

Chapter 4

RT Level SystemC

Zainalabedin Navabi

Slides prepared by: Hanieh Hashemi

© 2015-2016, Zainalabedin Navabi - RTL SystemC Modeling

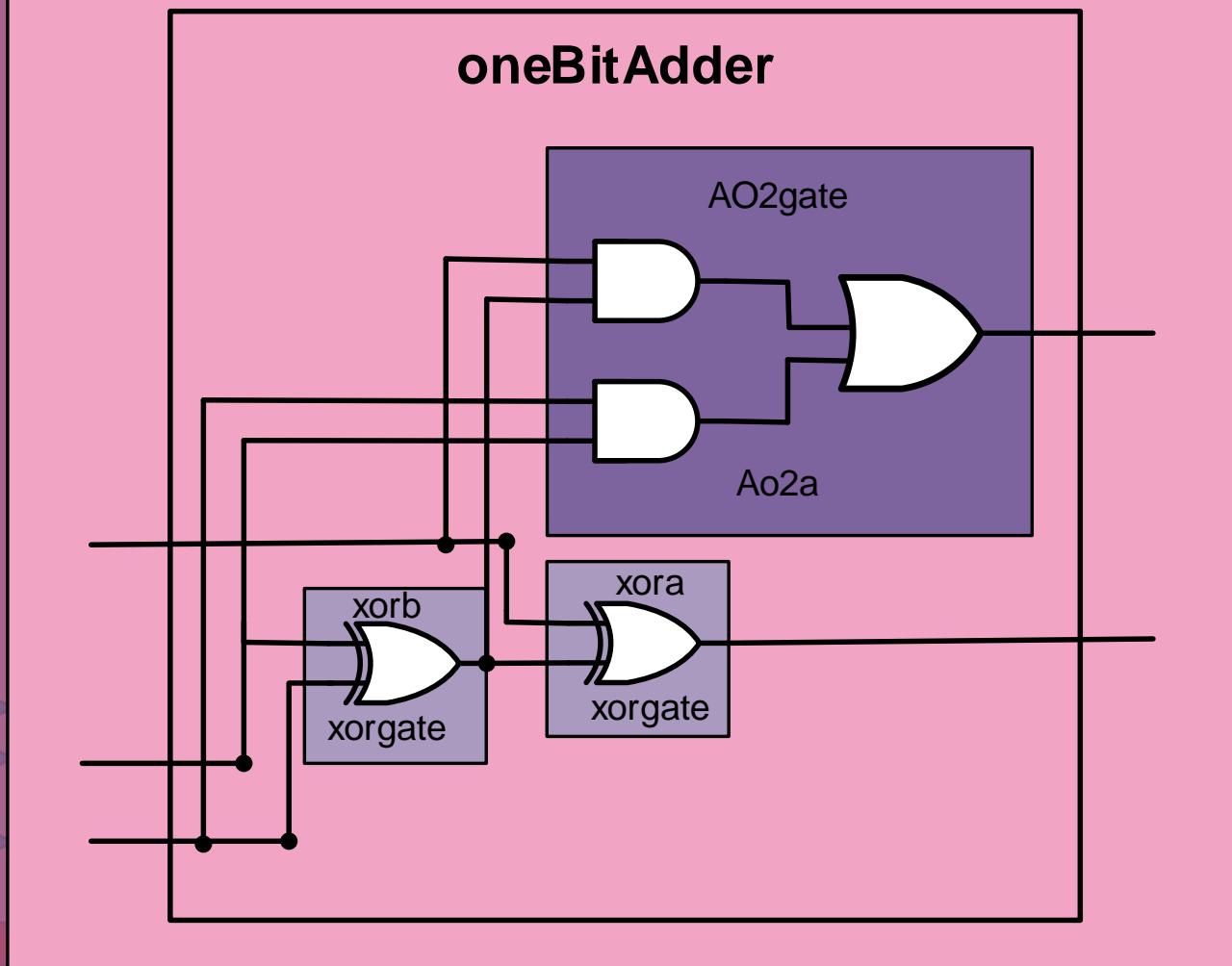
RT Level SystemC

- Taking Off From C++
- SystemC Modeling
- Simulation Environment
- Utilities for HDL Orientation
- Sequential Modeling and Timing
- SystemC FSM Modeling
- Components for RT Level Design
- A configurable Memory

RT Level SystemC

- Taking Off From c++
- SystemC Modeling
- Simulation Environment
- Utilities for HDL Orientation
- Sequential Modeling and Timing
- SystemC FSM Modeling
- Components for RT Level Design
- A configurable Memory

Hierarchical Structure Hardware



Low Level Component Functional models

All ports of our module
are bus pointers.

```
#include "classVectorPrimitives.h"
#include <iostream>
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 eval();
};

class AO2gate { ... };

class oneBitAdder {
    bus *i1, *i2, *i3,
        *o1, *o2;
    XORgate* XORA;
    XORgate* XORB;
    AO2gate* AO2a;

    bus x1;

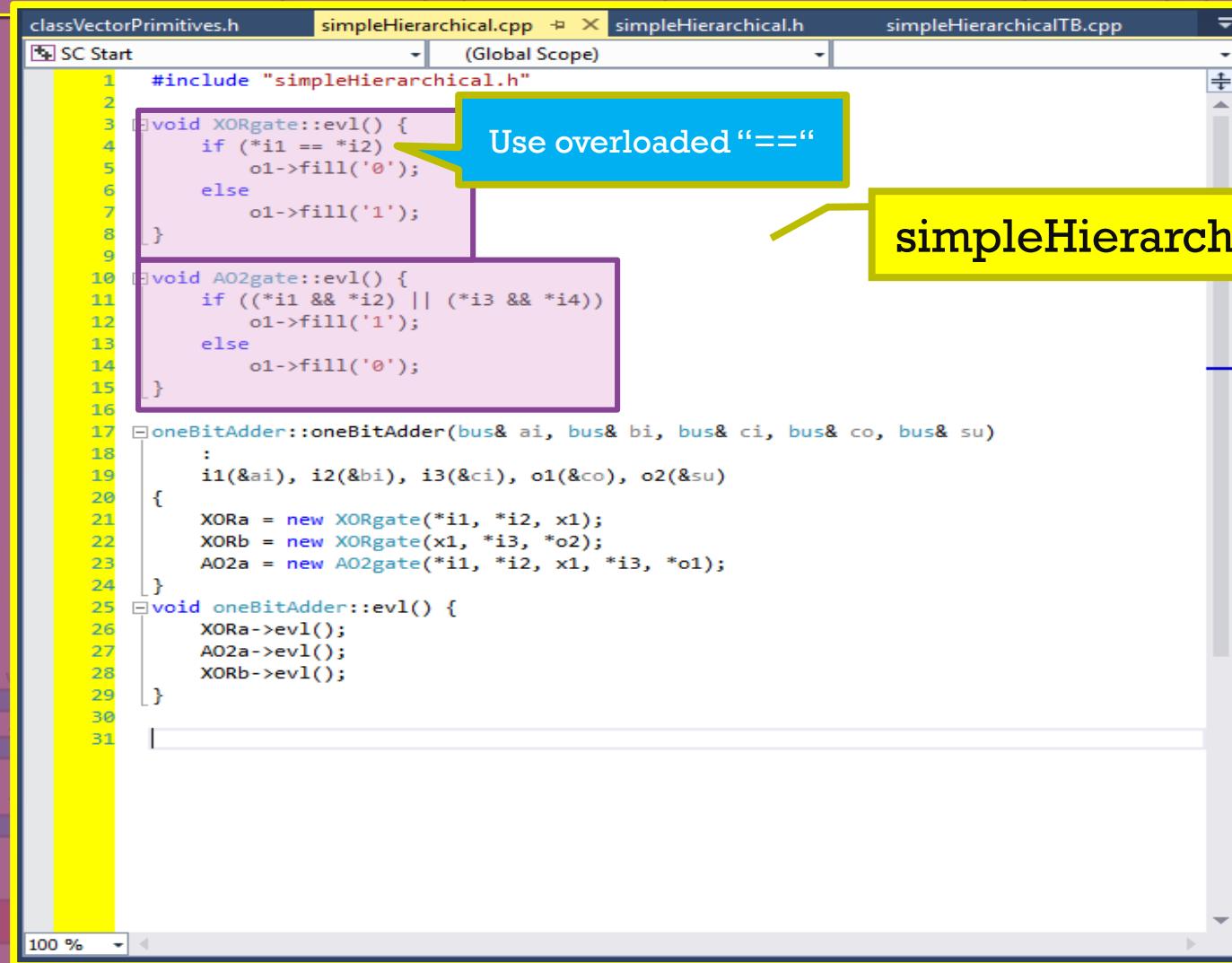
public:
    oneBitAdder(bus& ai, bus& bi, bus& ci, bus& co, bus& su);
    ~oneBitAdder();
    void eval();
};

SC Start (Global Scope)
```

