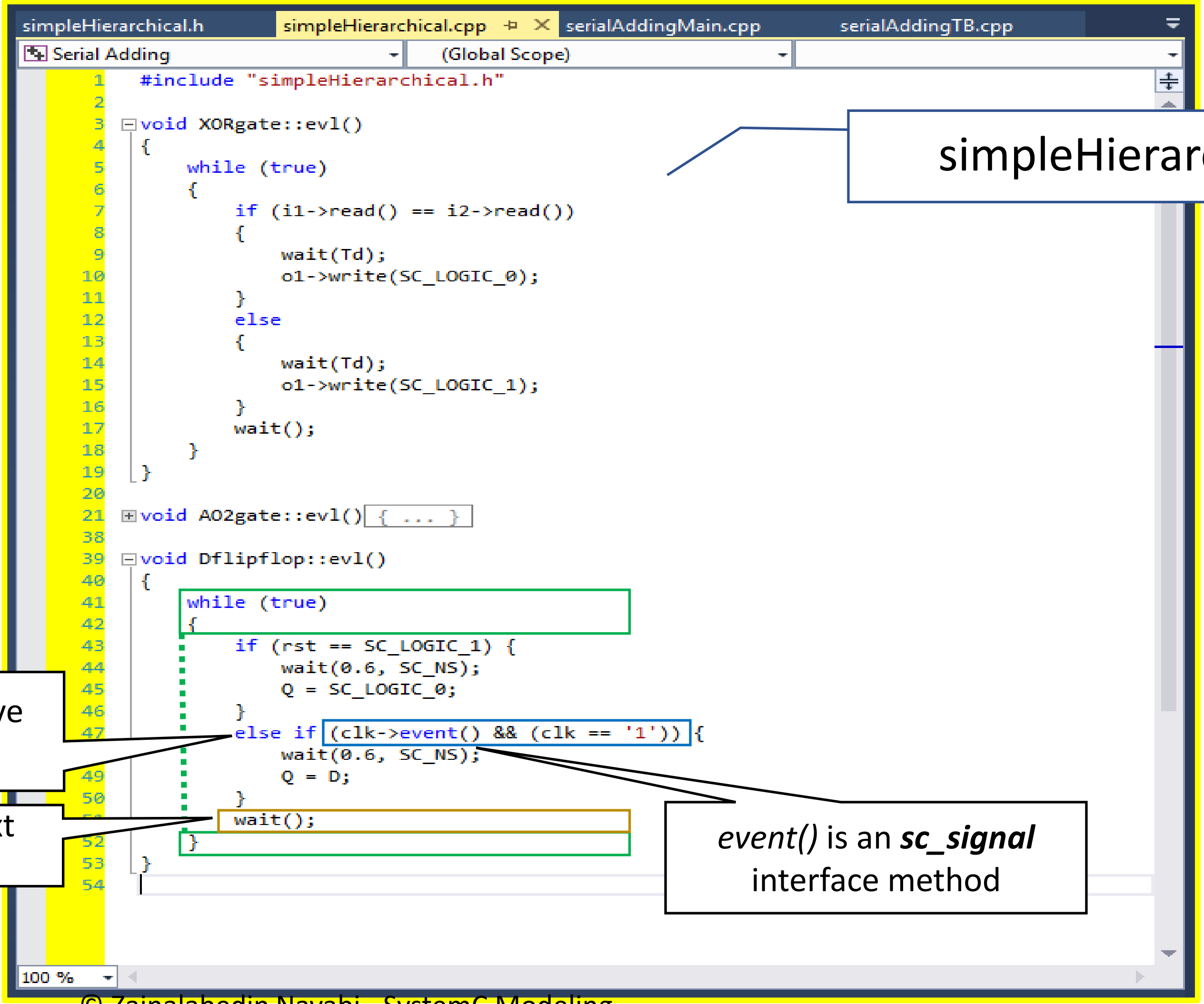
```
SC_MODULE(Dflipflop)
{
    sc_in<sc_logic> clk, rst, D;
    sc_out<sc_logic> Q;

    SC_CTOR(Dflipflop)
    {
        SC_THREAD(evl);
        sensitive << clk << rst;
    }
    void evl();
};


SC_MODULE(serialAdding)
{
    sc_in<sc_logic> ain, bin, reset, clock;
    sc_out<sc_logic> sum;

    sc_signal<sc_logic> co, ci;

    oneBitAdder* FA1;
    Dflipflop* FF1;

    SC_CTOR(serialAdding)
    {
        FA1 = new oneBitAdder("FA_instance");
            FA1->i1(ain);
            FA1->i2(bin);
            FA1->i3(ci);
            FA1->o1(co);
            FA1->o2(sum);
        FF1 = new Dflipflop("FF_instance");
            FF1->clk(clock);
            FF1->rst(reset);
            FF1->D(co);
            FF1->Q(ci);
    }
};
```

Sensitive to clock and reset

simpleHierarchical.h

Hierarchical timed design for Serial Adder

- Timed Components
  - D-Flip Flop



```
#include "simpleHierarchical.h"

void XORgate::evl()
{
    while (true)
    {
        if (i1->read() == i2->read())
        {
            wait(Td);
            o1->write(SC_LOGIC_0);
        }
        else
        {
            wait(Td);
            o1->write(SC_LOGIC_1);
        }
        wait();
    }
}

void AO2gate::evl() { ... }

void Dflipflop::evl()
{
    while (true)
    {
        if (rst == SC_LOGIC_1) {
            wait(0.6, SC_NS);
            Q = SC_LOGIC_0;
        }
        else if (clk->event() && (clk == '1')){
            wait(0.6, SC_NS);
            Q = D;
        }
        wait();
    }
}
```

simpleHierarchical.cpp

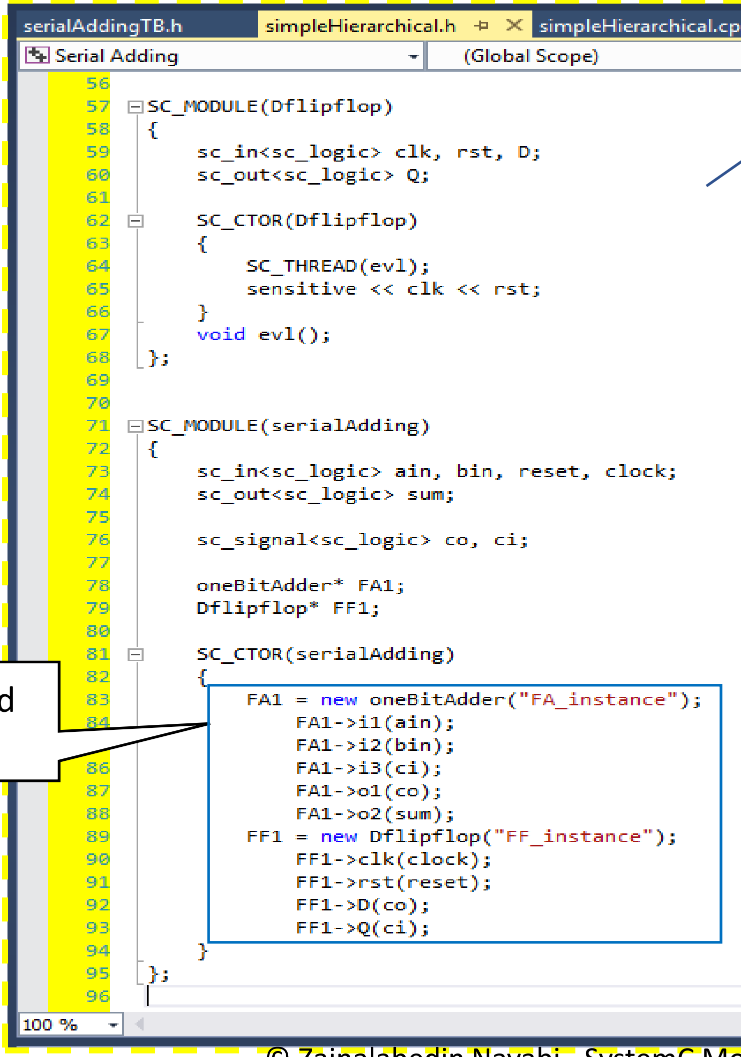
This means the positive edge of the clk

Suspend until the next event on clk or rst

event() is an ***sc_signal*** interface method

Hierarchical timed design for Serial Adder

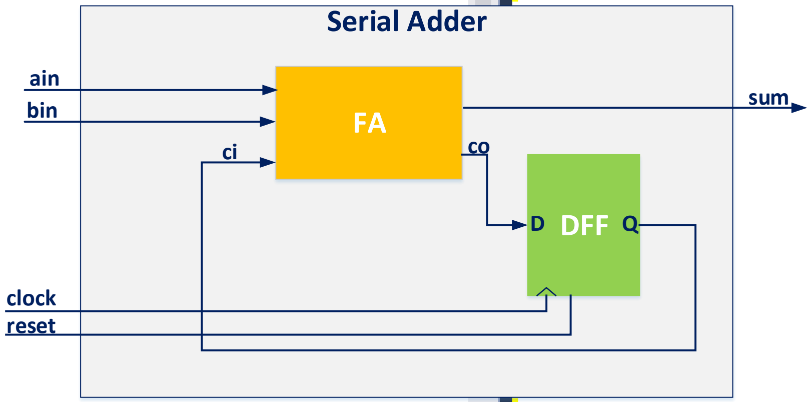
- Top-Level



```
SC_MODULE(Dflipflop)
{
    sc_in<sc_logic> clk, rst, D;
    sc_out<sc_logic> Q;

    SC_CTOR(Dflipflop)
    {
        SC_THREAD(evl);
        sensitive << clk << rst;
    }
    void evl();
};


SC_MODULE(serialAdding)
{
    sc_in<sc_logic> ain, bin, reset, clock;
    sc_out<sc_logic> sum;

    sc_signal<sc_logic> co, ci;

    oneBitAdder* FA1;
    Dflipflop* FF1;

    SC_CTOR(serialAdding)
    {
        FA1 = new oneBitAdder("FA_instance");
            FA1->i1(ain);
            FA1->i2(bin);
            FA1->i3(ci);
            FA1->o1(co);
            FA1->o2(sum);
        FF1 = new Dflipflop("FF_instance");
            FF1->clk(clock);
            FF1->rst(reset);
            FF1->D(co);
            FF1->Q(ci);
    }
};
```

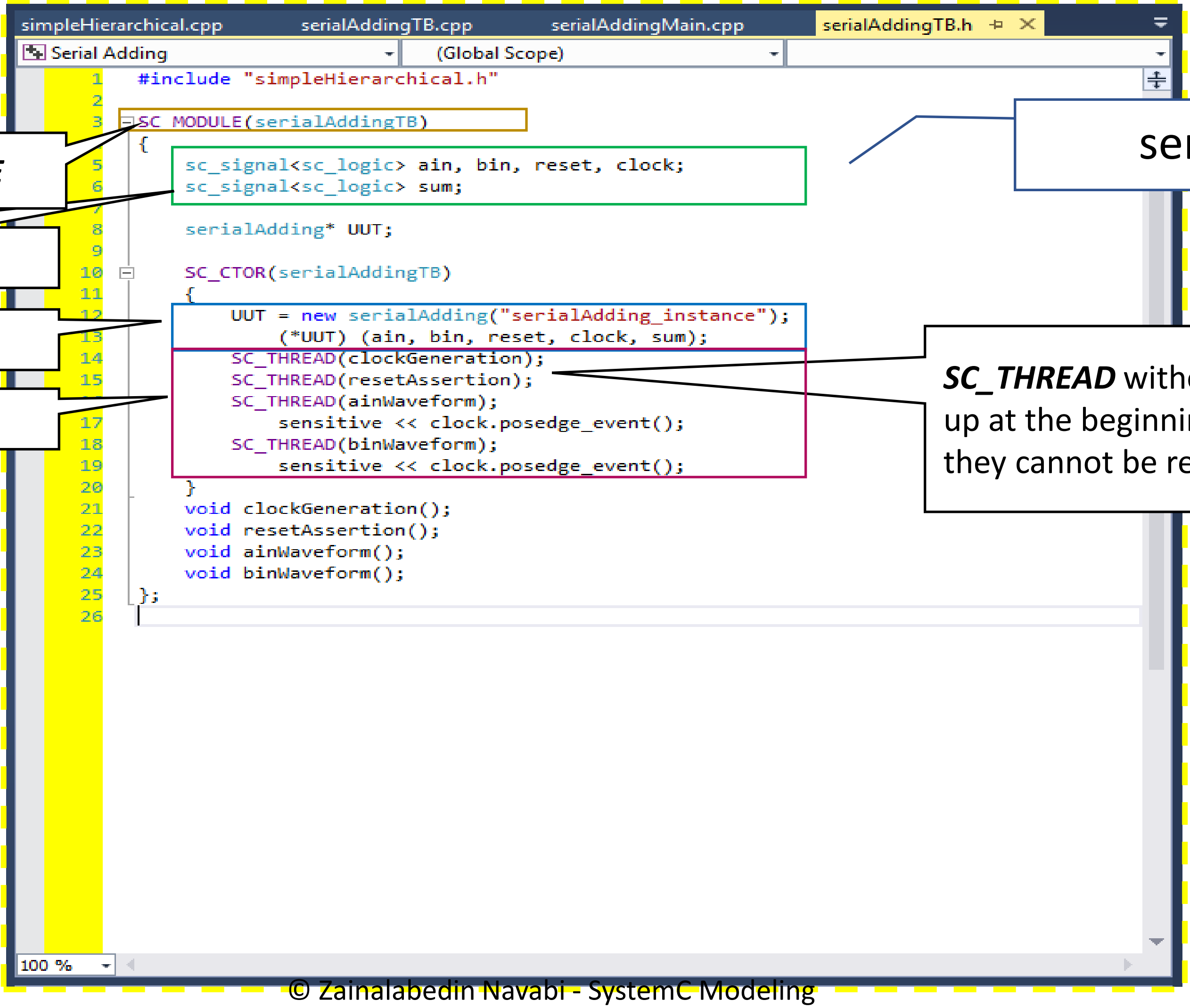
simpleHierarchical.h

Submodule instantiation and binding by name



Hierarchical timed design for Serial Adder

- Testbench



```
#include "simpleHierarchical.h"

SC_MODULE(serialAddingTB)
{
    sc_signal<sc_logic> ain, bin, reset, clock;
    sc_signal<sc_logic> sum;

    serialAdding* UUT;

    SC_CTOR(serialAddingTB)
    {
        UUT = new serialAdding("serialAdding_instance");
            (*UUT) (ain, bin, reset, clock, sum);
        SC_THREAD(clockGeneration);
        SC_THREAD(resetAssertion);
        SC_THREAD(ainWaveform);
            sensitive << clock.posedge_event();
        SC_THREAD(binWaveform);
            sensitive << clock.posedge_event();
    }
    void clockGeneration();
    void resetAssertion();
    void ainWaveform();
    void binWaveform();
};
```

serialAddingTB.h

Testbench as a ***SC_MODULE***

Local signals

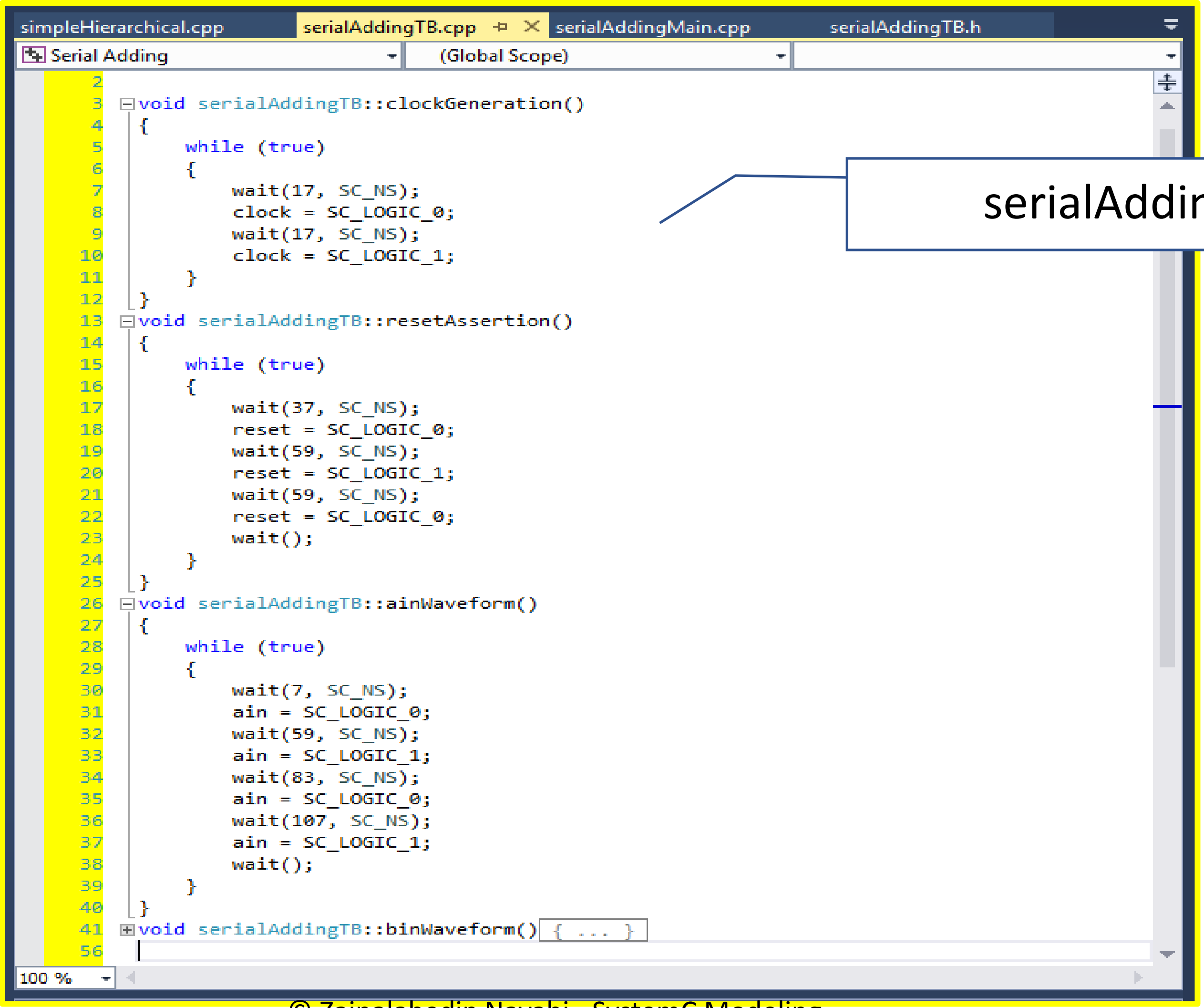
Top-Level instantiation

Data generation

SC_THREAD without sensitivity list: wake up at the beginning and if they suspend they cannot be resumed

Hierarchical timed design for Serial Adder

- Testbench

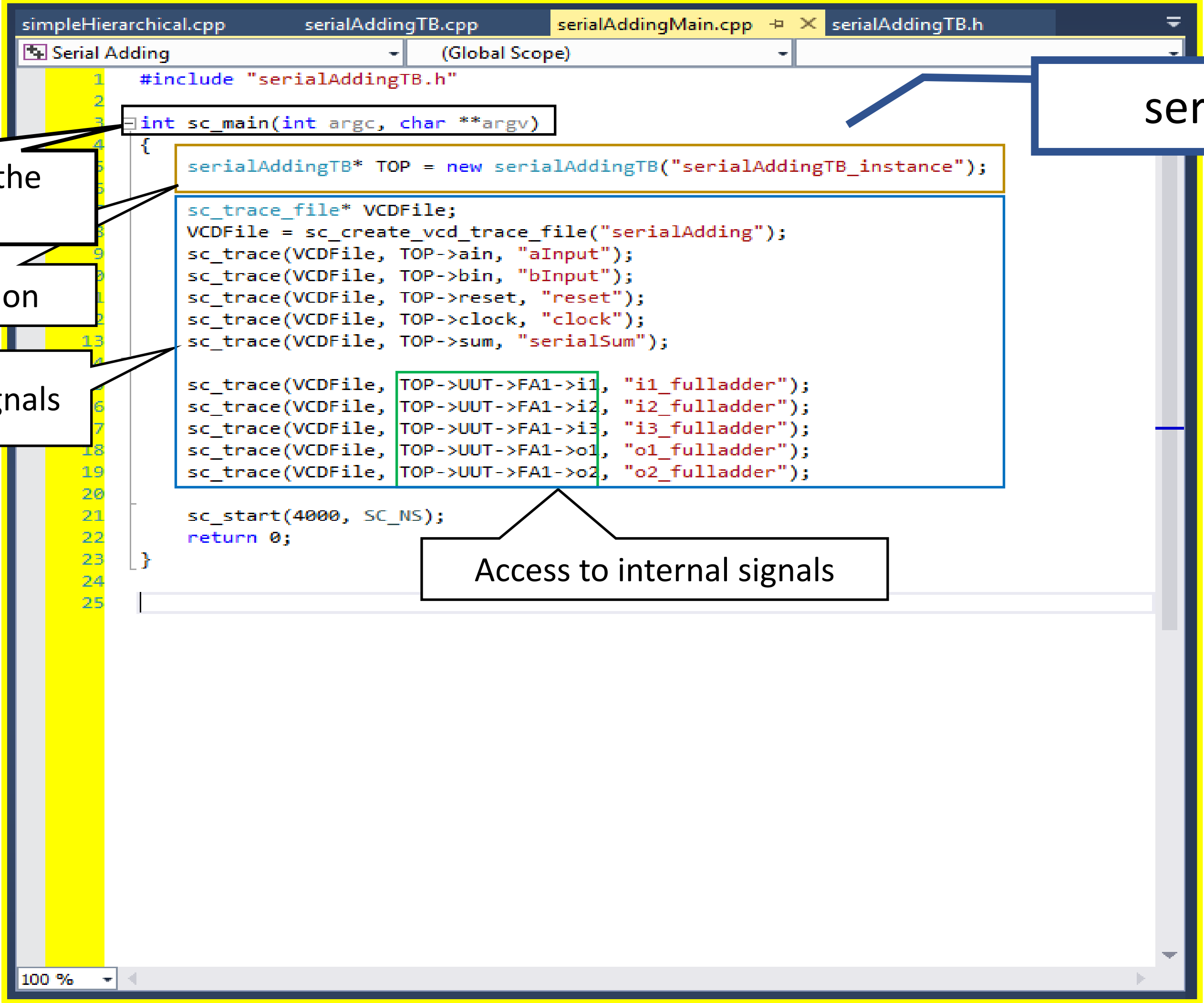


```

void serialAddingTB::clockGeneration()
{
    while (true)
    {
        wait(17, SC_NS);
        clock = SC_LOGIC_0;
        wait(17, SC_NS);
        clock = SC_LOGIC_1;
    }
}
void serialAddingTB::resetAssertion()
{
    while (true)
    {
        wait(37, SC_NS);
        reset = SC_LOGIC_0;
        wait(59, SC_NS);
        reset = SC_LOGIC_1;
        wait(59, SC_NS);
        reset = SC_LOGIC_0;
        wait();
    }
}
void serialAddingTB::ainWaveform()
{
    while (true)
    {
        wait(7, SC_NS);
        ain = SC_LOGIC_0;
        wait(59, SC_NS);
        ain = SC_LOGIC_1;
        wait(83, SC_NS);
        ain = SC_LOGIC_0;
        wait(107, SC_NS);
        ain = SC_LOGIC_1;
        wait();
    }
}
void serialAddingTB::binWaveform(){ ... }

```

serialAddingTB.cpp

Hierarchical timed design for Serial Adder

- sc_main



```
#include "serialAddingTB.h"

int sc_main(int argc, char **argv)
{
    serialAddingTB* TOP = new serialAddingTB("serialAddingTB_instance");

    sc_trace_file* VCDFile;
    VCDFile = sc_create_vcd_trace_file("serialAdding");
    sc_trace(VCDFile, TOP->ain, "aInput");
    sc_trace(VCDFile, TOP->bin, "bInput");
    sc_trace(VCDFile, TOP->reset, "reset");
    sc_trace(VCDFile, TOP->clock, "clock");
    sc_trace(VCDFile, TOP->sum, "serialSum");

    sc_trace(VCDFile, TOP->UUT->FA1->i1, "i1_fulladder");
    sc_trace(VCDFile, TOP->UUT->FA1->i2, "i2_fulladder");
    sc_trace(VCDFile, TOP->UUT->FA1->i3, "i3_fulladder");
    sc_trace(VCDFile, TOP->UUT->FA1->o1, "o1_fulladder");
    sc_trace(VCDFile, TOP->UUT->FA1->o2, "o2_fulladder");

    sc_start(4000, SC_NS);
    return 0;
}
```

serialAddingMain.cpp

function ***sc_main*** in the global namespace

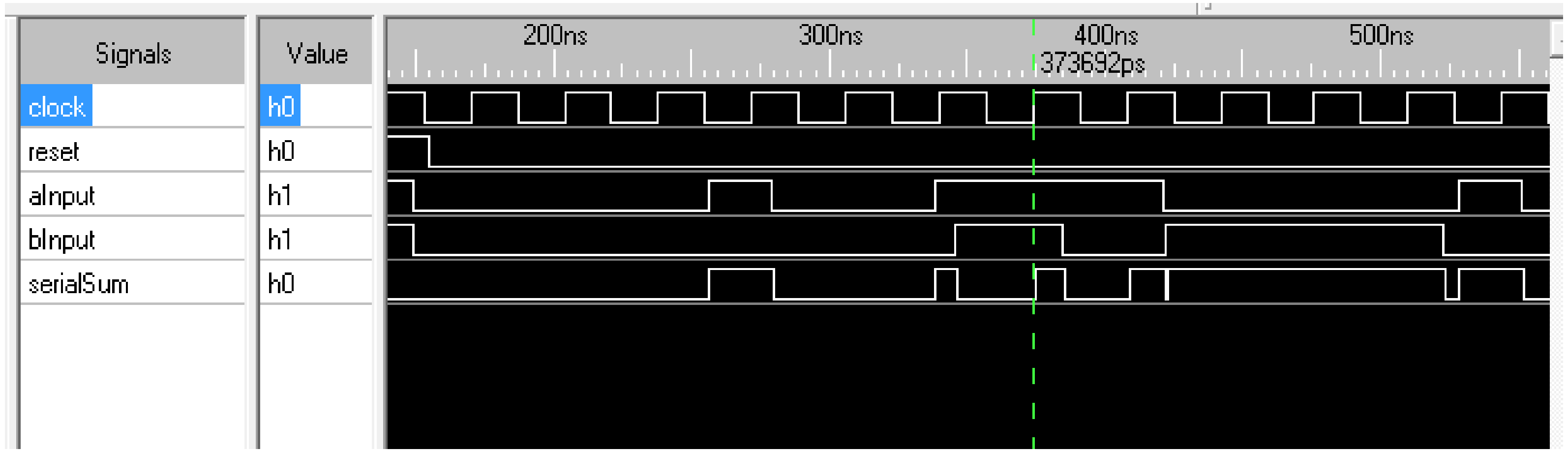
Testbench instantiation

VCD file and tracing signals

Access to internal signals

Hierarchical timed design for Serial Adder

- Waveform in a VCD viewer



Signals	Value
clock	h0
reset	h0
aInput	h1
bInput	h1
serialSum	h0

SystemC Modeling

- ✚ Taking Off From C++
- ✚ SystemC Gate-Level Modeling
- Utilities for HDL Orientation
- ✚ Timing & SystemC RT-Level Modeling
- – Components for RTL Design
  - – Combinational
    - Adder
    - Mux
    - Tri-State
  - – Sequential
    - D-Register
    - Counter
    - Shift-Register
- ✚ SystemC RTL Design - Example
- SystemC Functional Modeling
- ✚ SystemC Functional Design - Example
- Summary

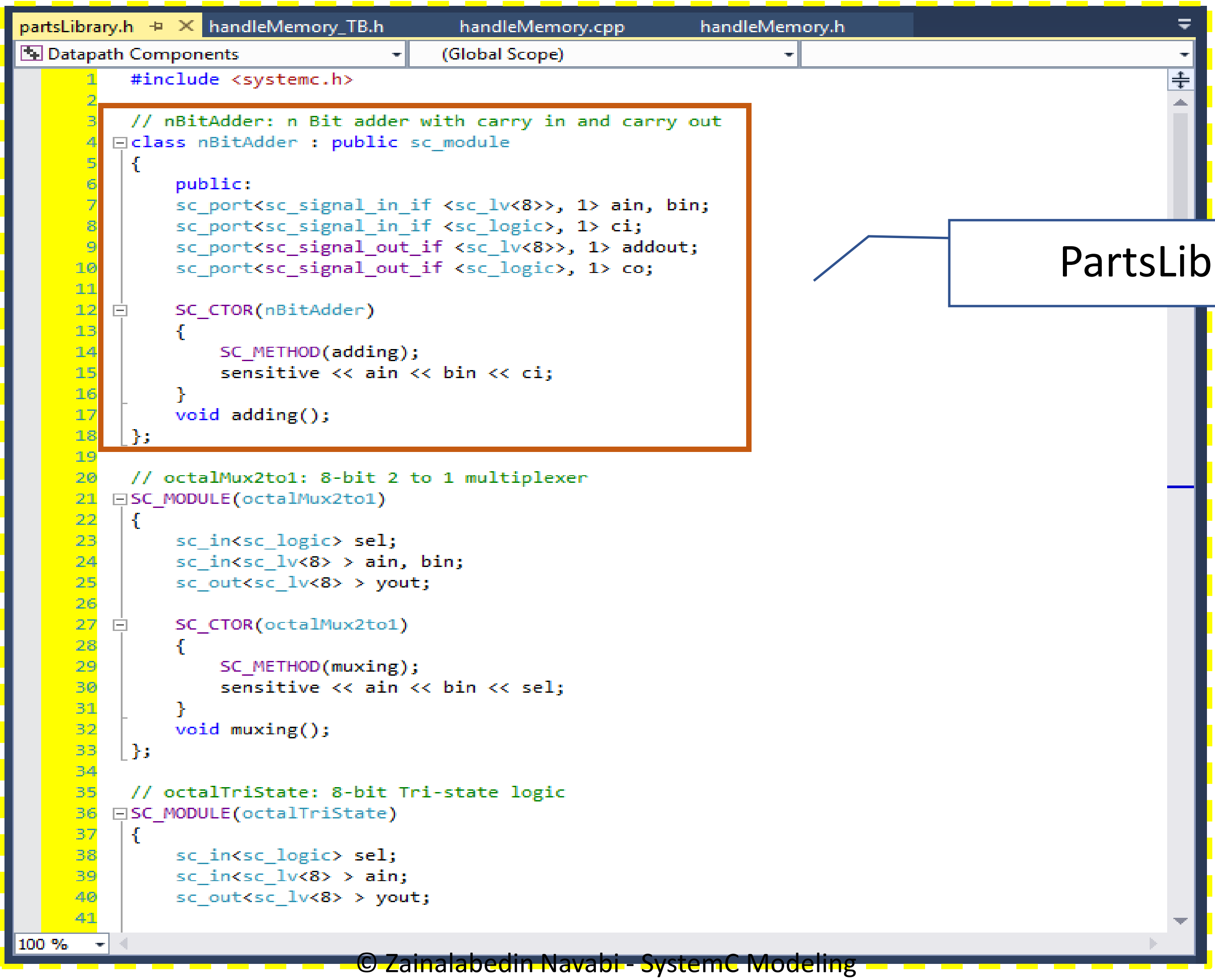
Four value logic

- *sc_logic* is four value logic
  - Logic value 0 and 1 are for standard logic values
  - Z represents high impedance
  - X represents unknown value
  - Vector version is *sc_lv<n>*
- *sc_bit* is two value logic
  - Logic value 0 and 1 are for standard logic values
  - Vector version is *sc_bv<n>*
- *sc_resolved* is four value logic which is used for the signals which can be simultaneously driven by several sources

Components for RTL Design

Combinational

- Adder

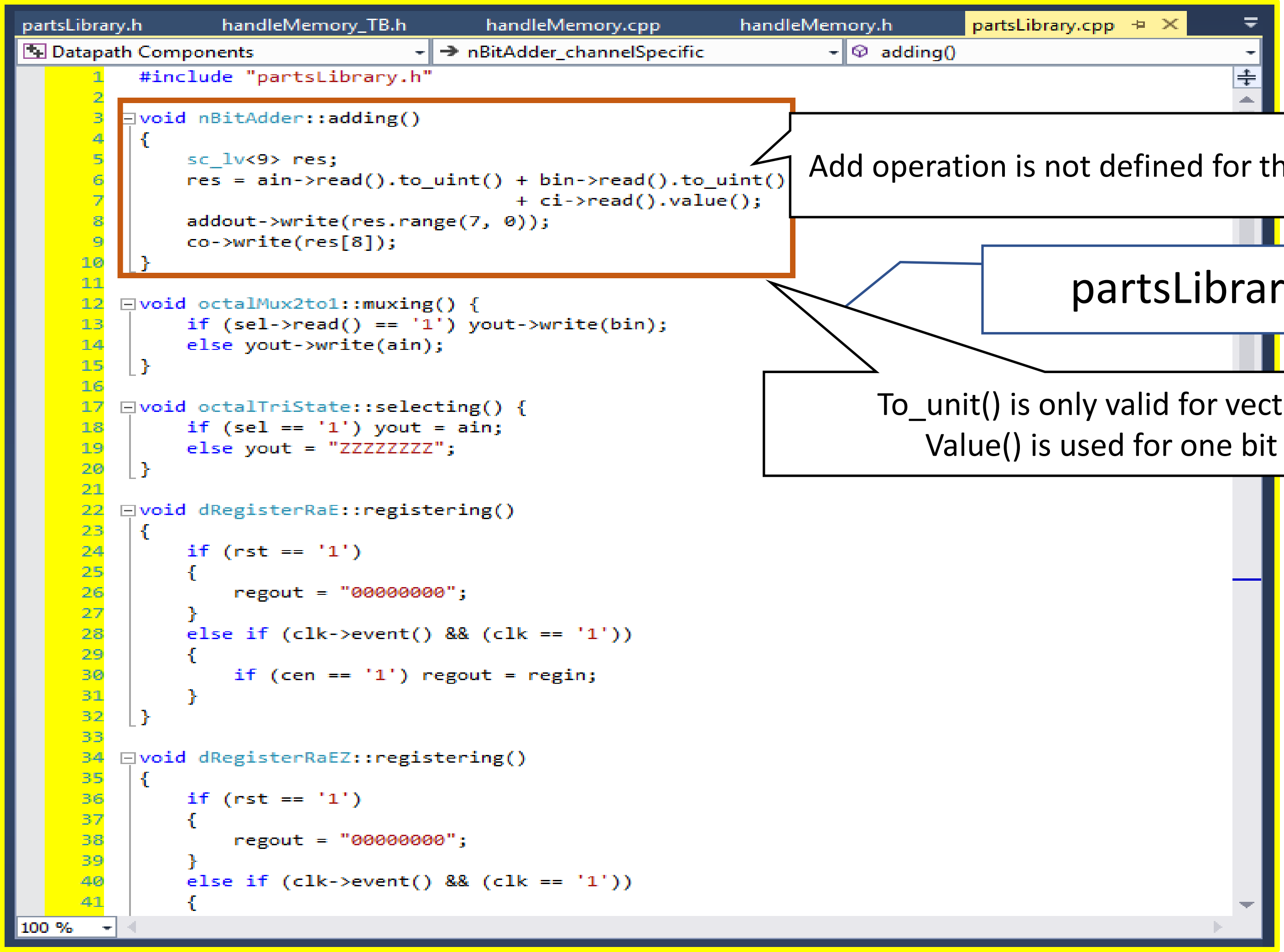


```
#include <systemc.h>

// nBitAdder: n Bit adder with carry in and carry out
class nBitAdder : public sc_module
{
    public:
    sc_port<sc_signal_in_if <sc_lv<8>>, 1> ain, bin;
    sc_port<sc_signal_in_if <sc_logic>, 1> ci;
    sc_port<sc_signal_out_if <sc_lv<8>>, 1> addout;
    sc_port<sc_signal_out_if <sc_logic>, 1> co;

    SC_CTOR(nBitAdder)
    {
        SC_METHOD(adding);
        sensitive << ain << bin << ci;
    }
    void adding();
};

// octalMux2to1: 8-bit 2 to 1 multiplexer
SC_MODULE(octalMux2to1)
{
    sc_in<sc_logic> sel;
    sc_in<sc_lv<8> > ain, bin;
    sc_out<sc_lv<8> > yout;

    SC_CTOR(octalMux2to1)
    {
        SC_METHOD(muxing);
        sensitive << ain << bin << sel;
    }
    void muxing();
};

// octalTriState: 8-bit Tri-state logic
SC_MODULE(octalTriState)
{
    sc_in<sc_logic> sel;
    sc_in<sc_lv<8> > ain;
    sc_out<sc_lv<8> > yout;

```

PartsLibrary.h

Combinational

- Adder



```
#include "partsLibrary.h"

void nBitAdder::adding()
{
    sc_lv<9> res;
    res = ain->read().to_uint() + bin->read().to_uint()
                                + ci->read().value();
    addout->write(res.range(7, 0));
    co->write(res[8]);
}

void octalMux2to1::muxing() {
    if (sel->read() == '1') yout->write(bin);
    else yout->write(ain);
}

void octalTriState::selecting() {
    if (sel == '1') yout = ain;
    else yout = "ZZZZZZZZ";
}

void dRegisterRaE::registering()
{
    if (rst == '1')
    {
        regout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
        if (cen == '1') regout = regin;
    }
}

void dRegisterRaEZ::registering()
{
    if (rst == '1')
    {
        regout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
```

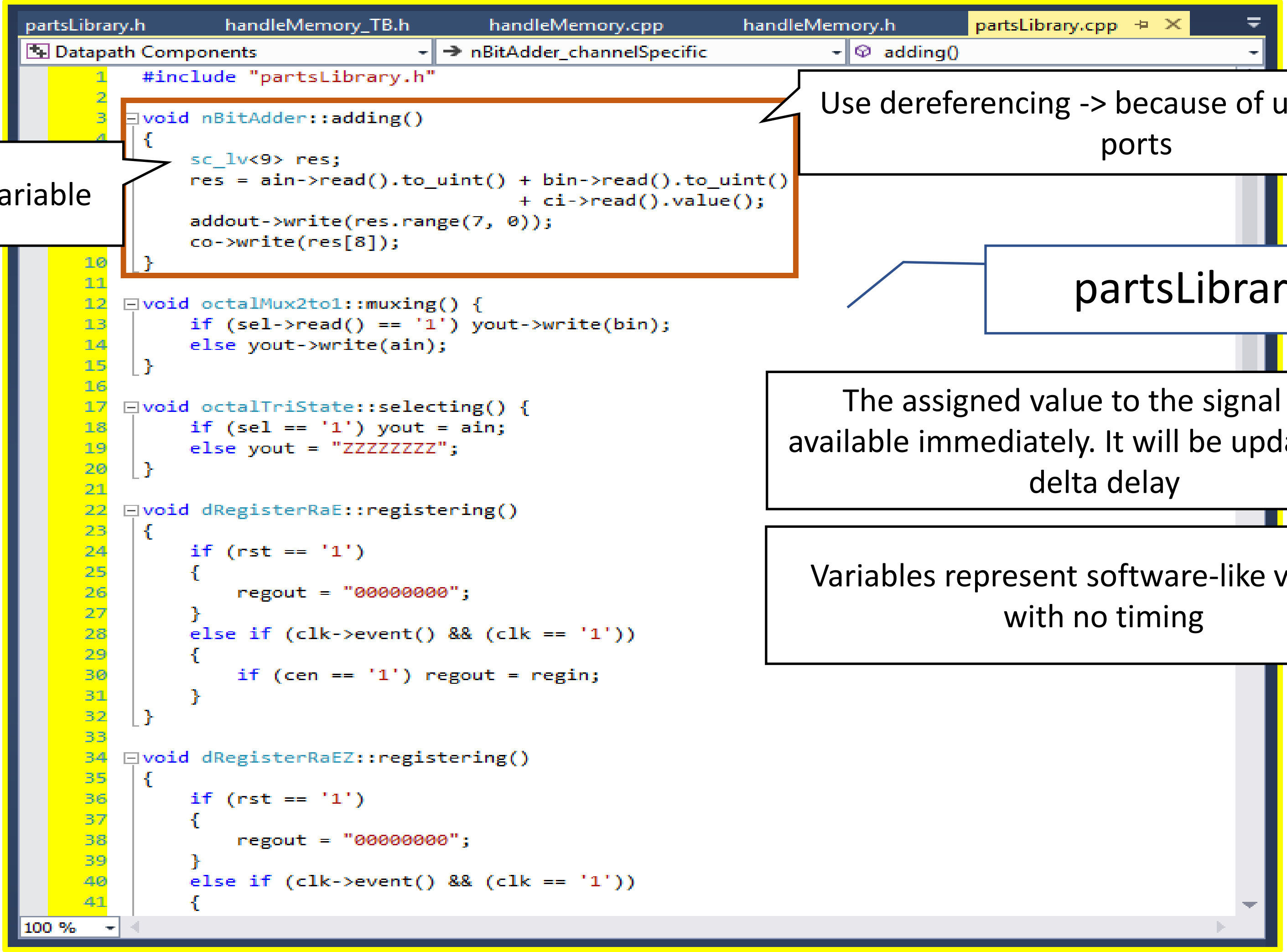
Add operation is not defined for the logic type

partsLibrary.cpp

To_unit() is only valid for vectors
Value() is used for one bit

Combinational

- Adder



```cpp
#include "partsLibrary.h"

void nBitAdder::adding()
{
    sc_lv<9> res;
    res = ain->read().to_uint() + bin->read().to_uint()
                                + ci->read().value();
    addout->write(res.range(7, 0));
    co->write(res[8]);
}

void octalMux2to1::muxing() {
    if (sel->read() == '1') yout->write(bin);
    else yout->write(ain);
}

void octalTriState::selecting() {
    if (sel == '1') yout = ain;
    else yout = "ZZZZZZZZ";
}

void dRegisterRaE::registering()
{
    if (rst == '1')
    {
        regout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
        if (cen == '1') regout = regin;
    }
}

void dRegisterRaEZ::registering()
{
    if (rst == '1')
    {
        regout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
```

variable

Use dereferencing -> because of using pointer ports

partsLibrary.cpp

The assigned value to the signal is not available immediately. It will be updated after delta delay