simpleHierarchical.h

Resembles standard
hardware description but
lacks the concurrency

Low Level Component Functional models

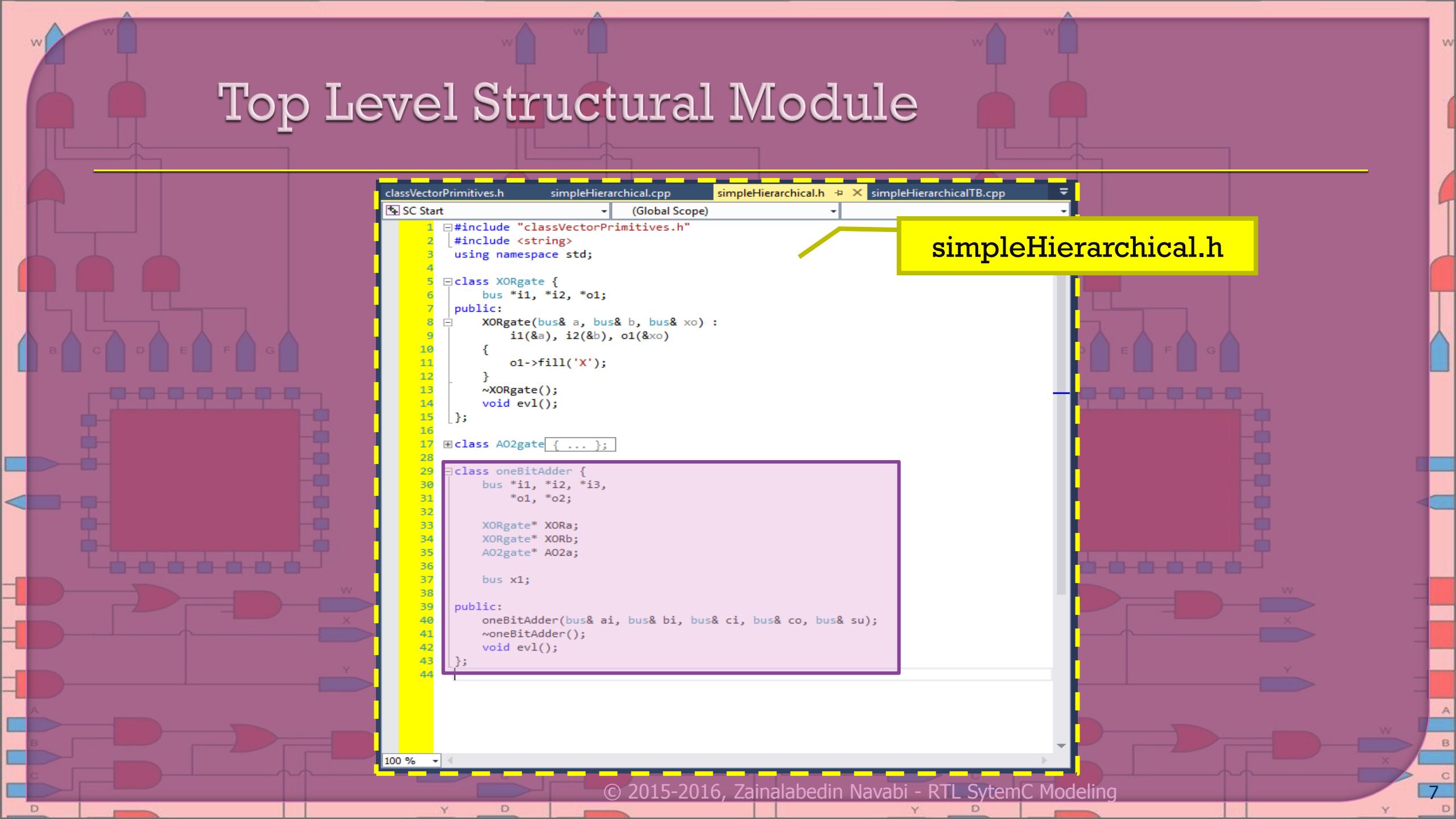


The screenshot shows a SystemC IDE interface with four tabs: classVectorPrimitives.h, simpleHierarchical.cpp, simpleHierarchical.h, and simpleHierarchicalTB.cpp. The simpleHierarchical.cpp tab is active, displaying the following code:

```
classVectorPrimitives.h          simpleHierarchical.cpp      simpleHierarchical.h      simpleHierarchicalTB.cpp
SC Start                         (Global Scope)           SC Start
1 #include "simpleHierarchical.h"
2
3 void XORgate::eval() {
4     if (*i1 == *i2)
5         o1->fill('0');
6     else
7         o1->fill('1');
8 }
9
10 void AO2gate::eval() {
11     if ((*i1 && *i2) || (*i3 && *i4))
12         o1->fill('1');
13     else
14         o1->fill('0');
15 }
16
17 oneBitAdder::oneBitAdder(bus& ai, bus& bi, bus& ci, bus& co, bus& su)
18     :
19     i1(&ai), i2(&bi), i3(&ci), o1(&co), o2(&su)
20 {
21     XORa = new XORgate(*i1, *i2, x1);
22     XORb = new XORgate(x1, *i3, *o2);
23     AO2a = new AO2gate(*i1, *i2, x1, *i3, *o1);
24 }
25
26 void oneBitAdder::eval() {
27     XORa->eval();
28     AO2a->eval();
29     XORb->eval();
30 }
31
```

A yellow callout box points to the line `if (*i1 == *i2)` with the text "Use overloaded ‘==’". Another yellow callout box points to the file tab "simpleHierarchical.cpp" with the text "simpleHierarchical.cpp".

Top Level Structural Module



```
#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 { ... };

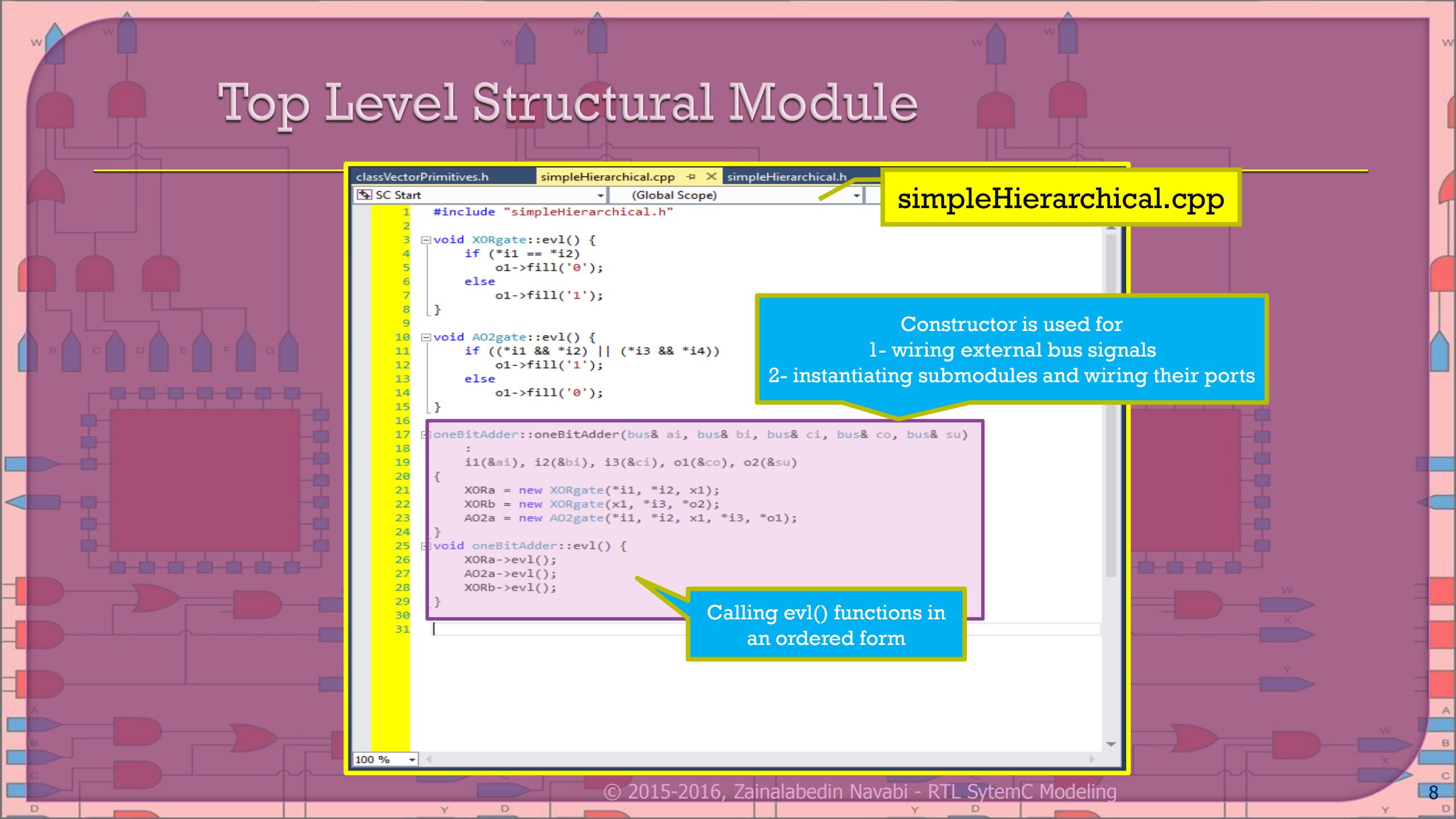
class oneBitAdder {
    bus *i1, *i2, *i3,
        *o1, *o2;
    XORgate* XORA;
    XORgate* XORB;
    AO2gate* AO2a;

    bus x1;

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

simpleHierarchical.h

Top Level Structural Module



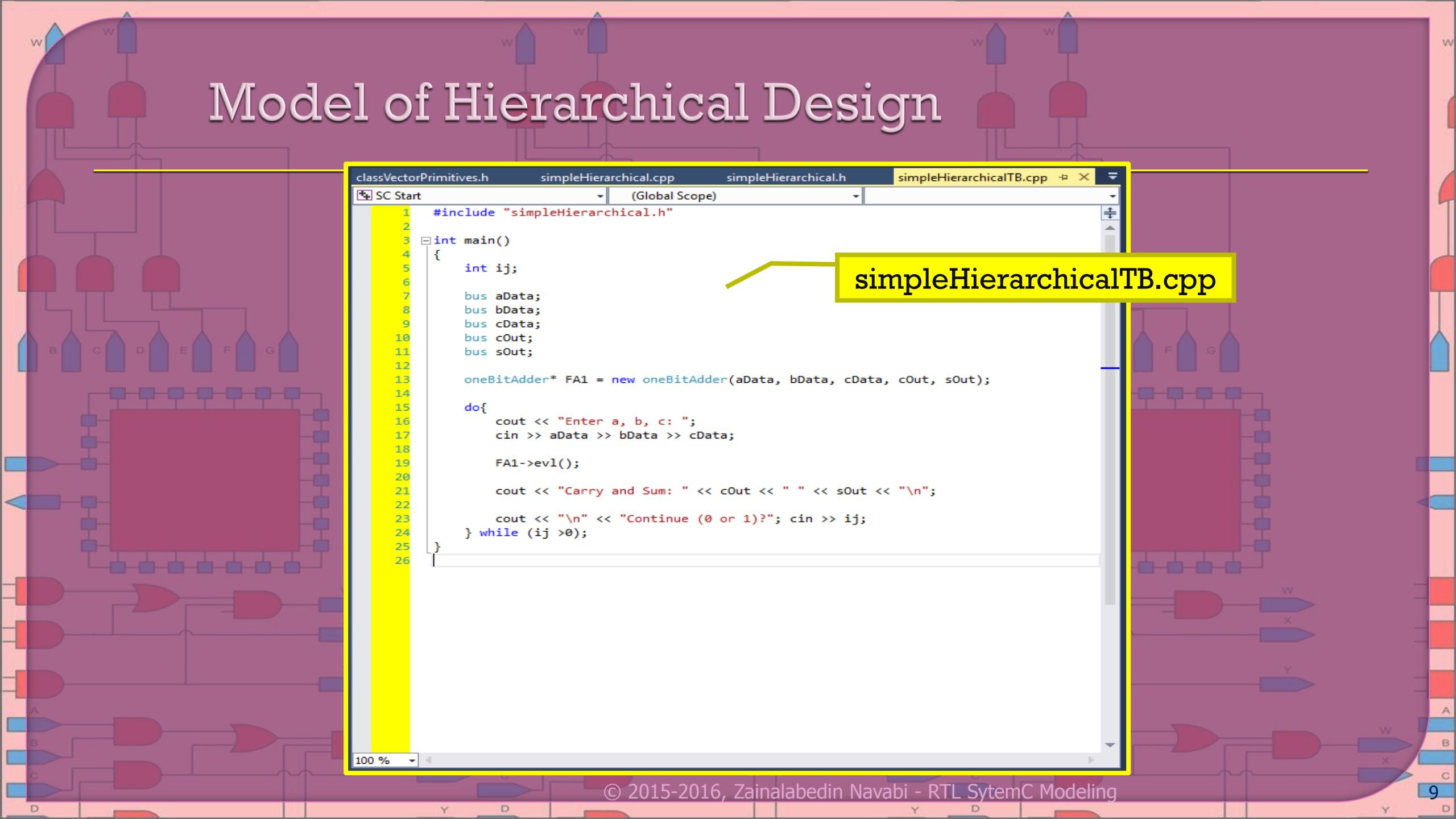
```
classVectorPrimitives.h    simpleHierarchical.cpp  simpleHierarchical.h
SC Start                  (Global Scope)
1 #include "simpleHierarchical.h"
2
3 void XORgate::evl() {
4     if (*i1 == *i2)
5         o1->fill('0');
6     else
7         o1->fill('1');
8 }
9
10 void AO2gate::evl() {
11     if ((*i1 && *i2) || (*i3 && *i4))
12         o1->fill('1');
13     else
14         o1->fill('0');
15 }
16
17 oneBitAdder::oneBitAdder(bus& ai, bus& bi, bus& ci, bus& co, bus& su)
18 :
19     i1(&ai), i2(&bi), i3(&ci), o1(&co), o2(&su)
20 {
21     XORa = new XORgate(*i1, *i2, x1);
22     XORb = new XORgate(x1, *i3, *o2);
23     AO2a = new AO2gate(*i1, *i2, x1, *i3, *o1);
24 }
25 void oneBitAdder::evl() {
26     XORa->evl();
27     AO2a->evl();
28     XORb->evl();
29 }
30
31
100 %
```

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

Model of Hierarchical Design



The screenshot shows a software interface for SystemC modeling. A central window displays a C++ code editor with the file `simpleHierarchicalTB.cpp` open. The code implements a one-bit adder and a testbench loop. A yellow callout box points from the text "simpleHierarchicalTB.cpp" to the code editor.

```
#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);
}
```

RT Level SystemC

- Taking Off From c++
- SystemC Modeling
- Simulation Environment
- Utilities for HDL Orientation
- Sequential Modeling and Timing
- SystemC FSM Modeling
- Components for RT Level Design
- A configurable Memory

sc_module

simpleHierarchicalTB.cpp simpleHierarchical.cpp simpleHierarchical.h

```
#include <systemc.h>
class XORgate : 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;

    SC_CTOR(XORgate)
    {
        SC_METHOD(ev1);
        sensitive << i1 << i2;
    }
    void ev1();
};

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);
};
```

sc_port, sc_signal, sc_logic

Mechanism for accessing methods of the channel given by sc_port

Channel communication and type of data

simpleHierarchical.h

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

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

```
simpleHierarchicalTB.cpp simpleHierarchical.cpp simpleHierarchical.h
SC Start (Global Scope)
1 #include <systemc.h>
2
3 class XORgate : public sc_module {
4 public:
5     sc_port<sc_signal_in_if<sc_logic>, 1> i1, i2;
6     sc_port<sc_signal_out_if<sc_logic>, 1> o1;
7
8     SC_CTOR(XORgate)
9     {
10        SC_METHOD(ev1);
11        sensitive << i1 << i2;
12    }
13    void ev1();
14
15    class A02gate { ... };
16
17    class oneBitAdder : public sc_module {
18    public:
19        sc_port<sc_signal_in_if<sc_logic>, 1> i1, i2, i3;
20        sc_port<sc_signal_out_if<sc_logic>, 1> o1, o2;
21
22        sc_signal<sc_logic> x1;
23
24        XORgate* XORa;
25        XORgate* XORb;
26        A02gate* A02a;
27
28        SC_HAS_PROCESS(oneBitAdder);
29        oneBitAdder(sc_module_name);
30    };
31
32
33
34
35
36
37
38
39
40
41
42
43 }
```

Module Inline Constructor

Evl function
is registered
as sc_method

Sc_ctocvr macro allows inline
definition of xorgate
constructorc

Evl() wakes up when an event
occurs on i1 & i2

simpleHierarchical.h

Sc_method also run once at the
beginning of the simulation

```
simpleHierarchicalTB.cpp simpleHierarchical.cpp simpleHierarchical.h
SC Start (Global Scope)
1 #include <systemc.h>
2
3 class XORgate : public sc_module {
4 public:
5     sc_port<sc_signal_in_if<sc_logic>> i1;
6     sc_port<sc_signal_out_if<sc_logic>> o1;
7
8     SC_CTOR(XORgate)
9     {
10        SC_METHOD(evl);
11        sensitive << i1 << i2;
12    }
13    void evl();
14
15
16 class AO2gate { ... };
17
18 class oneBitAdder : public sc_module {
19 public:
20     sc_port<sc_signal_in_if<sc_logic>> i1, i2, i3;
21     sc_port<sc_signal_out_if<sc_logic>> o1, o2;
22
23     sc_signal<sc_logic> x1;
24
25     XORgate* XORa;
26     XORgate* XORb;
27     AO2gate* AO2a;
28
29     SC_HAS_PROCESS(oneBitAdder);
30     oneBitAdder(sc_module_name);
31
32
33
34
35
36
37
38
39
40
41
42
43 }
```

sc_signal, sc_has_process

```
simpleHierarchicalTB.cpp simpleHierarchical.cpp simpleHierarchical.h
SC Start (Global Scope)

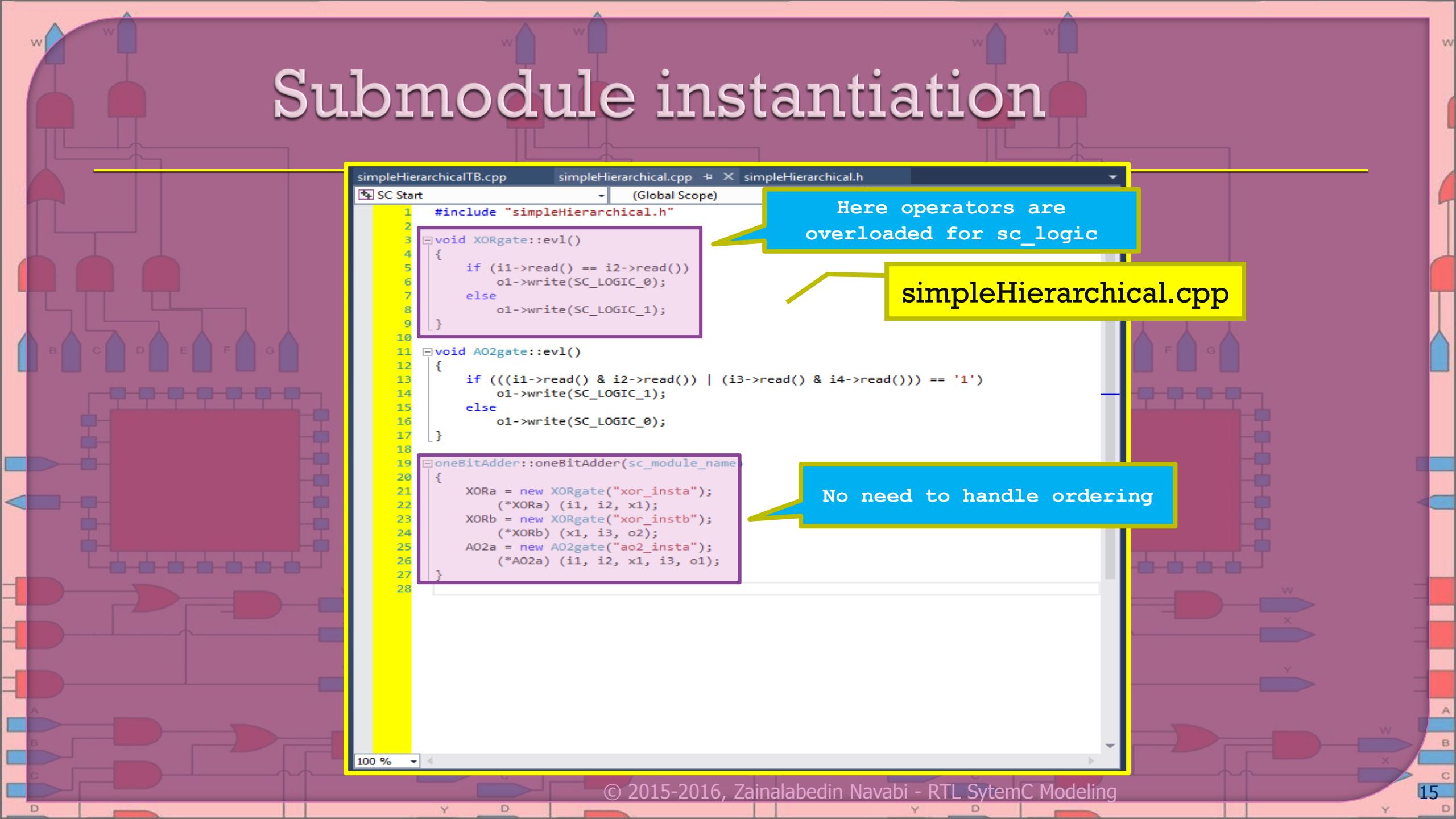
1 #include <systemc.h>
2
3 class XORgate : public sc_module {
4 public:
5     sc_port<sc_signal_in_if<sc_logic>, 1> i1, i2;
6     sc_port<sc_signal_out_if<sc_logic>, 1> o1;
7
8     SC_CTOR(XORgate)
9     {
10        SC_METHOD(ev1);
11        sensitive << i1 << i2;
12    }
13    void ev1();
14};
15
16 class AO2gate { ... };
17
18 class oneBitAdder : public sc_module {
19 public:
20     sc_port<sc_signal_in_if<sc_logic>, 1> i1, i2, i3;
21     sc_port<sc_signal_out_if<sc_logic>, 1> o1, o2;
22
23     sc_signal<sc_logic> x1;
24
25     XORgate* XORa;
26     XORgate* XORb;
27     AO2gate* AO2a;
28
29     SC_HAS_PROCESS(oneBitAdder);
30     oneBitAdder(sc_module_name);
31 };
32
33
34
35
36
37
38
39
40
41
42
43
```

Module ports that are bound to x1 have access to interfaces c

simpleHierarchical.h

This constructor can be separate from module declaration. Other arguments also can be passed.

Submodule instantiation



The background of the slide features a complex digital logic circuit diagram, likely a full adder or similar combinational logic, composed of AND, OR, and NOT gates.