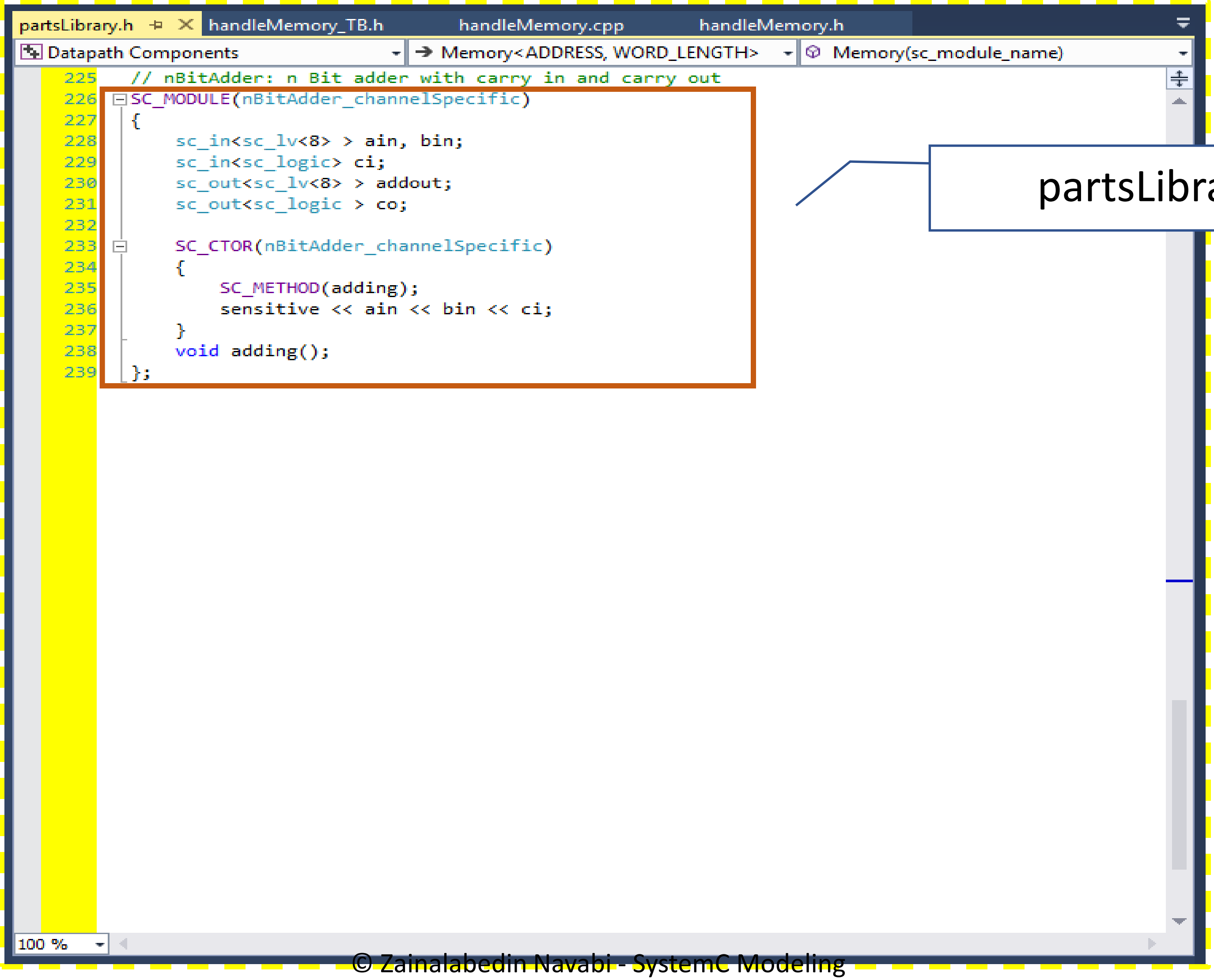
Variables represent software-like variables with no timing

Signals and variables

Components for RTL Design

Combinational

- Adder



```
// nBitAdder: n Bit adder with carry in and carry out
SC_MODULE(nBitAdder_channelSpecific)
{
    sc_in<sc_lv<8> > ain, bin;
    sc_in<sc_logic> ci;
    sc_out<sc_lv<8> > addout;
    sc_out<sc_logic > co;

    SC_CTOR(nBitAdder_channelSpecific)
    {
        SC_METHOD(adding);
        sensitive << ain << bin << ci;
    }
    void adding();
};
```

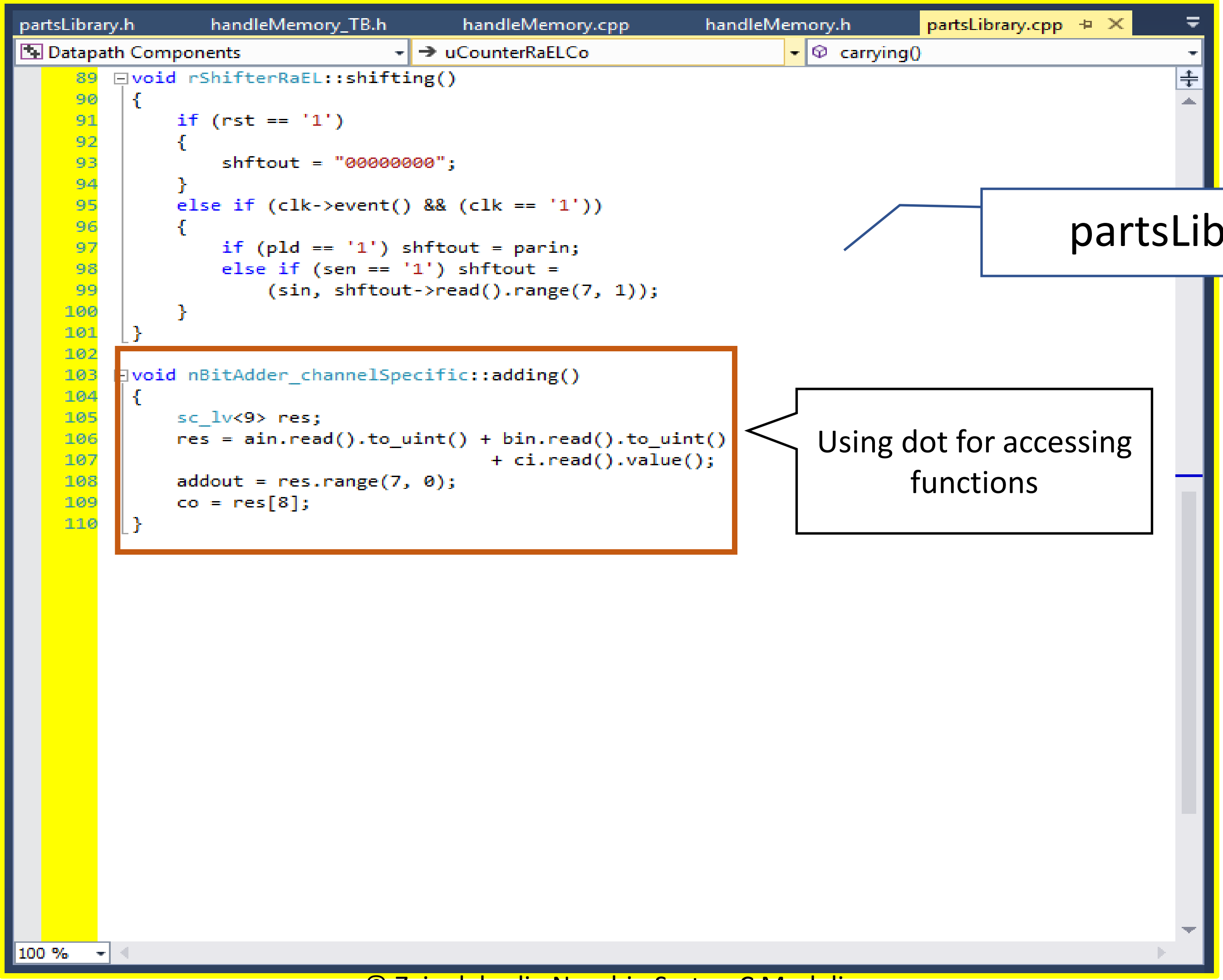
partsLibrary.h

Using sc_in and sc_out

Components for RTL Design

Combinational

- Adder



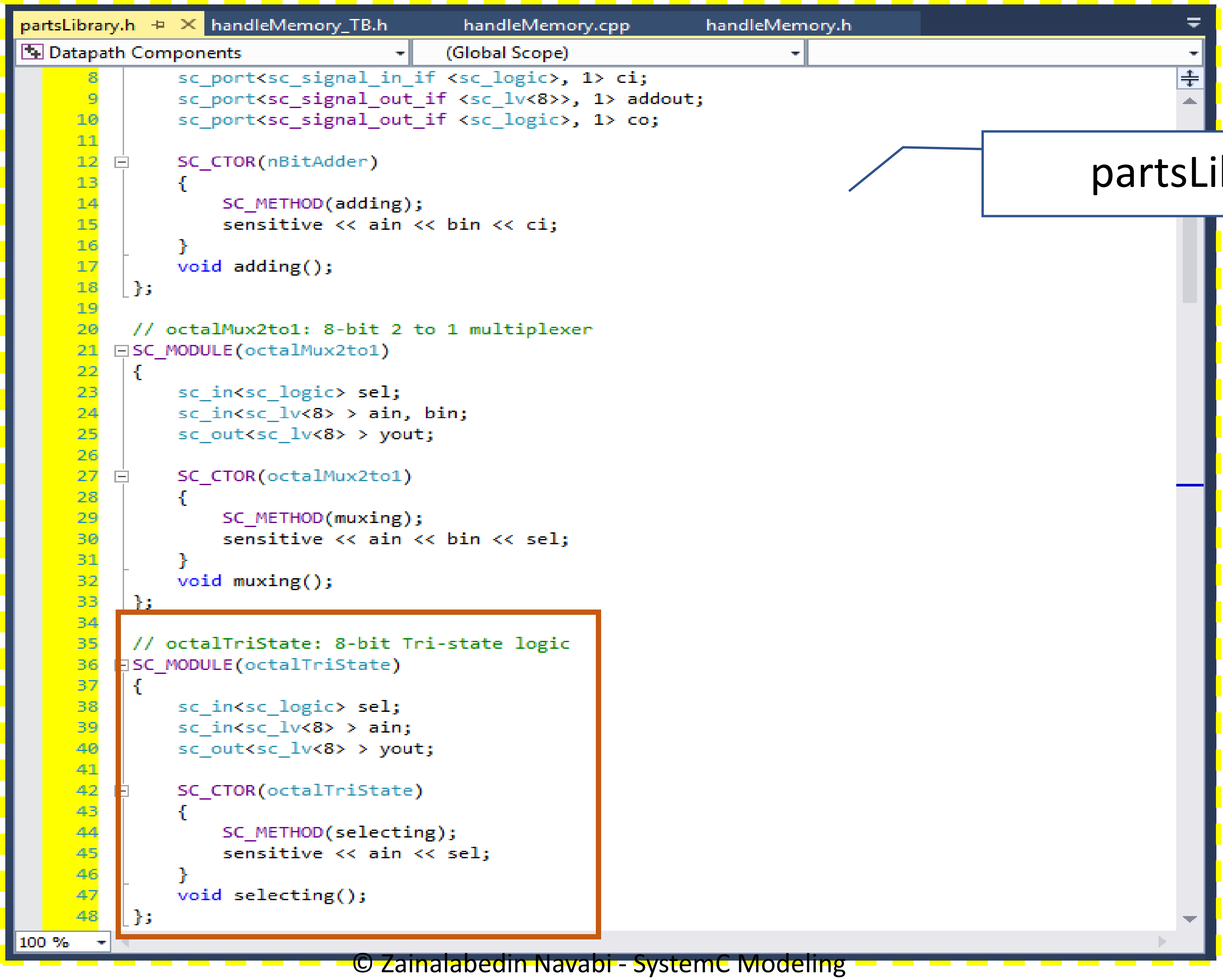
```
void rShifterRaEL::shifting()
{
    if (rst == '1')
    {
        shftout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
        if (pld == '1') shftout = parin;
        else if (sen == '1') shftout =
            (sin, shftout->read().range(7, 1));
    }
}

void nBitAdder_channelSpecific::adding()
{
    sc_lv<9> res;
    res = ain.read().to_uint() + bin.read().to_uint()
                                + ci.read().value();
    addout = res.range(7, 0);
    co = res[8];
}
```

partsLibrary.cpp

Using dot for accessing functions

Combinational

- Tri-State



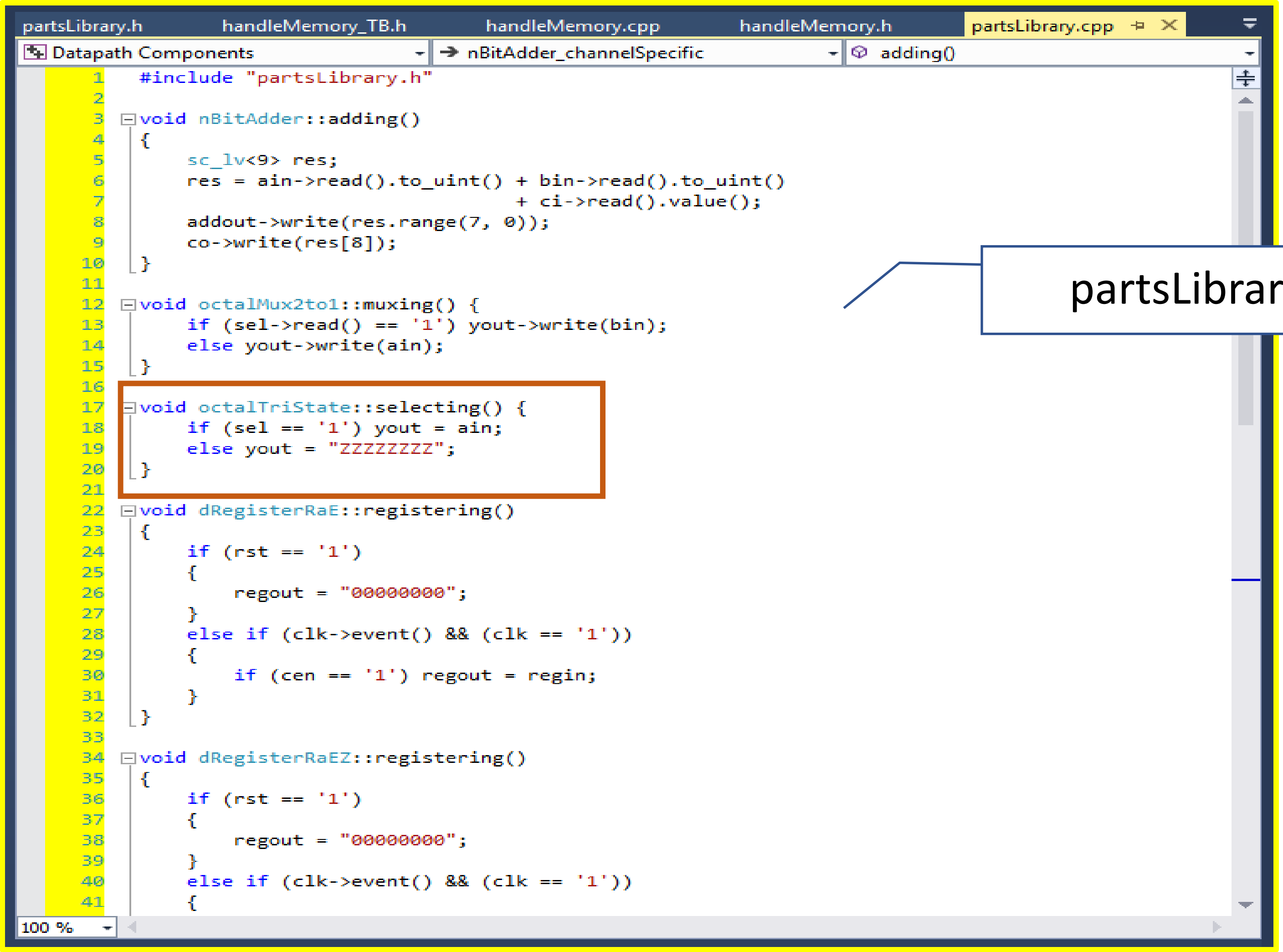
```
    sc_port<sc_signal_in_if <sc_logic>, 1> ci;
    sc_port<sc_signal_out_if <sc_lv<8>>, 1> addout;
    sc_port<sc_signal_out_if <sc_logic>, 1> co;

    SC_CTOR(nBitAdder)
    {
        SC_METHOD(adding);
        sensitive << ain << bin << ci;
    }
    void adding();
};

// octalMux2to1: 8-bit 2 to 1 multiplexer
SC_MODULE(octalMux2to1)
{
    sc_in<sc_logic> sel;
    sc_in<sc_lv<8> > ain, bin;
    sc_out<sc_lv<8> > yout;

    SC_CTOR(octalMux2to1)
    {
        SC_METHOD(muxing);
        sensitive << ain << bin << sel;
    }
    void muxing();
};

// octalTriState: 8-bit Tri-state logic
SC_MODULE(octalTriState)
{
    sc_in<sc_logic> sel;
    sc_in<sc_lv<8> > ain;
    sc_out<sc_lv<8> > yout;

    SC_CTOR(octalTriState)
    {
        SC_METHOD(selecting);
        sensitive << ain << sel;
    }
    void selecting();
};
```

partsLibrary.h

– Components for RTL Design

Combinational

- Tri-State

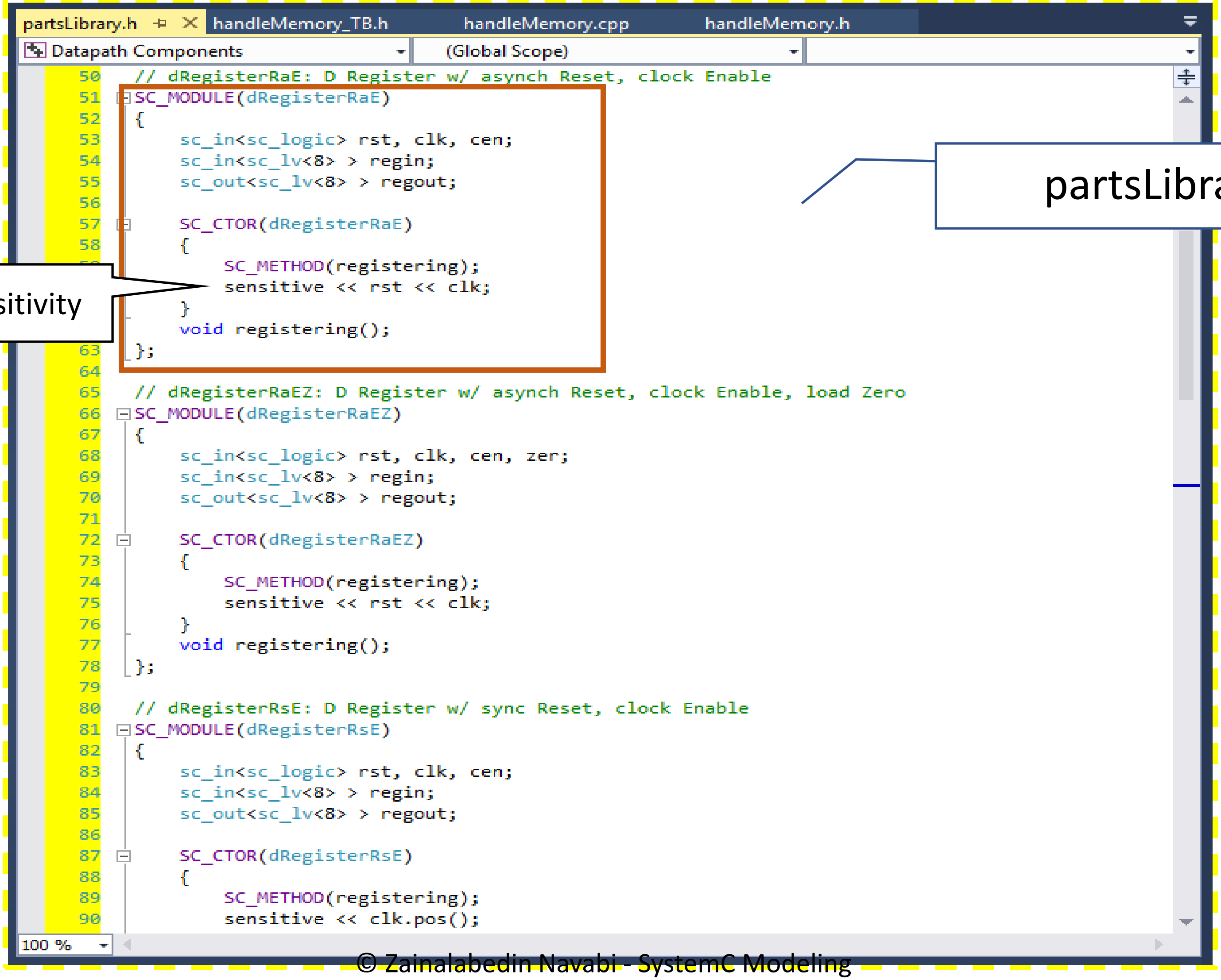


```
#include "partsLibrary.h"

void nBitAdder::adding()
{
    sc_lv<9> res;
    res = ain->read().to_uint() + bin->read().to_uint()
                            + ci->read().value();
    addout->write(res.range(7, 0));
    co->write(res[8]);
}

void octalMux2to1::muxing() {
    if (sel->read() == '1') yout->write(bin);
    else yout->write(ain);
}

void octalTriState::selecting() {
    if (sel == '1') yout = ain;
    else yout = "ZZZZZZZZ";
}

void dRegisterRaE::registering()
{
    if (rst == '1')
    {
        regout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
        if (cen == '1') regout = regin;
    }
}

void dRegisterRaEZ::registering()
{
    if (rst == '1')
    {
        regout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
```

partsLibrary.cpp

Sequential

- Register



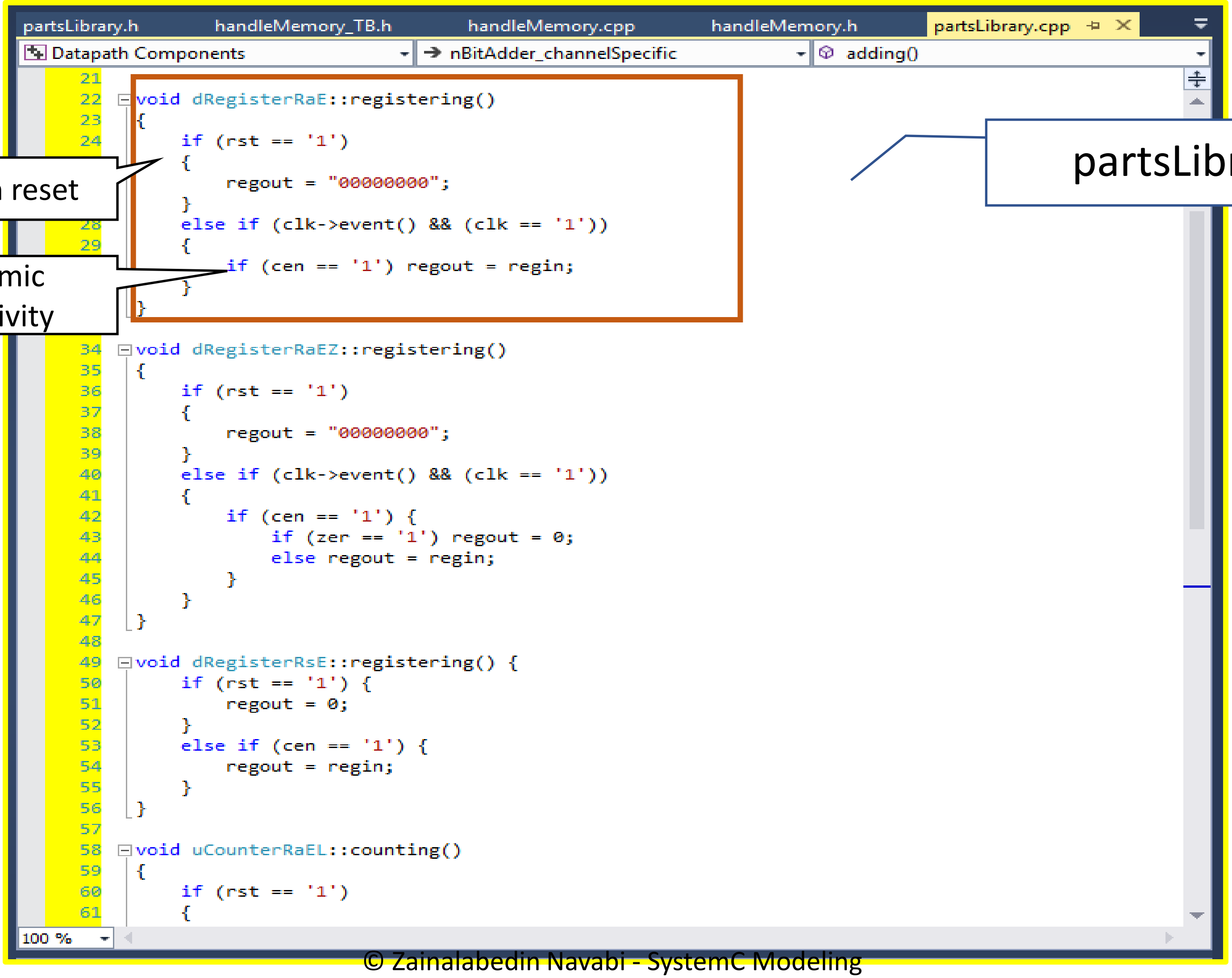
```
// dRegisterRaE: D Register w/ asynch Reset, clock Enable
SC_MODULE(dRegisterRaE)
{
    sc_in<sc_logic> rst, clk, cen;
    sc_in<sc_lv<8> > regin;
    sc_out<sc_lv<8> > regout;

    SC_CTOR(dRegisterRaE)
    {
        SC_METHOD(registering);
        sensitive << rst << clk;
    }
    void registering();
};

// dRegisterRaEZ: D Register w/ asynch Reset, clock Enable, load Zero
SC_MODULE(dRegisterRaEZ)
{
    sc_in<sc_logic> rst, clk, cen, zer;
    sc_in<sc_lv<8> > regin;
    sc_out<sc_lv<8> > regout;

    SC_CTOR(dRegisterRaEZ)
    {
        SC_METHOD(registering);
        sensitive << rst << clk;
    }
    void registering();
};

// dRegisterRsE: D Register w/ sync Reset, clock Enable
SC_MODULE(dRegisterRsE)
{
    sc_in<sc_logic> rst, clk, cen;
    sc_in<sc_lv<8> > regin;
    sc_out<sc_lv<8> > regout;

    SC_CTOR(dRegisterRsE)
    {
        SC_METHOD(registering);
        sensitive << clk.pos();
```

partsLibrary.h

Static sesitivity

Sequential

- Register



```
void dRegisterRaE::registering()
{
    if (rst == '1')
    {
        regout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
        if (cen == '1') regout = regin;
    }
}

void dRegisterRaEZ::registering()
{
    if (rst == '1')
    {
        regout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
        if (cen == '1') {
            if (zer == '1') regout = 0;
            else regout = regin;
        }
    }
}

void dRegisterRsE::registering() {
    if (rst == '1') {
        regout = 0;
    }
    else if (cen == '1') {
        regout = regin;
    }
}

void uCounterRaEL::counting()
{
    if (rst == '1')
    {
```

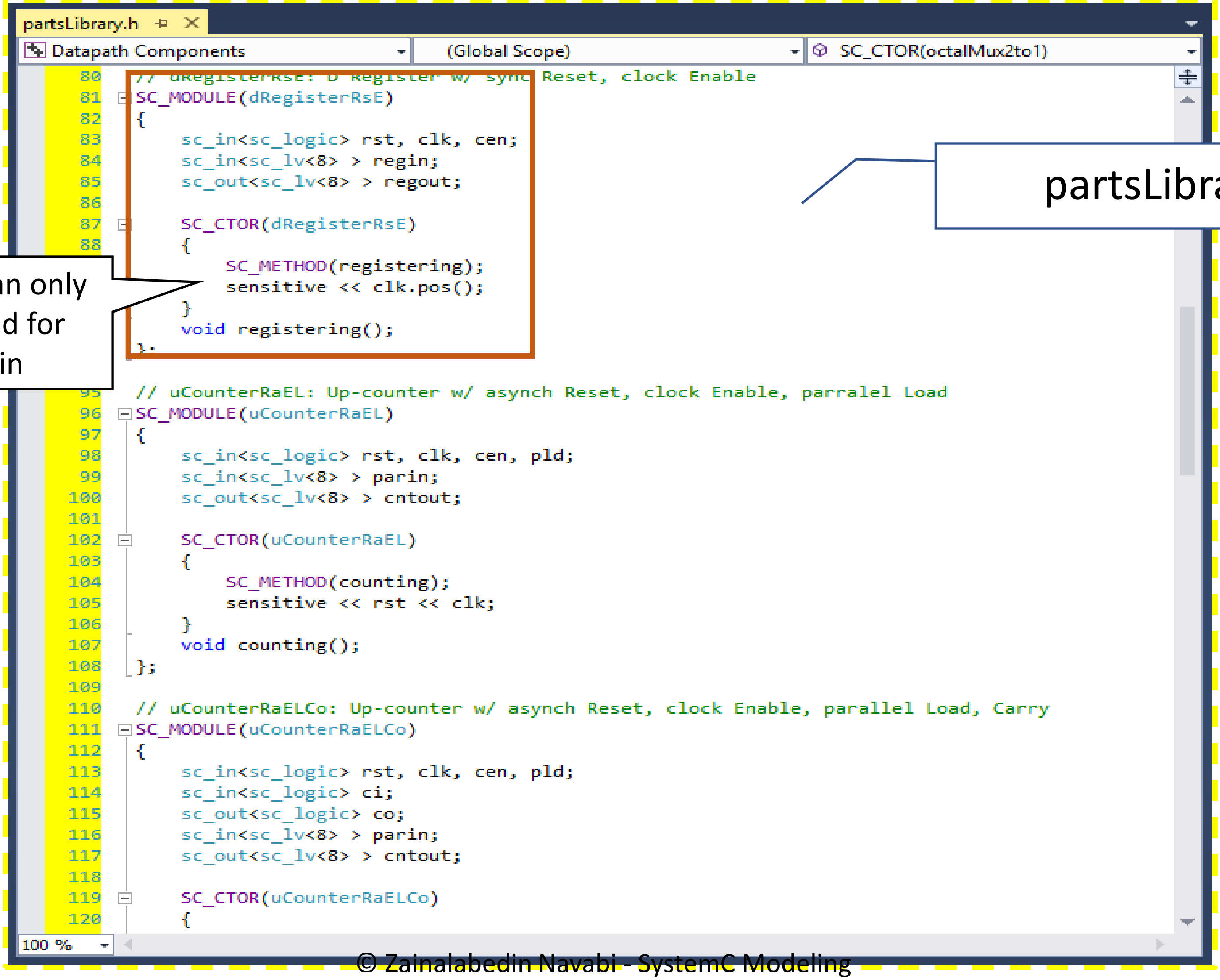
Asynch reset

Dynamic sensitivity

partsLibrary.cpp

Sequential

- Register



```
// dRegisterRsE: D Register w/ sync Reset, clock Enable
SC_MODULE(dRegisterRsE)
{
    sc_in<sc_logic> rst, clk, cen;
    sc_in<sc_lv<8> > regin;
    sc_out<sc_lv<8> > regout;

    SC_CTOR(dRegisterRsE)
    {
        SC_METHOD(registering);
        sensitive << clk.pos();
    }
    void registering();
};

// uCounterRaEL: Up-counter w/ asynch Reset, clock Enable, parralel Load
SC_MODULE(uCounterRaEL)
{
    sc_in<sc_logic> rst, clk, cen, pld;
    sc_in<sc_lv<8> > parin;
    sc_out<sc_lv<8> > cntout;

    SC_CTOR(uCounterRaEL)
    {
        SC_METHOD(counting);
        sensitive << rst << clk;
    }
    void counting();
};

// uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
SC_MODULE(uCounterRaELCo)
{
    sc_in<sc_logic> rst, clk, cen, pld;
    sc_in<sc_logic> ci;
    sc_out<sc_logic> co;
    sc_in<sc_lv<8> > parin;
    sc_out<sc_lv<8> > cntout;

    SC_CTOR(uCounterRaELCo)
    {
```

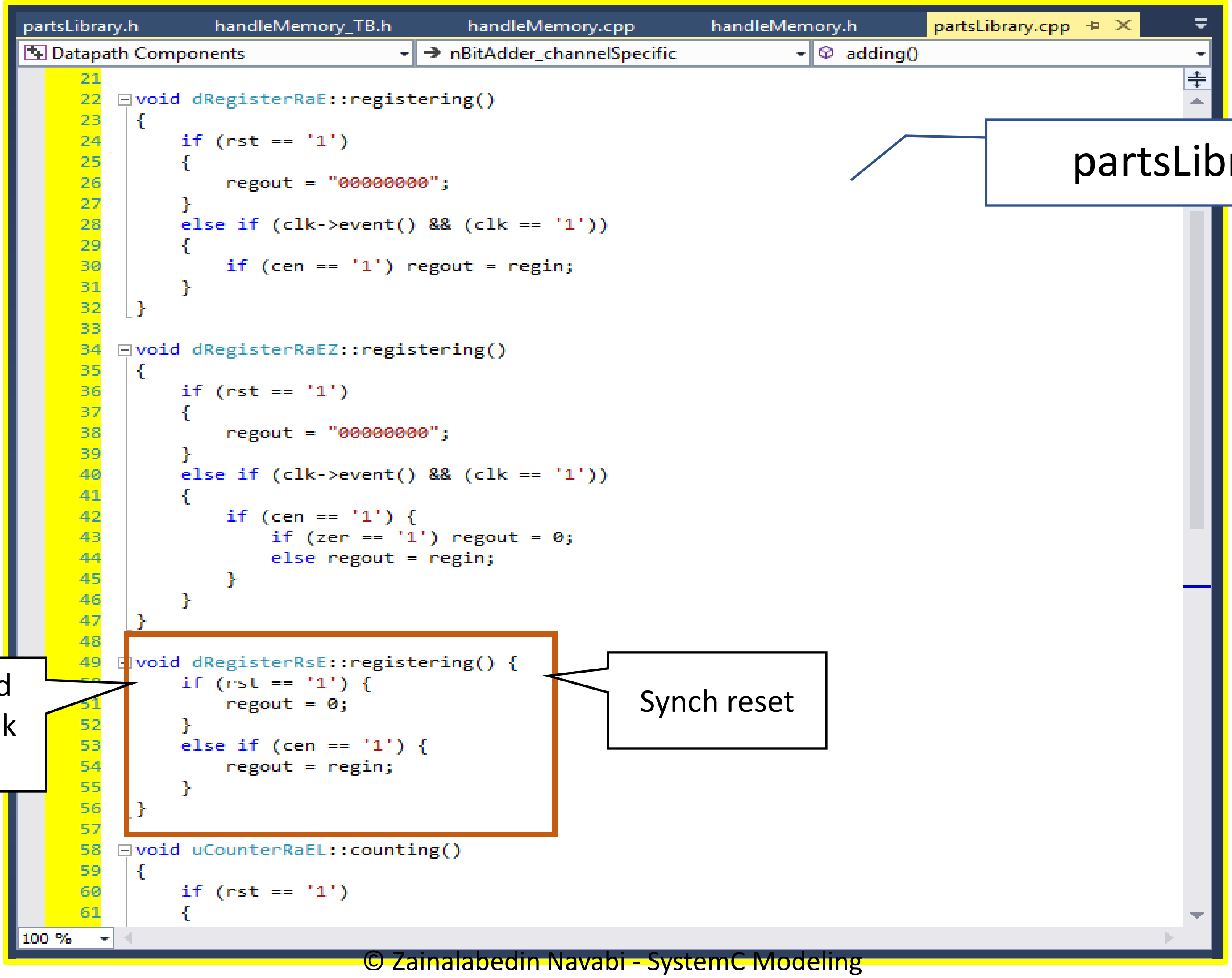
.pos() can only be used for sc_in

partsLibrary.h

Components for RTL Design

Sequential

- Register



```
void dRegisterRaE::registering()
{
    if (rst == '1')
    {
        regout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
        if (cen == '1') regout = regin;
    }
}

void dRegisterRaEZ::registering()
{
    if (rst == '1')
    {
        regout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
        if (cen == '1') {
            if (zer == '1') regout = 0;
            else regout = regin;
        }
    }
}

void dRegisterRsE::registering() {
    if (rst == '1') {
        regout = 0;
    }
    else if (cen == '1') {
        regout = regin;
    }
}

void uCounterRaEL::counting()
{
    if (rst == '1')
    {
```

partsLibrary.cpp

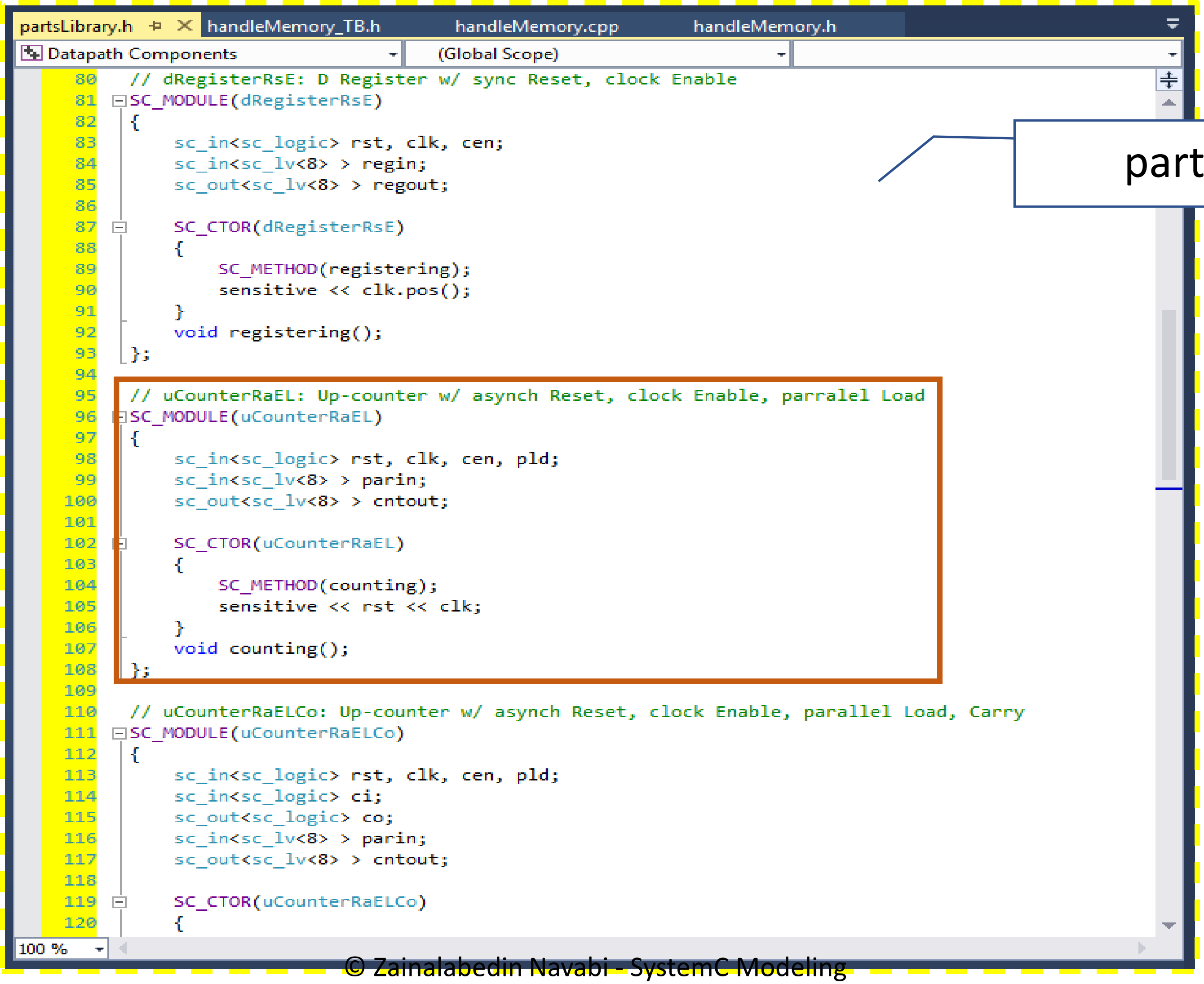
Doesn't need to check clock edge

Synch reset

Components for RTL Design

Sequential

- Counter

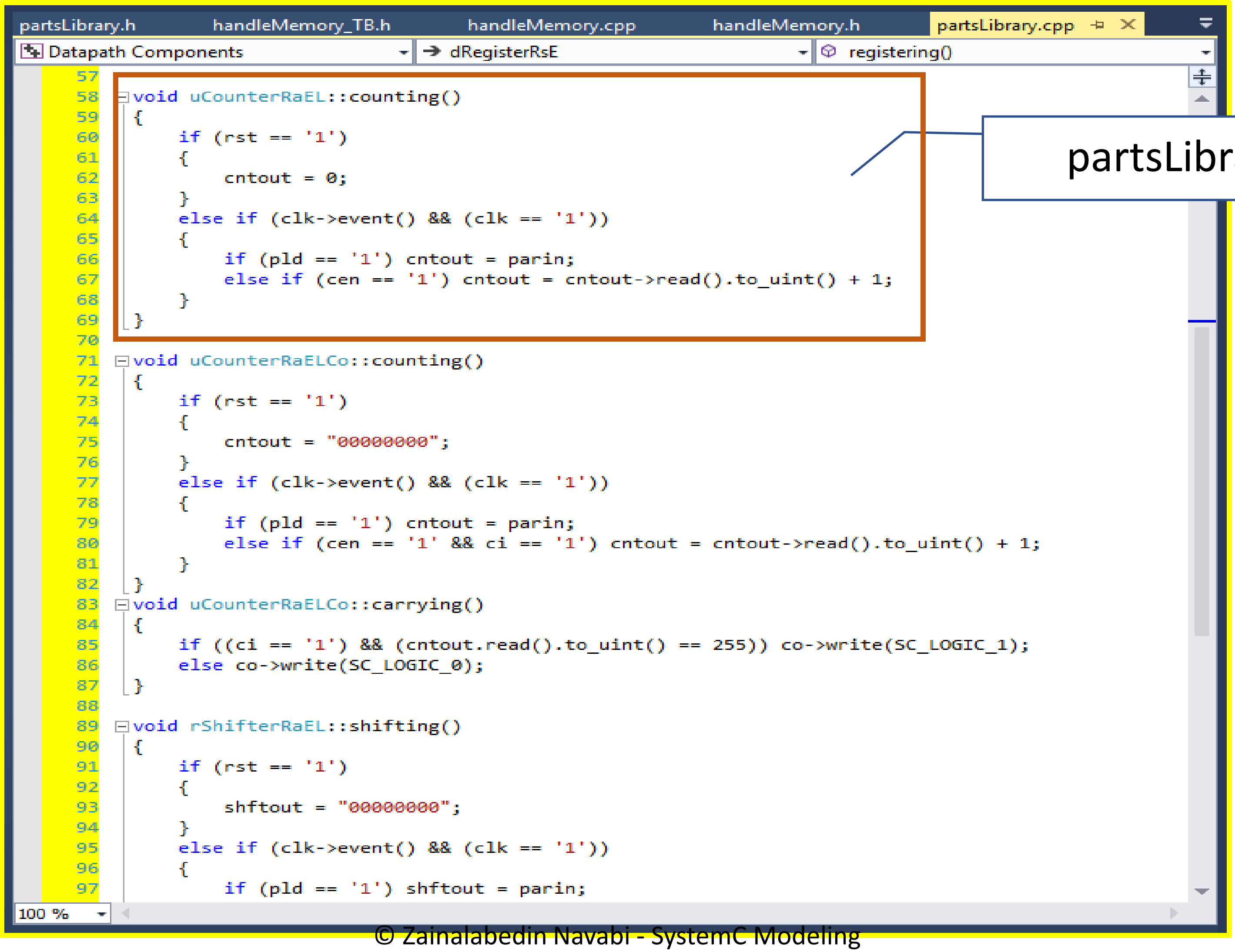


```
// dRegisterRsE: D Register w/ sync Reset, clock Enable
SC_MODULE(dRegisterRsE)
{
    sc_in<sc_logic> rst, clk, cen;
    sc_in<sc_lv<8> > regin;
    sc_out<sc_lv<8> > regout;

    SC_CTOR(dRegisterRsE)
    {
        SC_METHOD(registering);
        sensitive << clk.pos();
    }
    void registering();
};

// uCounterRaEL: Up-counter w/ asynch Reset, clock Enable, parralel Load
SC_MODULE(uCounterRaEL)
{
    sc_in<sc_logic> rst, clk, cen, pld;
    sc_in<sc_lv<8> > parin;
    sc_out<sc_lv<8> > cntout;

    SC_CTOR(uCounterRaEL)
    {
        SC_METHOD(counting);
        sensitive << rst << clk;
    }
    void counting();
};

// uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
SC_MODULE(uCounterRaELCo)
{
    sc_in<sc_logic> rst, clk, cen, pld;
    sc_in<sc_logic> ci;
    sc_out<sc_logic> co;
    sc_in<sc_lv<8> > parin;
    sc_out<sc_lv<8> > cntout;

    SC_CTOR(uCounterRaELCo)
    {
```