```
simpleHierarchicalTB.cpp    simpleHierarchical.cpp  simpleHierarchical.h
SC Start  (Global Scope)
1 #include "simpleHierarchical.h"
2
3 void XORgate::eval()
4 {
5     if (i1->read() == i2->read())
6         o1->write(SC_LOGIC_0);
7     else
8         o1->write(SC_LOGIC_1);
9 }
10
11 void AO2gate::eval()
12 {
13     if (((i1->read() & i2->read()) | (i3->read() & i4->read())) == '1')
14         o1->write(SC_LOGIC_1);
15     else
16         o1->write(SC_LOGIC_0);
17 }
18
19 oneBitAdder::oneBitAdder(sc_module_name)
20 {
21     XORa = new XORgate("xor_insta");
22     (*XORa) (i1, i2, x1);
23     XORb = new XORgate("xor_instb");
24     (*XORb) (x1, i3, o2);
25     AO2a = new AO2gate("ao2_insta");
26     (*AO2a) (i1, i2, x1, i3, o1);
27 }
```

Here operators are overloaded for sc_logic

simpleHierarchical.cpp

No need to handle ordering

RT Level SystemC

- Taking Off From c++
- SystemC Modeling
- Simulation Environment
- Utilities for HDL Orientation
- Sequential Modeling and Timing
- SystemC FSM Modeling
- Components for RT Level Design
- A configurable Memory

sc_main

The screenshot shows a SystemC IDE interface with three tabs: simpleHierarchicalTB.cpp, simpleHierarchical.cpp, and simpleHierarchical.h. The simpleHierarchicalTB.cpp tab is active, displaying the following code:

```
1 #include "simpleHierarchical.h"
2
3 int sc_main(int argc, char **argv)
4 {
5     sc_signal<sc_logic> aData;
6     sc_signal<sc_logic> bData;
7     sc_signal<sc_logic> cData;
8     sc_signal<sc_logic> cOut;
9     sc_signal<sc_logic> sOut;
10
11     oneBitAdder* FA1 = new oneBitAdder("FA1_instance");
12         (*FA1) (aData, bData, cData, cOut, sOut);
13
14     sc_trace_file* VCDFile;
15     VCDFile = sc_create_vcd_trace_file("simpleHierarchical");
16         sc_trace(VCDFile, aData, "aIn");
17         sc_trace(VCDFile, bData, "bIn");
18         sc_trace(VCDFile, cData, "carryIn");
19         sc_trace(VCDFile, cOut, "carryOut");
20         sc_trace(VCDFile, sOut, "sumOut");
21
22     ...
23
24     return 0;
25 }
```

A yellow callout box points to the code with the text "simpleHierarchicalTB.cpp". Another yellow callout box points to the "sc_trace" and "sc_create_vcd_trace_file" calls with the text "VCD file and tracing signals".

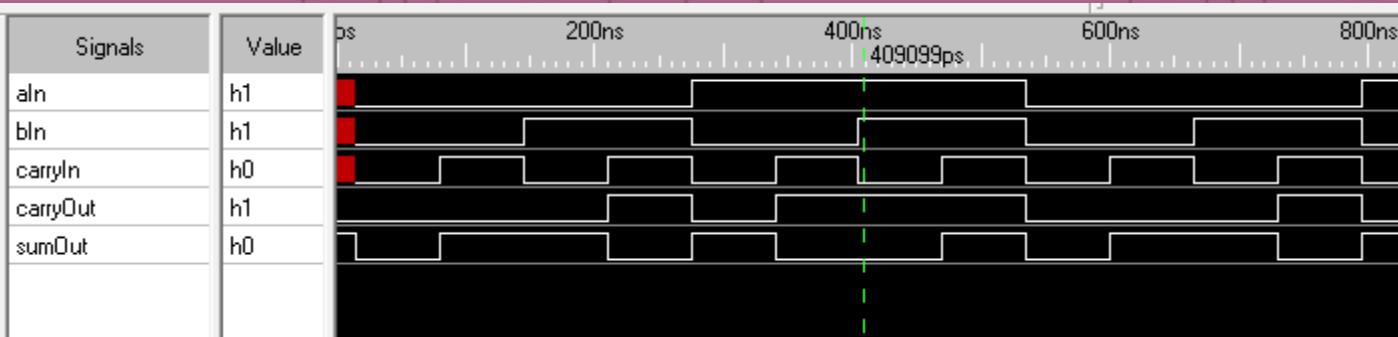
Simulation Run Timing

```
21 sc_int<3> intData;
22 sc_lv<3> abcData;
23
24 sc_start(15, SC_NS);
25
26 intData = 0;
27 int ij=0;
28 do {
29     abcData = intData;
30     aData = abcData[2];
31     bData = abcData[1];
32     cData = abcData[0];
33     sc_start(15, SC_NS);
34     intData = intData + 1;
35     sc_start(50, SC_NS);
36 } while (++ij < 40);
37
38 sc_start(100,SC_NS);
39
40
```

simpleHierarchicalTB.cpp

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

Waveform in VCD



RT Level SystemC

- Taking Off From c++
- SystemC Modeling
- Simulation Environment
- Utilities for HDL Orientation
- Sequential Modeling and Timing
- SystemC FSM Modeling
- Components for RT Level Design
- A configurable Memory

sc_module, sc_in, sc_out

Only valid for
sc_signal

simpleHierarchical.h

Port
association by
position

```
simpleHierarchical.cpp simpleHierarchical.h ✘ X
SC_Next (Global Scope)
1 #include <systemc.h>
2
3 SC_MODULE(XORgate)
4 {
5     sc_in<sc_logic> i1, i2;
6     sc_out<sc_logic> o1;
7
8     SC_CTOR(XORgate)
9     {
10        SC_METHOD(ev1);
11        sensitive << i1 << i2;
12    }
13    void ev1();
14}
15
16 SC_MODULE(AO2gate) { ... }
17
18 SC_MODULE(oneBitAdder)
19 {
20     sc_in<sc_logic> i1, i2, i3;
21     sc_out<sc_logic> o1, o2;
22
23     sc_signal<sc_logic> x1;
24
25     XORgate* XORa;
26     XORgate* XORb;
27     AO2gate* AO2a;
28
29     SC_CTOR(oneBitAdder)
30     {
31         XORa = new XORgate("xor_insta");
32         (*XORa) (i1, i2, x1);
33         XORb = new XORgate("xor_instb");
34         (*XORb) (x1, i3, o2);
35         AO2a = new AO2gate("ao2_insta");
36         (*AO2a) (i1, i2, x1, i3, o1);
37     }
38
39
40
41
42
43
44
45
46
47
48
49
50
```

Evl()

```
simpleHierarchical.cpp  X  simpleHierarchical.h
SC Next  (Global Scope)

1 #include "simpleHierarchical.h"
2
3 class XORgate {
4 public:
5     void evl();
6 };
7
8 class AO2gate {
9 public:
10    void evl();
11 };
12
13
14
15
16
17
18
19
20
```

simpleHierarchical.cpp

RT Level SystemC

- Taking Off From c++
- SystemC Modeling
- Simulation Environment
- Utilities for HDL Orientation
- Sequential Modeling and Timing
- SystemC FSM Modeling
- Components for RT Level Design
- A configurable Memory

Gates with propagation delay

Use `sc_has_process` for passing delay argument

```
serialAddingTB.h    simpleHierarchical.h  simpleHierarchical.cpp  serialAddingTB.cpp
Serial Adding      (Global Scope)          (Global Scope)        (Global Scope)
1 #include <systemc.h>
2
3 SC_MODULE(XORgate)
4 {
5     sc_in<sc_logic> i1, i2;
6     sc_out<sc_logic> o1;
7
8     sc_time Td;
9     SC_HAS_PROCESS(XORgate);
10    XORgate::XORgate(sc_module_name, sc_time delay)
11    {
12        Td = delay;
13        SC_THREAD(evl);
14        sensitive << i1 << i2;
15    }
16    void evl();
17}
18
19 SC_MODULE(AO2gate) { ... }
20
21 SC_MODULE(oneBitAdder)
22 {
23     sc_in<sc_logic> i1, i2, i3;
24     sc_out<sc_logic> o1, o2; // Carry, Sum
25
26     sc_signal<sc_logic> x1;
27
28     XORgate* XORa;
29     XORgate* XORb;
30     AO2gate* AO2a;
31
32     SC_CTOR(oneBitAdder)
33     {
34         XORa = new XORgate("xor_insta", sc_time(0.5, SC_NS));
35         (*XORa) (i1, i2, x1);
36         XORb = new XORgate("xor_instb", sc_time(0.5, SC_NS));
37         (*XORb) (x1, i3, o2);
38         AO2a = new AO2gate("ao2_insta", sc_time(0.4, SC_NS));
39         (*AO2a) (i1, i2, x1, i3, o1);
40     }
41 }
```

simpleHierarchical.h

In order to use delay in the functionality `sc_thread` is used

`Sc_method` cannot be suspended temporarily by use of delay to wait

sc_thread

```
#include "simpleHierarchical.h"

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

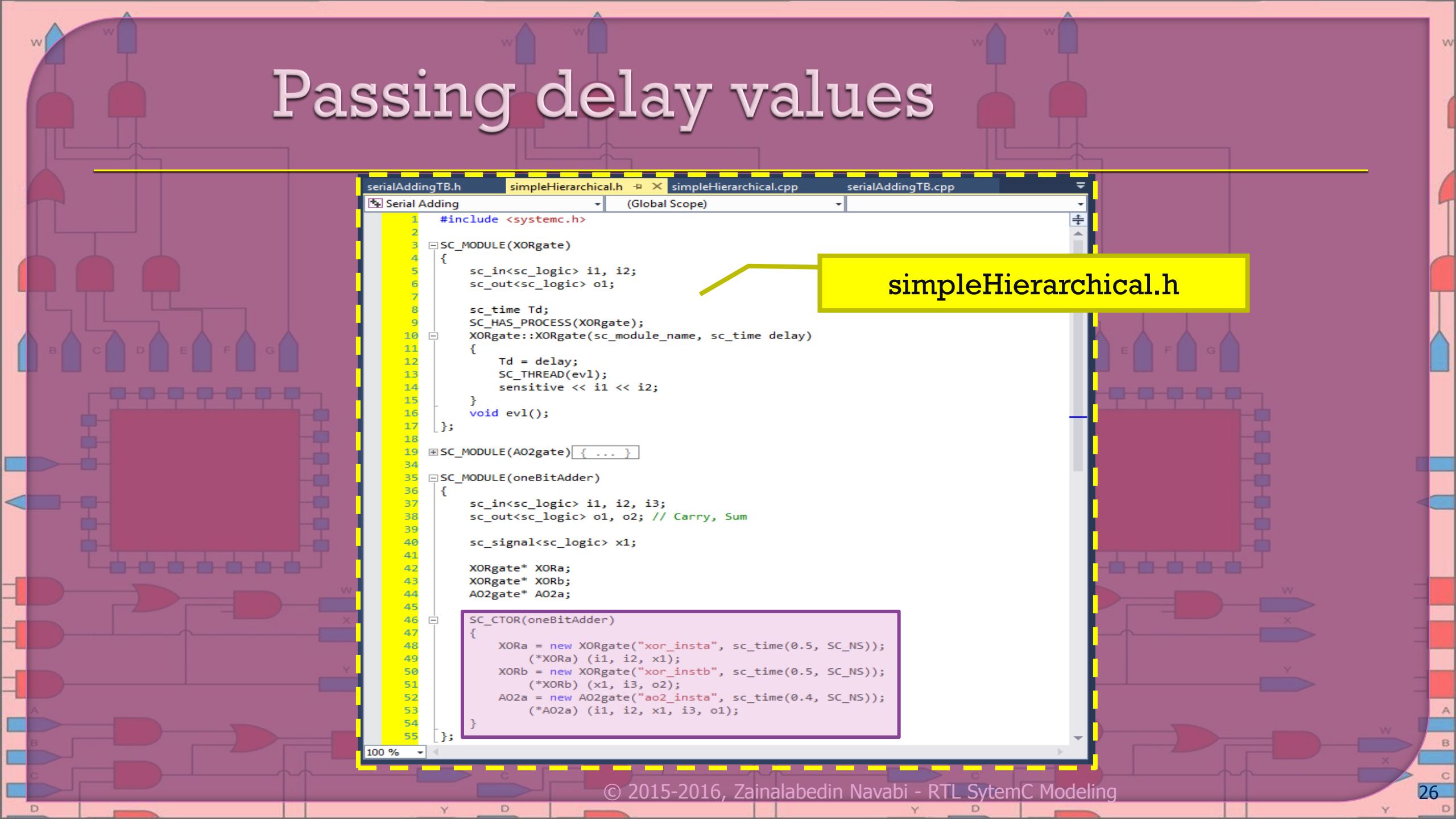
void AO2gate::eval() { ... }

void Dflipflop::eval()
{
    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();
    }
}
```

classVectorPrimitives.cpp

Suspend until the next
event on i1 & i2

Passing delay values



The image shows a digital circuit design in the background, featuring various logic gates like XOR, AND, and OR gates, along with wires and connections. In the foreground, there is a code editor window with four tabs: `serialAddingTB.h`, `simpleHierarchical.h`, `simpleHierarchical.cpp`, and `serialAddingTB.cpp`. The `simpleHierarchical.h` tab is highlighted with a yellow box.

```
#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

Clocking and flip flop declaration

```
serialAddingTB.h    simpleHierarchical.h  simpleHierarchical.cpp  serialAddingTB.cpp
Serial Adding      (Global Scope)

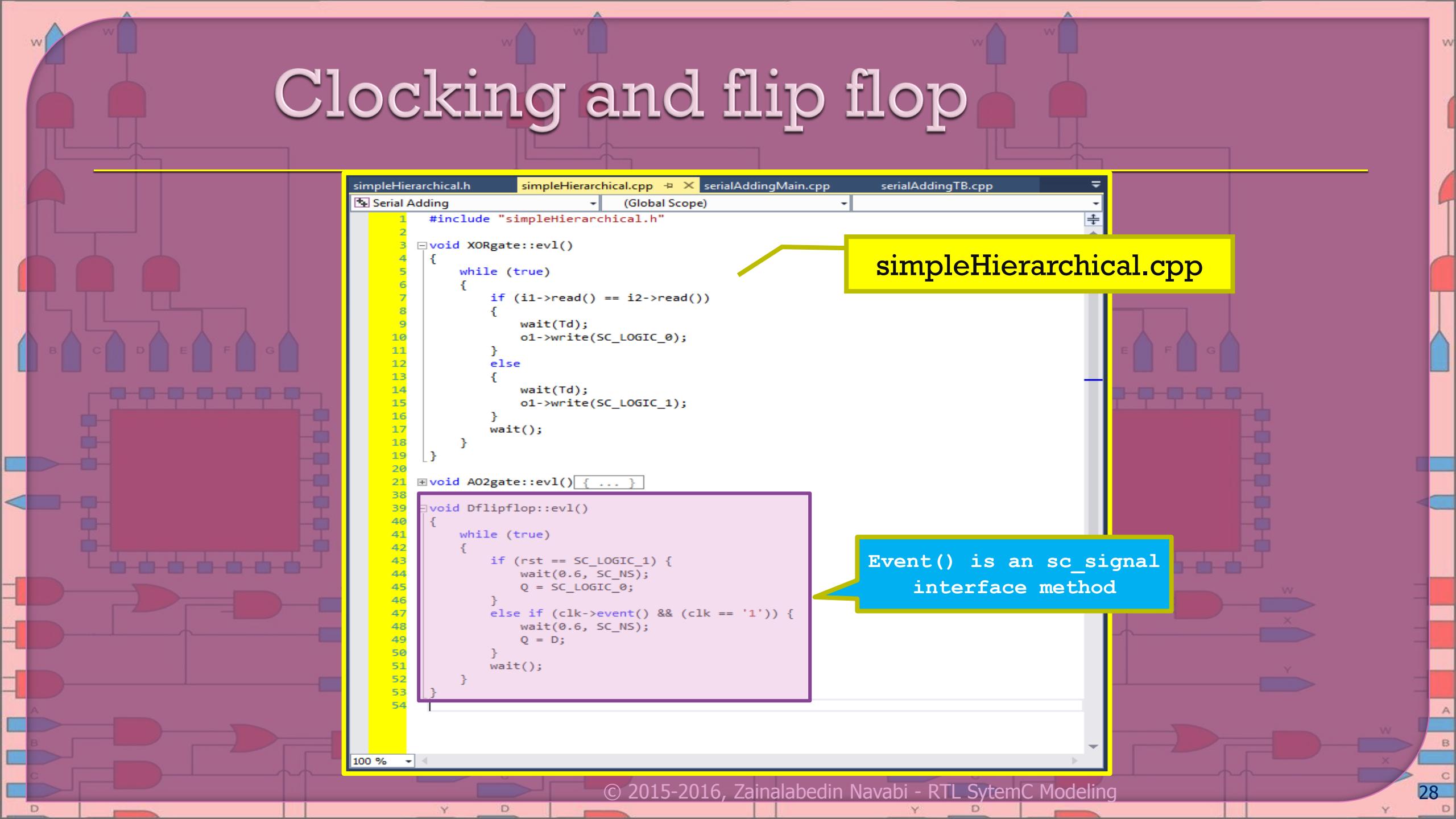
56
57     SC_MODULE(Dflipflop)
58     {
59         sc_in<sc_logic> clk, rst, D;
60         sc_out<sc_logic> Q;
61
62     SC_CTOR(Dflipflop)
63     {
64         SC_THREAD(ev1);
65         sensitive << clk << rst;
66     }
67     void ev1();
68 }
69
70
71     SC_MODULE(serialAdding)
72     {
73         sc_in<sc_logic> ain, bin, reset, clock;
74         sc_out<sc_logic> sum;
75
76         sc_signal<sc_logic> co, ci;
77
78         oneBitAdder* FA1;
79         Dflipflop* FF1;
80
81     SC_CTOR(serialAdding)
82     {
83         FA1 = new oneBitAdder("FA_instance");
84         FA1->i1(ain);
85         FA1->i2(bin);
86         FA1->i3(ci);
87         FA1->o1(co);
88         FA1->o2(sum);
89
90         FF1 = new Dflipflop("FF_instance");
91         FF1->clk(clock);
92         FF1->rst(reset);
93         FF1->D(co);
94         FF1->Q(ci);
95     }
96 }
```