partsLibrary.h

Sequential

- Counter

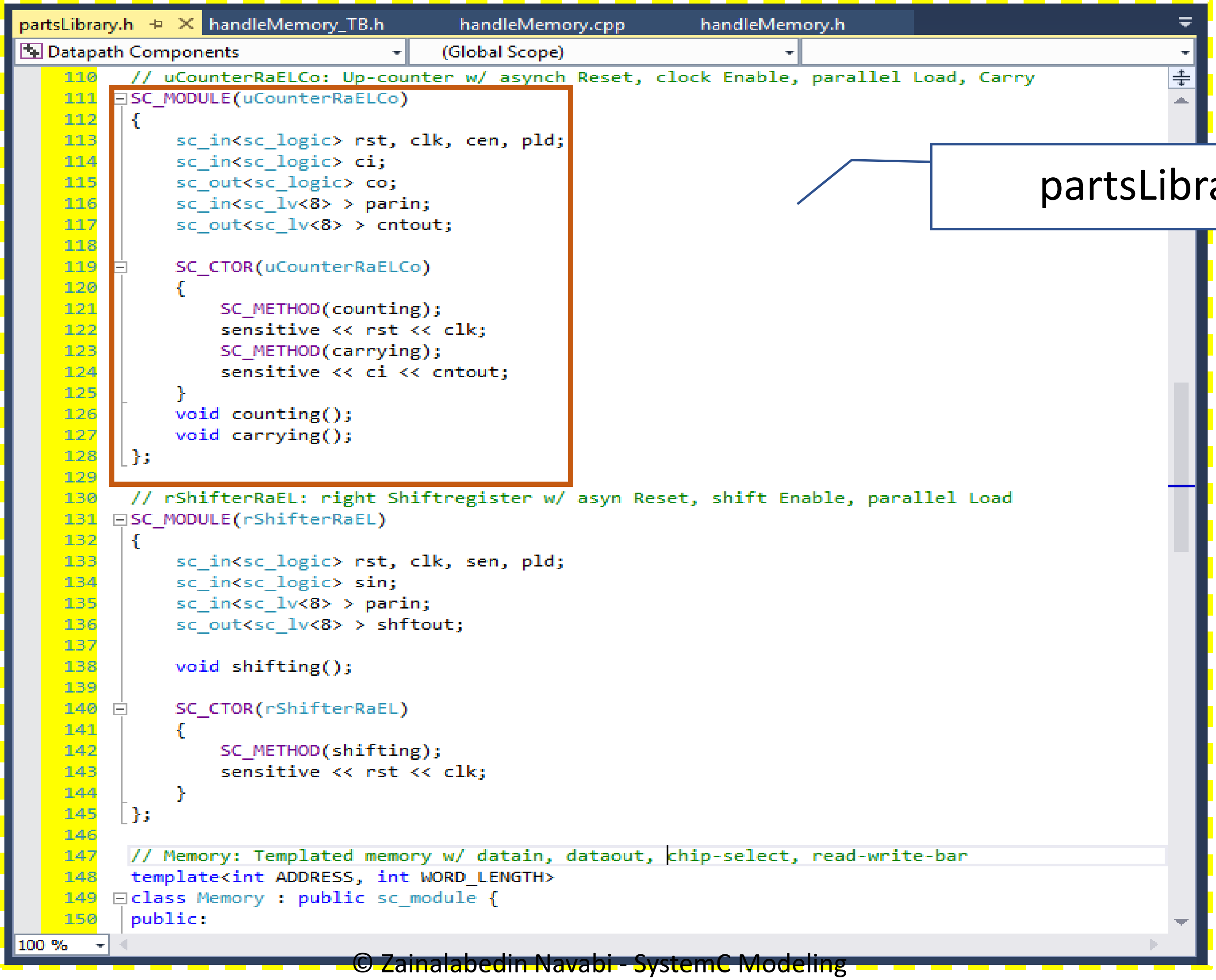


```

void uCounterRaEL::counting()
{
	if (rst == '1')
	{
		cntout = 0;
	}
	else if (clk->event() && (clk == '1'))
	{
		if (pld == '1') cntout = parin;
		else if (cen == '1') cntout = cntout->read().to_uint() + 1;
	}
}

void uCounterRaELCo::counting()
{
	if (rst == '1')
	{
		cntout = "00000000";
	}
	else if (clk->event() && (clk == '1'))
	{
		if (pld == '1') cntout = parin;
		else if (cen == '1' && ci == '1') cntout = cntout->read().to_uint() + 1;
	}
}
void uCounterRaELCo::carrying()
{
	if ((ci == '1') && (cntout.read().to_uint() == 255)) co->write(SC_LOGIC_1);
	else co->write(SC_LOGIC_0);
}

void rShifterRaEL::shifting()
{
	if (rst == '1')
	{
		shftout = "00000000";
	}
	else if (clk->event() && (clk == '1'))
	{
		if (pld == '1') shftout = parin;
```

partsLibrary.cpp

Sequential

- Counter

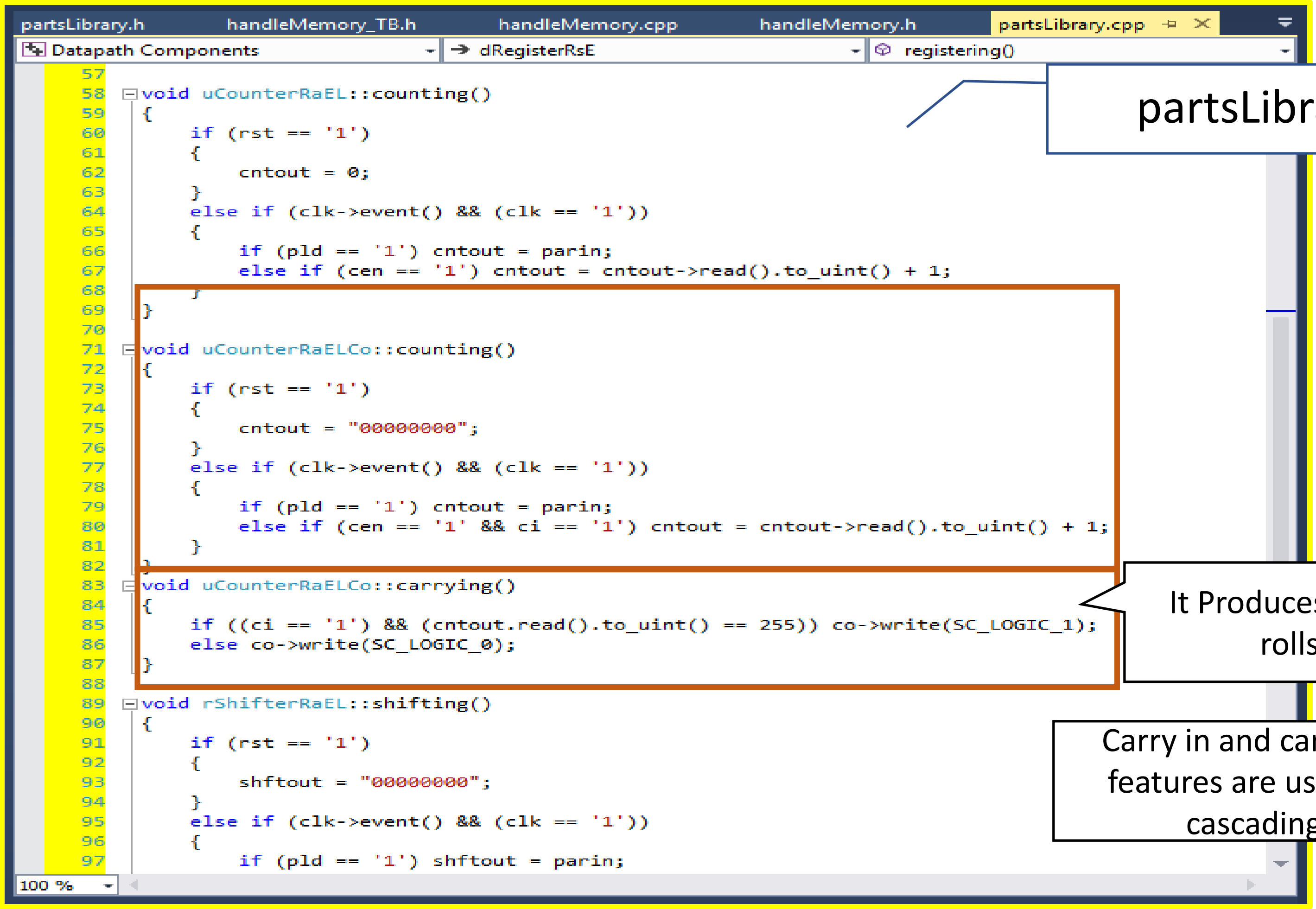


```
// uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
SC_MODULE(uCounterRaELCo)
{
    sc_in<sc_logic> rst, clk, cen, pld;
    sc_in<sc_logic> ci;
    sc_out<sc_logic> co;
    sc_in<sc_lv<8> > parin;
    sc_out<sc_lv<8> > cntout;

    SC_CTOR(uCounterRaELCo)
    {
        SC_METHOD(counting);
        sensitive << rst << clk;
        SC_METHOD(carrying);
        sensitive << ci << cntout;
    }
    void counting();
    void carrying();
};

// rShifterRaEL: right Shiftregister w/ asyn Reset, shift Enable, parallel Load
SC_MODULE(rShifterRaEL)
{
    sc_in<sc_logic> rst, clk, sen, pld;
    sc_in<sc_logic> sin;
    sc_in<sc_lv<8> > parin;
    sc_out<sc_lv<8> > shftout;

    void shifting();

    SC_CTOR(rShifterRaEL)
    {
        SC_METHOD(shifting);
        sensitive << rst << clk;
    }
};

// Memory: Templated memory w/ datain, dataout, chip-select, read-write-bar
template<int ADDRESS, int WORD_LENGTH>
class Memory : public sc_module {
public:
```

partsLibrary.h

Sequential

- Counter



```

void uCounterRaEL::counting()
{
    if (rst == '1')
    {
        cntout = 0;
    }
    else if (clk->event() && (clk == '1'))
    {
        if (pld == '1') cntout = parin;
        else if (cen == '1') cntout = cntout->read().to_uint() + 1;
    }
}

void uCounterRaELCo::counting()
{
    if (rst == '1')
    {
        cntout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
        if (pld == '1') cntout = parin;
        else if (cen == '1' && ci == '1') cntout = cntout->read().to_uint() + 1;
    }
}
void uCounterRaELCo::carrying()
{
    if ((ci == '1') && (cntout.read().to_uint() == 255)) co->write(SC_LOGIC_1);
    else co->write(SC_LOGIC_0);
}

void rShifterRaEL::shifting()
{
    if (rst == '1')
    {
        shftout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
        if (pld == '1') shftout = parin;
```

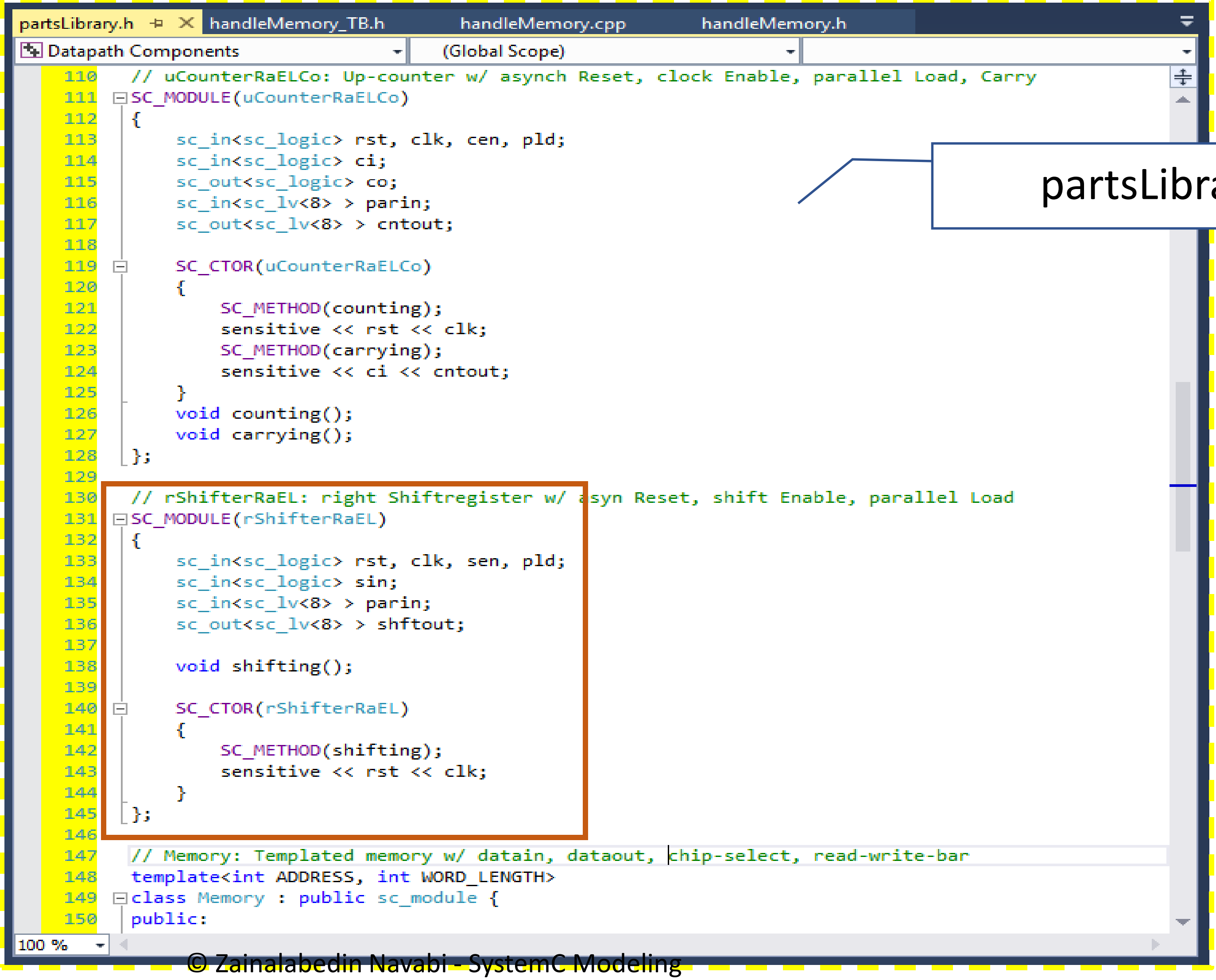
partsLibrary.cpp

It Produces co when it rolls over

Carry in and carry out features are used for cascading

Sequential

- Shift-Register



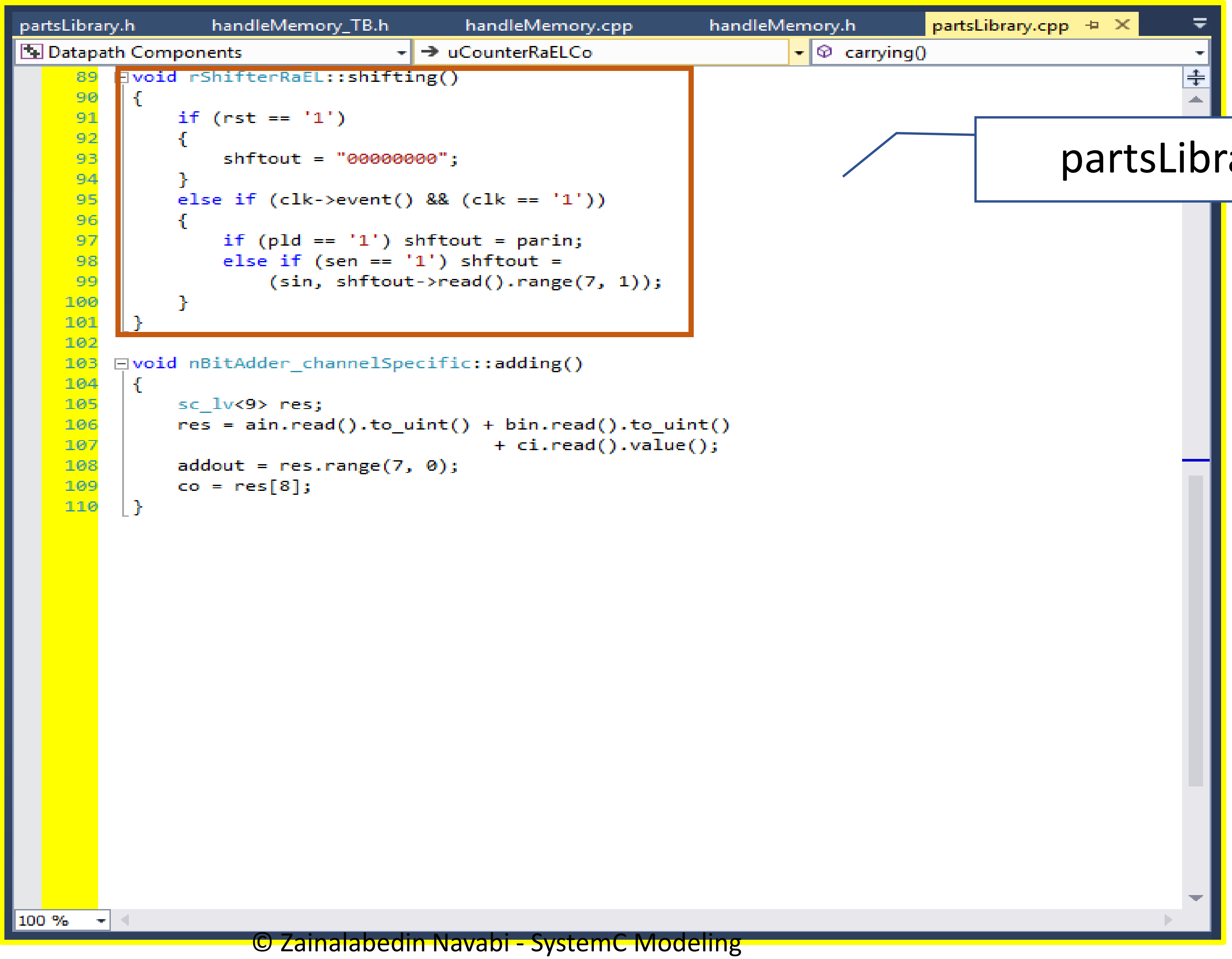
```
// uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
SC_MODULE(uCounterRaELCo)
{
    sc_in<sc_logic> rst, clk, cen, pld;
    sc_in<sc_logic> ci;
    sc_out<sc_logic> co;
    sc_in<sc_lv<8> > parin;
    sc_out<sc_lv<8> > cntout;

    SC_CTOR(uCounterRaELCo)
    {
        SC_METHOD(counting);
        sensitive << rst << clk;
        SC_METHOD(carrying);
        sensitive << ci << cntout;
    }
    void counting();
    void carrying();
};

// rShifterRaEL: right Shiftregister w/ asyn Reset, shift Enable, parallel Load
SC_MODULE(rShifterRaEL)
{
    sc_in<sc_logic> rst, clk, sen, pld;
    sc_in<sc_logic> sin;
    sc_in<sc_lv<8> > parin;
    sc_out<sc_lv<8> > shftout;

    void shifting();

    SC_CTOR(rShifterRaEL)
    {
        SC_METHOD(shifting);
        sensitive << rst << clk;
    }
};

// Memory: Templated memory w/ datain, dataout, chip-select, read-write-bar
template<int ADDRESS, int WORD_LENGTH>
class Memory : public sc_module {
public:
```

partsLibrary.h

Components for RTL Design

Sequential

- Shift-Register



```
void rShifterRaEL::shifting()
{
    if (rst == '1')
    {
        shftout = "00000000";
    }
    else if (clk->event() && (clk == '1'))
    {
        if (pld == '1') shftout = parin;
        else if (sen == '1') shftout =
            (sin, shftout->read().range(7, 1));
    }
}

void nBitAdder_channelSpecific::adding()
{
    sc_lv<9> res;
    res = ain.read().to_uint() + bin.read().to_uint()
                                + ci.read().value();
    addout = res.range(7, 0);
    co = res[8];
}
```

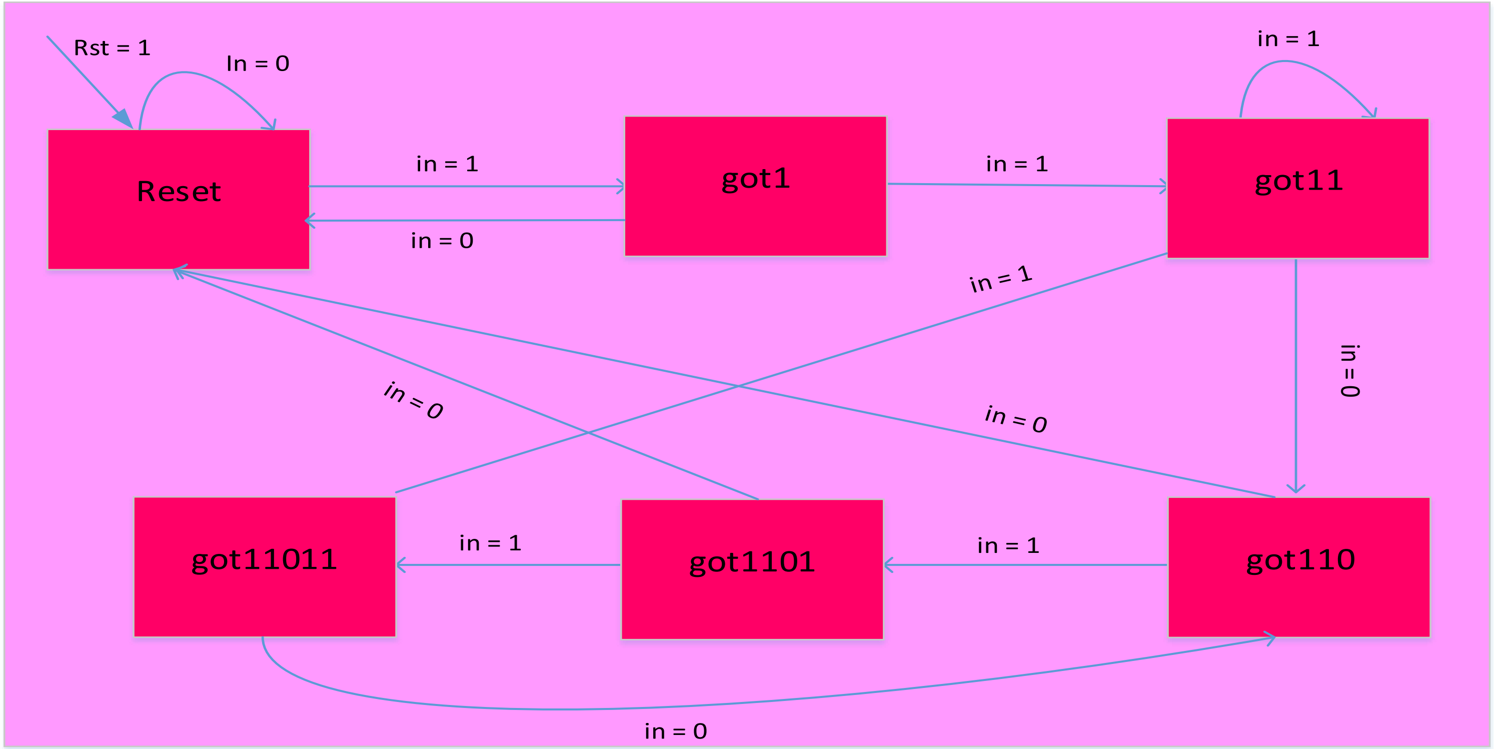
partsLibrary.cpp

SystemC Modeling

+ Taking Off From C++
+ SystemC Gate-Level Modeling
  Utilities for HDL Orientation
+ Timing & SystemC RT-Level Modeling
+ Components for RTL Design
– SystemC RTL Design – Example
    - Sequence Detector 11011
    - A configurable Memory
    - Exponential Circuit
  SystemC Functional Modeling
+ SystemC Functional Design - Example
  Summary

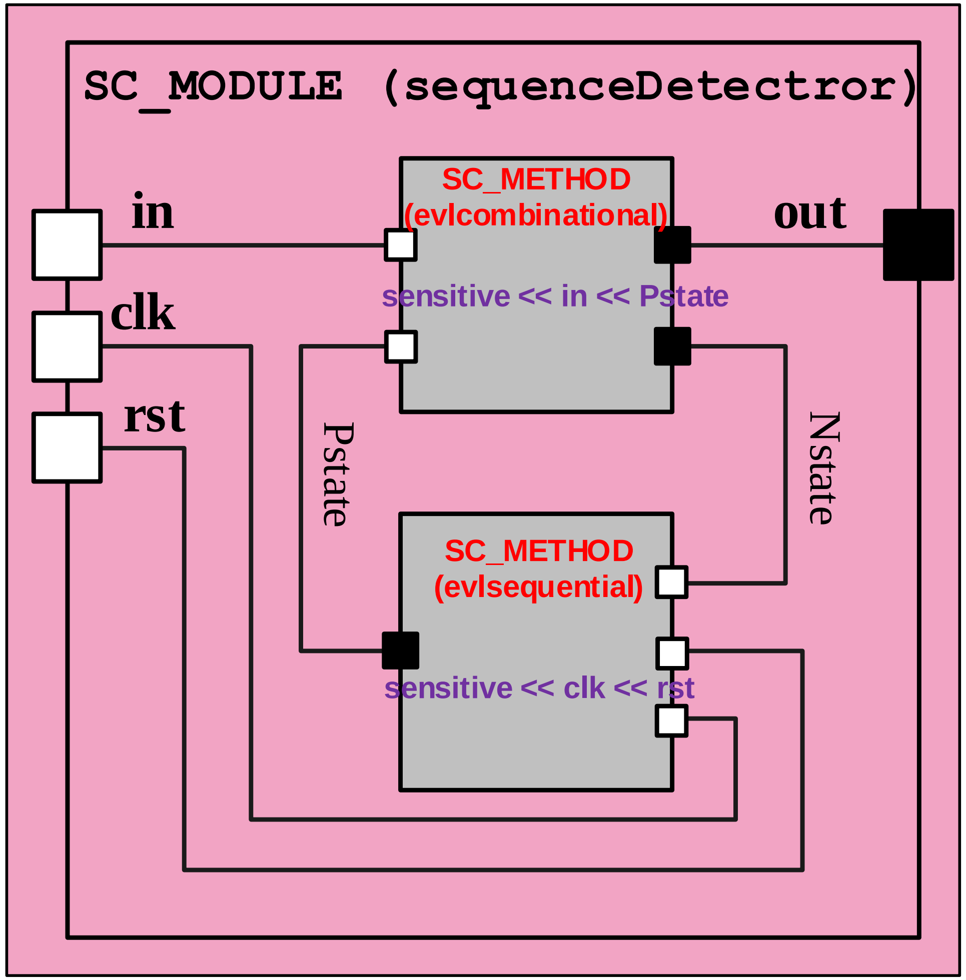
Sequence Detector 11011

- State Diagram

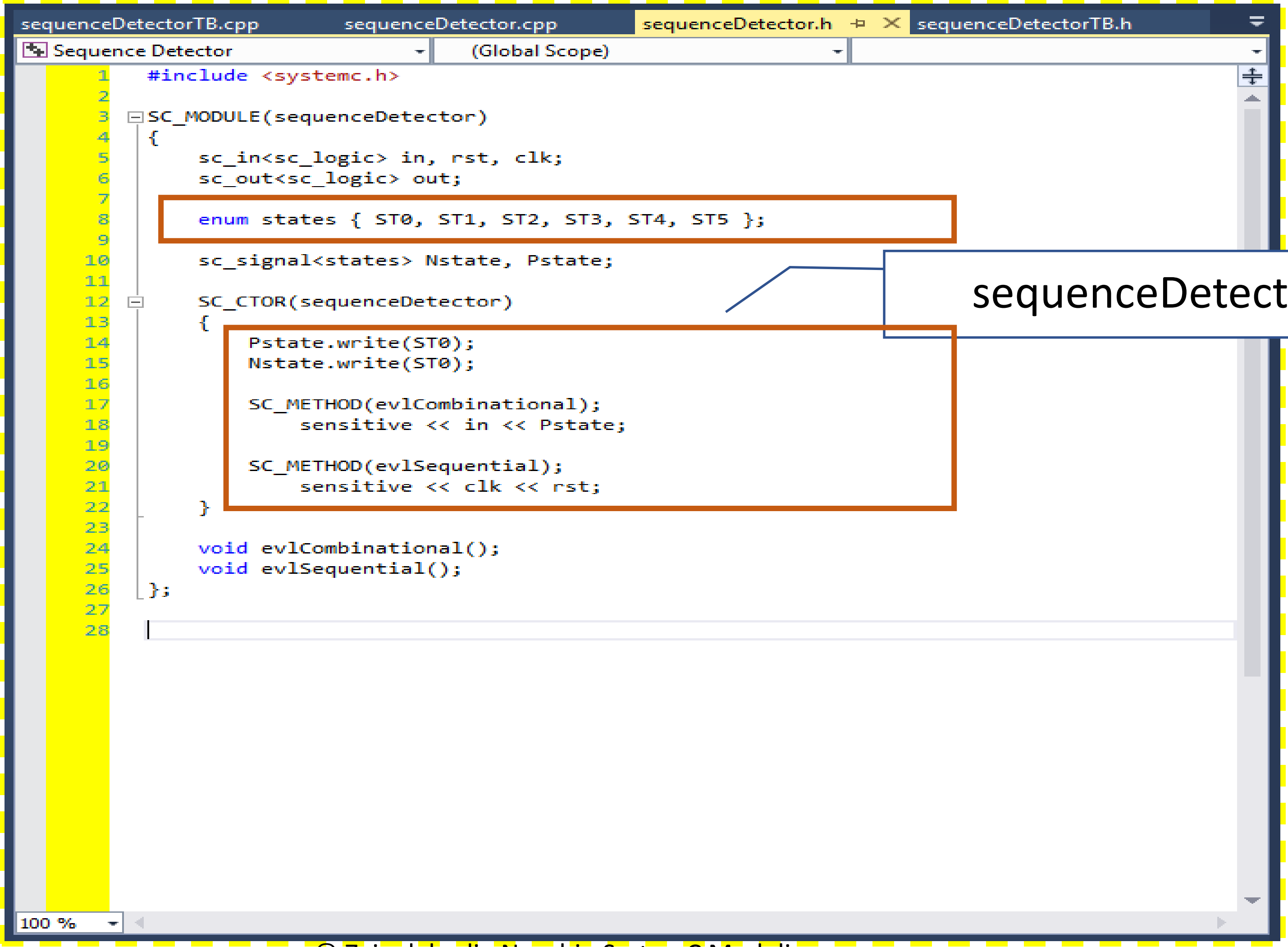


Sequence Detector 11011

- Huffman Model



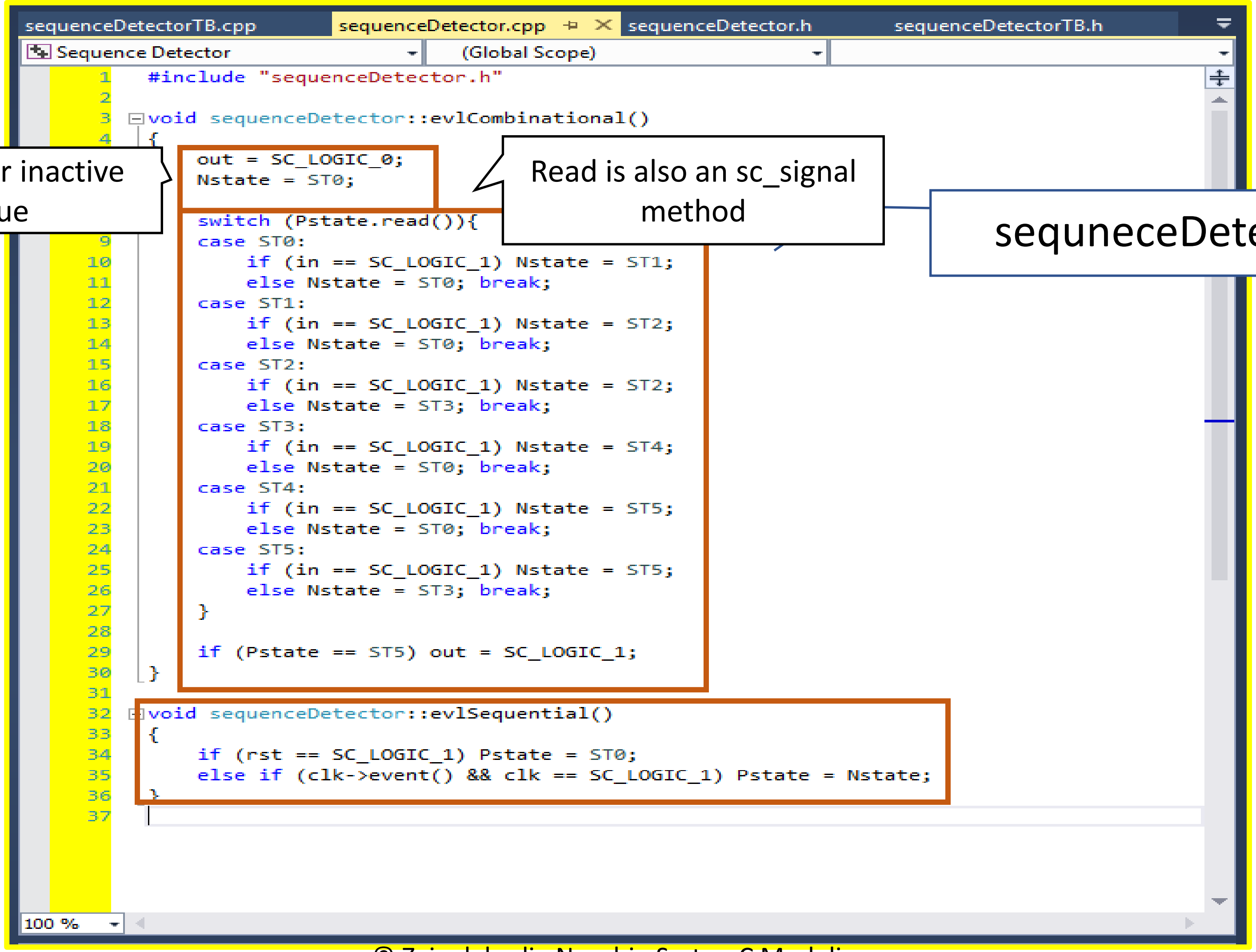
Sequence Detector 11011



```
#include <systemc.h>

SC_MODULE(sequenceDetector)
{
    sc_in<sc_logic> in, rst, clk;
    sc_out<sc_logic> out;

    enum states { ST0, ST1, ST2, ST3, ST4, ST5 };

    sc_signal<states> Nstate, Pstate;

    SC_CTOR(sequenceDetector)
    {
        Pstate.write(ST0);
        Nstate.write(ST0);

        SC_METHOD(evlCombinational);
            sensitive << in << Pstate;

        SC_METHOD(evlSequential);
            sensitive << clk << rst;
    }

    void evlCombinational();
    void evlSequential();
};

```

sequenceDetector.h

Sequence Detector 11011



```
#include "sequenceDetector.h"

void sequenceDetector::evlCombinational()
{
    out = SC_LOGIC_0;
    Nstate = ST0;

    switch (Pstate.read()){
    case ST0:
        if (in == SC_LOGIC_1) Nstate = ST1;
        else Nstate = ST0; break;
    case ST1:
        if (in == SC_LOGIC_1) Nstate = ST2;
        else Nstate = ST0; break;
    case ST2:
        if (in == SC_LOGIC_1) Nstate = ST2;
        else Nstate = ST3; break;
    case ST3:
        if (in == SC_LOGIC_1) Nstate = ST4;
        else Nstate = ST0; break;
    case ST4:
        if (in == SC_LOGIC_1) Nstate = ST5;
        else Nstate = ST0; break;
    case ST5:
        if (in == SC_LOGIC_1) Nstate = ST5;
        else Nstate = ST3; break;
    }

    if (Pstate == ST5) out = SC_LOGIC_1;
}

void sequenceDetector::evlSequential()
{
    if (rst == SC_LOGIC_1) Pstate = ST0;
    else if (clk->event() && clk == SC_LOGIC_1) Pstate = Nstate;
}
```