simpleHierarchical.h

Sensitive to clk
and reset



Clocking and flip flop



```
simpleHierarchical.h    simpleHierarchical.cpp  serialAddingMain.cpp  serialAddingTB.cpp
Serial Adding          (Global Scope)          

1 #include "simpleHierarchical.h"
2
3 void XORgate::evl()
4 {
5     while (true)
6     {
7         if (i1->read() == i2->read())
8         {
9             wait(Td);
10            o1->write(SC_LOGIC_0);
11        }
12        else
13        {
14            wait(Td);
15            o1->write(SC_LOGIC_1);
16        }
17        wait();
18    }
19
20
21 void AO2gate::evl(){ ... }
22
23 void Dflipflop::evl()
24 {
25     while (true)
26     {
27         if (rst == SC_LOGIC_1) {
28             wait(0.6, SC_NS);
29             Q = SC_LOGIC_0;
30         }
31         else if (clk->event() && (clk == '1')) {
32             wait(0.6, SC_NS);
33             Q = D;
34         }
35         wait();
36     }
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
100 %
```

simpleHierarchical.cpp

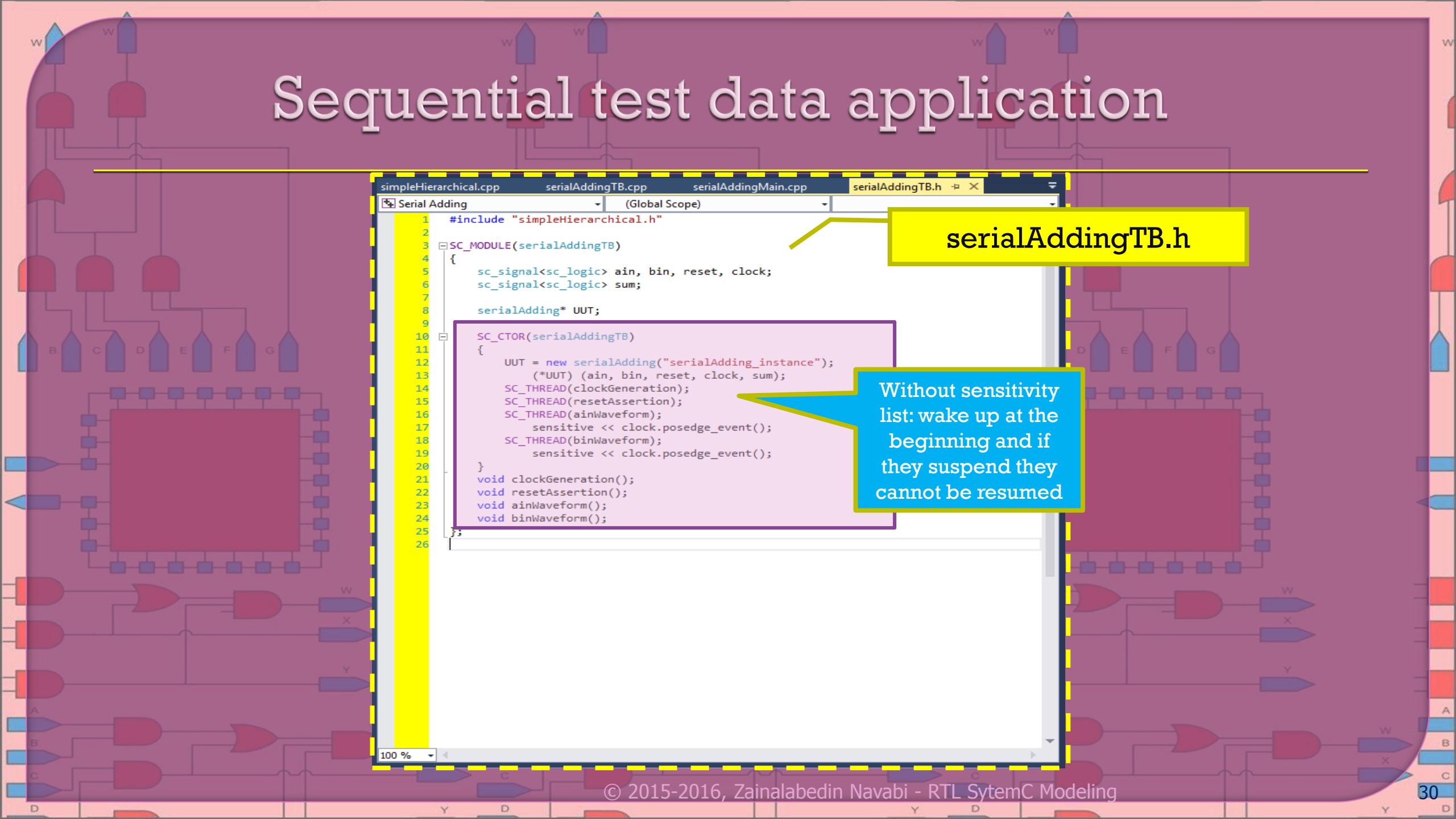
Event() is an sc_signal interface method

Clocking and flip flop declaration

```
serialAddingTB.h    simpleHierarchical.h  simpleHierarchical.cpp  serialAddingTB.cpp
Serial Adding      (Global Scope)

56
57 SC_MODULE(Dflipflop)
58 {
59     sc_in<sc_logic> clk, rst, D;
60     sc_out<sc_logic> Q;
61
62     SC_CTOR(Dflipflop)
63     {
64         SC_THREAD(ev1);
65         sensitive << clk << rst;
66     }
67     void ev1();
68 }
69
70
71 SC_MODULE(serialAdding)
72 {
73     sc_in<sc_logic> ain, bin, reset, clock;
74     sc_out<sc_logic> sum;
75
76     sc_signal<sc_logic> co, ci;
77
78     oneBitAdder* FA1;
79     Dflipflop* FF1;
80
81     SC_CTOR(serialAdding)
82     {
83         FA1 = new oneBitAdder("FA_instance");
84         FA1->i1(ain);
85         FA1->i2(bin);
86         FA1->i3(ci);
87         FA1->o1(co);
88         FA1->o2(sum);
89         FF1 = new Dflipflop("FF_instance");
90         FF1->clk(clock);
91         FF1->rst(reset);
92         FF1->D(co);
93         FF1->Q(ci);
94     }
95 }
```

Sequential test data application

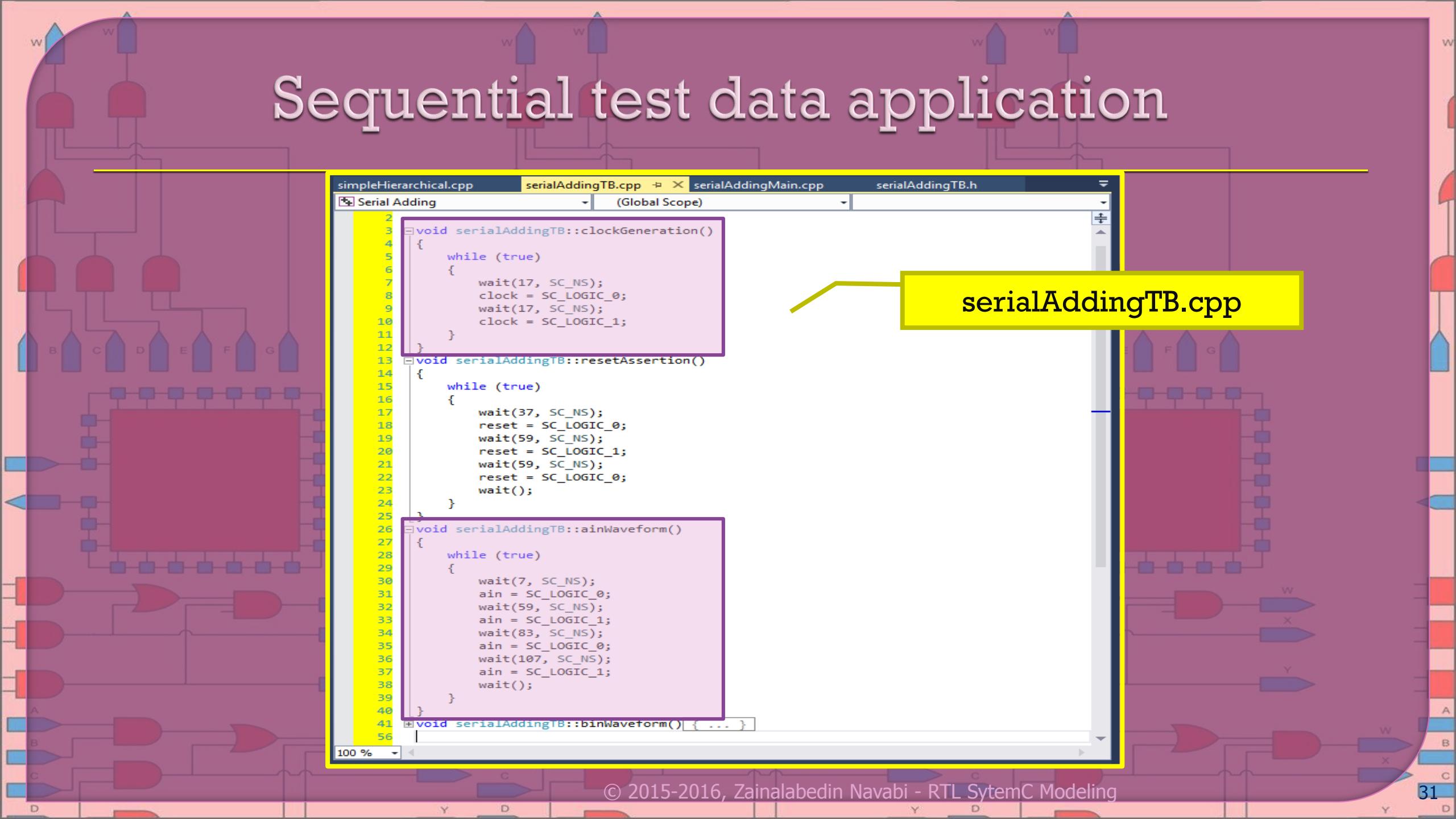


```
simpleHierarchical.cpp      serialAddingTB.cpp      serialAddingMain.cpp      serialAddingTB.h
Serial Adding              (Global Scope)          X
1 #include "simpleHierarchical.h"
2
3 SC_MODULE(serialAddingTB)
4 {
5     sc_signal<sc_logic> ain, bin, reset, clock;
6     sc_signal<sc_logic> sum;
7
8     serialAdding* UUT;
9
10 SC_CTOR(serialAddingTB)
11 {
12     UUT = new serialAdding("serialAdding_instance");
13     (*UUT) (ain, bin, reset, clock, sum);
14     SC_THREAD(clockGeneration);
15     SC_THREAD(resetAssertion);
16     SC_THREAD(ainWaveform);
17         sensitive << clock.posedge_event();
18     SC_THREAD(binWaveform);
19         sensitive << clock.posedge_event();
20 }
21 void clockGeneration();
22 void resetAssertion();
23 void ainWaveform();
24 void binWaveform();
25 }
26 
```

serialAddingTB.h

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

Sequential test data application



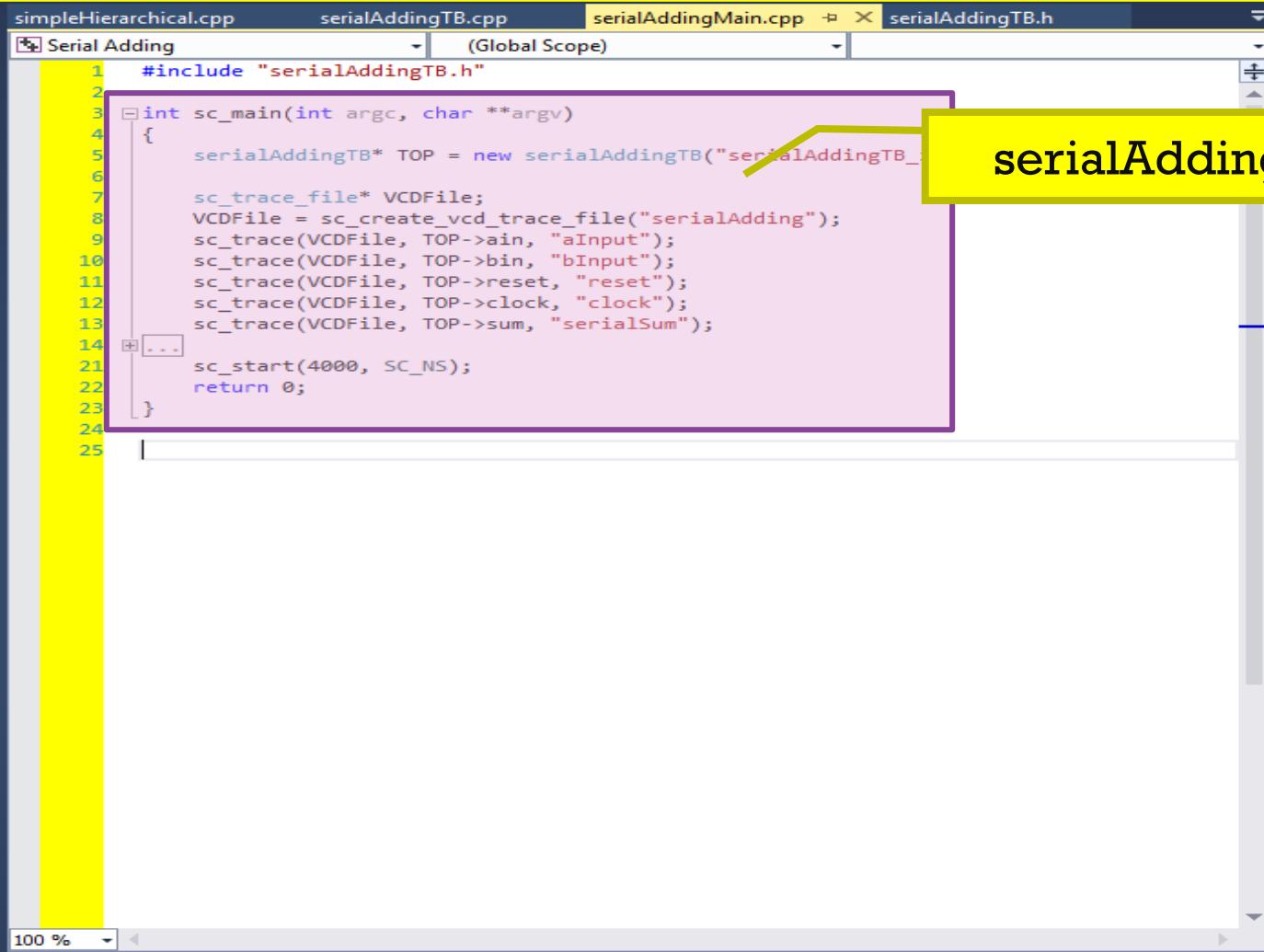
A screenshot of a software interface showing SystemC code for a testbench. The code is contained within a tab labeled `serialAddingTB.cpp`. The code defines three methods: `clockGeneration()`, `resetAssertion()`, and `ainWaveform()`. The `ainWaveform()` method is highlighted with a yellow box.

```
simpleHierarchical.cpp    serialAddingTB.cpp + X    serialAddingMain.cpp    serialAddingTB.h
Serial Adding            (Global Scope)          

2
3 void serialAddingTB::clockGeneration()
4 {
5     while (true)
6     {
7         wait(17, SC_NS);
8         clock = SC_LOGIC_0;
9         wait(17, SC_NS);
10        clock = SC_LOGIC_1;
11    }
12}
13 void serialAddingTB::resetAssertion()
14 {
15     while (true)
16     {
17         wait(37, SC_NS);
18         reset = SC_LOGIC_0;
19         wait(59, SC_NS);
20         reset = SC_LOGIC_1;
21         wait(59, SC_NS);
22         reset = SC_LOGIC_0;
23         wait();
24     }
25}
26 void serialAddingTB::ainWaveform()
27 {
28     while (true)
29     {
30         wait(7, SC_NS);
31         ain = SC_LOGIC_0;
32         wait(59, SC_NS);
33         ain = SC_LOGIC_1;
34         wait(83, SC_NS);
35         ain = SC_LOGIC_0;
36         wait(107, SC_NS);
37         ain = SC_LOGIC_1;
38         wait();
39     }
40}
41 void serialAddingTB::binWaveform() { ... }
```

serialAddingTB.cpp

sc_main



```
#include "serialAddingTB.h"
int sc_main(int argc, char **argv)
{
    serialAddingTB* TOP = new serialAddingTB("serialAdding");
    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_start(4000, SC_NS);
    return 0;
}
```

serialAddingMain.cpp

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");

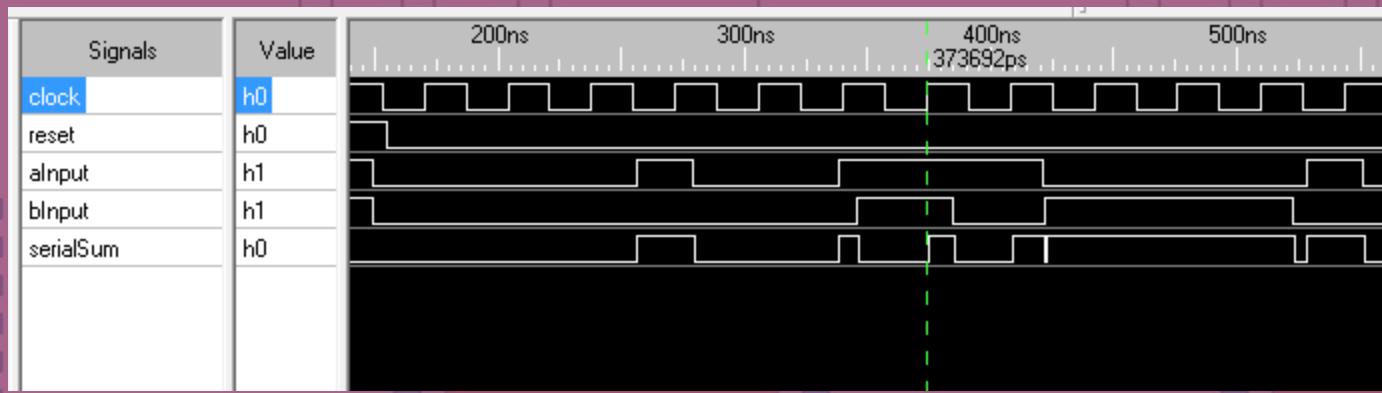
    sc_trace(VCDFile, TOP->UUT->FA1->i1, "i1_fulladder");
    sc_trace(VCDFile, TOP->UUT->FA1->i2, "i2_fulladder");
    sc_trace(VCDFile, TOP->UUT->FA1->i3, "i3_fulladder");
    sc_trace(VCDFile, TOP->UUT->FA1->o1, "o1_fulladder");
    sc_trace(VCDFile, TOP->UUT->FA1->o2, "o2_fulladder");