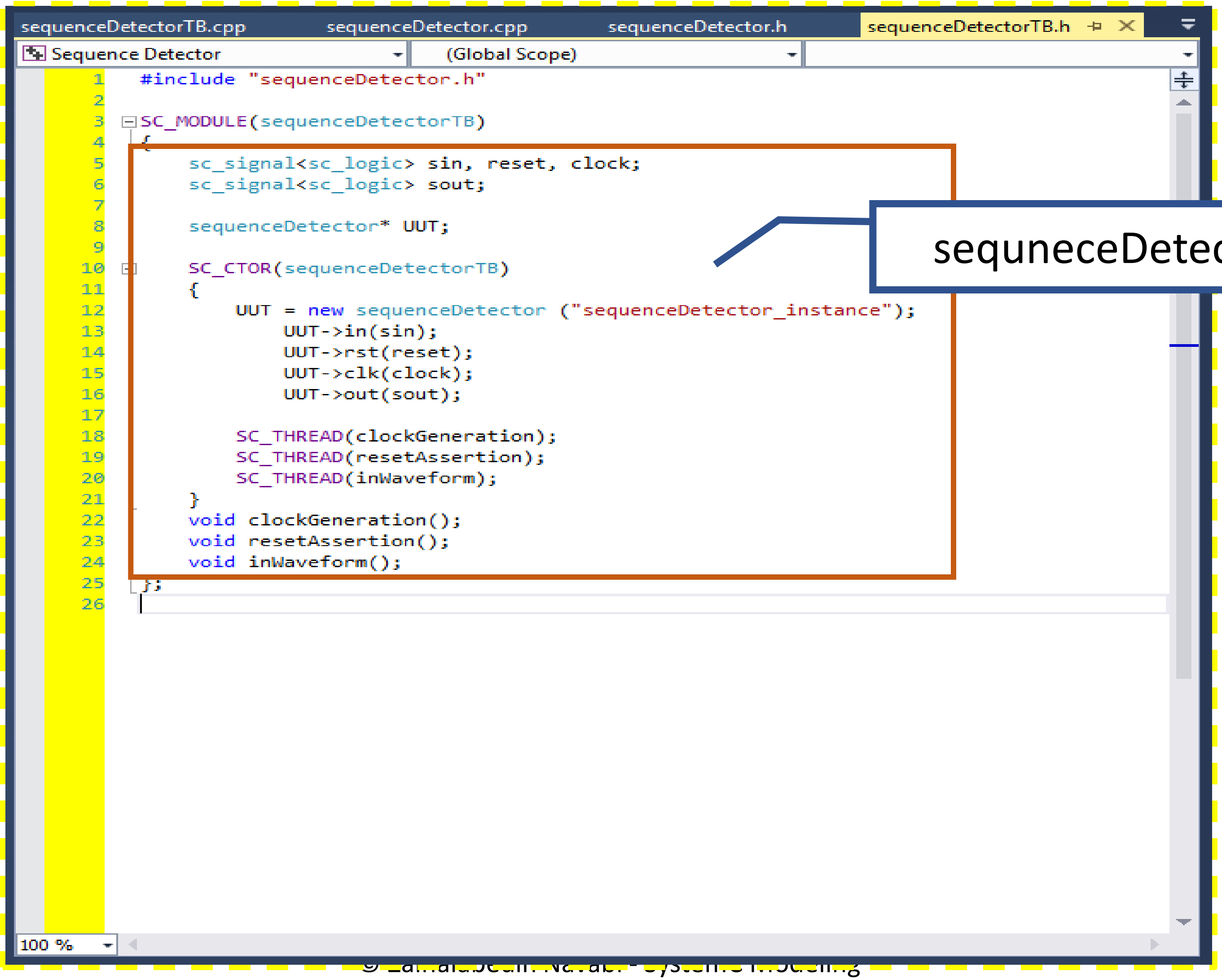
Set to their inactive value

Read is also an sc_signal method

sequneceDetector.cpp

Sequence Detector 11011

- Testbench

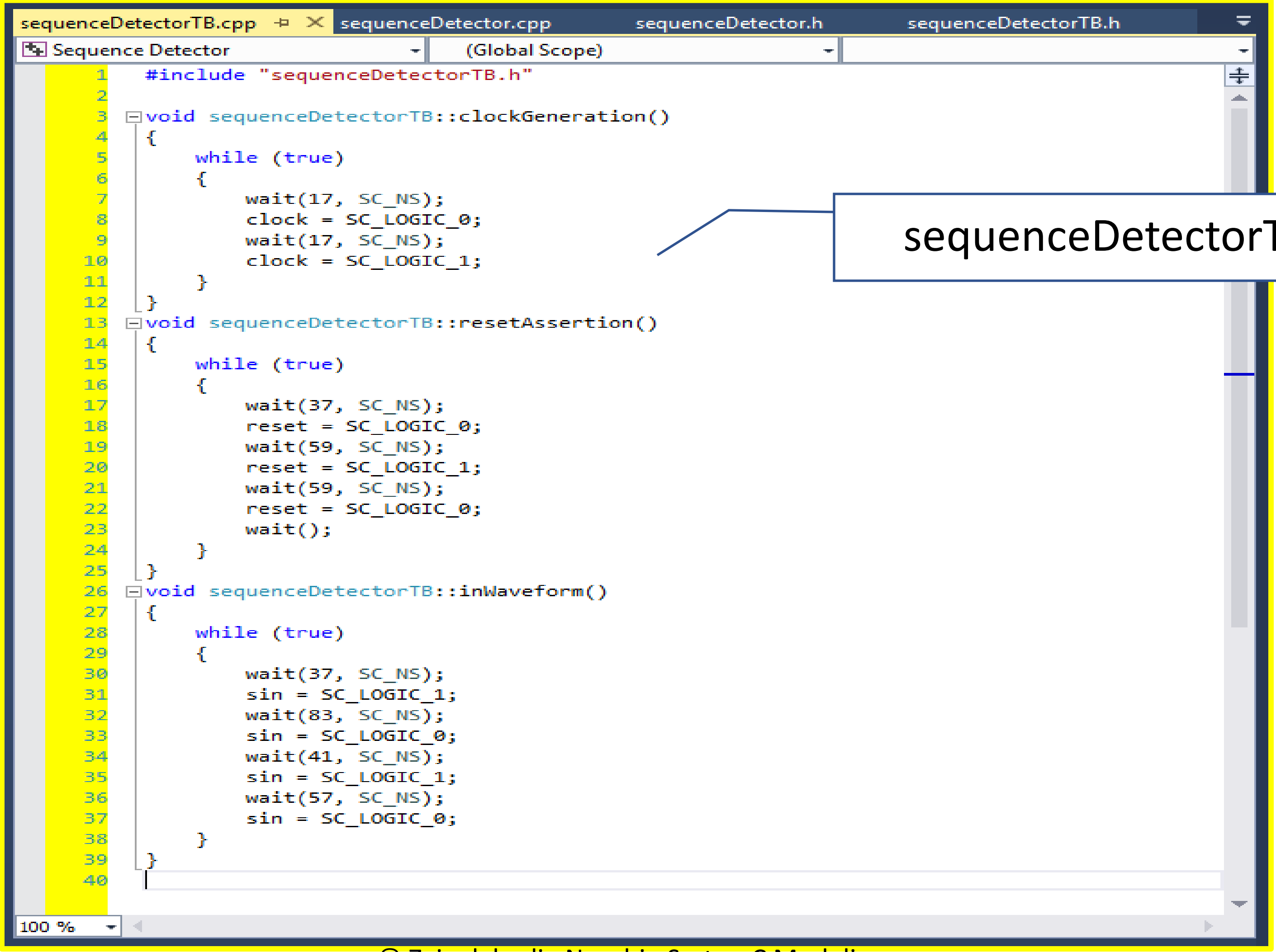


```
#include "sequenceDetector.h"

SC_MODULE(sequenceDetectorTB)
{
    sc_signal<sc_logic> sin, reset, clock;
    sc_signal<sc_logic> sout;

    sequenceDetector* UUT;

    SC_CTOR(sequenceDetectorTB)
    {
        UUT = new sequenceDetector ("sequenceDetector_instance");
            UUT->in(sin);
            UUT->rst(reset);
            UUT->clk(clock);
            UUT->out(sout);

        SC_THREAD(clockGeneration);
        SC_THREAD(resetAssertion);
        SC_THREAD(inWaveform);
    }
    void clockGeneration();
    void resetAssertion();
    void inWaveform();
};
```

sequneceDetectorTB.h

Sequence Detector 11011

- Testbench

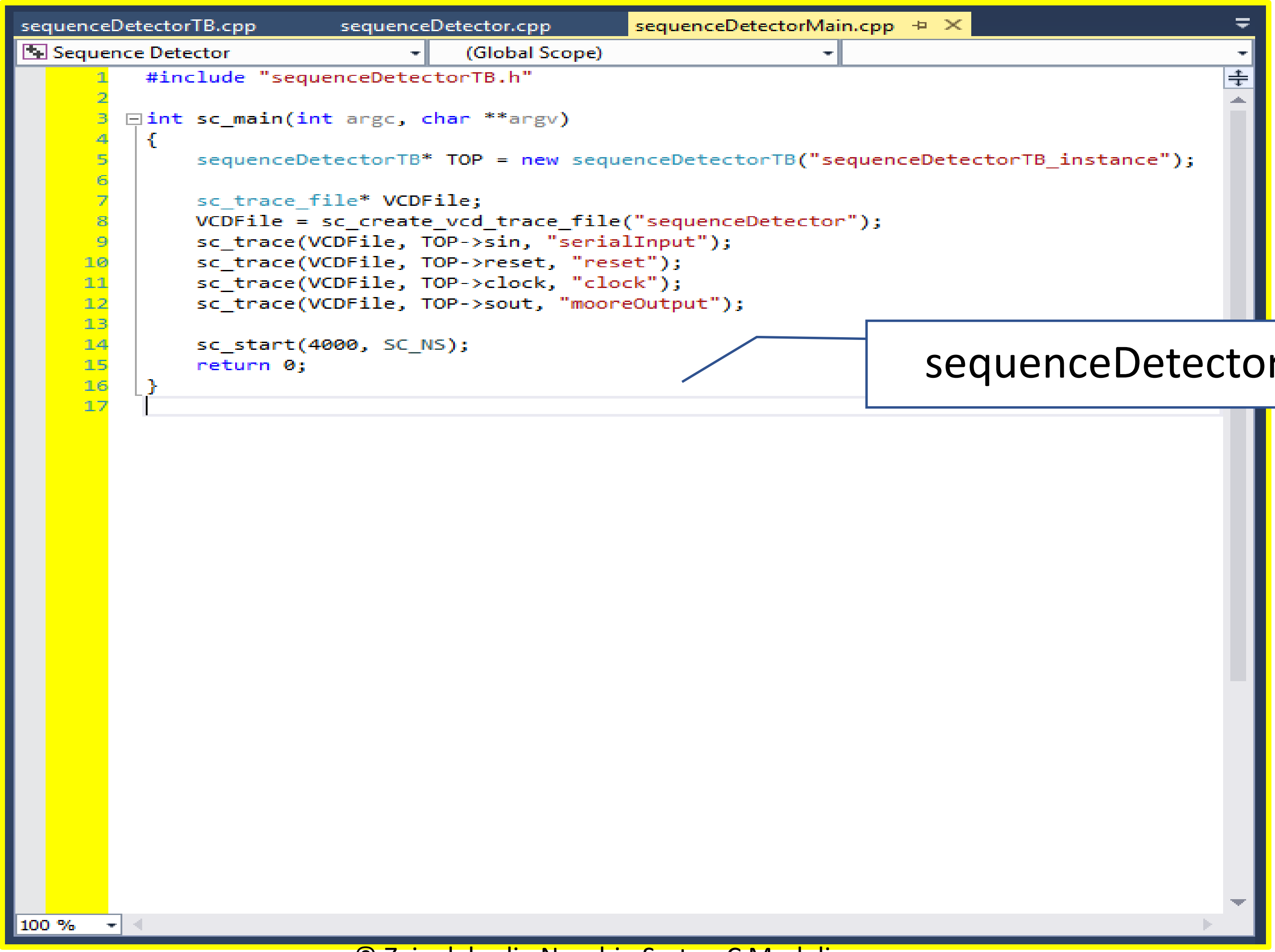


```
#include "sequenceDetectorTB.h"

void sequenceDetectorTB::clockGeneration()
{
    while (true)
    {
        wait(17, SC_NS);
        clock = SC_LOGIC_0;
        wait(17, SC_NS);
        clock = SC_LOGIC_1;
    }
}
void sequenceDetectorTB::resetAssertion()
{
    while (true)
    {
        wait(37, SC_NS);
        reset = SC_LOGIC_0;
        wait(59, SC_NS);
        reset = SC_LOGIC_1;
        wait(59, SC_NS);
        reset = SC_LOGIC_0;
        wait();
    }
}
void sequenceDetectorTB::inWaveform()
{
    while (true)
    {
        wait(37, SC_NS);
        sin = SC_LOGIC_1;
        wait(83, SC_NS);
        sin = SC_LOGIC_0;
        wait(41, SC_NS);
        sin = SC_LOGIC_1;
        wait(57, SC_NS);
        sin = SC_LOGIC_0;
    }
}

```

sequenceDetectorTB.cpp

Sequence Detector 11011

- sc_main

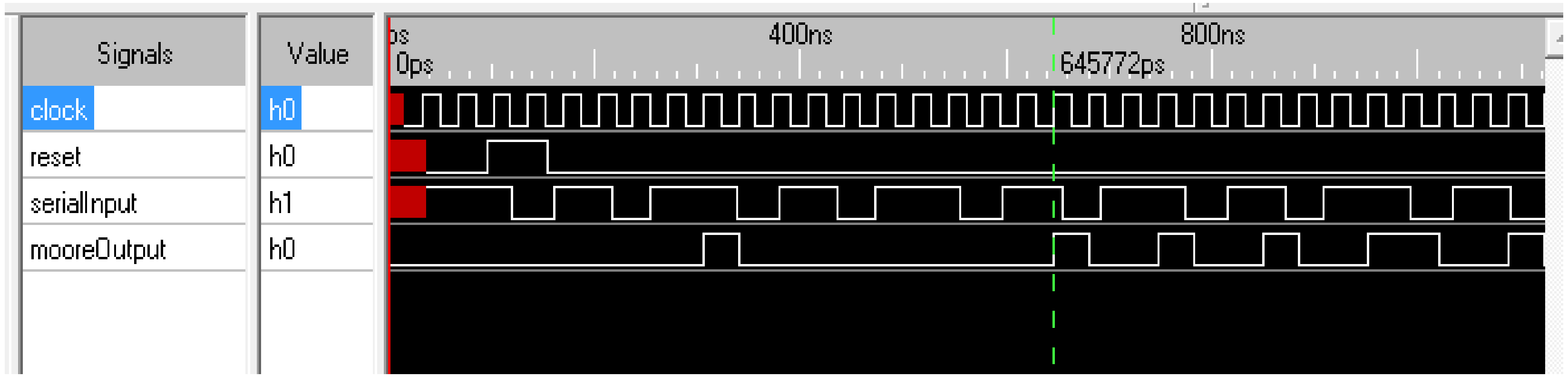


```
#include "sequenceDetectorTB.h"

int sc_main(int argc, char **argv)
{
    sequenceDetectorTB* TOP = new sequenceDetectorTB("sequenceDetectorTB_instance");

    sc_trace_file* VCDFile;
    VCDFile = sc_create_vcd_trace_file("sequenceDetector");
    sc_trace(VCDFile, TOP->sin, "serialInput");
    sc_trace(VCDFile, TOP->reset, "reset");
    sc_trace(VCDFile, TOP->clock, "clock");
    sc_trace(VCDFile, TOP->sout, "mooreOutput");

    sc_start(4000, SC_NS);
    return 0;
}
```

sequenceDetectorMain.cpp

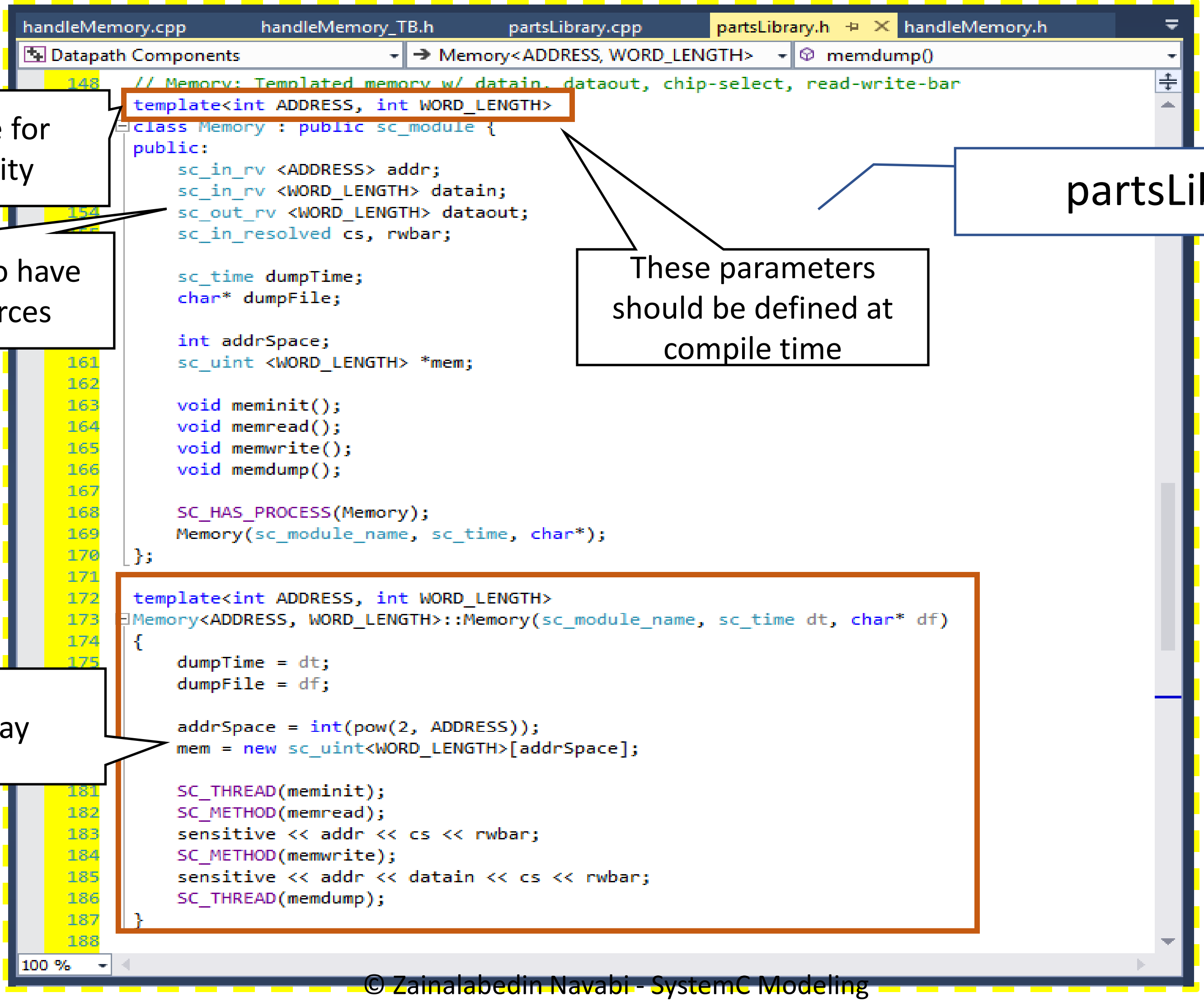
Sequence Detector 11011

- VCD waveform



Signals	Value
clock	h0
reset	h0
serialInput	h1
mooreOutput	h0

A Configurable Memory



```
// Memory: Templated memory w/ datain, dataout, chip-select, read-write-bar
template<int ADDRESS, int WORD_LENGTH>
class Memory : public sc_module {
public:
    sc_in_rv <ADDRESS> addr;
    sc_in_rv <WORD_LENGTH> datain;
    sc_out_rv <WORD_LENGTH> dataout;
    sc_in_resolved cs, rwbar;

    sc_time dumpTime;
    char* dumpFile;

    int addrSpace;
    sc_uint <WORD_LENGTH> *mem;

    void meminit();
    void memread();
    void memwrite();
    void memdump();

    SC_HAS_PROCESS(Memory);
    Memory(sc_module_name, sc_time, char*);
};

template<int ADDRESS, int WORD_LENGTH>
Memory<ADDRESS, WORD_LENGTH>::Memory(sc_module_name, sc_time dt, char* df)
{
    dumpTime = dt;
    dumpFile = df;

    addrSpace = int(pow(2, ADDRESS));
    mem = new sc_uint<WORD_LENGTH>[addrSpace];

    SC_THREAD(meminit);
    SC_METHOD(memread);
    sensitive << addr << cs << rwbar;
    SC_METHOD(memwrite);
    sensitive << addr << datain << cs << rwbar;
    SC_THREAD(memdump);
}
```

Use Template for Configurability

Use resolved to have multiple sources

These parameters should be defined at compile time

partsLibrary.h

Memory Array

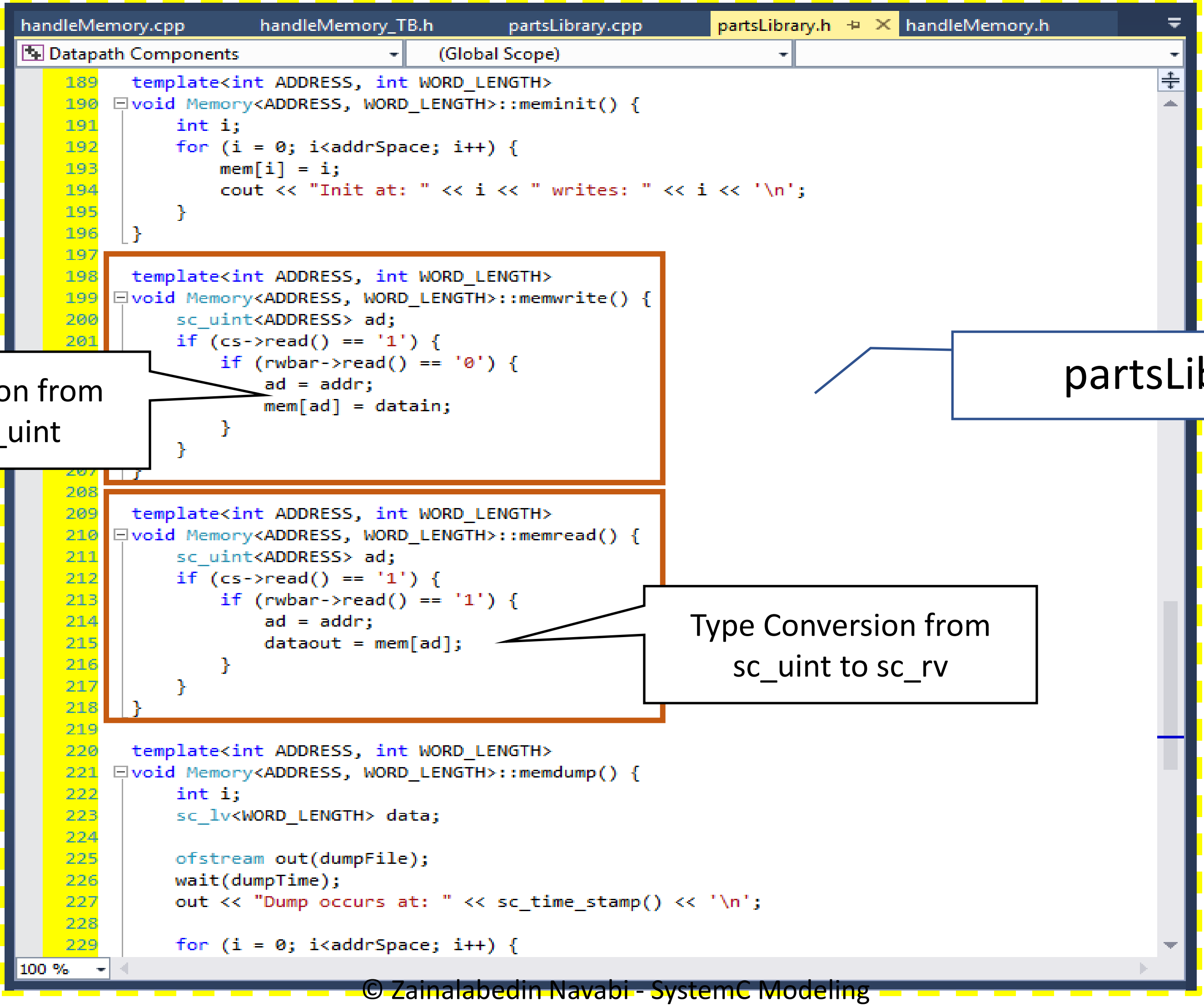
A Configurable Memory

- Resolved value

Driving Value 2 \ Driving Value 1	X	0	1	Z
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

Z is the weakest and X is the strongest

A Configurable Memory



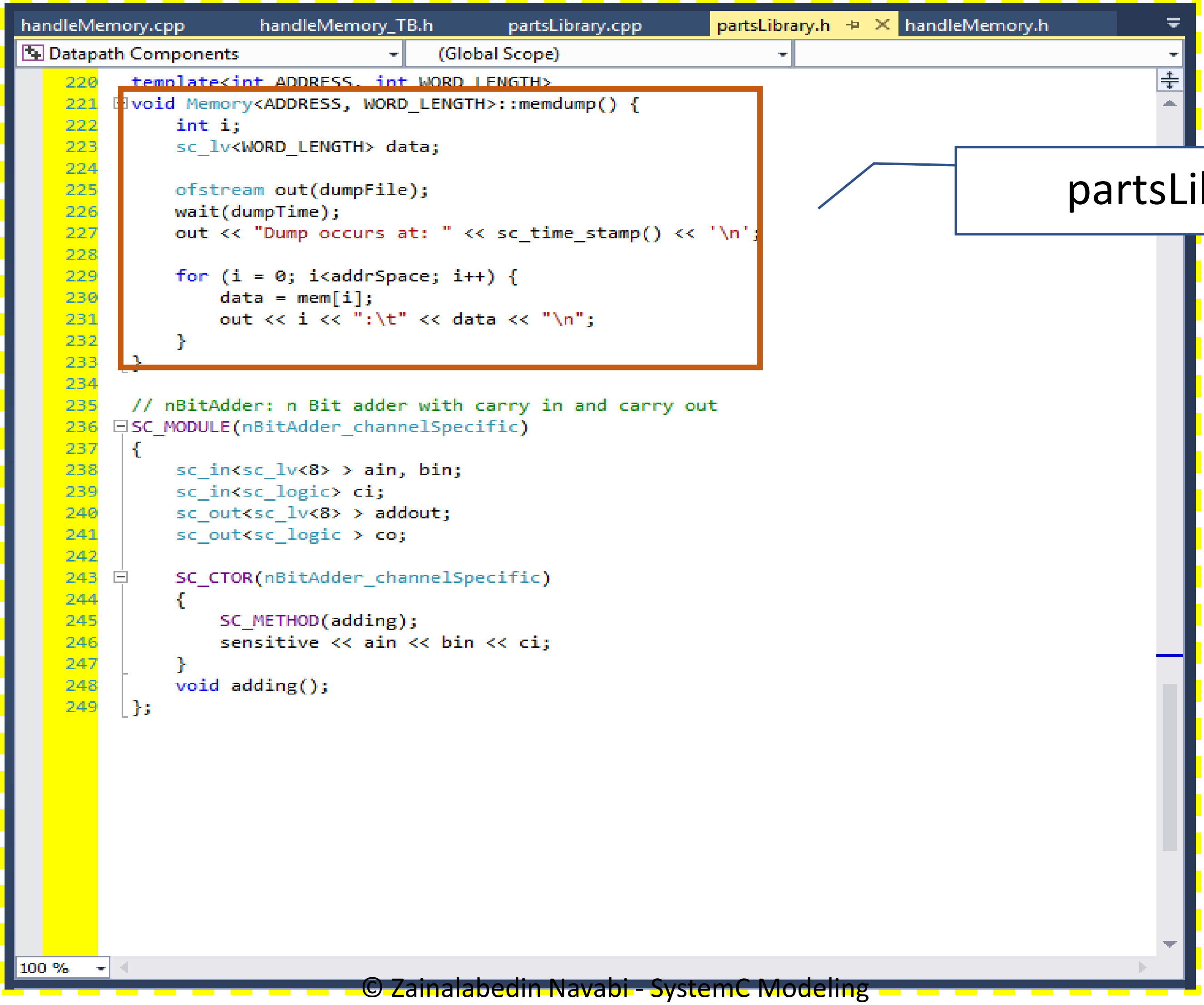
```
template<int ADDRESS, int WORD_LENGTH>
void Memory<ADDRESS, WORD_LENGTH>::meminit() {
    int i;
    for (i = 0; i<addrSpace; i++) {
        mem[i] = i;
        cout << "Init at: " << i << " writes: " << i << '\n';
    }
}

template<int ADDRESS, int WORD_LENGTH>
void Memory<ADDRESS, WORD_LENGTH>::memwrite() {
    sc_uint<ADDRESS> ad;
    if (cs->read() == '1') {
        if (rwbar->read() == '0') {
            ad = addr;
            mem[ad] = datain;
        }
    }
}

template<int ADDRESS, int WORD_LENGTH>
void Memory<ADDRESS, WORD_LENGTH>::memread() {
    sc_uint<ADDRESS> ad;
    if (cs->read() == '1') {
        if (rwbar->read() == '1') {
            ad = addr;
            dataout = mem[ad];
        }
    }
}

template<int ADDRESS, int WORD_LENGTH>
void Memory<ADDRESS, WORD_LENGTH>::memdump() {
    int i;
    sc_lv<WORD_LENGTH> data;

    ofstream out(dumpFile);
    wait(dumpTime);
    out << "Dump occurs at: " << sc_time_stamp() << '\n';

    for (i = 0; i<addrSpace; i++) {
```

Type Conversion from sc_rv to sc_uint

partsLibrary.h

Type Conversion from sc_uint to sc_rv

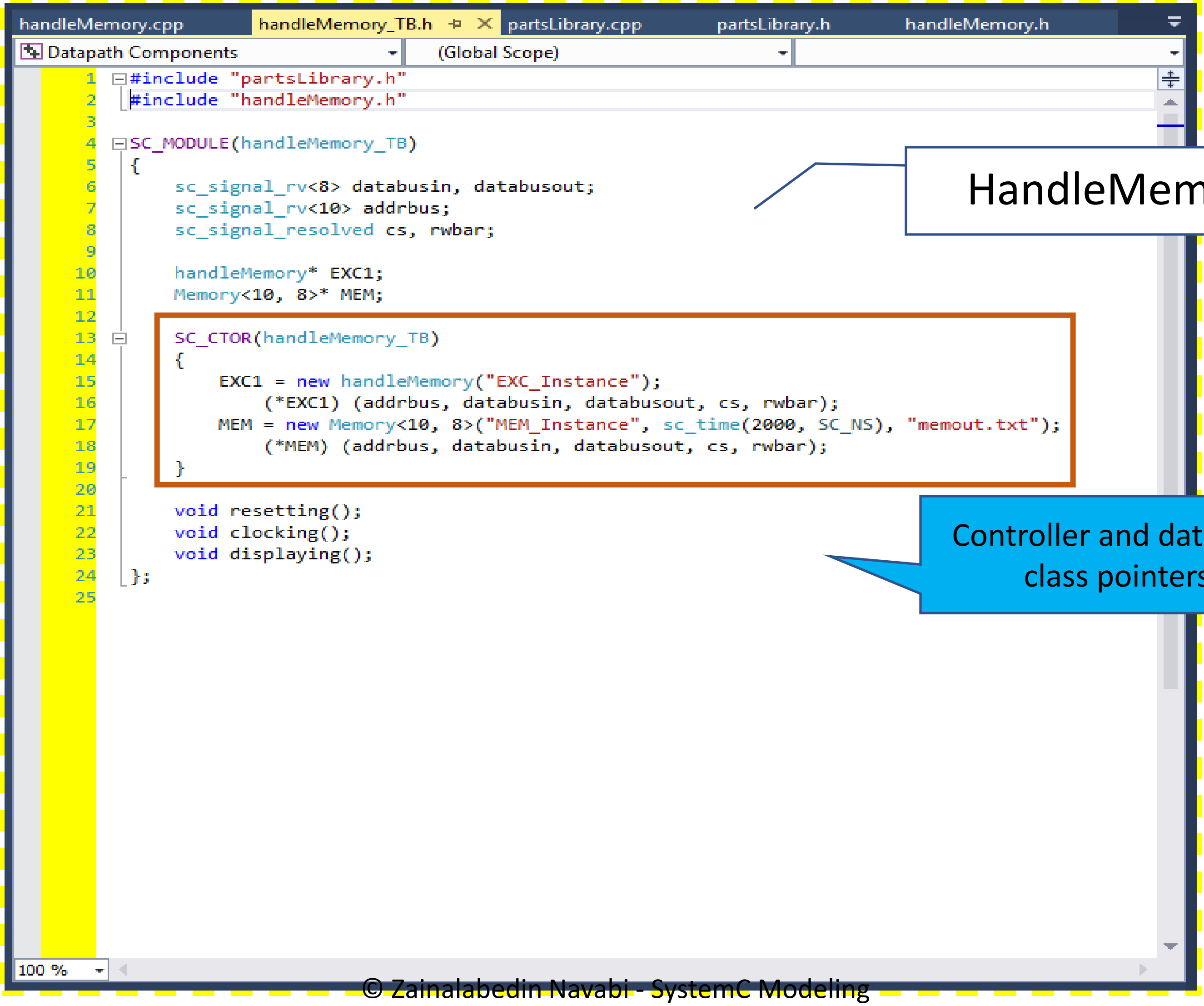
A Configurable Memory



```
template<int ADDRESS, int WORD_LENGTH>
void Memory<ADDRESS, WORD_LENGTH>::memdump() {
    int i;
    sc_lv<WORD_LENGTH> data;

    ofstream out(dumpFile);
    wait(dumpTime);
    out << "Dump occurs at: " << sc_time_stamp() << '\n';

    for (i = 0; i<addrSpace; i++) {
        data = mem[i];
        out << i << ":\t" << data << "\n";
    }
}

// nBitAdder: n Bit adder with carry in and carry out
SC_MODULE(nBitAdder_channelSpecific)
{
    sc_in<sc_lv<8> > ain, bin;
    sc_in<sc_logic> ci;
    sc_out<sc_lv<8> > addout;
    sc_out<sc_logic > co;

    SC_CTOR(nBitAdder_channelSpecific)
    {
        SC_METHOD(adding);
        sensitive << ain << bin << ci;
    }
    void adding();
};
```

partsLibrary.h

A Configurable Memory



```
#include "partsLibrary.h"
#include "handleMemory.h"

SC_MODULE(handleMemory_TB)
{
    sc_signal_rv<8> databusin, databusout;
    sc_signal_rv<10> addrbus;
    sc_signal_resolved cs, rwbar;

    handleMemory* EXC1;
    Memory<10, 8>* MEM;

    SC_CTOR(handleMemory_TB)
    {
        EXC1 = new handleMemory("EXC_Instance");
            (*EXC1) (addrbus, databusin, databusout, cs, rwbar);
        MEM = new Memory<10, 8>("MEM_Instance", sc_time(2000, SC_NS), "memout.txt");
            (*MEM) (addrbus, databusin, databusout, cs, rwbar);
    }

    void resetting();
    void clocking();
    void displaying();
};
```

HandleMemory_TB.h

Controller and datapath class pointers

Exponential Circuit

$$\mathrm{e}^{x} = \sum_{k=0}^{\infty} \frac{x^{k}}{k!} = 1 + \frac{x^{1}}{1!} + \frac{x^{2}}{2!} + \frac{x^{3}}{3!} + \cdots$$

```
term = 1;
exp = 1;
for( i = 1; i < n; i++ ) {
    term = term × x × ( 1 / i );
    exp = exp + term;
}
```

Exponential Circuit

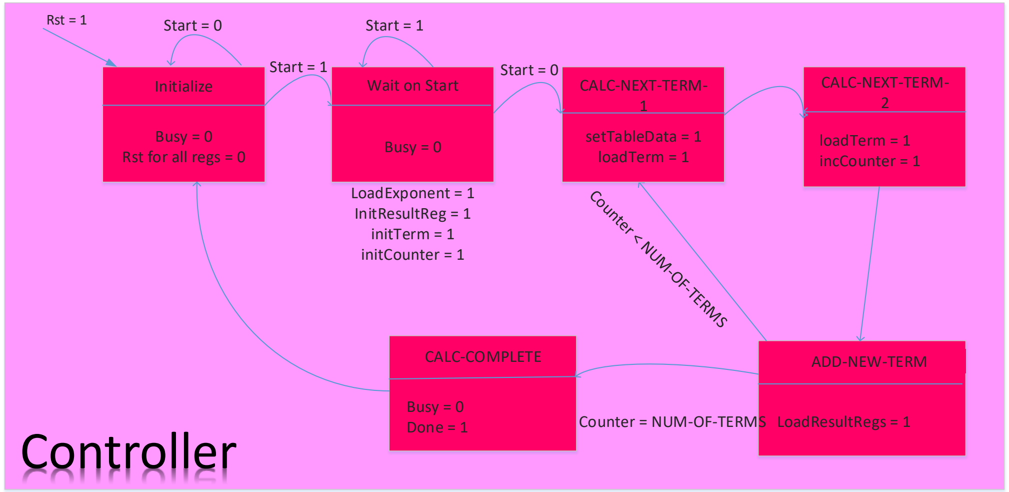
$$e^{x} = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

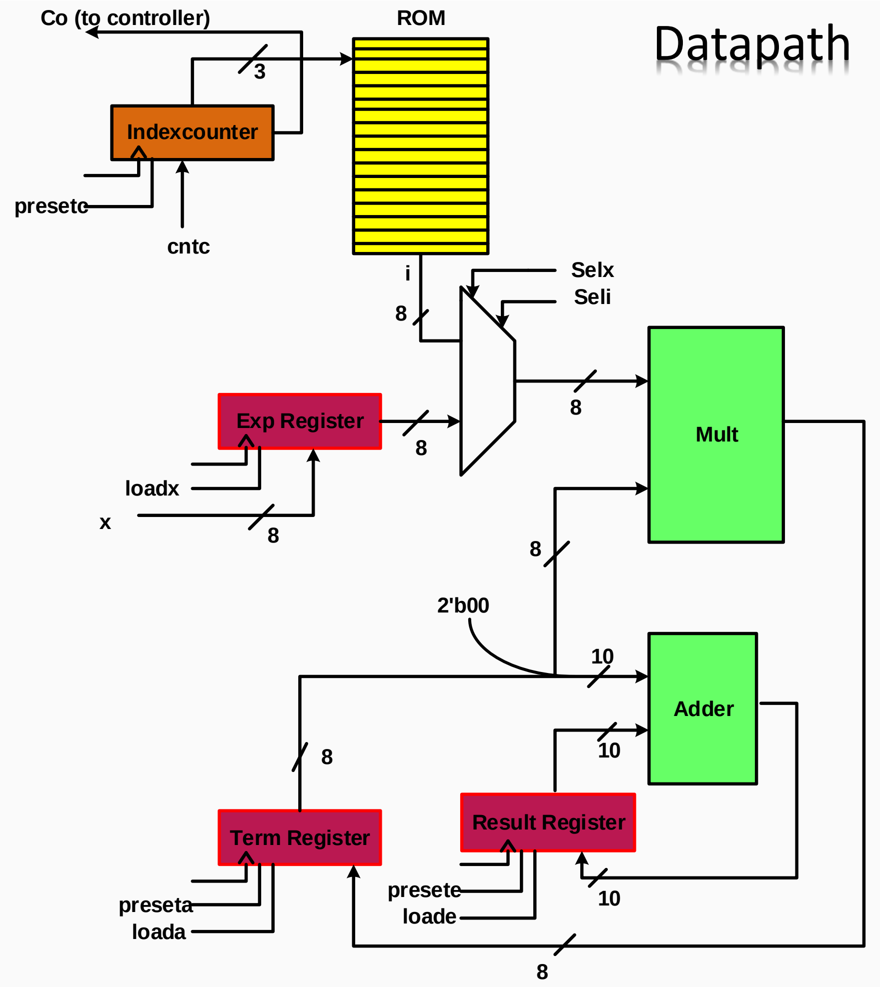
Using only one Multiplier

```
term = 1;
exp = 1;
for( i = 1; i < n; i++ ) {
   term = term × ( 1 / i );
   term = term × x;
   exp = exp + term;
}
```

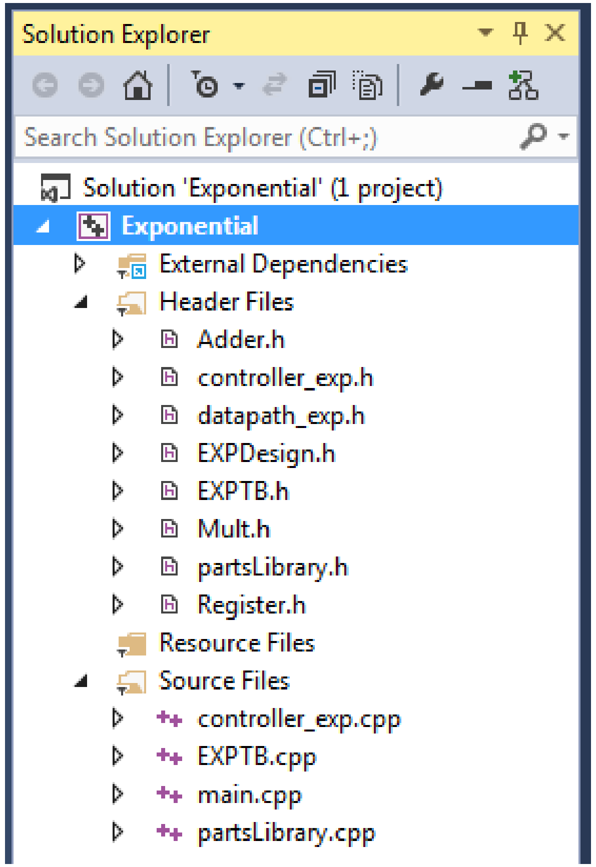
Exponential Circuit

- Datapath & Controller





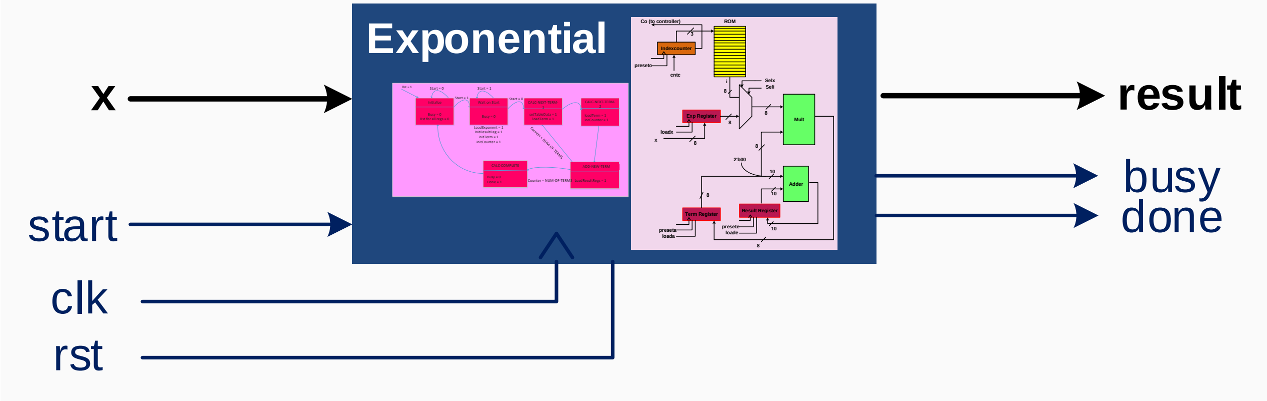
Exponential Circuit



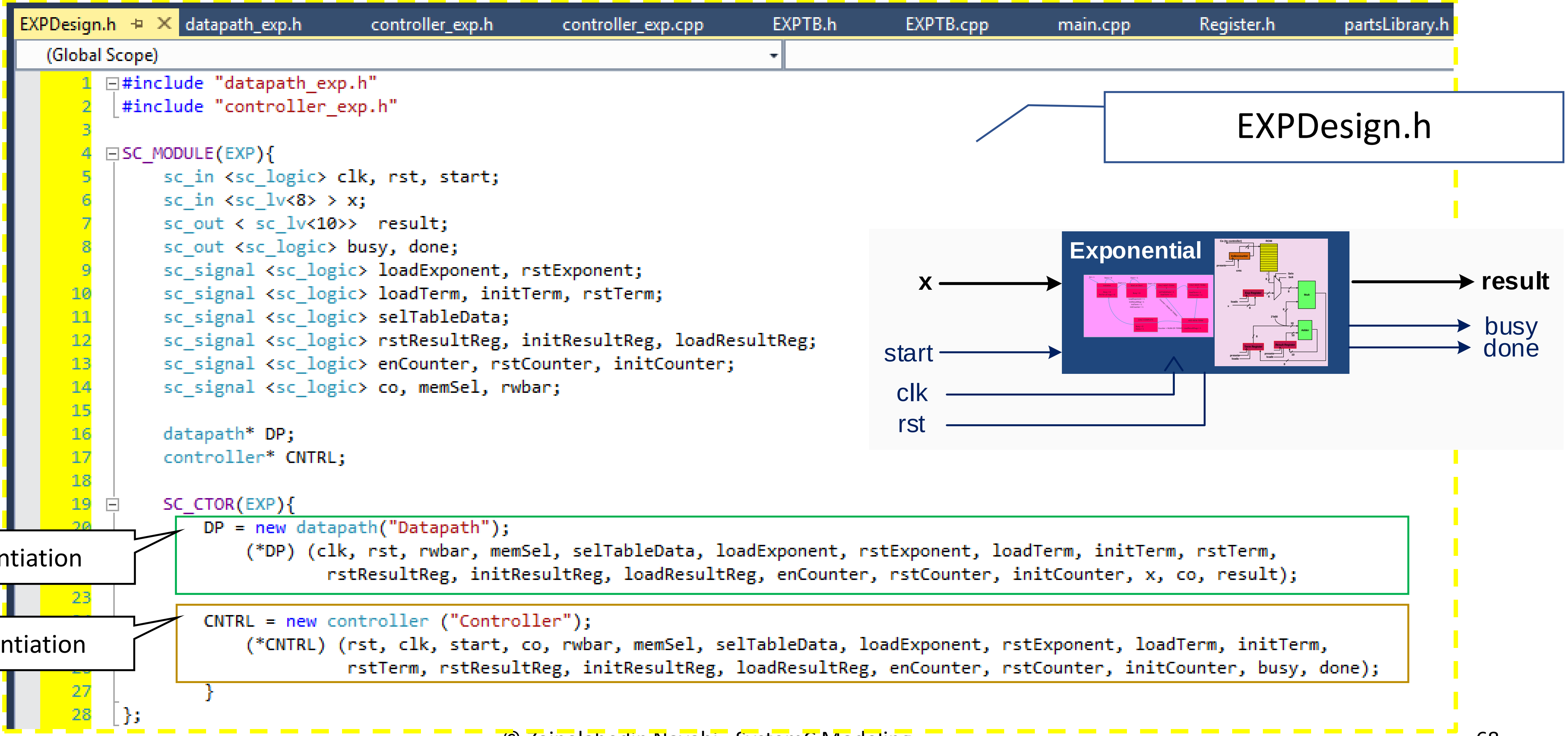
Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'Exponential' (1 project)
  - Exponential
    - External Dependencies
    - Header Files
      - Adder.h
      - controller_exp.h
      - datapath_exp.h
      - EXPDesign.h
      - EXPTB.h
      - Mult.h
      - partsLibrary.h
      - Register.h
    - Resource Files
    - Source Files
      - controller_exp.cpp
      - EXPTB.cpp
      - main.cpp
      - partsLibrary.cpp



Exponential Circuit



```
#include "datapath_exp.h"
#include "controller_exp.h"

SC_MODULE(EXP){
    sc_in <sc_logic> clk, rst, start;
    sc_in <sc_lv<8> > x;
    sc_out < sc_lv<10>>  result;
    sc_out <sc_logic> busy, done;
    sc_signal <sc_logic> loadExponent, rstExponent;
    sc_signal <sc_logic> loadTerm, initTerm, rstTerm;
    sc_signal <sc_logic> selTableData;
    sc_signal <sc_logic> rstResultReg, initResultReg, loadResultReg;
    sc_signal <sc_logic> enCounter, rstCounter, initCounter;
    sc_signal <sc_logic> co, memSel, rwbar;

    datapath* DP;
    controller* CNTRL;

    SC_CTOR(EXP){
        DP = new datapath("Datapath");
            (*DP) (clk, rst, rwbar, memSel, selTableData, loadExponent, rstExponent, loadTerm, initTerm, rstTerm,
                    rstResultReg, initResultReg, loadResultReg, enCounter, rstCounter, initCounter, x, co, result);

        CNTRL = new controller ("Controller");
            (*CNTRL) (rst, clk, start, co, rwbar, memSel, selTableData, loadExponent, rstExponent, loadTerm, initTerm,
                      rstTerm, rstResultReg, initResultReg, loadResultReg, enCounter, rstCounter, initCounter, busy, done);
        }
};
```

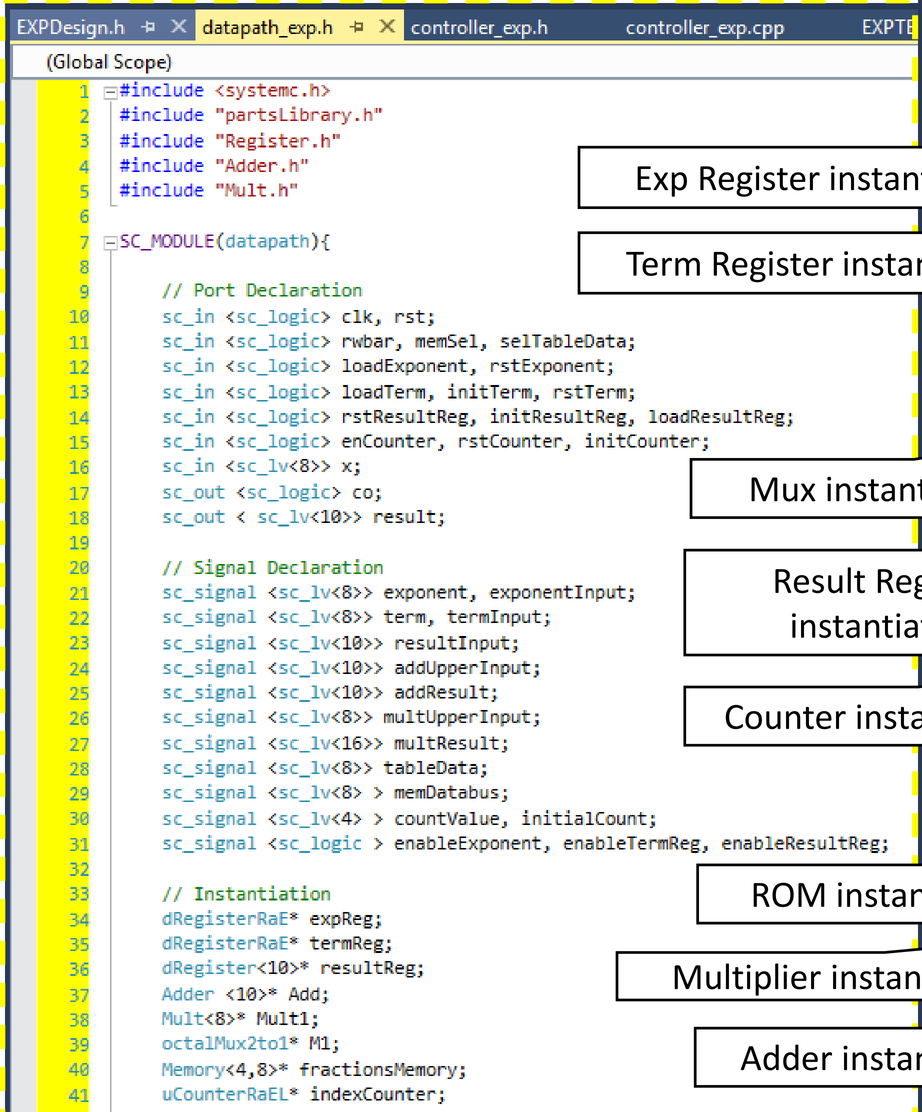
EXPDesign.h

Datapath instantiation

Controller instantiation

SystemC RTL Design - Example

Exponential Circuit



```
#include <systemc.h>
#include "partsLibrary.h"
#include "Register.h"
#include "Adder.h"
#include "Mult.h"

SC_MODULE(datapath){

    // Port Declaration
    sc_in <sc_logic> clk, rst;
    sc_in <sc_logic> rwbar, memSel, selTableData;
    sc_in <sc_logic> loadExponent, rstExponent;
    sc_in <sc_logic> loadTerm, initTerm, rstTerm;
    sc_in <sc_logic> rstResultReg, initResultReg, loadResultReg;
    sc_in <sc_logic> enCounter, rstCounter, initCounter;
    sc_in <sc_lv<8>> x;
    sc_out <sc_logic> co;
    sc_out < sc_lv<10>> result;

    // Signal Declaration
    sc_signal <sc_lv<8>> exponent, exponentInput;
    sc_signal <sc_lv<8>> term, termInput;
    sc_signal <sc_lv<10>> resultInput;
    sc_signal <sc_lv<10>> addUpperInput;
    sc_signal <sc_lv<10>> addResult;
    sc_signal <sc_lv<8>> multUpperInput;
    sc_signal <sc_lv<16>> multResult;
    sc_signal <sc_lv<8>> tableData;
    sc_signal <sc_lv<8> > memDatabus;
    sc_signal <sc_lv<4> > countValue, initialCount;
    sc_signal <sc_logic > enableExponent, enableTermReg, enableResultReg;

    // Instantiation
    dRegisterRaE* expReg;
    dRegisterRaE* termReg;
    dRegister<10>* resultReg;
    Adder <10>* Add;
    Mult<8>* Mult1;
    octalMux2to1* M1;
    Memory<4,8>* fractionsMemory;
    uCounterRaEL* indexCounter;
```

Exp Register instantiation

Term Register instantiation

Mux instantiation

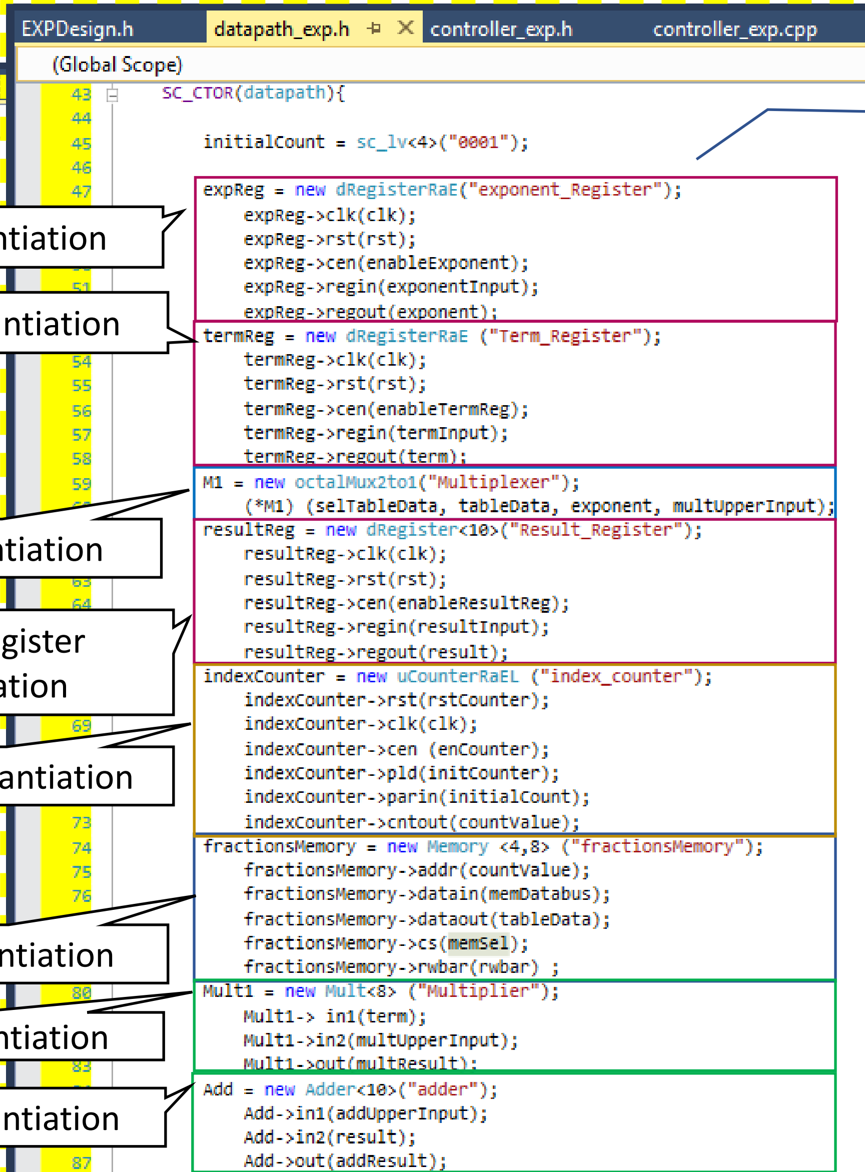
Result Register instantiation

Counter instantiation

ROM instantiation

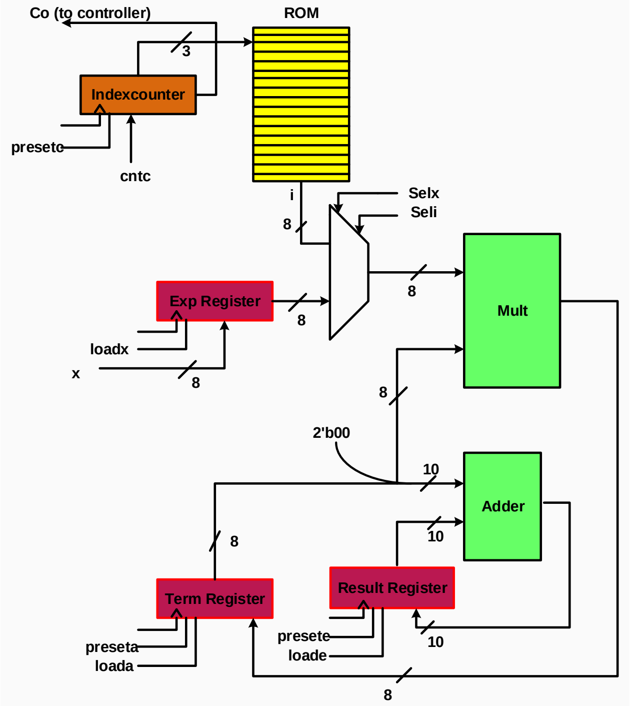
Multiplier instantiation

Adder instantiation



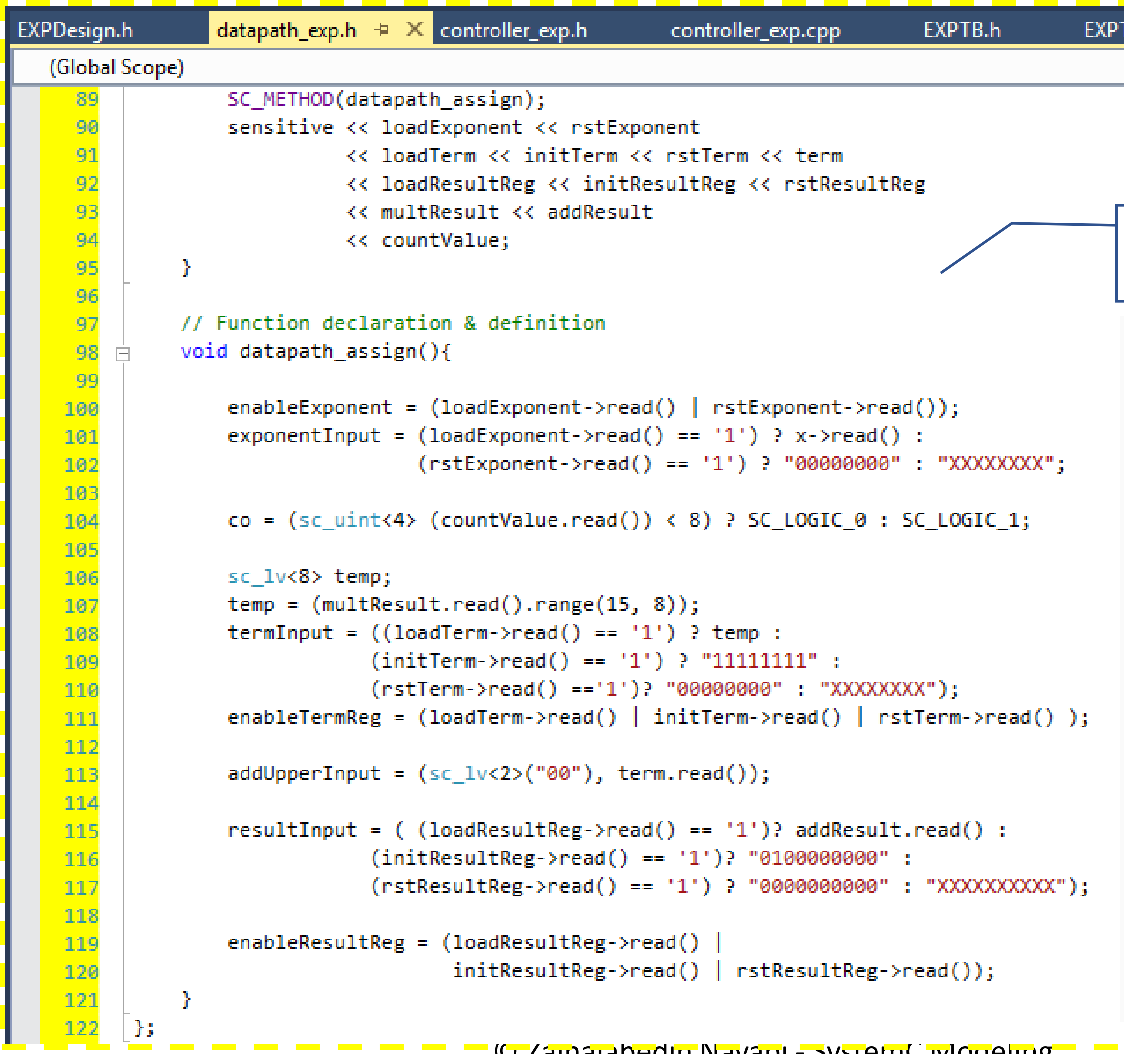
```
    SC_CTOR(datapath){

        initialCount = sc_lv<4>("0001");

        expReg = new dRegisterRaE("exponent_Register");
            expReg->clk(clk);
            expReg->rst(rst);
            expReg->cen(enableExponent);
            expReg->regin(exponentInput);
            expReg->regout(exponent);
        termReg = new dRegisterRaE ("Term_Register");
            termReg->clk(clk);
            termReg->rst(rst);
            termReg->cen(enableTermReg);
            termReg->regin(termInput);
            termReg->regout(term);
        M1 = new octalMux2to1("Multiplexer");
            (*M1) (selTableData, tableData, exponent, multUpperInput);
        resultReg = new dRegister<10>("Result_Register");
            resultReg->clk(clk);
            resultReg->rst(rst);
            resultReg->cen(enableResultReg);
            resultReg->regin(resultInput);
            resultReg->regout(result);
        indexCounter = new uCounterRaEL ("index_counter");
            indexCounter->rst(rstCounter);
            indexCounter->clk(clk);
            indexCounter->cen (enCounter);
            indexCounter->pld(initCounter);
            indexCounter->parin(initialCount);
            indexCounter->cntout(countValue);
        fractionsMemory = new Memory <4,8> ("fractionsMemory");
            fractionsMemory->addr(countValue);
            fractionsMemory->datain(memDatabus);
            fractionsMemory->dataout(tableData);
            fractionsMemory->cs(memSel);
            fractionsMemory->rwbar(rwbar) ;
        Mult1 = new Mult<8> ("Multiplier");
            Mult1-> in1(term);
            Mult1->in2(multUpperInput);
            Mult1->out(multResult);
        Add = new Adder<10>("adder");
            Add->in1(addUpperInput);
            Add->in2(result);
            Add->out(addResult);
```

datapath_exp.h



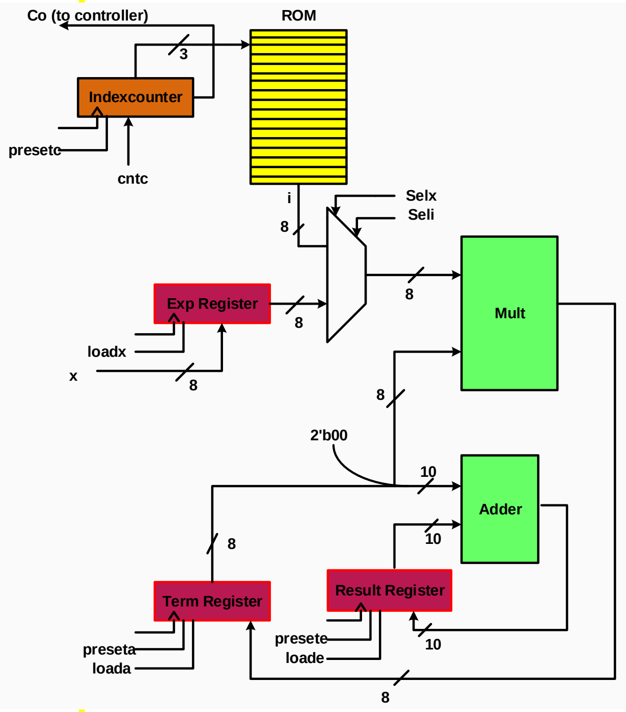
SystemC RTL Design - Example

Exponential Circuit

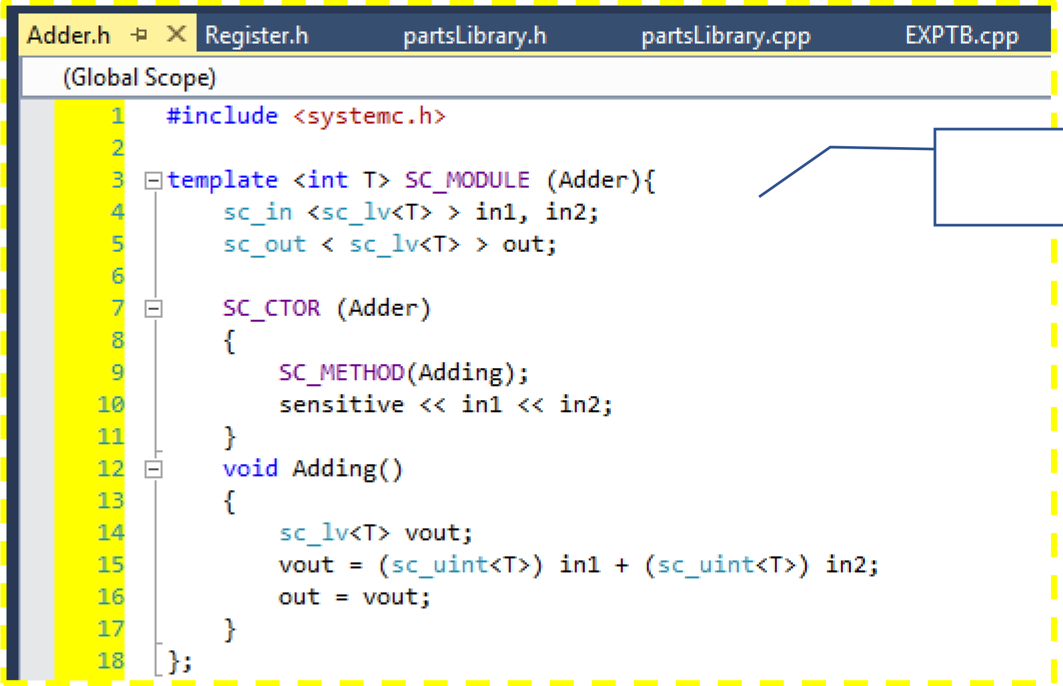


```
        SC_METHOD(datapath_assign);
        sensitive << loadExponent << rstExponent
                  << loadTerm << initTerm << rstTerm << term
                  << loadResultReg << initResultReg << rstResultReg
                  << multResult << addResult
                  << countValue;
    }

    // Function declaration & definition
    void datapath_assign(){

        enableExponent = (loadExponent->read() | rstExponent->read());
        exponentInput = (loadExponent->read() == '1') ? x->read() :
                        (rstExponent->read() == '1') ? "00000000" : "XXXXXXXX";

        co = (sc_uint<4> (countValue.read()) < 8) ? SC_LOGIC_0 : SC_LOGIC_1;

        sc_lv<8> temp;
        temp = (multResult.read().range(15, 8));
        termInput = ((loadTerm->read() == '1') ? temp :
                    (initTerm->read() == '1') ? "11111111" :
                    (rstTerm->read() =='1')? "00000000" : "XXXXXXXX");
        enableTermReg = (loadTerm->read() | initTerm->read() | rstTerm->read() );

        addUpperInput = (sc_lv<2>("00"), term.read());

        resultInput = ( (loadResultReg->read() == '1')? addResult.read() :
                    (initResultReg->read() == '1')? "0100000000" :
                    (rstResultReg->read() == '1') ? "0000000000" : "XXXXXXXXXX");

        enableResultReg = (loadResultReg->read() |
                           initResultReg->read() | rstResultReg->read());
    }
};
```

datapath_exp.h



Exponential Circuit

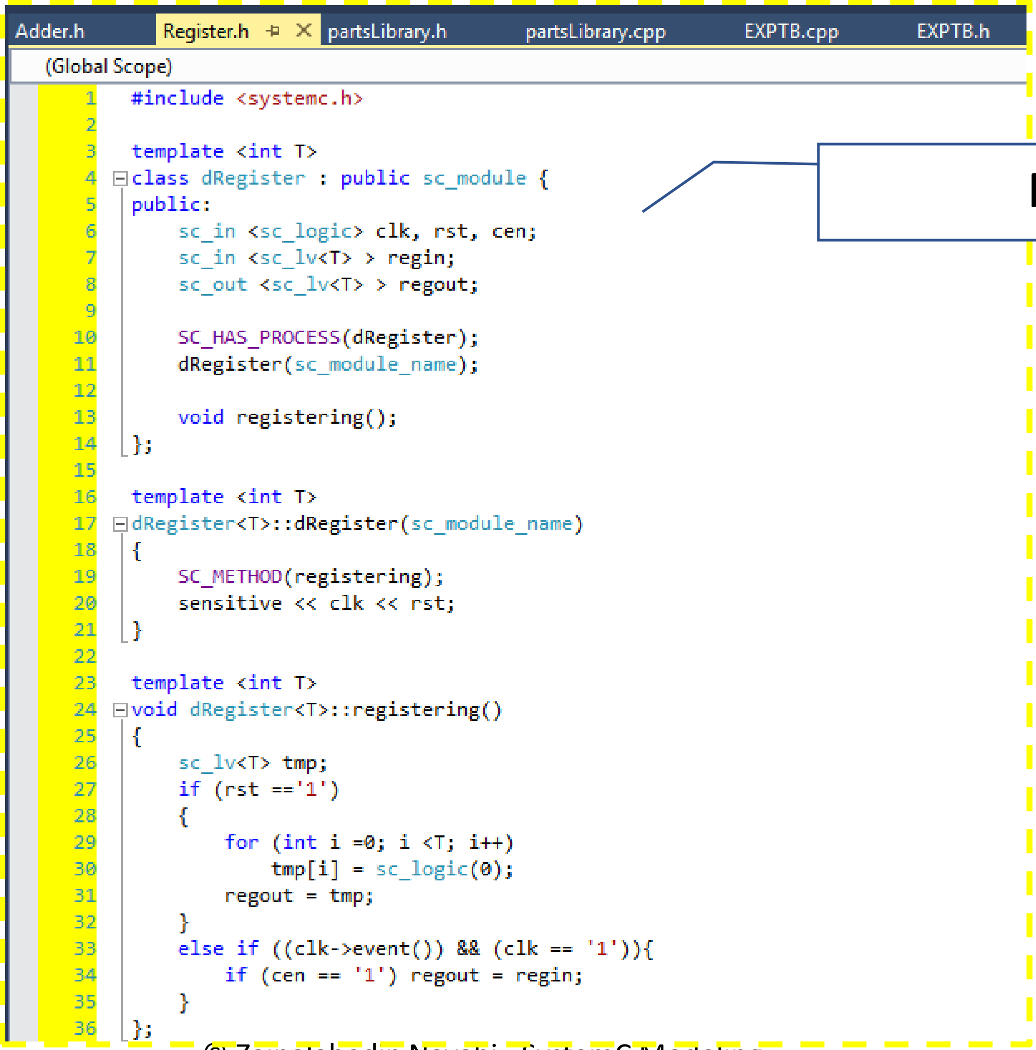


```
#include <systemc.h>

template <int T> SC_MODULE (Adder){
    sc_in <sc_lv<T> > in1, in2;
    sc_out < sc_lv<T> > out;

    SC_CTOR (Adder)
    {
        SC_METHOD(Adding);
        sensitive << in1 << in2;
    }
    void Adding()
    {
        sc_lv<T> vout;
        vout = (sc_uint<T>) in1 + (sc_uint<T>) in2;
        out = vout;
    }
};
```

Adder.h

SystemC RTL Design - Example

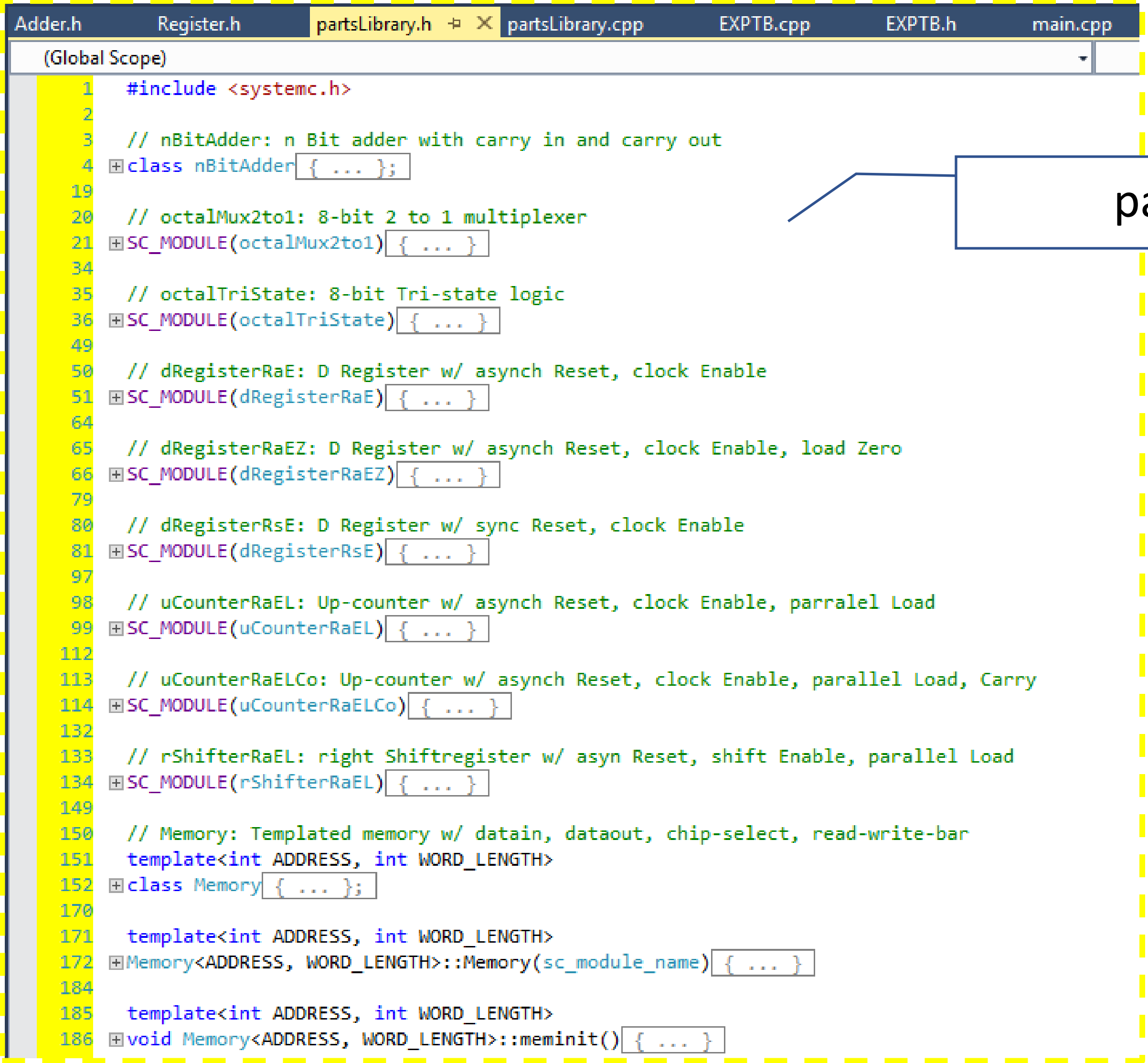
Exponential Circuit



```
#include <systemc.h>

template <int T>
class dRegister : public sc_module {
public:
    sc_in <sc_logic> clk, rst, cen;
    sc_in <sc_lv<T> > regin;
    sc_out <sc_lv<T> > regout;

    SC_HAS_PROCESS(dRegister);
    dRegister(sc_module_name);

    void registering();
};

template <int T>
dRegister<T>::dRegister(sc_module_name)
{
    SC_METHOD(registering);
    sensitive << clk << rst;
}

template <int T>
void dRegister<T>::registering()
{
    sc_lv<T> tmp;
    if (rst =='1')
    {
        for (int i =0; i <T; i++)
            tmp[i] = sc_logic(0);
        regout = tmp;
    }
    else if ((clk->event()) && (clk == '1')){
        if (cen == '1') regout = regin;
    }
};
```

Register.h

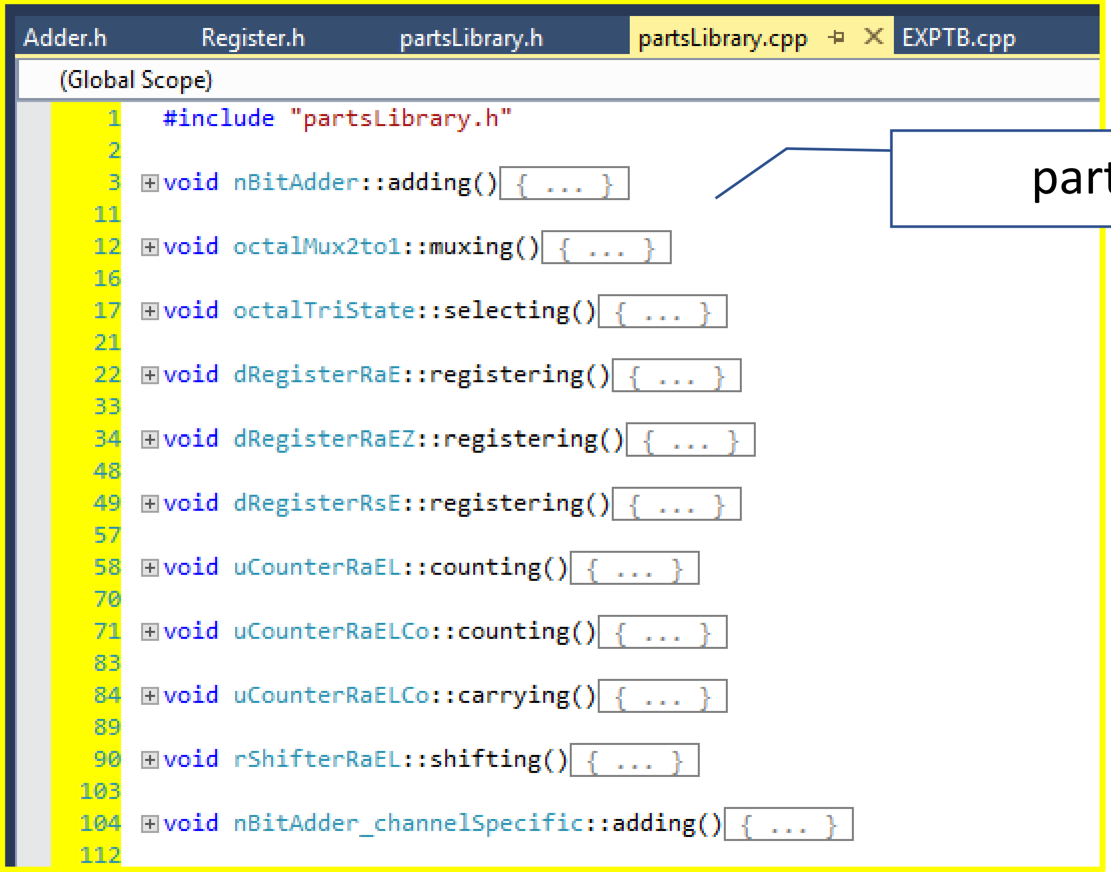
Exponential Circuit



```
#include <systemc.h>

// nBitAdder: n Bit adder with carry in and carry out
class nBitAdder{ ... };

// octalMux2to1: 8-bit 2 to 1 multiplexer
SC_MODULE(octalMux2to1){ ... }

// octalTriState: 8-bit Tri-state logic
SC_MODULE(octalTriState){ ... }

// dRegisterRaE: D Register w/ asynch Reset, clock Enable
SC_MODULE(dRegisterRaE){ ... }

// dRegisterRaEZ: D Register w/ asynch Reset, clock Enable, load Zero
SC_MODULE(dRegisterRaEZ){ ... }

// dRegisterRsE: D Register w/ sync Reset, clock Enable
SC_MODULE(dRegisterRsE){ ... }

// uCounterRaEL: Up-counter w/ asynch Reset, clock Enable, parralel Load
SC_MODULE(uCounterRaEL){ ... }

// uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
SC_MODULE(uCounterRaELCo){ ... }

// rShifterRaEL: right Shiftregister w/ asyn Reset, shift Enable, parallel Load
SC_MODULE(rShifterRaEL){ ... }

// Memory: Templated memory w/ datain, dataout, chip-select, read-write-bar
template<int ADDRESS, int WORD_LENGTH>
class Memory{ ... };

template<int ADDRESS, int WORD_LENGTH>
Memory<ADDRESS, WORD_LENGTH>::Memory(sc_module_name){ ... }

template<int ADDRESS, int WORD_LENGTH>
void Memory<ADDRESS, WORD_LENGTH>::meminit(){ ... }
```

partsLibrary.h

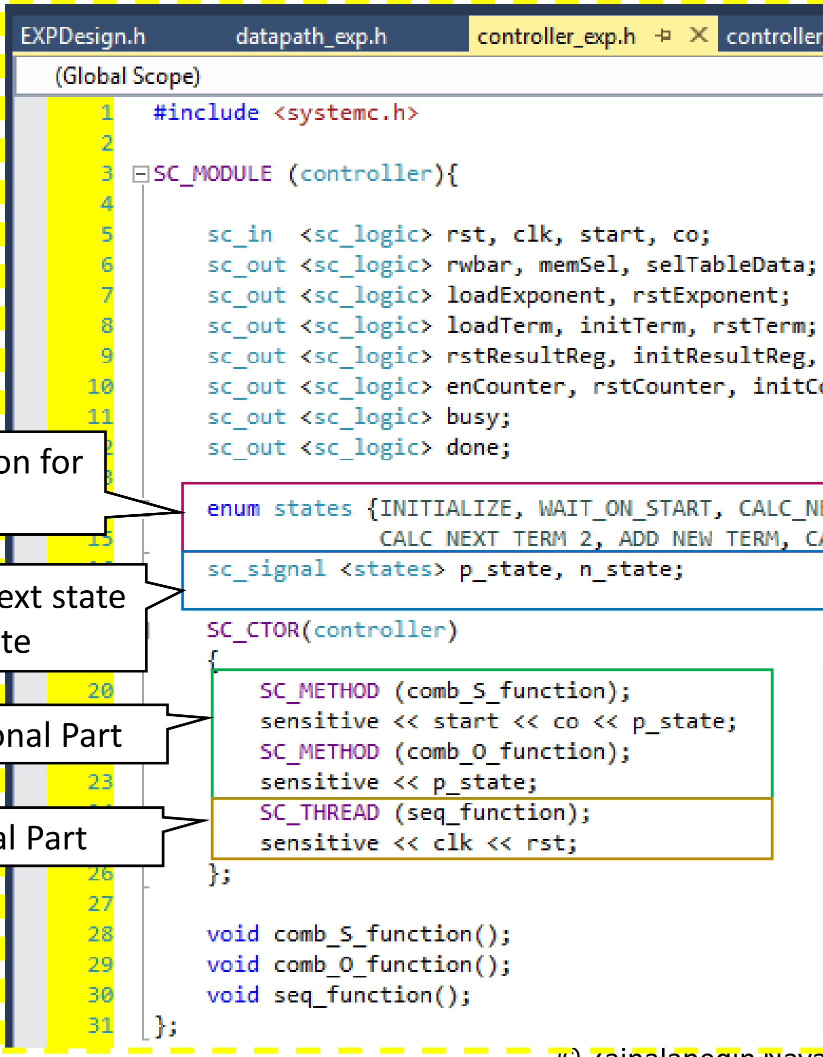
Exponential Circuit



```
#include "partsLibrary.h"

void nBitAdder::adding() { ... }

void octalMux2to1::muxing() { ... }

void octalTriState::selecting() { ... }

void dRegisterRaE::registering() { ... }

void dRegisterRaEZ::registering() { ... }

void dRegisterRsE::registering() { ... }

void uCounterRaEL::counting() { ... }

void uCounterRaELCo::counting() { ... }

void uCounterRaELCo::carrying() { ... }

void rShifterRaEL::shifting() { ... }

void nBitAdder_channelSpecific::adding() { ... }
```

partsLibrary.cpp

Exponential Circuit



```
#include <systemc.h>

SC_MODULE (controller){

    sc_in  <sc_logic> rst, clk, start, co;
    sc_out <sc_logic> rwbar, memSel, selTableData;
    sc_out <sc_logic> loadExponent, rstExponent;
    sc_out <sc_logic> loadTerm, initTerm, rstTerm;
    sc_out <sc_logic> rstResultReg, initResultReg, loadResultReg;
    sc_out <sc_logic> enCounter, rstCounter, initCounter;
    sc_out <sc_logic> busy;
    sc_out <sc_logic> done;

    enum states {INITIALIZE, WAIT_ON_START, CALC_NEXT_TERM_1,
                 CALC_NEXT_TERM_2, ADD_NEW_TERM, CALC_COMPLETE};
    sc_signal <states> p_state, n_state;

    SC_CTOR(controller)
    {
        SC_METHOD (comb_S_function);
        sensitive << start << co << p_state;
        SC_METHOD (comb_O_function);
        sensitive << p_state;
        SC_THREAD (seq_function);
        sensitive << clk << rst;
    };

    void comb_S_function();
    void comb_O_function();
    void seq_function();
};
```

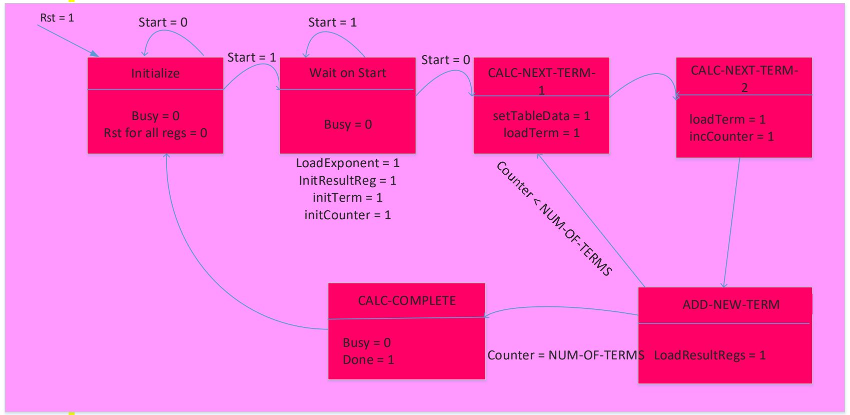
controller_exp.h

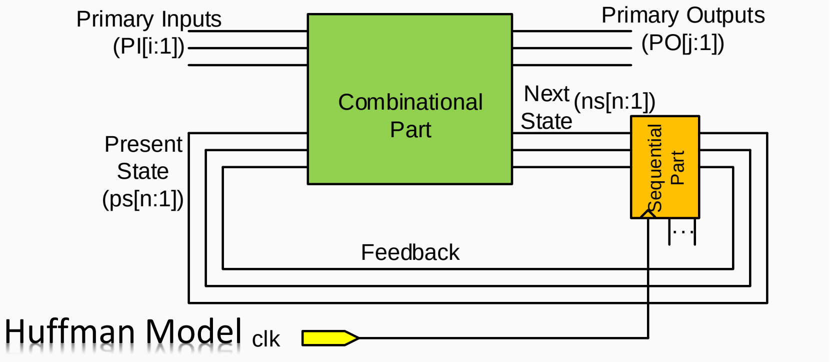
enum definition for states

Signal definition for next state and present state

Combinational Part

Sequential Part

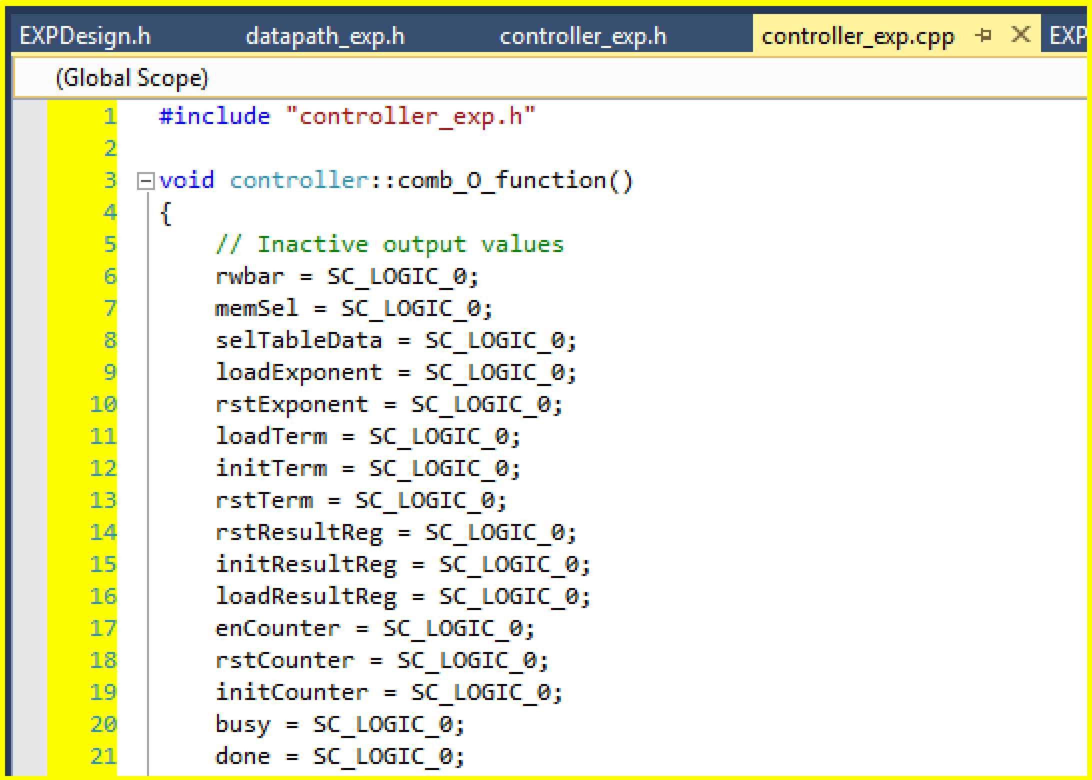




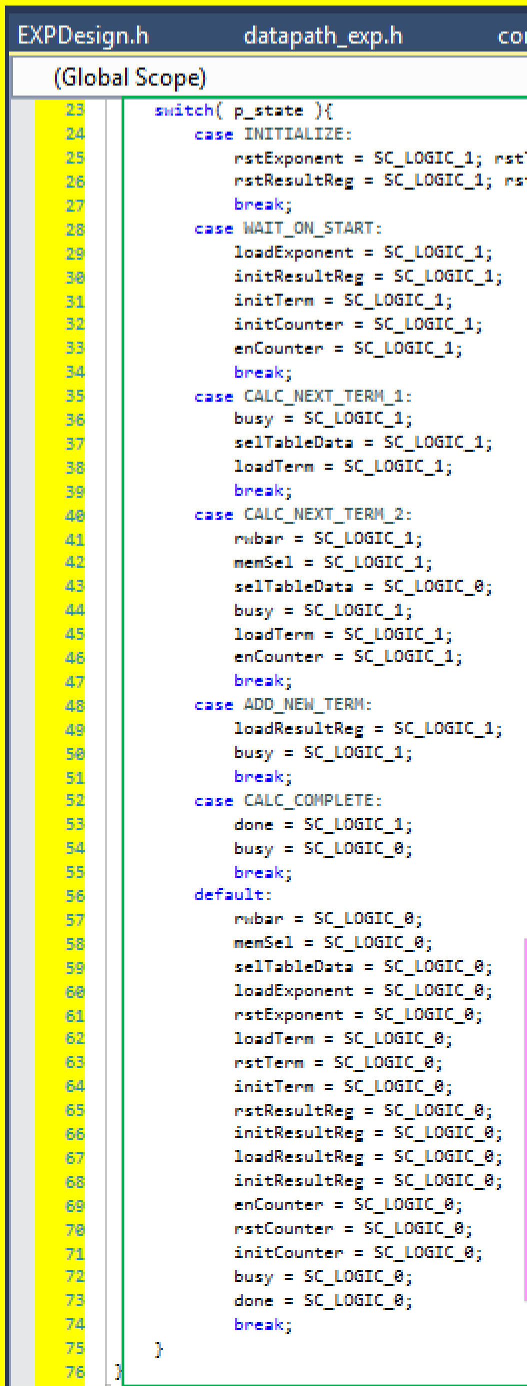
Huffman Model

SystemC RTL Design - Example

Exponential Circuit

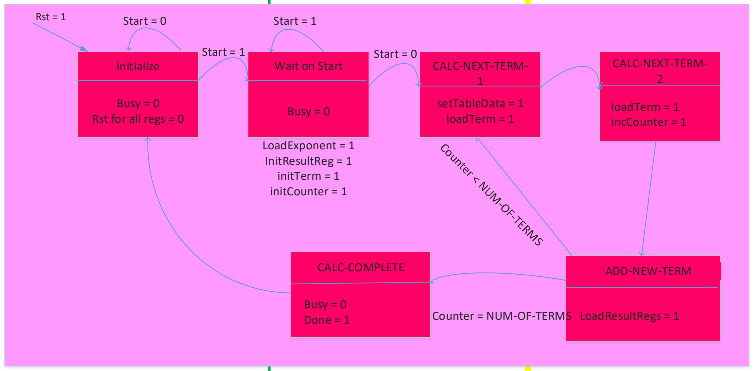


```cpp
#include "controller_exp.h"

void controller::comb_O_function()
{
    // Inactive output values
    rwbar = SC_LOGIC_0;
    memSel = SC_LOGIC_0;
    selTableData = SC_LOGIC_0;
    loadExponent = SC_LOGIC_0;
    rstExponent = SC_LOGIC_0;
    loadTerm = SC_LOGIC_0;
    initTerm = SC_LOGIC_0;
    rstTerm = SC_LOGIC_0;
    rstResultReg = SC_LOGIC_0;
    initResultReg = SC_LOGIC_0;
    loadResultReg = SC_LOGIC_0;
    enCounter = SC_LOGIC_0;
    rstCounter = SC_LOGIC_0;
    initCounter = SC_LOGIC_0;
    busy = SC_LOGIC_0;
    done = SC_LOGIC_0;
```

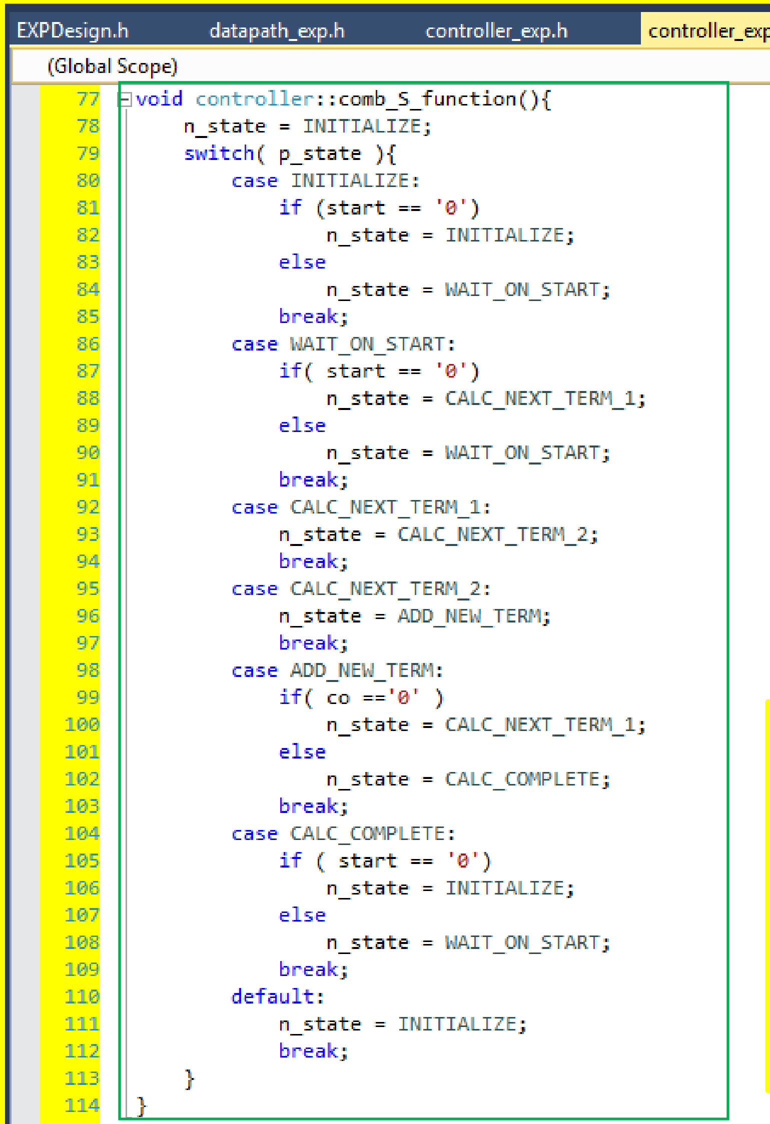


```cpp
    switch( p_state ){
        case INITIALIZE:
            rstExponent = SC_LOGIC_1; rstTerm  = SC_LOGIC_1;
            rstResultReg = SC_LOGIC_1; rstCounter = SC_LOGIC_1;
            break;
        case WAIT_ON_START:
            loadExponent = SC_LOGIC_1;
            initResultReg = SC_LOGIC_1;
            initTerm = SC_LOGIC_1;
            initCounter = SC_LOGIC_1;
            enCounter = SC_LOGIC_1;
            break;
        case CALC_NEXT_TERM_1:
            busy = SC_LOGIC_1;
            selTableData = SC_LOGIC_1;
            loadTerm = SC_LOGIC_1;
            break;
        case CALC_NEXT_TERM_2:
            rwbar = SC_LOGIC_1;
            memSel = SC_LOGIC_1;
            selTableData = SC_LOGIC_0;
            busy = SC_LOGIC_1;
            loadTerm = SC_LOGIC_1;
            enCounter = SC_LOGIC_1;
            break;
        case ADD_NEW_TERM:
            loadResultReg = SC_LOGIC_1;
            busy = SC_LOGIC_1;
            break;
        case CALC_COMPLETE:
            done = SC_LOGIC_1;
            busy = SC_LOGIC_0;
            break;
        default:
            rwbar = SC_LOGIC_0;
            memSel = SC_LOGIC_0;
            selTableData = SC_LOGIC_0;
            loadExponent = SC_LOGIC_0;
            rstExponent = SC_LOGIC_0;
            loadTerm = SC_LOGIC_0;
            rstTerm = SC_LOGIC_0;
            initTerm = SC_LOGIC_0;
            rstResultReg = SC_LOGIC_0;
            initResultReg = SC_LOGIC_0;
            loadResultReg = SC_LOGIC_0;
            initResultReg = SC_LOGIC_0;
            enCounter = SC_LOGIC_0;
            rstCounter = SC_LOGIC_0;
            initCounter = SC_LOGIC_0;
            busy = SC_LOGIC_0;
            done = SC_LOGIC_0;
            break;
    }
}
```

controller_exp.cpp

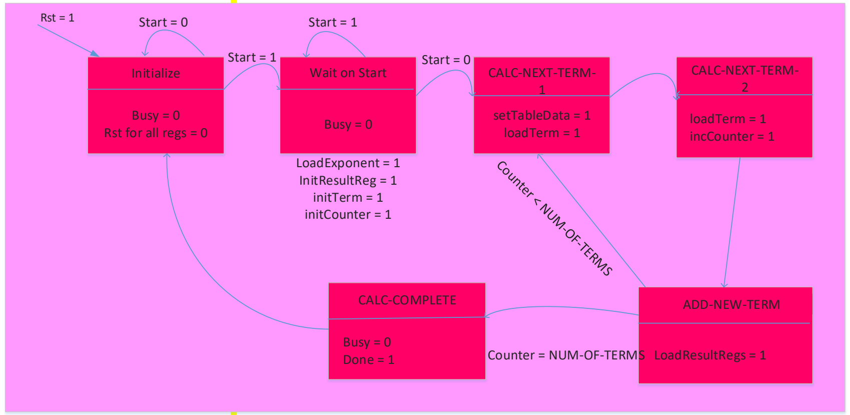


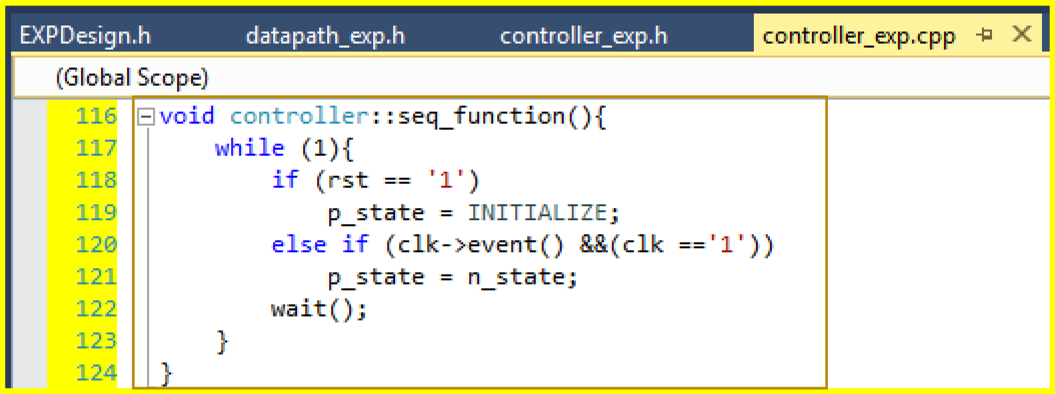
Exponential Circuit



```
void controller::comb_S_function(){
    n_state = INITIALIZE;
    switch( p_state ){
        case INITIALIZE:
            if (start == '0')
                n_state = INITIALIZE;
            else
                n_state = WAIT_ON_START;
            break;
        case WAIT_ON_START:
            if( start == '0')
                n_state = CALC_NEXT_TERM_1;
            else
                n_state = WAIT_ON_START;
            break;
        case CALC_NEXT_TERM_1:
            n_state = CALC_NEXT_TERM_2;
            break;
        case CALC_NEXT_TERM_2:
            n_state = ADD_NEW_TERM;
            break;
        case ADD_NEW_TERM:
            if( co =='0' )
                n_state = CALC_NEXT_TERM_1;
            else
                n_state = CALC_COMPLETE;
            break;
        case CALC_COMPLETE:
            if ( start == '0')
                n_state = INITIALIZE;
            else
                n_state = WAIT_ON_START;
            break;
        default:
            n_state = INITIALIZE;
            break;
    }
}
```

controller_exp.cpp

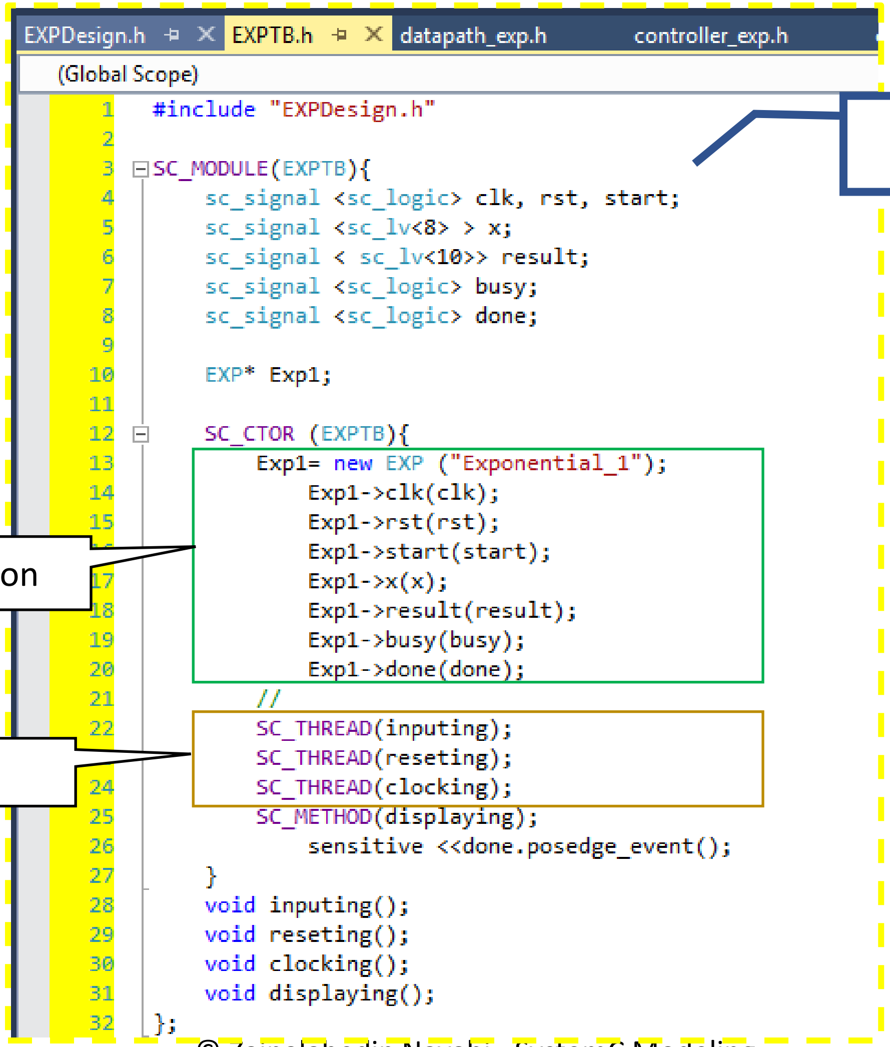




```
void controller::seq_function(){
    while (1){
        if (rst == '1')
            p_state = INITIALIZE;
        else if (clk->event() &&(clk =='1'))
            p_state = n_state;
        wait();
    }
}
```

SystemC RTL Design - Example

Exponential Circuit



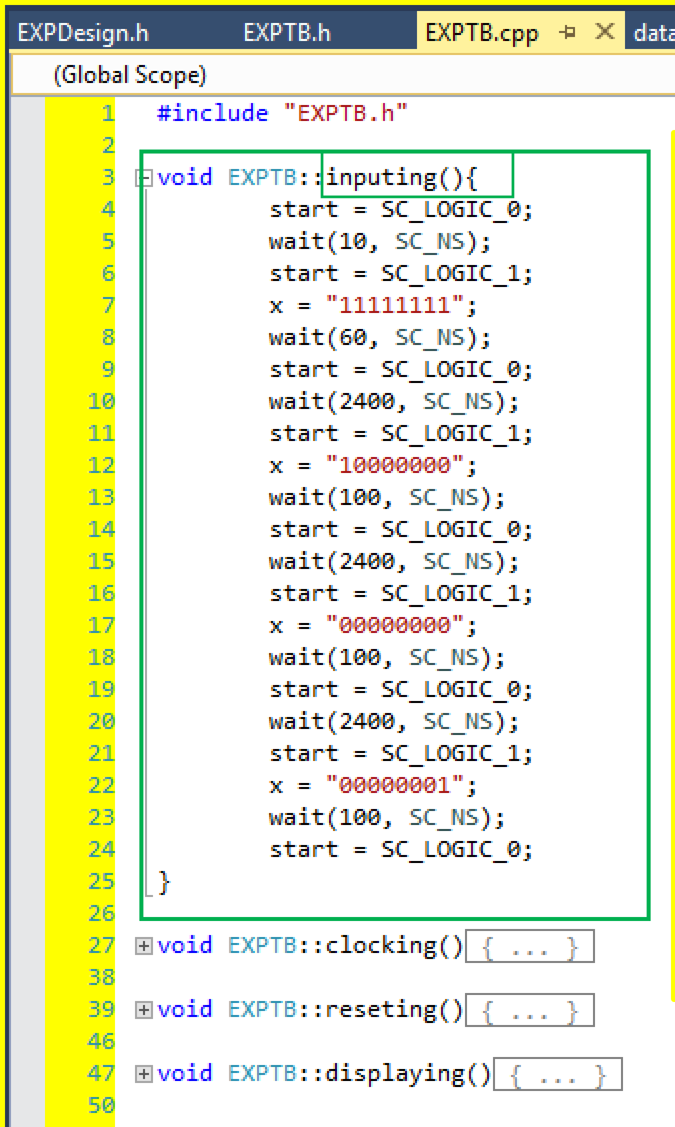
```
#include "EXPDesign.h"

SC_MODULE(EXPTB){
    sc_signal <sc_logic> clk, rst, start;
    sc_signal <sc_lv<8> > x;
    sc_signal < sc_lv<10>> result;
    sc_signal <sc_logic> busy;
    sc_signal <sc_logic> done;

    EXP* Exp1;

    SC_CTOR (EXPTB){
        Exp1= new EXP ("Exponential_1");
            Exp1->clk(clk);
            Exp1->rst(rst);
            Exp1->start(start);
            Exp1->x(x);
            Exp1->result(result);
            Exp1->busy(busy);
            Exp1->done(done);
        //
        SC_THREAD(inputing);
        SC_THREAD(reseting);
        SC_THREAD(clocking);
        SC_METHOD(displaying);
            sensitive <<done.posedge_event();
    }
    void inputing();
    void reseting();
    void clocking();
    void displaying();
};
```

EXPTB.h

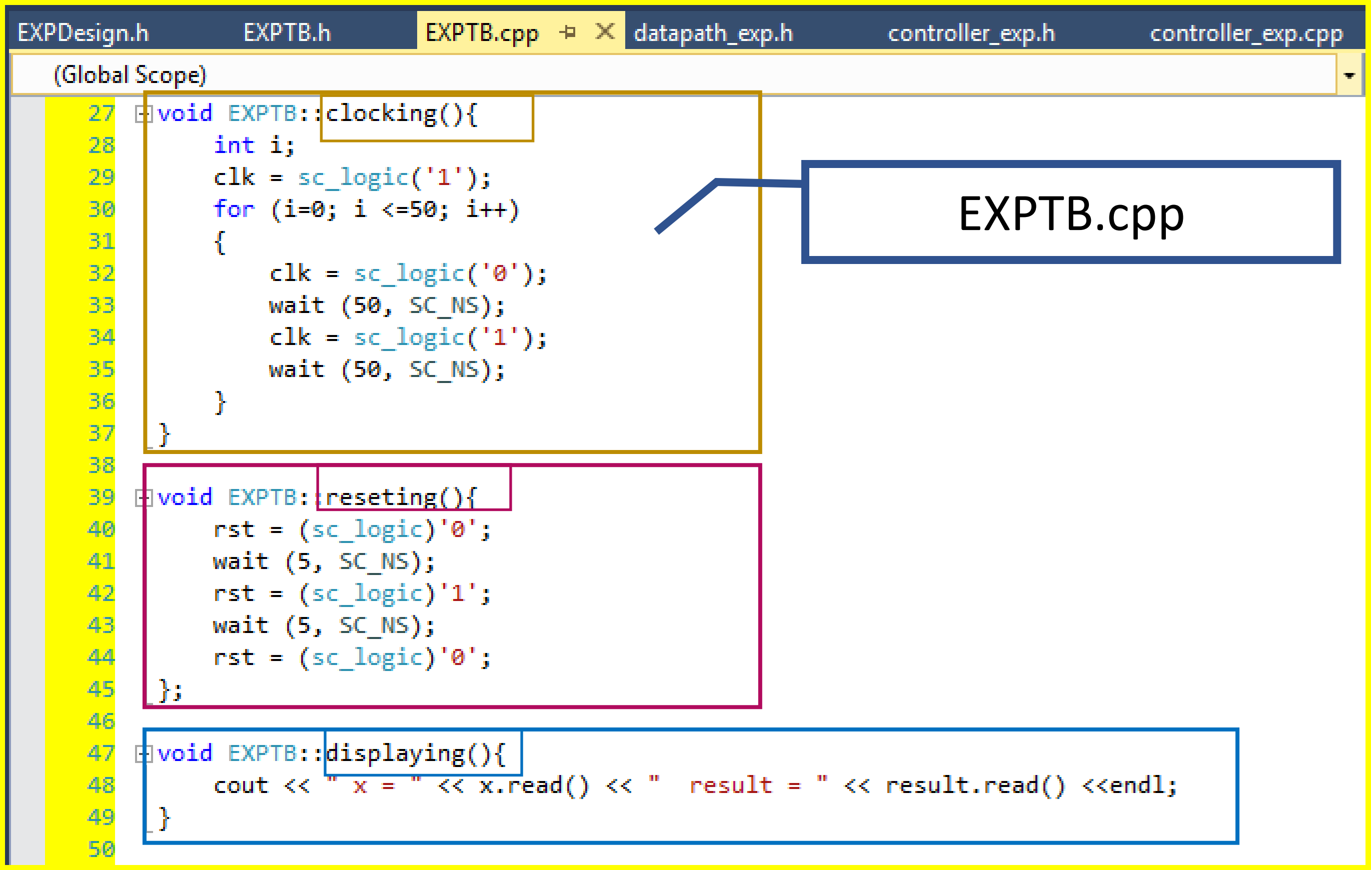
Exponential instantiation

Input generation

Exponential Circuit



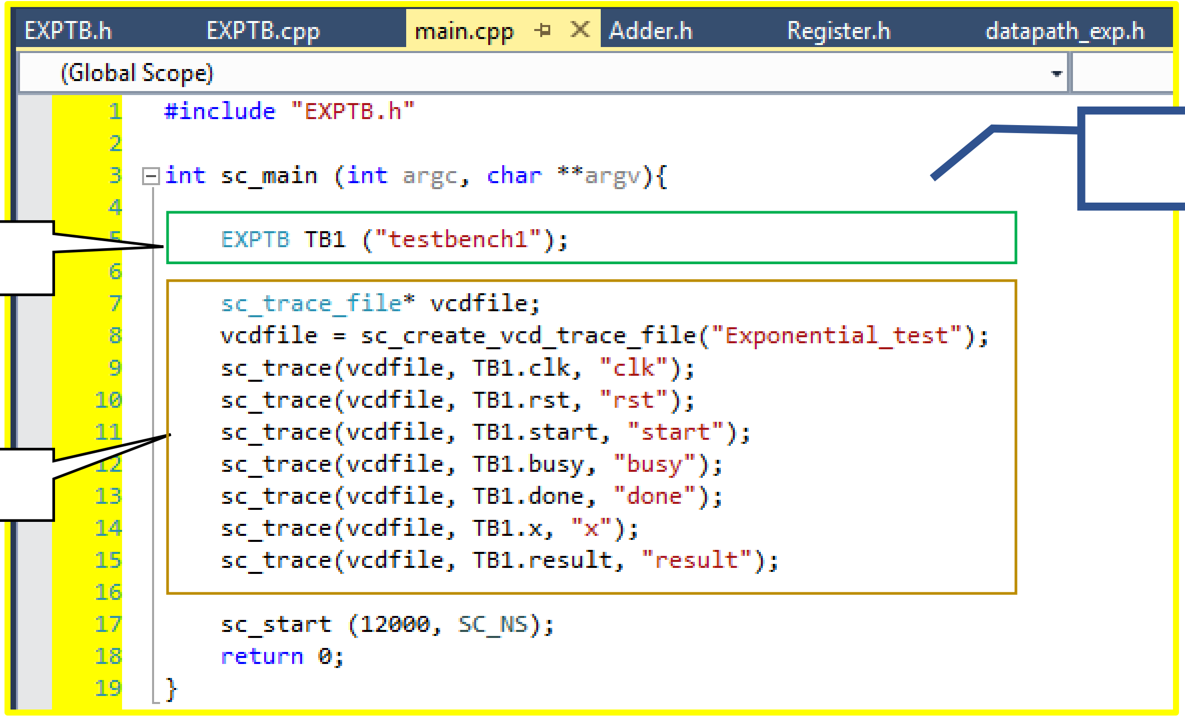
```
#include "EXPTB.h"

void EXPTB::inputing(){
        start = SC_LOGIC_0;
        wait(10, SC_NS);
        start = SC_LOGIC_1;
        x = "11111111";
        wait(60, SC_NS);
        start = SC_LOGIC_0;
        wait(2400, SC_NS);
        start = SC_LOGIC_1;
        x = "10000000";
        wait(100, SC_NS);
        start = SC_LOGIC_0;
        wait(2400, SC_NS);
        start = SC_LOGIC_1;
        x = "00000000";
        wait(100, SC_NS);
        start = SC_LOGIC_0;
        wait(2400, SC_NS);
        start = SC_LOGIC_1;
        x = "00000001";
        wait(100, SC_NS);
        start = SC_LOGIC_0;
}

void EXPTB::clocking(){ ... }

void EXPTB::reseting(){ ... }

void EXPTB::displaying(){ ... }
```



```
void EXPTB::clocking(){
    int i;
    clk = sc_logic('1');
    for (i=0; i <=50; i++)
    {
        clk = sc_logic('0');
        wait (50, SC_NS);
        clk = sc_logic('1');
        wait (50, SC_NS);
    }
}

void EXPTB::reseting(){
    rst = (sc_logic)'0';
    wait (5, SC_NS);
    rst = (sc_logic)'1';
    wait (5, SC_NS);
    rst = (sc_logic)'0';
};

void EXPTB::displaying(){
    cout << " x = " << x.read() << "  result = " << result.read() <<endl;
}
```

EXPTB.cpp

Exponential Circuit



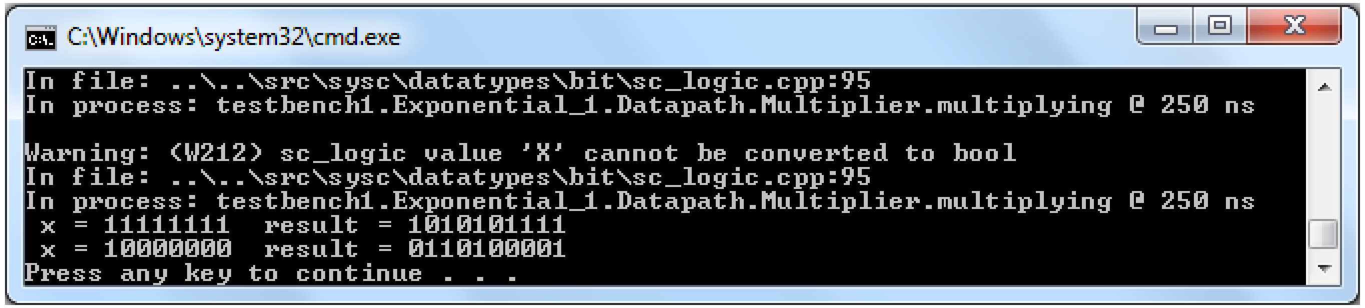
```
#include "EXPTB.h"

int sc_main (int argc, char **argv){

    EXPTB TB1 ("testbench1");

    sc_trace_file* vcdfile;
    vcdfile = sc_create_vcd_trace_file("Exponential_test");
    sc_trace(vcdfile, TB1.clk, "clk");
    sc_trace(vcdfile, TB1.rst, "rst");
    sc_trace(vcdfile, TB1.start, "start");
    sc_trace(vcdfile, TB1.busy, "busy");
    sc_trace(vcdfile, TB1.done, "done");
    sc_trace(vcdfile, TB1.x, "x");
    sc_trace(vcdfile, TB1.result, "result");

    sc_start (12000, SC_NS);
    return 0;
}
```

Main.cpp

Top-Level instantiation

VCD file and tracing signals

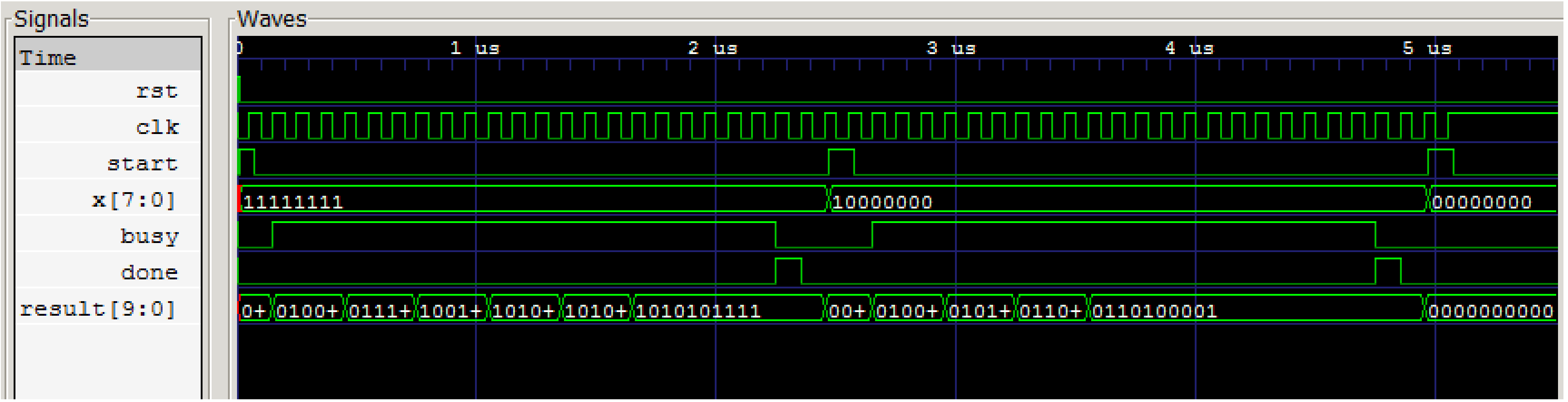
Exponential Circuit



```
In file: ..\..\src\sysc\datatypes\bit\sc_logic.cpp:95
In process: testbench1.Exponential_1.Datapath.Multiplier.multiplying @ 250 ns

Warning: (W212) sc_logic value 'X' cannot be converted to bool
In file: ..\..\src\sysc\datatypes\bit\sc_logic.cpp:95
In process: testbench1.Exponential_1.Datapath.Multiplier.multiplying @ 250 ns
 x = 11111111  result = 1010101111
 x = 10000000  result = 0110100001
Press any key to continue . . .
```

Exponential Circuit

- Waveform in a VCD viewer



SystemC Modeling

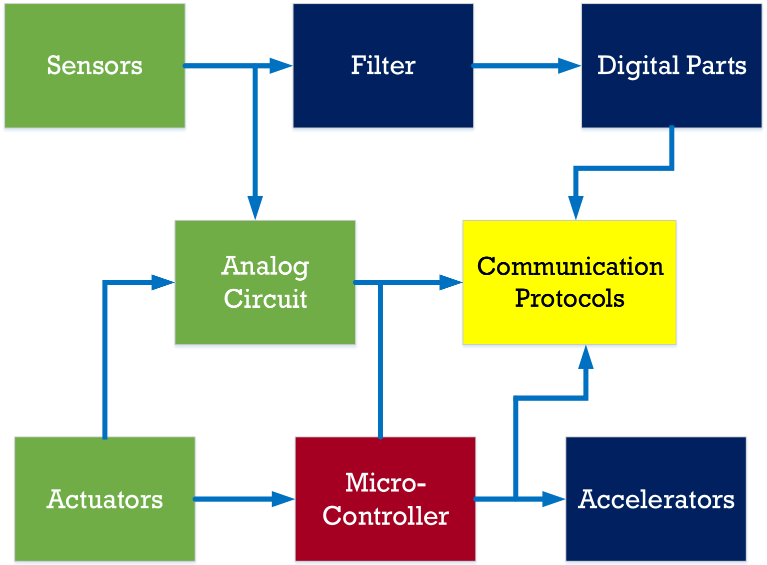
- + Taking Off From C++
- + SystemC Gate-Level Modeling
- Utilities for HDL Orientation
- + Timing & SystemC RT-Level Modeling
- + Components for RTL Design
- + SystemC RTL Design - Example
- SystemC Functional Modeling
- + SystemC Functional Design - Example
- Summary

SystemC Functional Modeling

- Functional Modeling
  - You just describe its behavior or function without caring about the hardware
  - Non-Synthesizable model, only for simulation
  - Faster-to-simulate and easy-to-debug
  - You can perform Design Space Exploration (DSE)
  - To model, you use loop, delay, task, wait statement, incomplete if…else

SystemC Functional Modeling

- Terminology, Other Functional Models
  - Bus Functional Modeling (BFM)
    - A bridge between functional interface (accepts transactions) and pin interfaces (operates the requisite bus protocol)
  - Cycle-accurate timing => mimic clock
  - Wrapper
    - A complete design for synchronizing and data handling

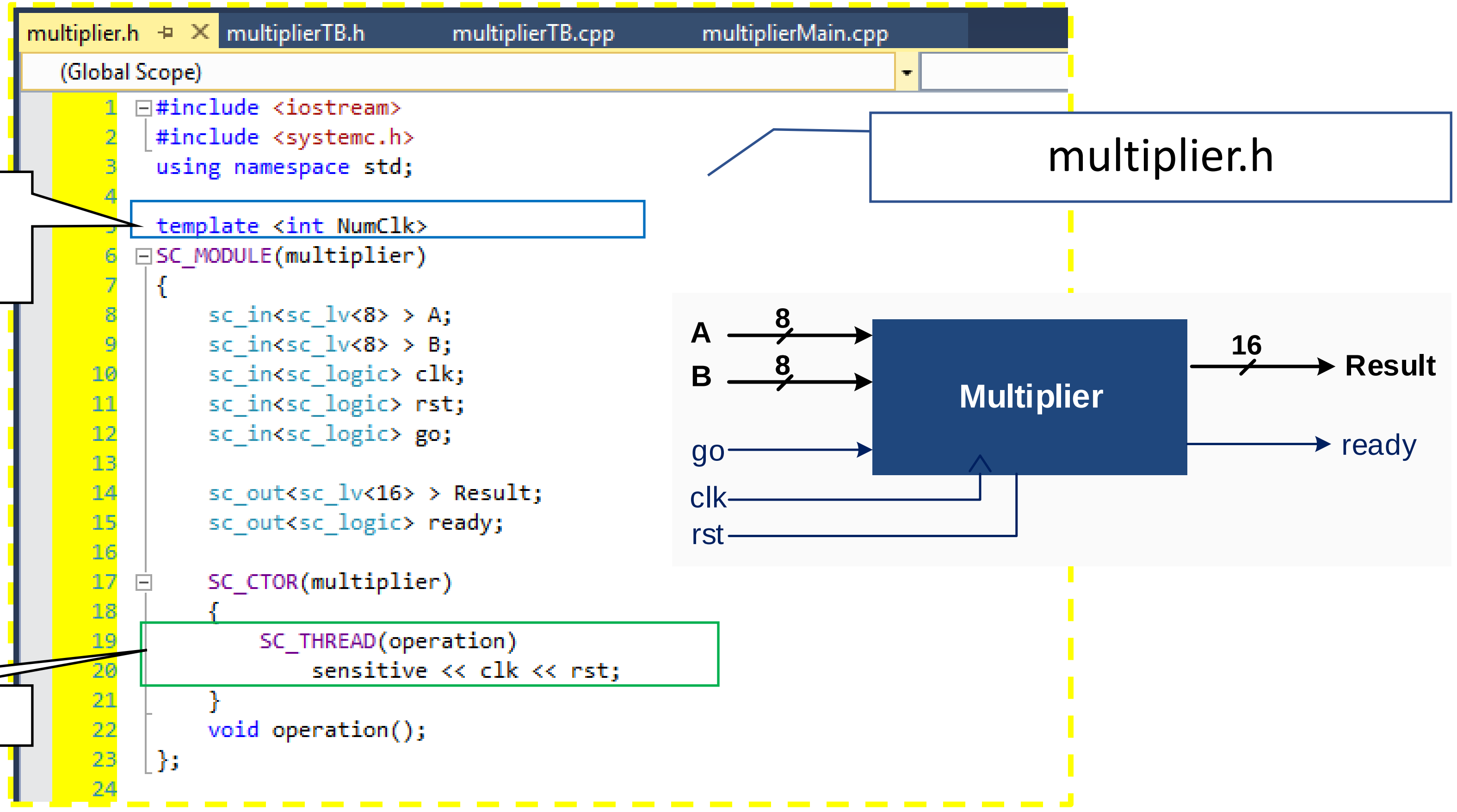


[Reference for BFM: https://semiengineering.com/knowledge_centers/eda-design/models/bus-functional-model/](https://semiengineering.com/knowledge_centers/eda-design/models/bus-functional-model/)

SystemC Modeling

- + Taking Off From C++
- + SystemC Gate-Level Modeling
- Utilities for HDL Orientation
- + Timing & SystemC RT-Level Modeling
- + Components for RTL Design
- + SystemC RTL Design - Example
- SystemC Functional Modeling
- – SystemC Functional Design - Example
  - Multiplier
  - Divider
  - Exponential Circuit
- Summary

Multiplier



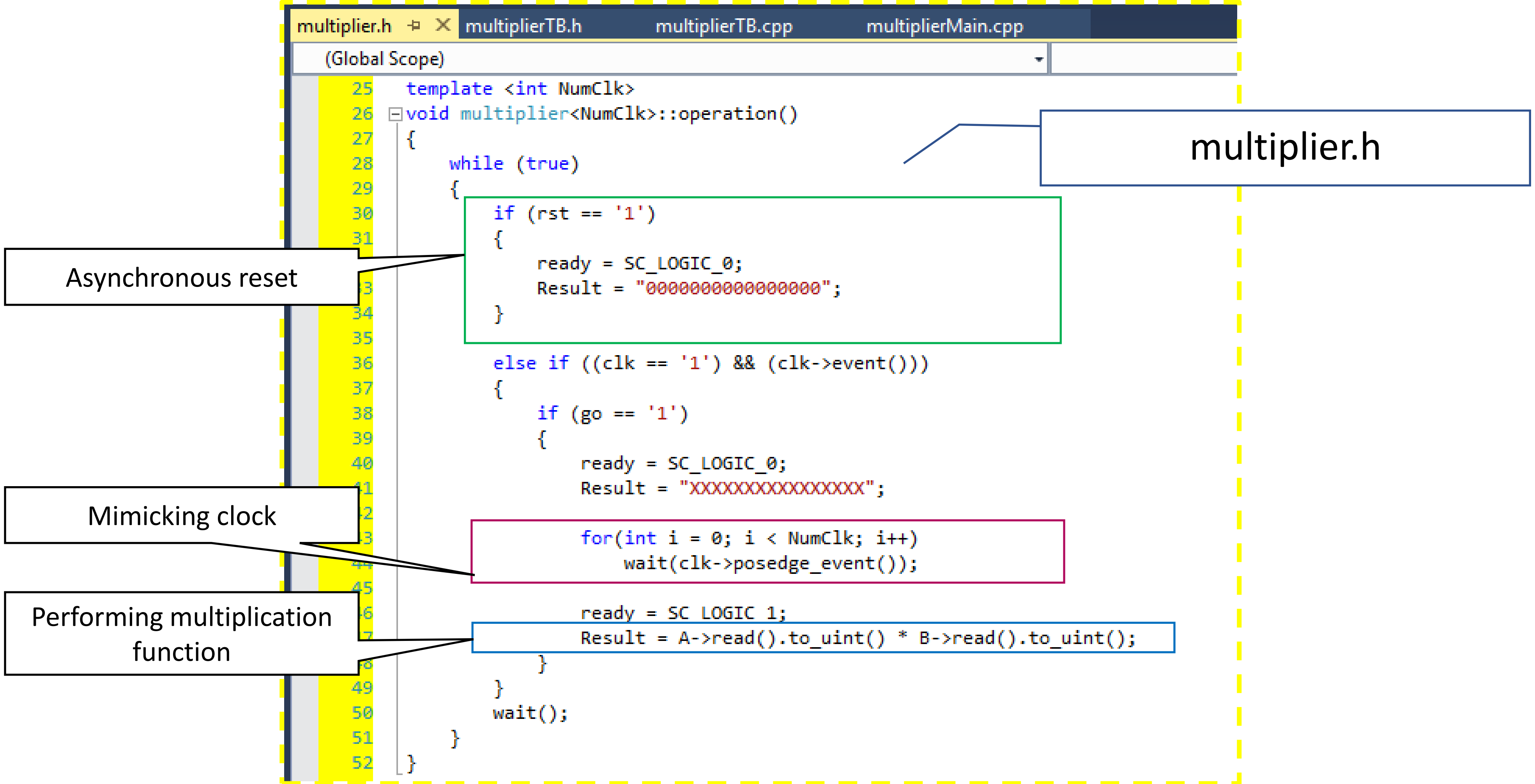
```
#include <iostream>
#include <systemc.h>
using namespace std;

template <int NumClk>
SC_MODULE(multiplier)
{
    sc_in<sc_lv<8> > A;
    sc_in<sc_lv<8> > B;
    sc_in<sc_logic> clk;
    sc_in<sc_logic> rst;
    sc_in<sc_logic> go;

    sc_out<sc_lv<16> > Result;
    sc_out<sc_logic> ready;

    SC_CTOR(multiplier)
    {
        SC_THREAD(operation)
            sensitive << clk << rst;
    }
    void operation();
};
```

multiplier.h

Templated ***SC_MODULE***
NumClk: Number of clocks

Sensitive to clock and reset

Multiplier



```
template <int NumClk>
void multiplier<NumClk>::operation()
{
    while (true)
    {
        if (rst == '1')
        {
            ready = SC_LOGIC_0;
            Result = "0000000000000000";
        }

        else if ((clk == '1') && (clk->event()))
        {
            if (go == '1')
            {
                ready = SC_LOGIC_0;
                Result = "XXXXXXXXXXXXXXXX";

                for(int i = 0; i < NumClk; i++)
                    wait(clk->posedge_event());

                ready = SC_LOGIC_1;
                Result = A->read().to_uint() * B->read().to_uint();
            }
        }
        wait();
    }
}
```

multiplier.h

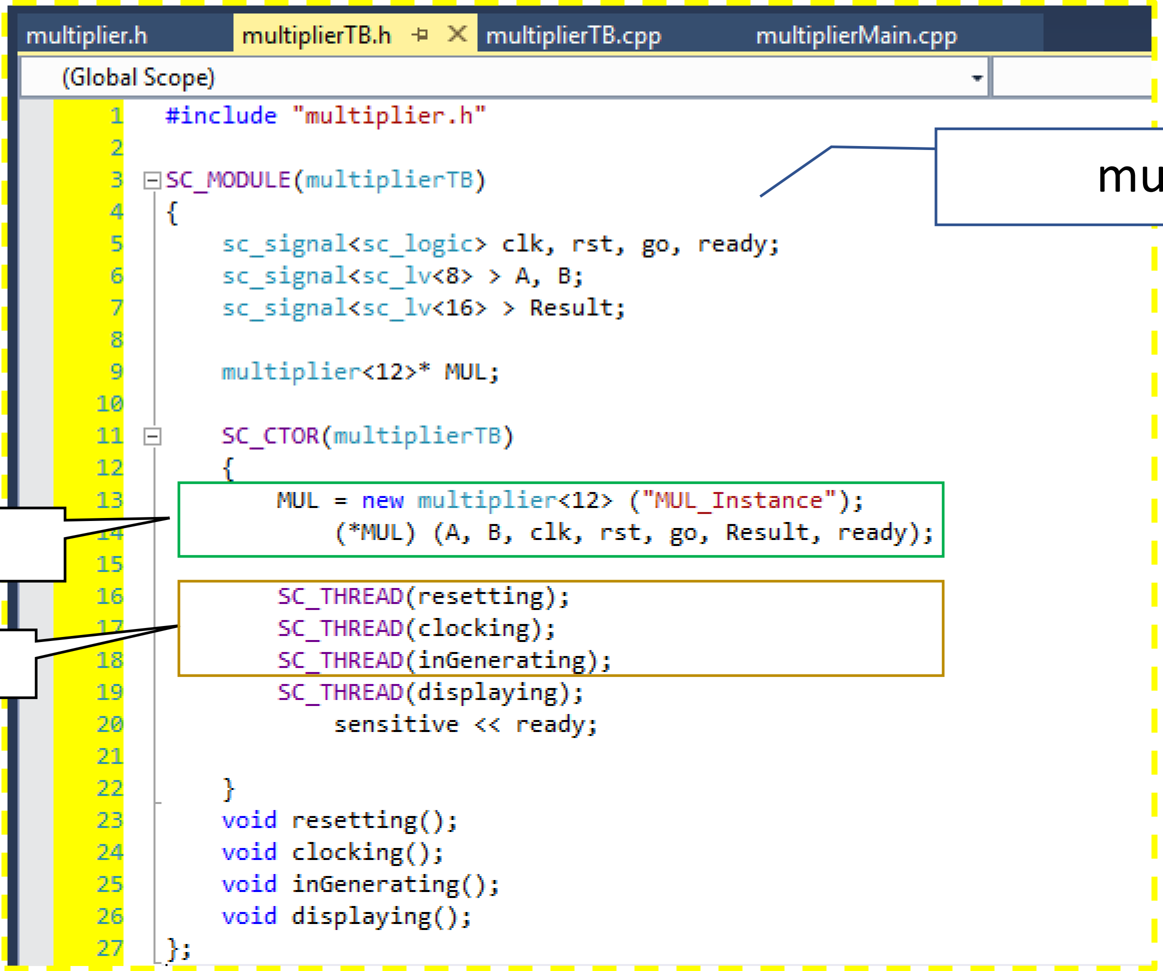
Asynchronous reset

Mimicking clock

Performing multiplication function

SystemC Functional Design - Example

Multiplier



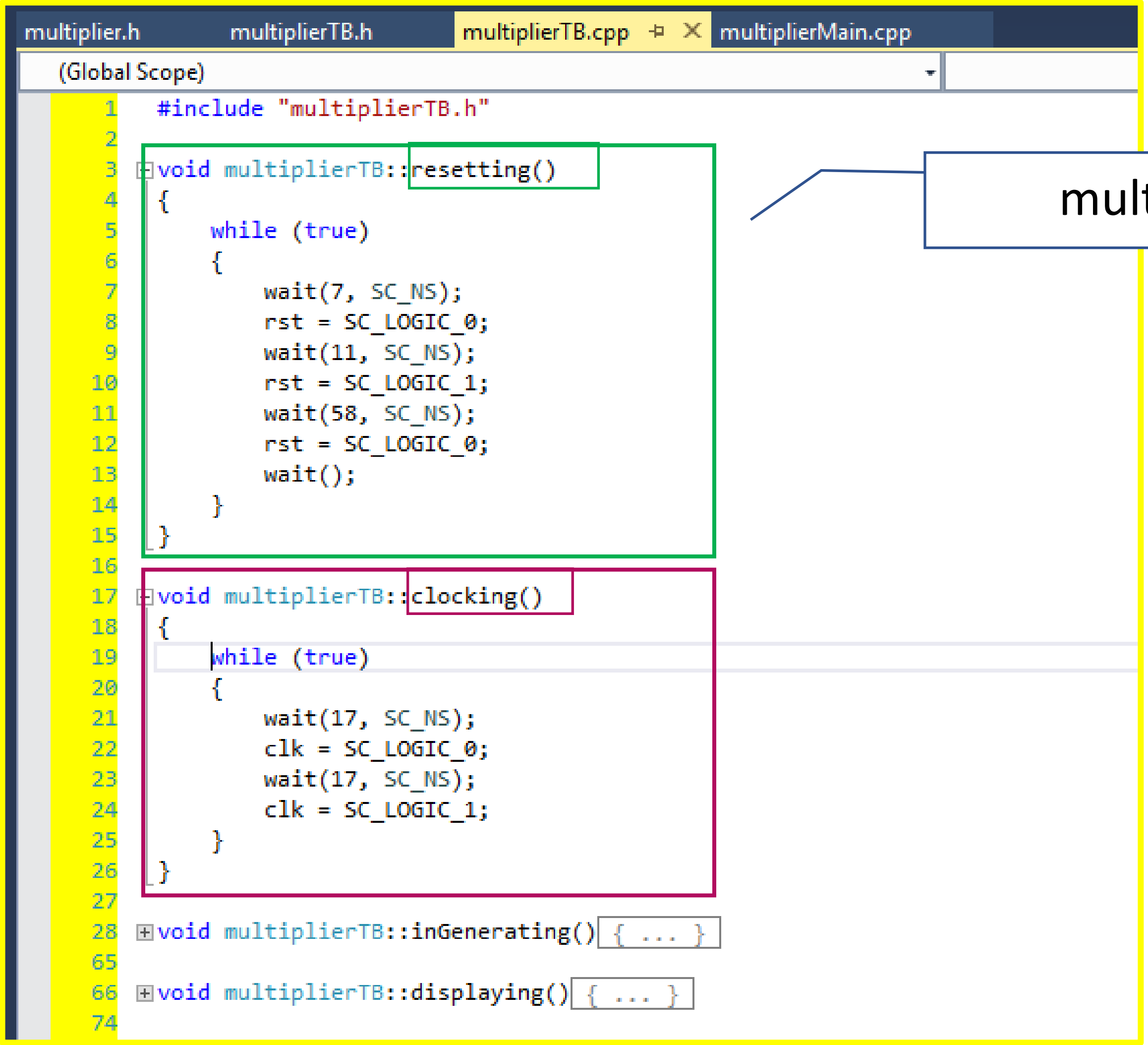
```
#include "multiplier.h"

SC_MODULE(multiplierTB)
{
    sc_signal<sc_logic> clk, rst, go, ready;
    sc_signal<sc_lv<8> > A, B;
    sc_signal<sc_lv<16> > Result;

    multiplier<12>* MUL;

    SC_CTOR(multiplierTB)
    {
        MUL = new multiplier<12> ("MUL_Instance");
            (*MUL) (A, B, clk, rst, go, Result, ready);

        SC_THREAD(resetting);
        SC_THREAD(clocking);
        SC_THREAD(inGenerating);
        SC_THREAD(displaying);
            sensitive << ready;

    }
    void resetting();
    void clocking();
    void inGenerating();
    void displaying();
};
```

multiplierTB.h

Multiplier instantiation

Input generation

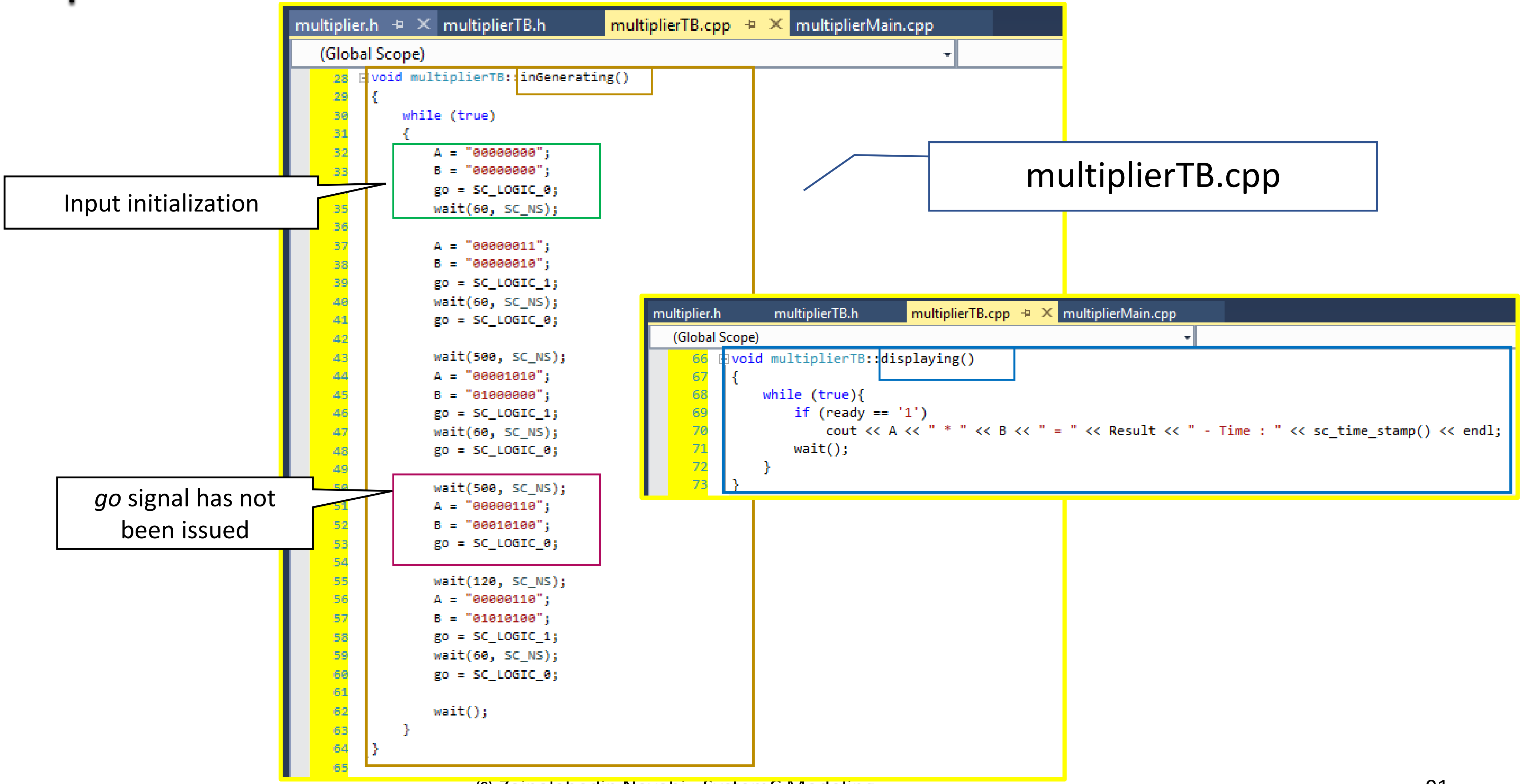
Multiplier



```
#include "multiplierTB.h"

void multiplierTB::resetting()
{
    while (true)
    {
        wait(7, SC_NS);
        rst = SC_LOGIC_0;
        wait(11, SC_NS);
        rst = SC_LOGIC_1;
        wait(58, SC_NS);
        rst = SC_LOGIC_0;
        wait();
    }
}

void multiplierTB::clocking()
{
    while (true)
    {
        wait(17, SC_NS);
        clk = SC_LOGIC_0;
        wait(17, SC_NS);
        clk = SC_LOGIC_1;
    }
}

void multiplierTB::inGenerating(){ ... }

void multiplierTB::displaying(){ ... }
```

multiplierTB.cpp

Multiplier



multiplierTB.cpp

Input initialization

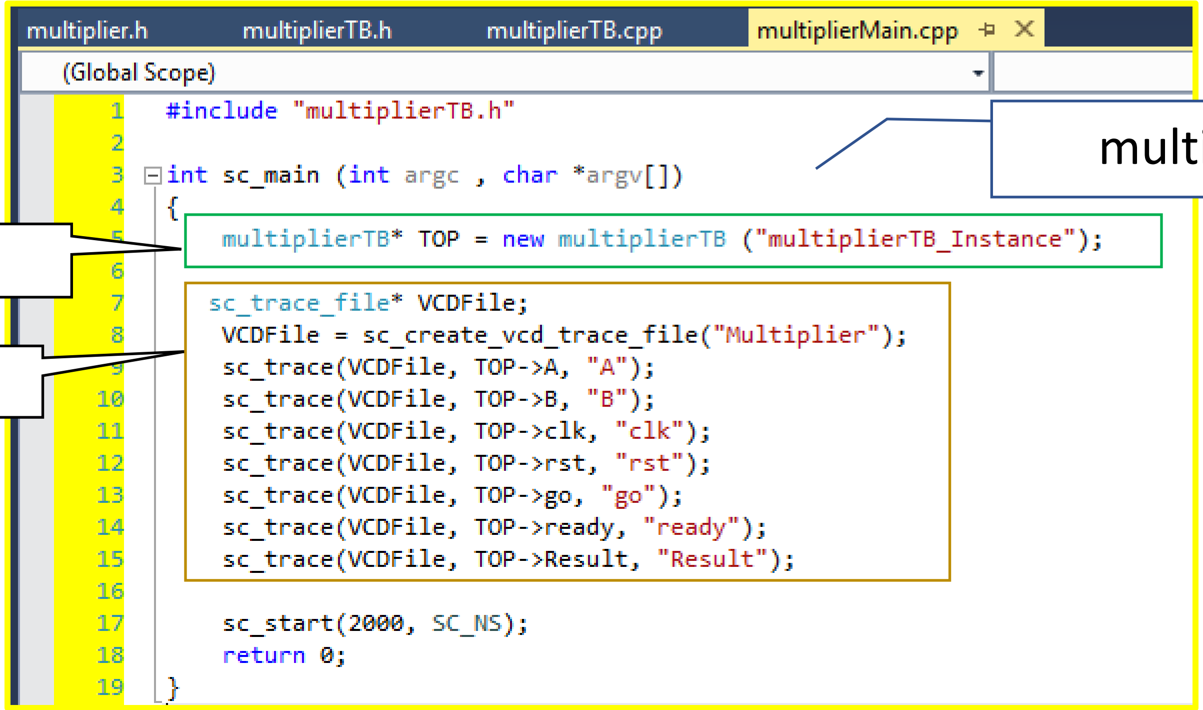
go signal has not been issued

```
void multiplierTB::inGenerating()
{
    while (true)
    {
        A = "00000000";
        B = "00000000";
        go = SC_LOGIC_0;
        wait(60, SC_NS);

        A = "00000011";
        B = "00000010";
        go = SC_LOGIC_1;
        wait(60, SC_NS);
        go = SC_LOGIC_0;

        wait(500, SC_NS);
        A = "00001010";
        B = "01000000";
        go = SC_LOGIC_1;
        wait(60, SC_NS);
        go = SC_LOGIC_0;

        wait(500, SC_NS);
        A = "00000110";
        B = "00010100";
        go = SC_LOGIC_0;

        wait(120, SC_NS);
        A = "00000110";
        B = "01010100";
        go = SC_LOGIC_1;
        wait(60, SC_NS);
        go = SC_LOGIC_0;

        wait();
    }
}
```

```
void multiplierTB::displaying()
{
    while (true){
        if (ready == '1')
            cout << A << " * " << B << " = " << Result << " - Time : " << sc_time_stamp() << endl;
        wait();
    }
}
```

SystemC Functional Design - Example

Multiplier



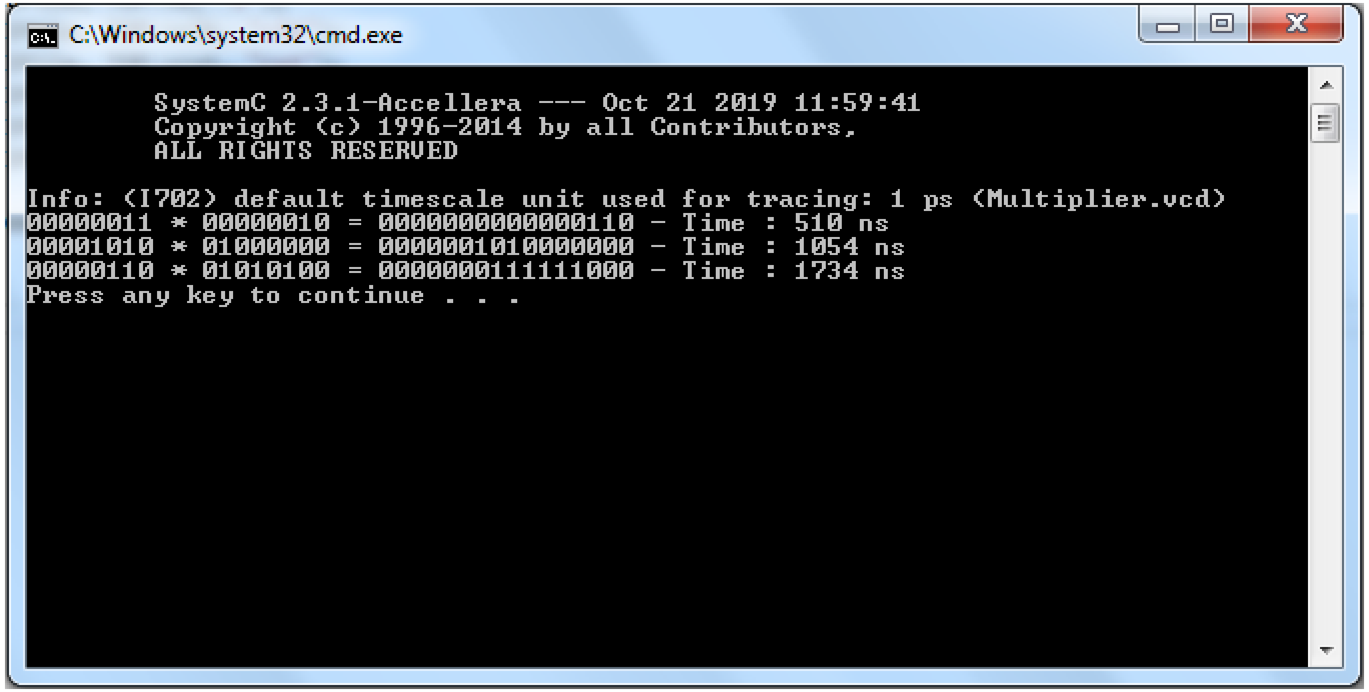
```
#include "multiplierTB.h"

int sc_main (int argc , char *argv[])
{
    multiplierTB* TOP = new multiplierTB ("multiplierTB_Instance");

   sc_trace_file* VCDFile;
    VCDFile = sc_create_vcd_trace_file("Multiplier");
    sc_trace(VCDFile, TOP->A, "A");
    sc_trace(VCDFile, TOP->B, "B");
    sc_trace(VCDFile, TOP->clk, "clk");
    sc_trace(VCDFile, TOP->rst, "rst");
    sc_trace(VCDFile, TOP->go, "go");
    sc_trace(VCDFile, TOP->ready, "ready");
    sc_trace(VCDFile, TOP->Result, "Result");

    sc_start(2000, SC_NS);
    return 0;
}
```

multiplierMain.cpp

Top-Level instantiation

VCD file and tracing signals

Multiplier

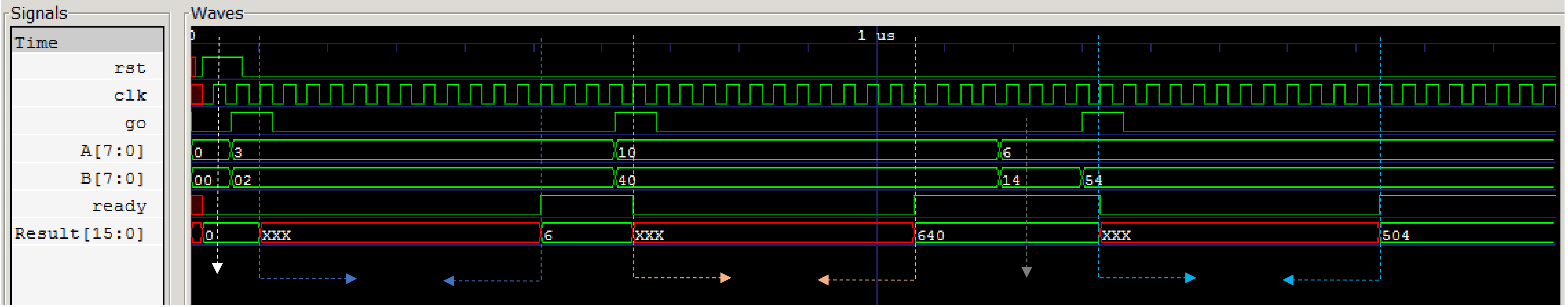


```
        SystemC 2.3.1-Accellera --- Oct 21 2019 11:59:41
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED

Info: (I702) default timescale unit used for tracing: 1 ps (Multiplier.vcd)
00000011 * 00000010 = 0000000000000110 - Time : 510 ns
00001010 * 01000000 = 0000001010000000 - Time : 1054 ns
00000110 * 01010100 = 0000000111111000 - Time : 1734 ns
Press any key to continue . . .
```

SystemC Functional Design - Example

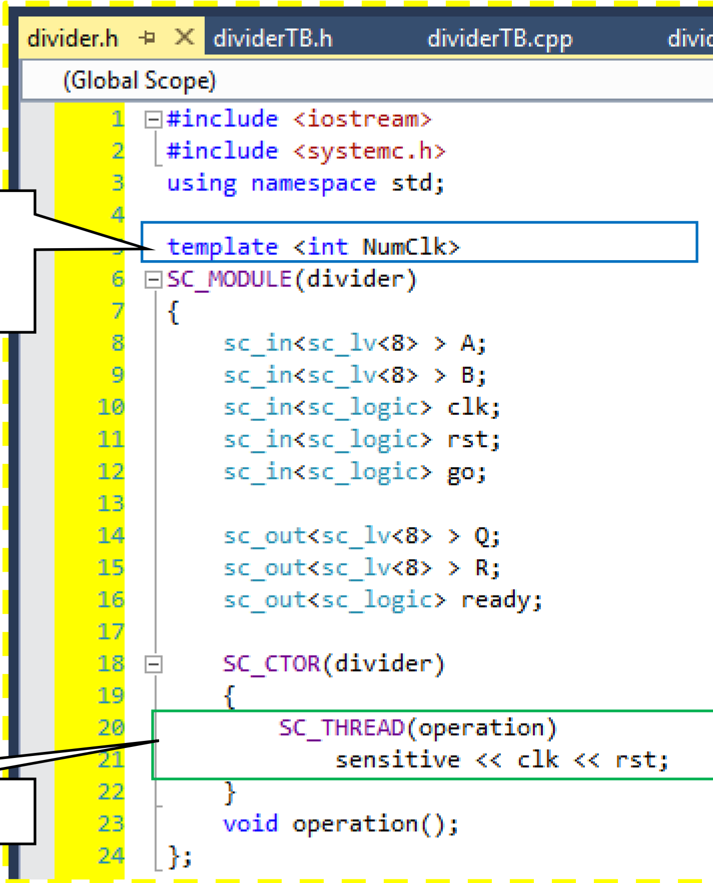
Multiplier

- Waveform in a VCD viewer



— SystemC Functional Design - Example

Divider

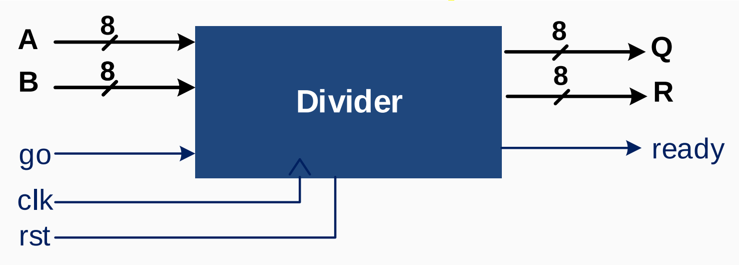


```
#include <iostream>
#include <systemc.h>
using namespace std;

template <int NumClk>
SC_MODULE(divider)
{
    sc_in<sc_lv<8> > A;
    sc_in<sc_lv<8> > B;
    sc_in<sc_logic> clk;
    sc_in<sc_logic> rst;
    sc_in<sc_logic> go;

    sc_out<sc_lv<8> > Q;
    sc_out<sc_lv<8> > R;
    sc_out<sc_logic> ready;

    SC_CTOR(divider)
    {
        SC_THREAD(operation)
            sensitive << clk << rst;
    }
    void operation();
};
```

Templated ***SC_MODULE***
NumClk: Number of clocks

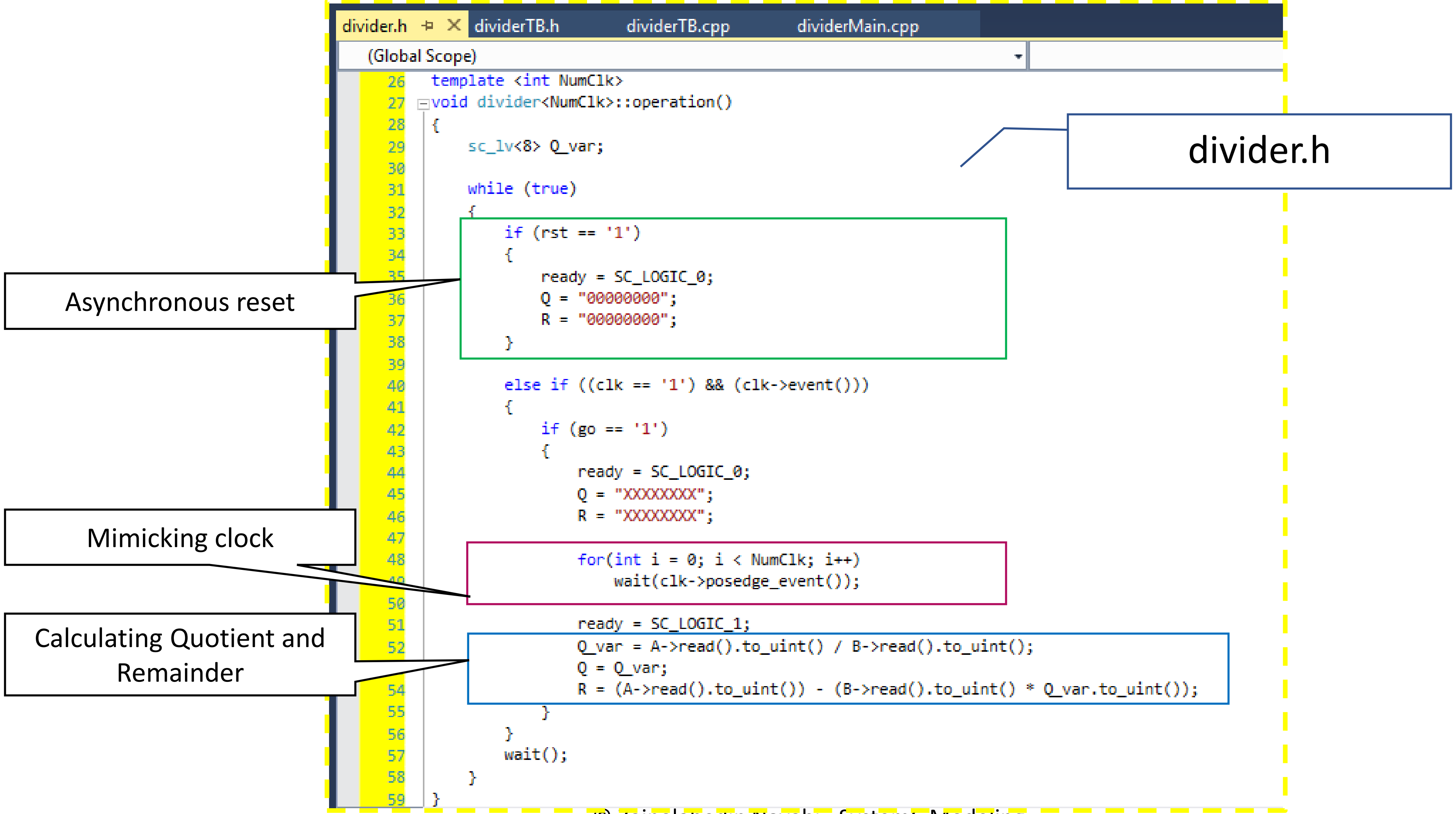
divider.h

Sensitive to clock and reset



SystemC Functional Design - Example

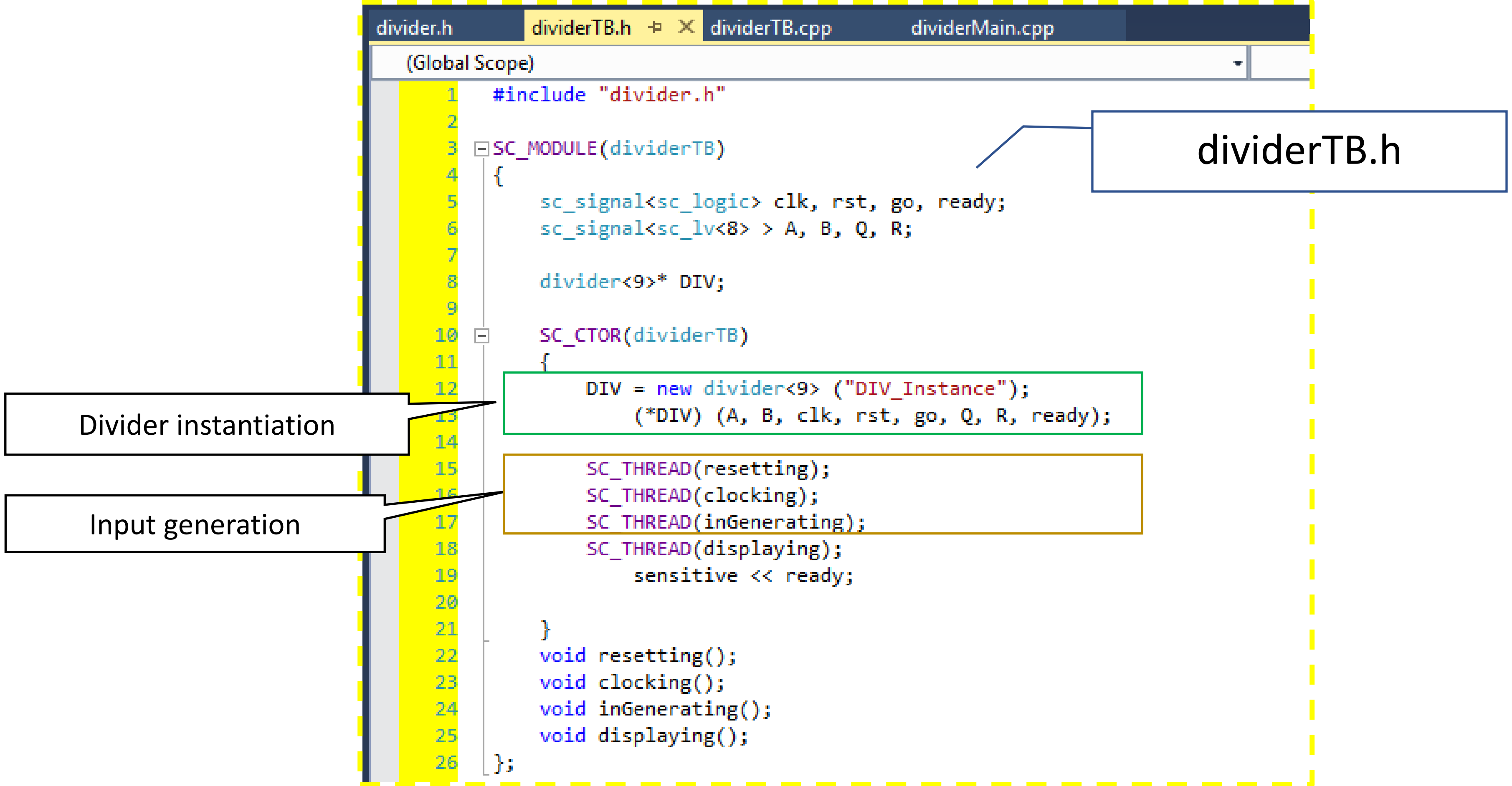
Divider



```
template <int NumClk>
void divider<NumClk>::operation()
{
    sc_lv<8> Q_var;

    while (true)
    {
        if (rst == '1')
        {
            ready = SC_LOGIC_0;
            Q = "00000000";
            R = "00000000";
        }

        else if ((clk == '1') && (clk->event()))
        {
            if (go == '1')
            {
                ready = SC_LOGIC_0;
                Q = "XXXXXXXX";
                R = "XXXXXXXX";

                for(int i = 0; i < NumClk; i++)
                    wait(clk->posedge_event());

                ready = SC_LOGIC_1;
                Q_var = A->read().to_uint() / B->read().to_uint();
                Q = Q_var;
                R = (A->read().to_uint()) - (B->read().to_uint() * Q_var.to_uint());
            }
        }
        wait();
    }
}
```

SystemC Functional Design - Example

Divider



dividerTB.h

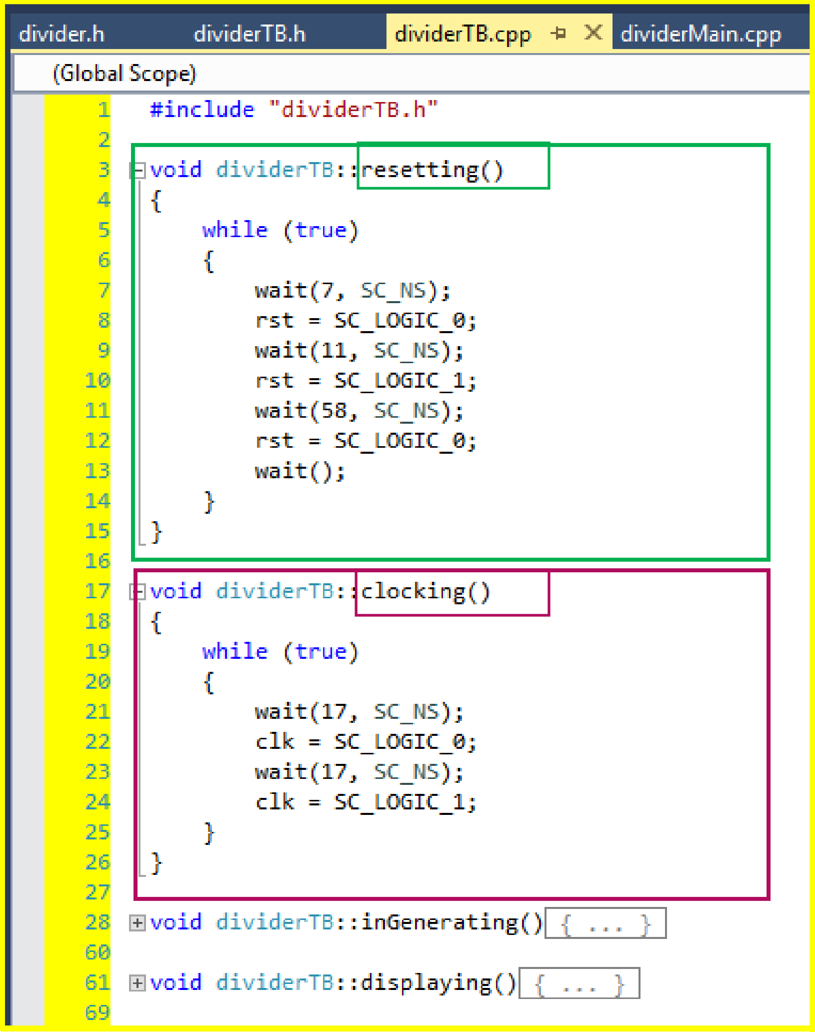
Divider instantiation

Input generation

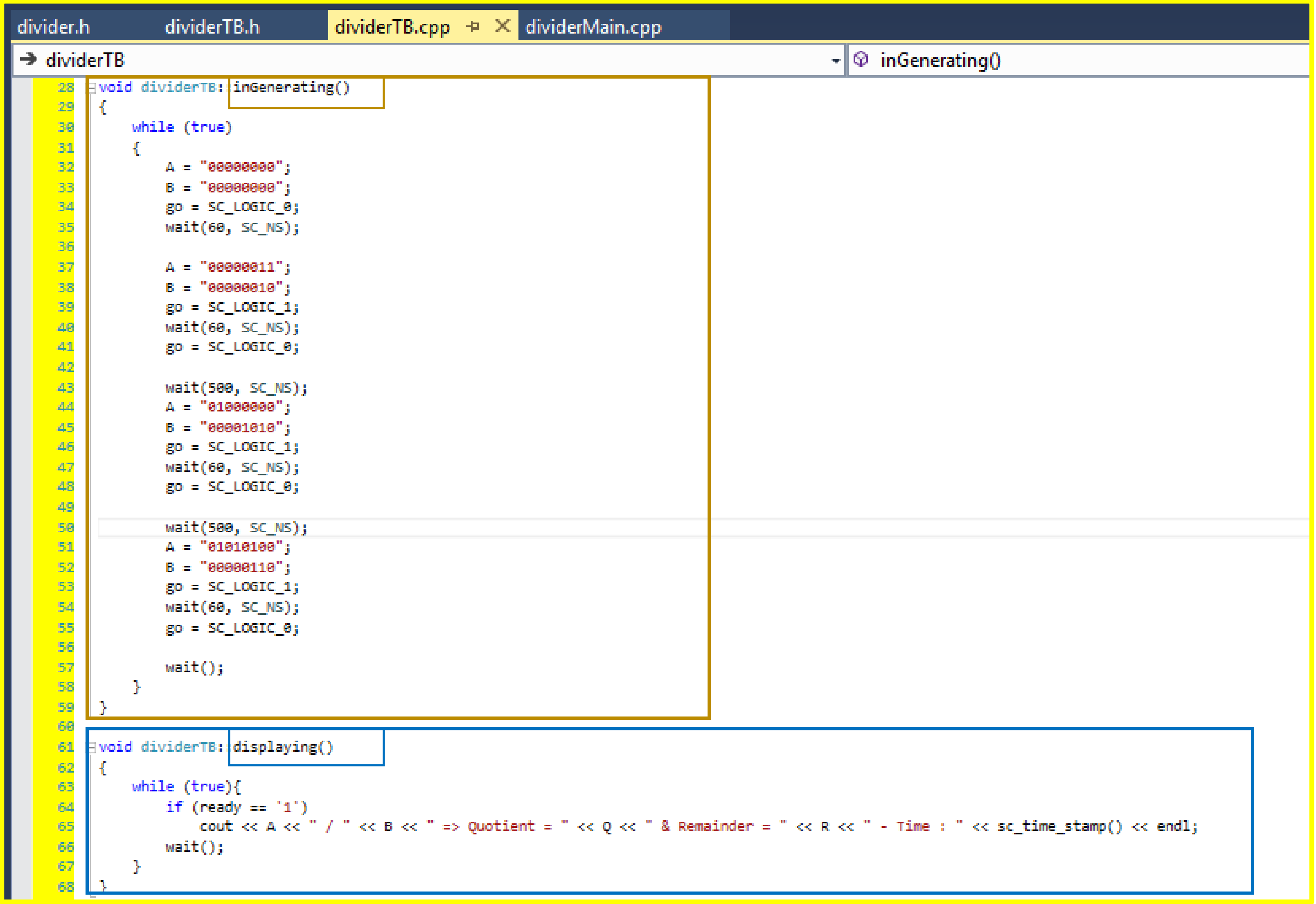
```
#include "divider.h"

SC_MODULE(dividerTB)
{
    sc_signal<sc_logic> clk, rst, go, ready;
    sc_signal<sc_lv<8> > A, B, Q, R;

    divider<9>* DIV;

    SC_CTOR(dividerTB)
    {
        DIV = new divider<9> ("DIV_Instance");
            (*DIV) (A, B, clk, rst, go, Q, R, ready);

        SC_THREAD(resetting);
        SC_THREAD(clocking);
        SC_THREAD(inGenerating);
        SC_THREAD(displaying);
            sensitive << ready;

    }
    void resetting();
    void clocking();
    void inGenerating();
    void displaying();
};
```

SystemC Functional Design - Example

Divider



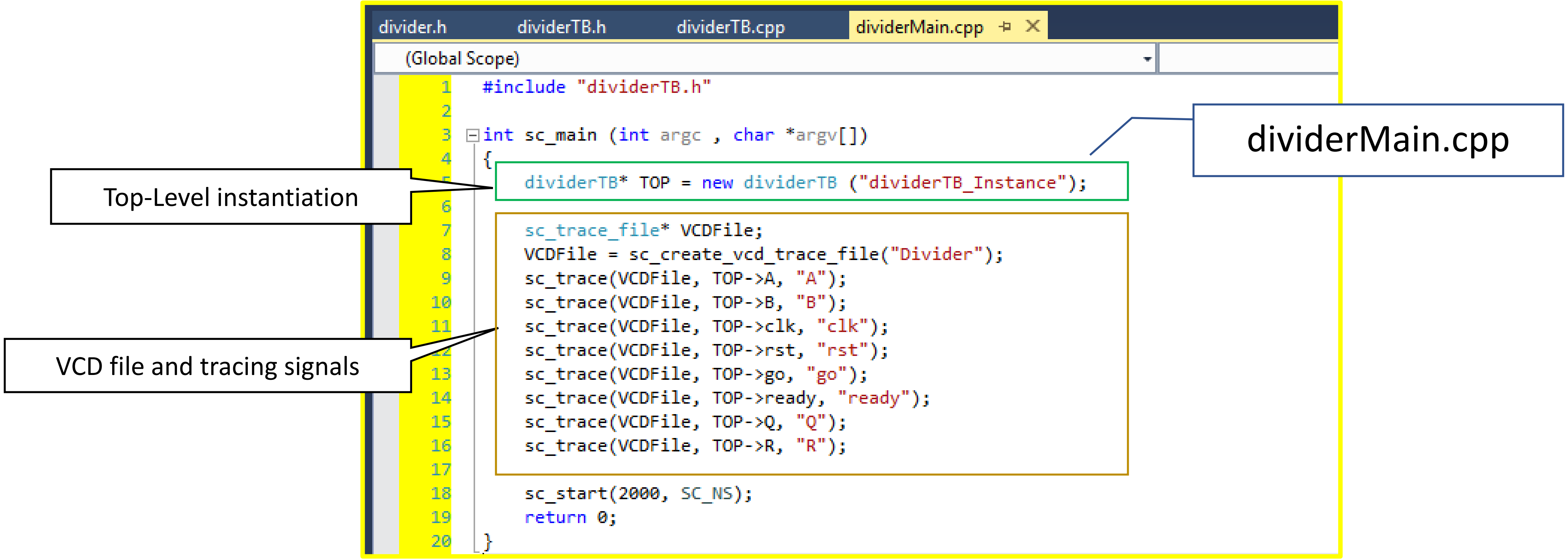
```
#include "dividerTB.h"

void dividerTB::resetting()
{
    while (true)
    {
        wait(7, SC_NS);
        rst = SC_LOGIC_0;
        wait(11, SC_NS);
        rst = SC_LOGIC_1;
        wait(58, SC_NS);
        rst = SC_LOGIC_0;
        wait();
    }
}

void dividerTB::clocking()
{
    while (true)
    {
        wait(17, SC_NS);
        clk = SC_LOGIC_0;
        wait(17, SC_NS);
        clk = SC_LOGIC_1;
    }
}

void dividerTB::inGenerating() { ... }

void dividerTB::displaying() { ... }
```



```
void dividerTB::inGenerating()
{
    while (true)
    {
        A = "00000000";
        B = "00000000";
        go = SC_LOGIC_0;
        wait(60, SC_NS);

        A = "00000011";
        B = "00000010";
        go = SC_LOGIC_1;
        wait(60, SC_NS);
        go = SC_LOGIC_0;

        wait(500, SC_NS);
        A = "01000000";
        B = "00001010";
        go = SC_LOGIC_1;
        wait(60, SC_NS);
        go = SC_LOGIC_0;

        wait(500, SC_NS);
        A = "01010100";
        B = "00000110";
        go = SC_LOGIC_1;
        wait(60, SC_NS);
        go = SC_LOGIC_0;

        wait();
    }
}

void dividerTB::displaying()
{
    while (true){
        if (ready == '1')
            cout << A << " / " << B << " => Quotient = " << Q << " & Remainder = " << R << " - Time : " << sc_time_stamp() << endl;
        wait();
    }
}
```

SystemC Functional Design - Example

Divider



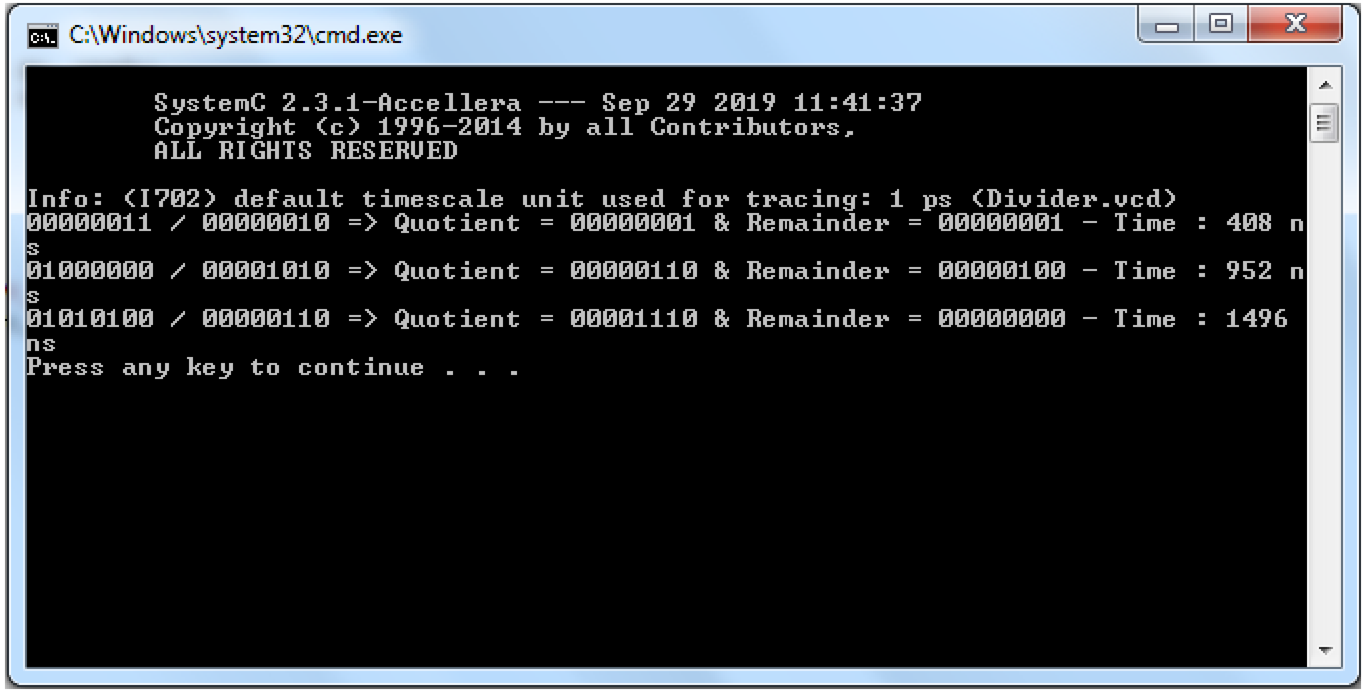
dividerMain.cpp

```
#include "dividerTB.h"

int sc_main (int argc , char *argv[])
{
    dividerTB* TOP = new dividerTB ("dividerTB_Instance");

    sc_trace_file* VCDFile;
    VCDFile = sc_create_vcd_trace_file("Divider");
    sc_trace(VCDFile, TOP->A, "A");
    sc_trace(VCDFile, TOP->B, "B");
    sc_trace(VCDFile, TOP->clk, "clk");
    sc_trace(VCDFile, TOP->rst, "rst");
    sc_trace(VCDFile, TOP->go, "go");
    sc_trace(VCDFile, TOP->ready, "ready");
    sc_trace(VCDFile, TOP->Q, "Q");
    sc_trace(VCDFile, TOP->R, "R");

    sc_start(2000, SC_NS);
    return 0;
}
```

Top-Level instantiation

VCD file and tracing signals

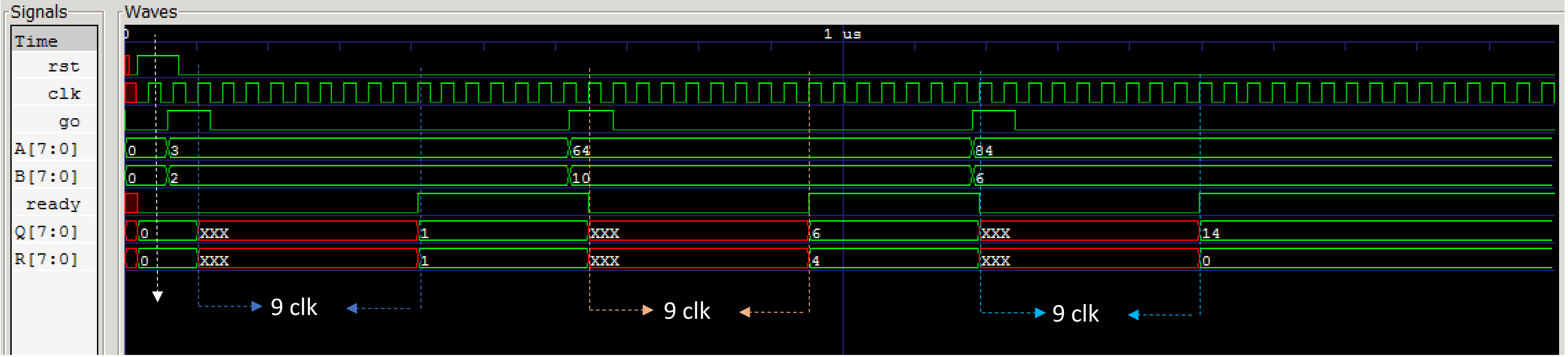
Divider



```
        SystemC 2.3.1-Accellera --- Sep 29 2019 11:41:37
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED

Info: (I702) default timescale unit used for tracing: 1 ps (Divider.vcd)
00000011 / 00000010 => Quotient = 00000001 & Remainder = 00000001 - Time : 408 n
s
01000000 / 00001010 => Quotient = 00000110 & Remainder = 00000100 - Time : 952 n
s
01010100 / 00000110 => Quotient = 00001110 & Remainder = 00000000 - Time : 1496
ns
Press any key to continue . . .
```

Divider

- Waveform in a VCD viewer

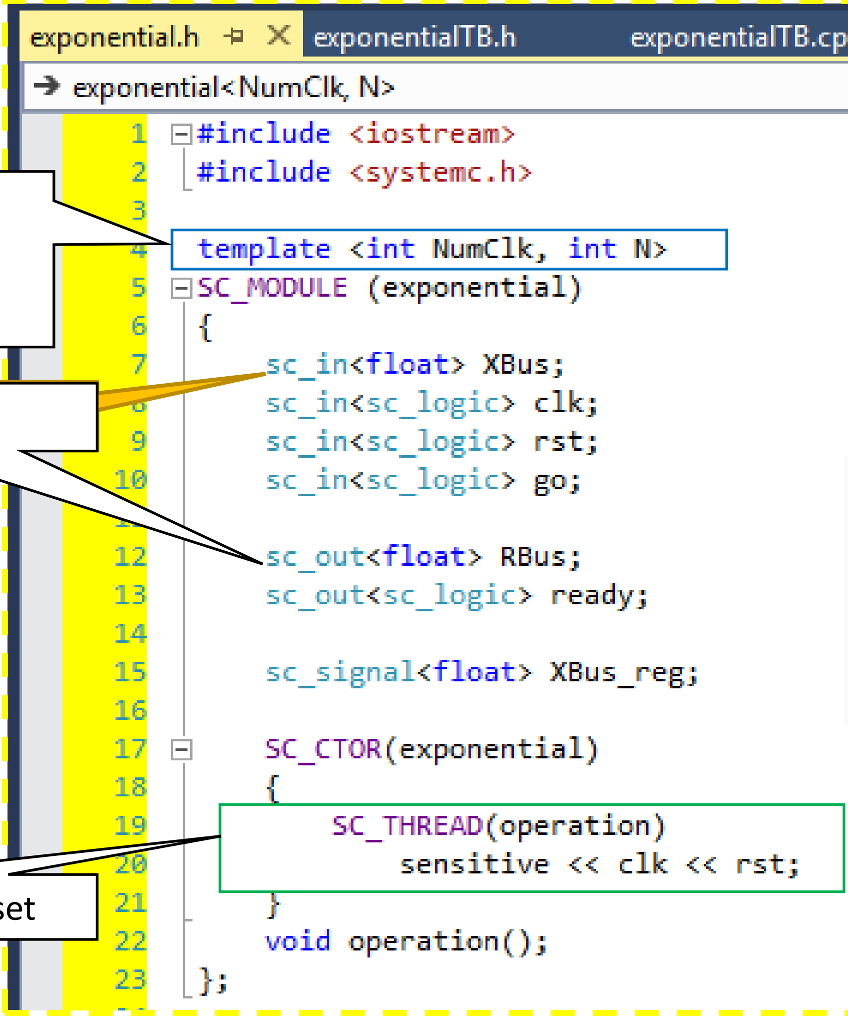


Exponential

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

```
term = 1;
exp = 1;
for( i = 1; i < n; i++ ) {
   term = term × x × ( 1 / i );
   exp = exp + term;
}
```

Exponential



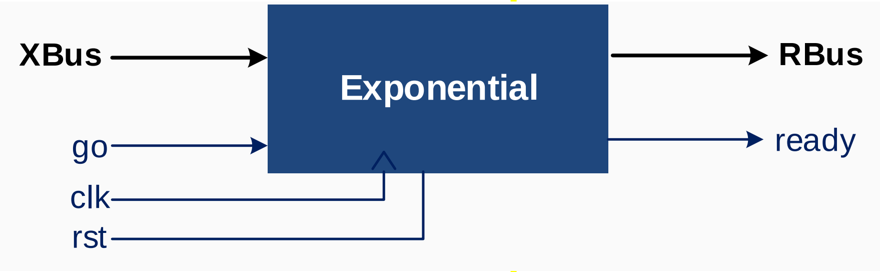
```
#include <iostream>
#include <systemc.h>

template <int NumClk, int N>
SC_MODULE (exponential)
{
        sc_in<float> XBus;
        sc_in<sc_logic> clk;
        sc_in<sc_logic> rst;
        sc_in<sc_logic> go;

        sc_out<float> RBus;
        sc_out<sc_logic> ready;

        sc_signal<float> XBus_reg;

        SC_CTOR(exponential)
        {
            SC_THREAD(operation)
                sensitive << clk << rst;
        }
        void operation();
};
```

Templated ***SC_MODULE***
NumClk: Number of clocks
N: Number of iterations

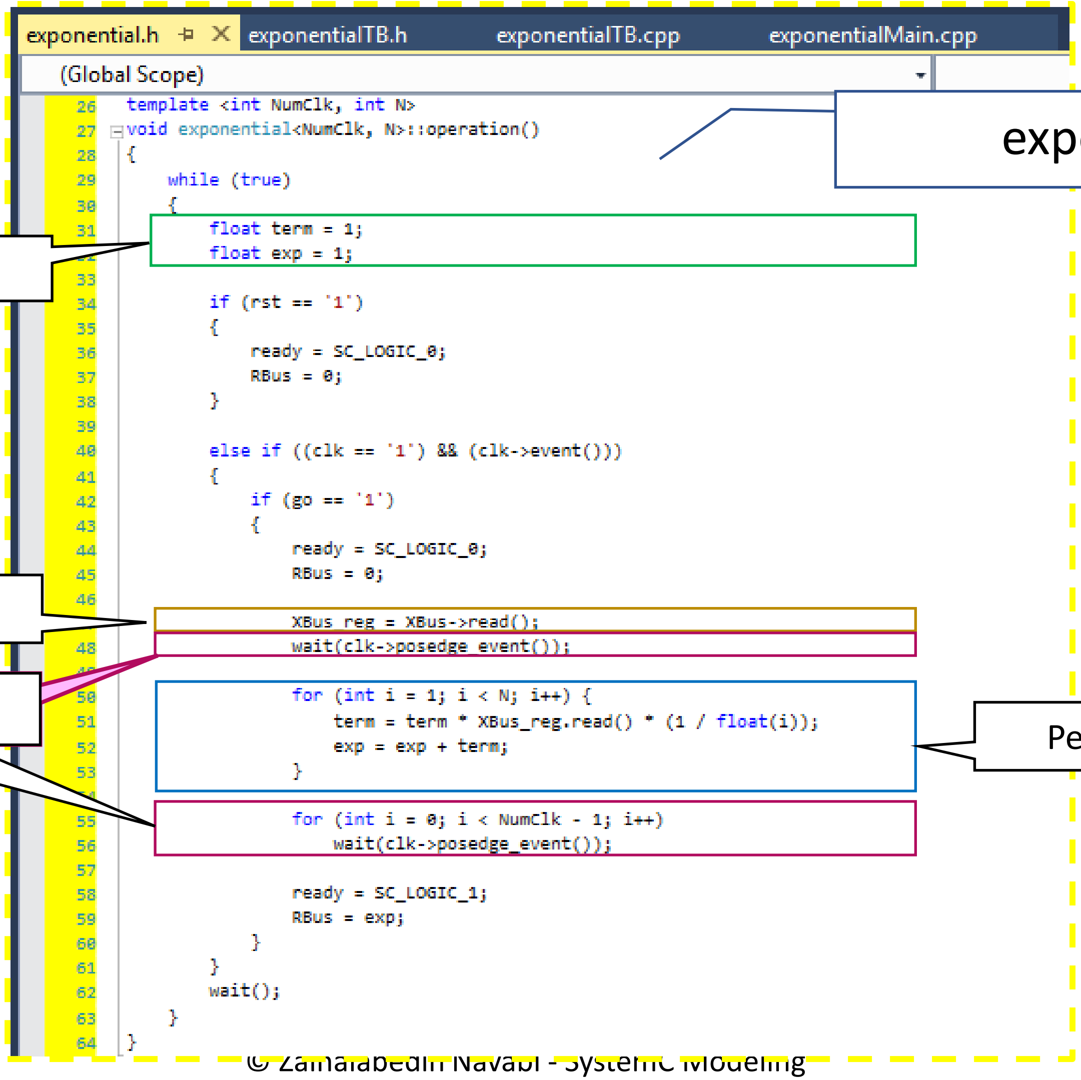
C++ data type

Sensitive to clock and reset

exponential.h



Exponential



```
template <int NumClk, int N>
void exponential<NumClk, N>::operation()
{
    while (true)
    {
        float term = 1;
        float exp = 1;

        if (rst == '1')
        {
            ready = SC_LOGIC_0;
            RBus = 0;
        }

        else if ((clk == '1') && (clk->event()))
        {
            if (go == '1')
            {
                ready = SC_LOGIC_0;
                RBus = 0;

                XBus_reg = XBus->read();
                wait(clk->posedge_event());

                for (int i = 1; i < N; i++) {
                    term = term * XBus_reg.read() * (1 / float(i));
                    exp = exp + term;
                }

                for (int i = 0; i < NumClk - 1; i++)
                    wait(clk->posedge_event());

                ready = SC_LOGIC_1;
                RBus = exp;
            }
        }
        wait();
    }
}
```

exponential.h

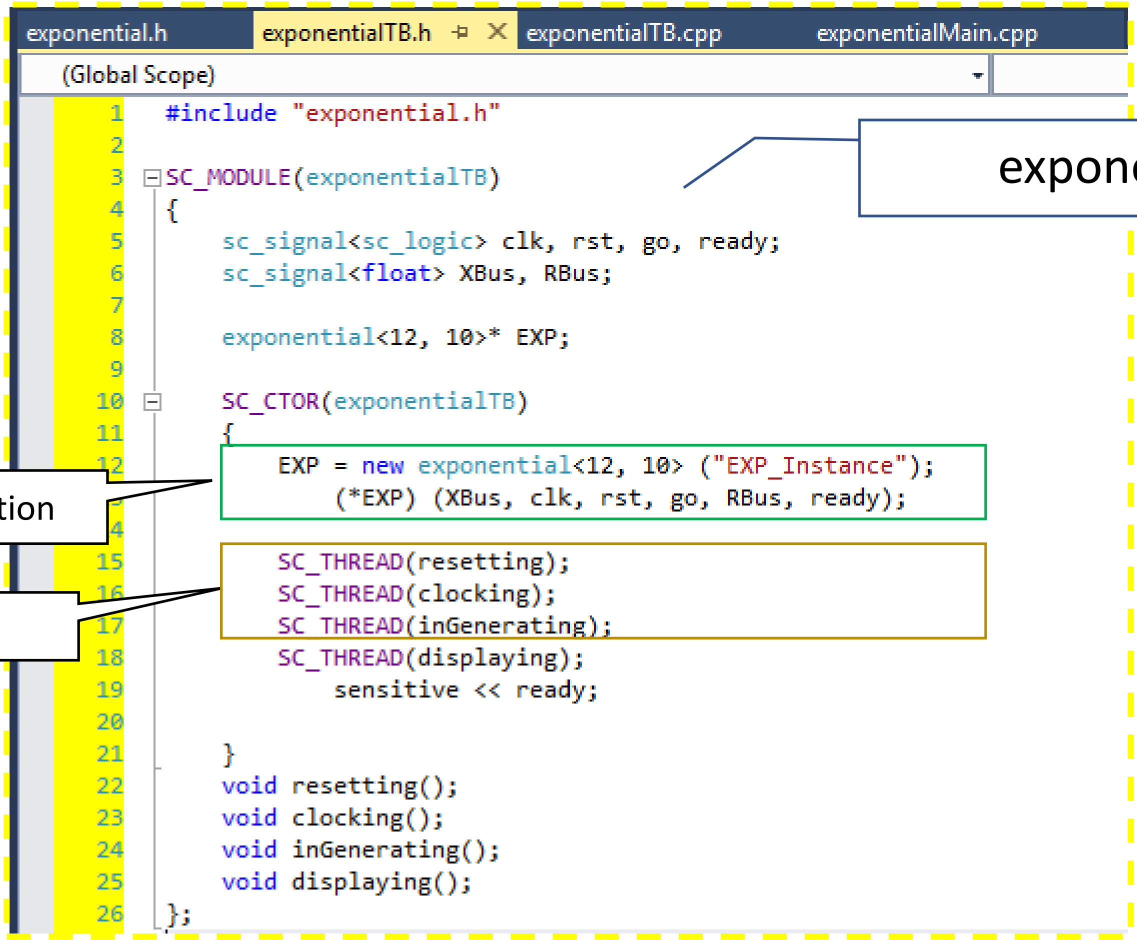
Variable declaration and initialization

Input data registration

Mimicking clock

Performing e^x function

Exponential



```
#include "exponential.h"

SC_MODULE(exponentialTB)
{
    sc_signal<sc_logic> clk, rst, go, ready;
    sc_signal<float> XBus, RBus;

    exponential<12, 10>* EXP;

    SC_CTOR(exponentialTB)
    {
        EXP = new exponential<12, 10> ("EXP_Instance");
            (*EXP) (XBus, clk, rst, go, RBus, ready);

        SC_THREAD(resetting);
        SC_THREAD(clocking);
        SC_THREAD(inGenerating);
        SC_THREAD(displaying);
            sensitive << ready;

    }
    void resetting();
    void clocking();
    void inGenerating();
    void displaying();
};
```

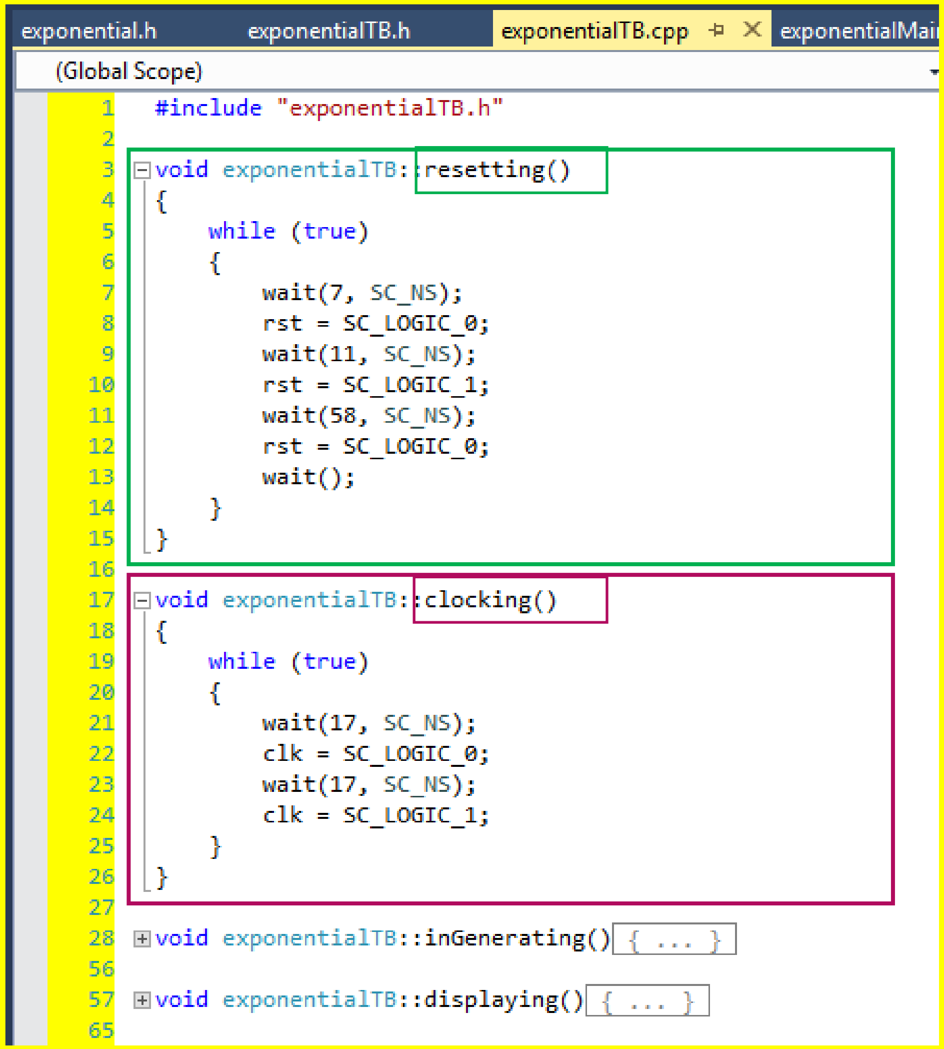
exponentialTB.h

Exponential instantiation

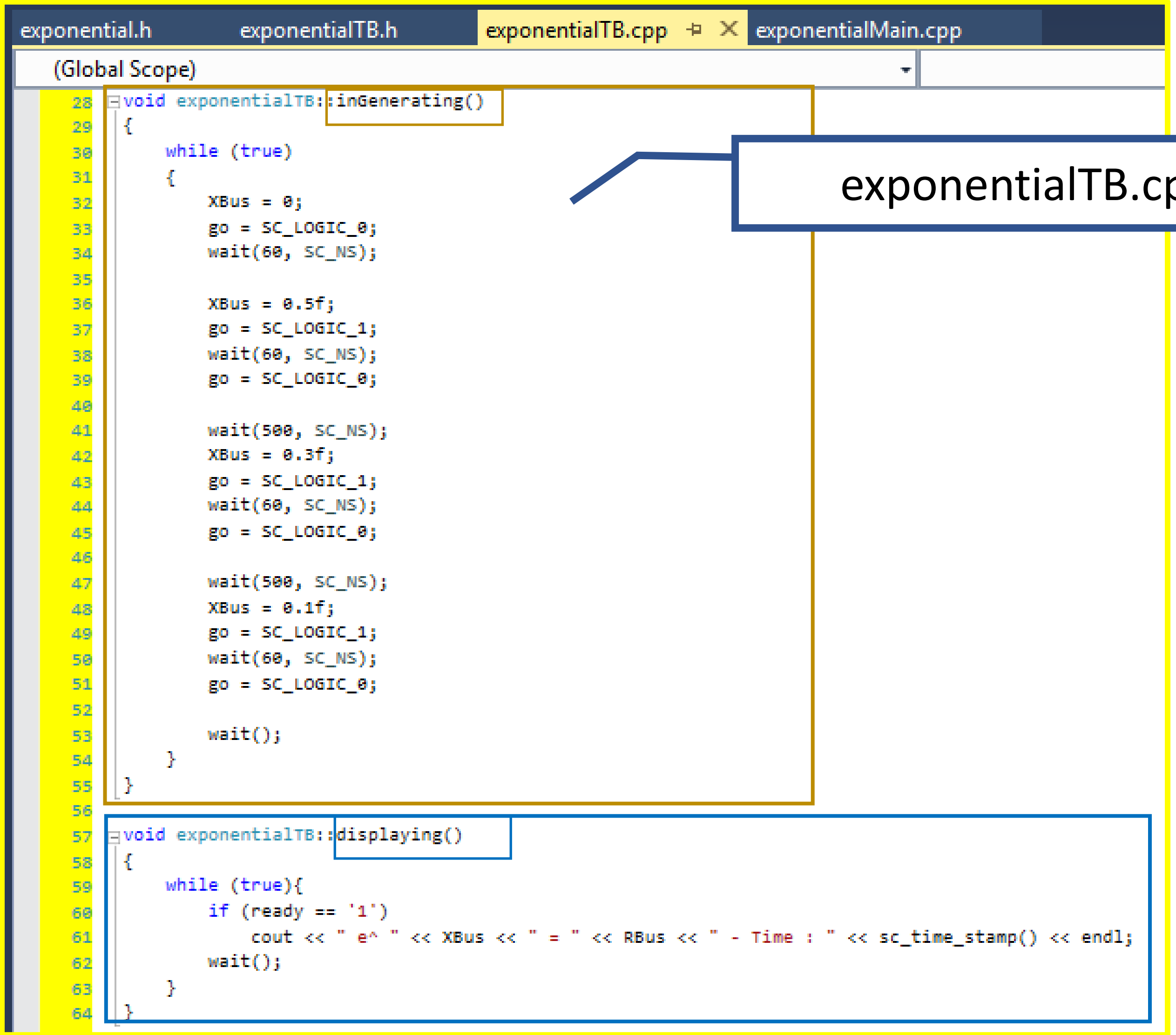
Input generation

SystemC Functional Design - Example

Exponential



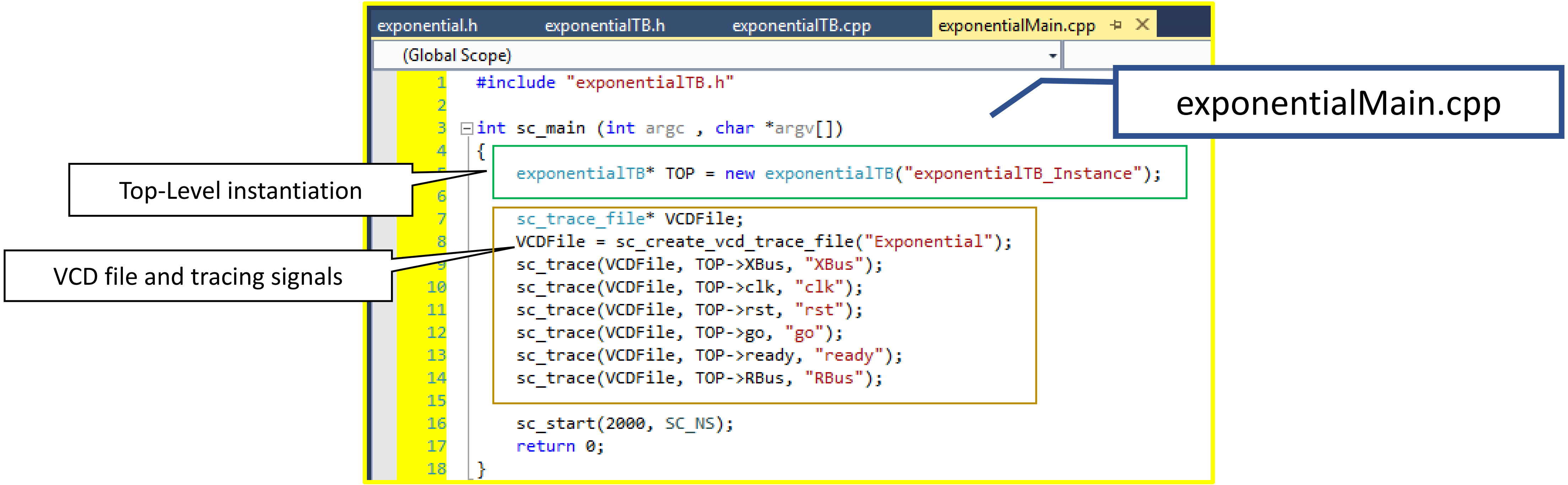
```cpp
#include "exponentialTB.h"

void exponentialTB::resetting()
{
    while (true)
    {
        wait(7, SC_NS);
        rst = SC_LOGIC_0;
        wait(11, SC_NS);
        rst = SC_LOGIC_1;
        wait(58, SC_NS);
        rst = SC_LOGIC_0;
        wait();
    }
}

void exponentialTB::clocking()
{
    while (true)
    {
        wait(17, SC_NS);
        clk = SC_LOGIC_0;
        wait(17, SC_NS);
        clk = SC_LOGIC_1;
    }
}

void exponentialTB::inGenerating(){ ... }

void exponentialTB::displaying(){ ... }
```



```cpp
void exponentialTB::inGenerating()
{
    while (true)
    {
        XBus = 0;
        go = SC_LOGIC_0;
        wait(60, SC_NS);

        XBus = 0.5f;
        go = SC_LOGIC_1;
        wait(60, SC_NS);
        go = SC_LOGIC_0;

        wait(500, SC_NS);
        XBus = 0.3f;
        go = SC_LOGIC_1;
        wait(60, SC_NS);
        go = SC_LOGIC_0;

        wait(500, SC_NS);
        XBus = 0.1f;
        go = SC_LOGIC_1;
        wait(60, SC_NS);
        go = SC_LOGIC_0;

        wait();
    }
}

void exponentialTB::displaying()
{
    while (true){
        if (ready == '1')
            cout << " e^ " << XBus << " = " << RBus << " - Time : " << sc_time_stamp() << endl;
        wait();
    }
}
```

exponentialTB.cpp

Exponential



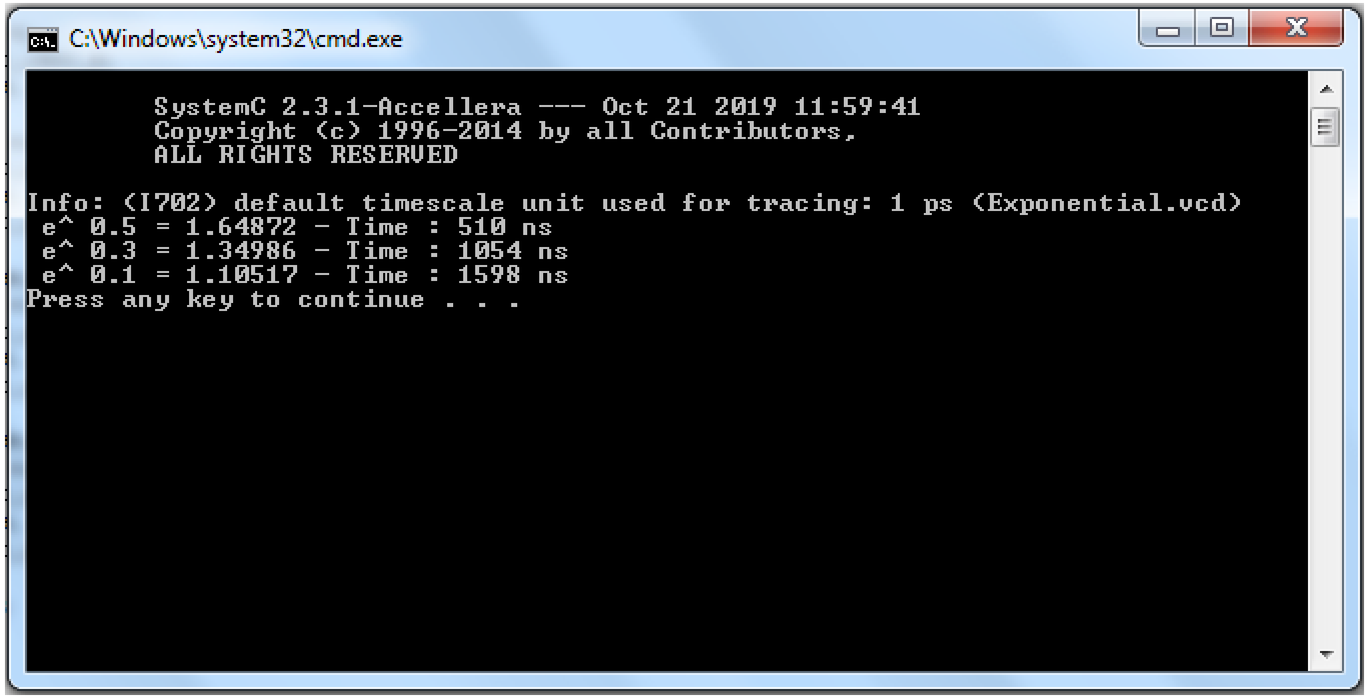
```
#include "exponentialTB.h"

int sc_main (int argc , char *argv[])
{
    exponentialTB* TOP = new exponentialTB("exponentialTB_Instance");

    sc_trace_file* VCDFile;
    VCDFile = sc_create_vcd_trace_file("Exponential");
    sc_trace(VCDFile, TOP->XBus, "XBus");
    sc_trace(VCDFile, TOP->clk, "clk");
    sc_trace(VCDFile, TOP->rst, "rst");
    sc_trace(VCDFile, TOP->go, "go");
    sc_trace(VCDFile, TOP->ready, "ready");
    sc_trace(VCDFile, TOP->RBus, "RBus");

    sc_start(2000, SC_NS);
    return 0;
}
```

exponentialMain.cpp

Top-Level instantiation

VCD file and tracing signals

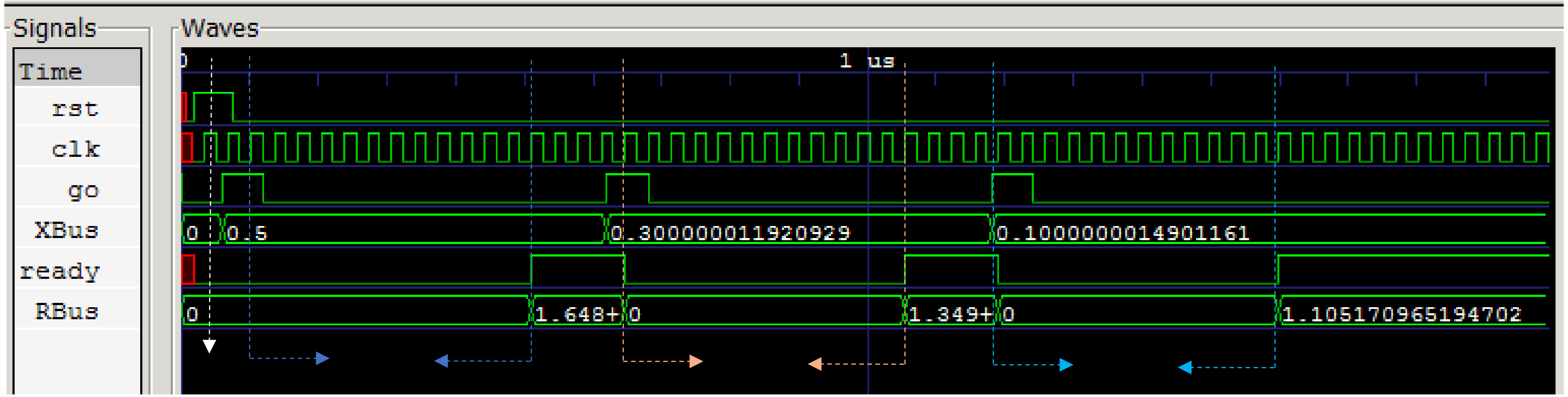
Exponential



```
        SystemC 2.3.1-Accellera --- Oct 21 2019 11:59:41
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED

Info: (I702) default timescale unit used for tracing: 1 ps (Exponential.vcd)
 e^ 0.5 = 1.64872 - Time : 510 ns
 e^ 0.3 = 1.34986 - Time : 1054 ns
 e^ 0.1 = 1.10517 - Time : 1598 ns
Press any key to continue . . .
```

Exponential

- Waveform in a VCD viewer



SystemC Modeling

- + Taking Off From C++
- + SystemC Gate-Level Modeling
- Utilities for HDL Orientation
- + Timing & SystemC RT-Level Modeling
- + Components for RTL Design
- + SystemC RTL Design - Example
- SystemC Functional Modeling
- + SystemC Functional Design - Example
- Summary

Summary

- Taking Off From C++
  - C++ modeling of 1-bit Adder
- SystemC Gate-level Modeling
  - SystemC modeling of 1-bit Adder

Utilities for HDL Orientation

- Timing & SystemC RT-Level Modeling
  - Hierarchical timed design for Serial Adder
- Components for RTL Design
  - Combinational
  - Sequential
- SystemC RTL Design - Examples

  Sequence Detector 11011

  A configurable Memory

  Exponential Circuit

SystemC Functional Modeling

- SystemC Functional Design - Example

  Multiplier

  Divider

  Exponential Circuit