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

serialAddingMain.cpp

Access internal
signals

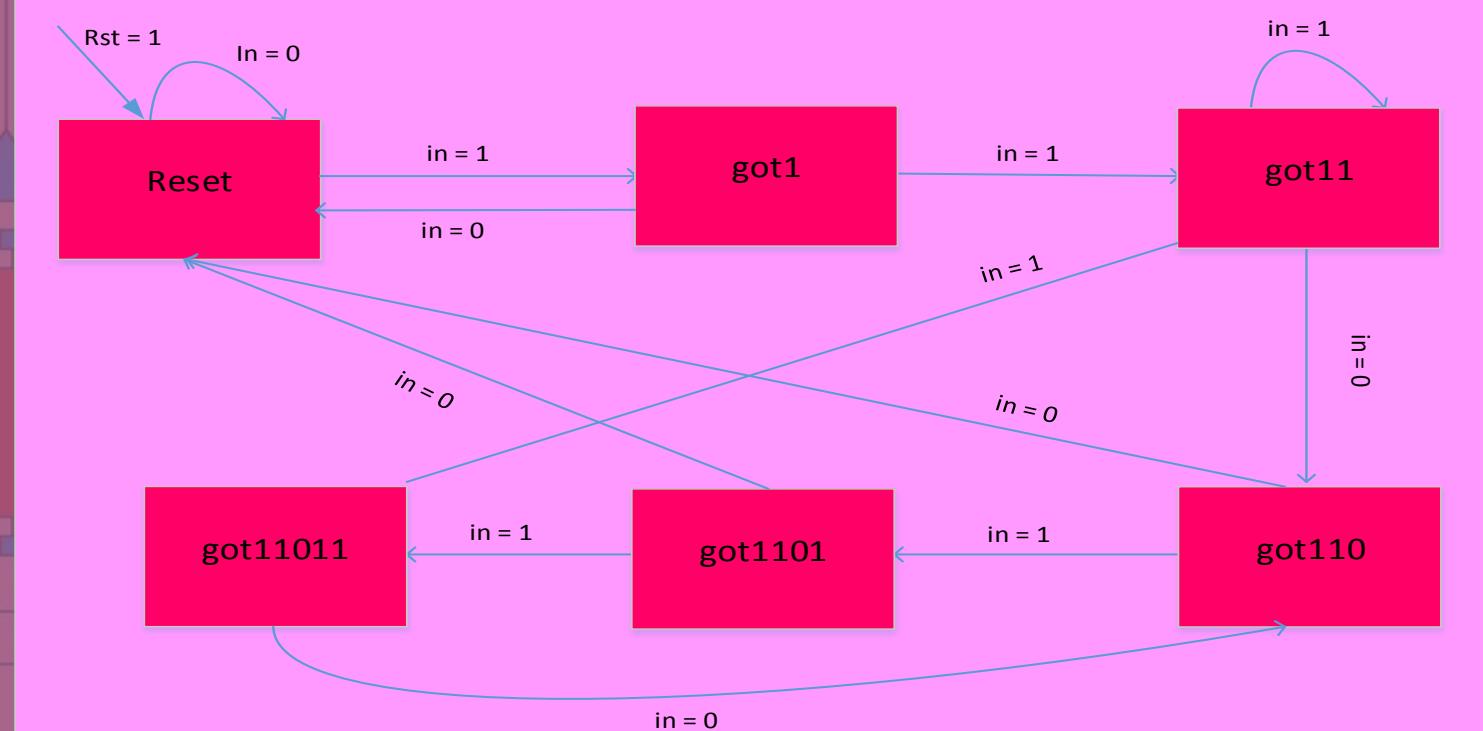
VCD waveform



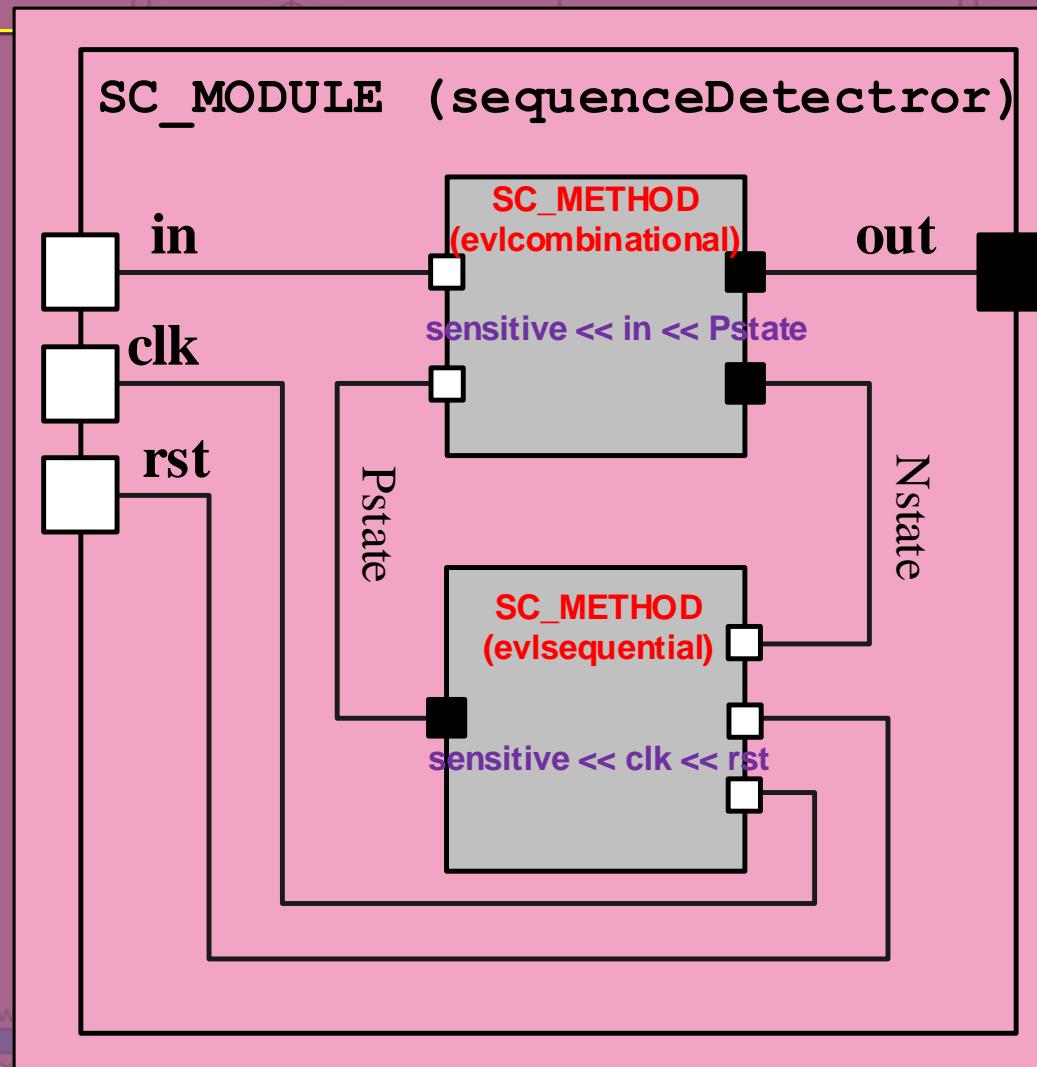
RT Level SystemC

- Taking Off From c++
- SystemC Modeling
- Simulation Environment
- Utilities for HDL Orientation
- Sequential Modeling and Timing
- SystemC FSM Modeling
- Components for RT Level Design
- A configurable Memory

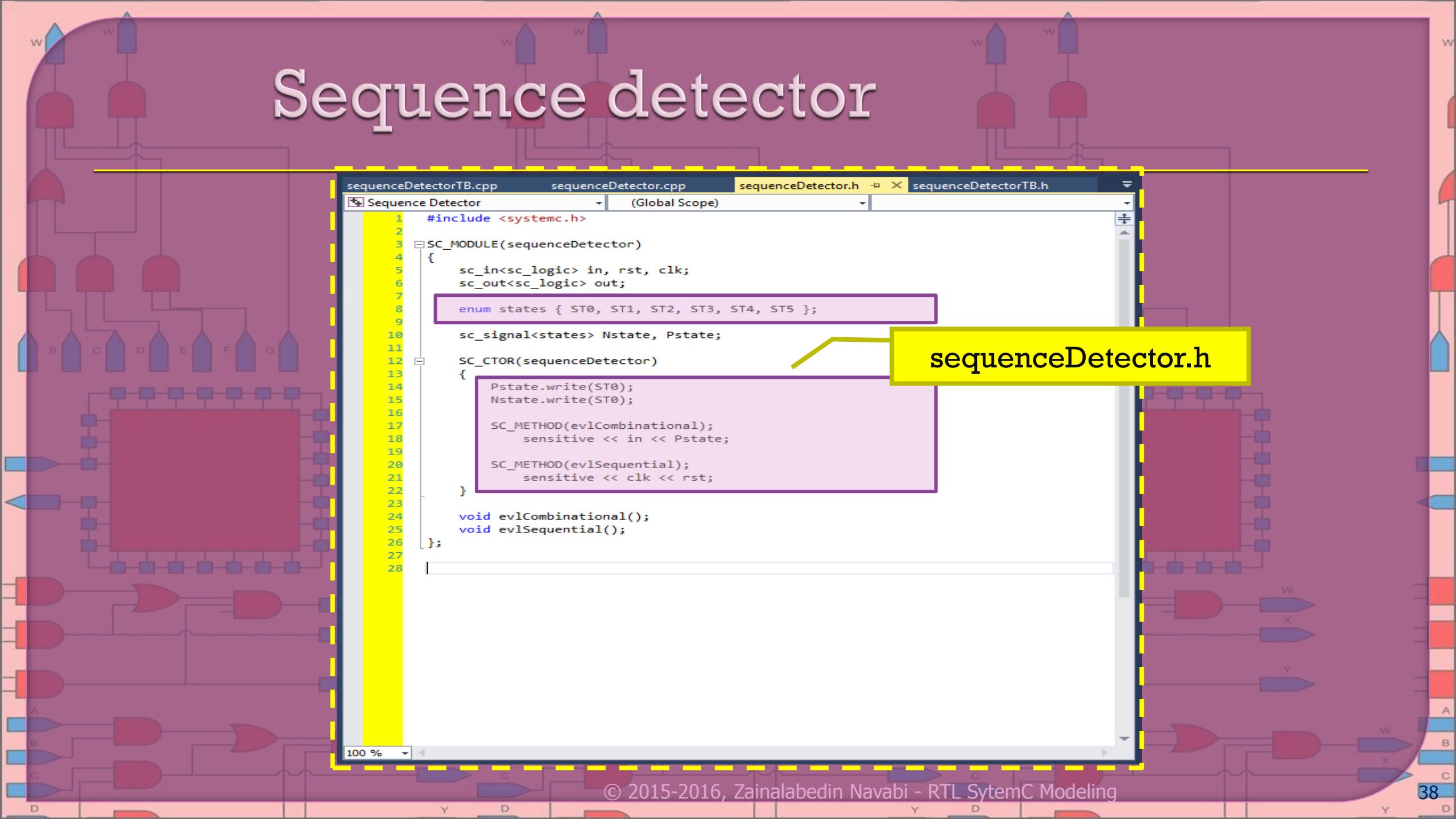
SystemC FSM Modeling (sequence detector 11011)



Huffman Model



Sequence detector



```
#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

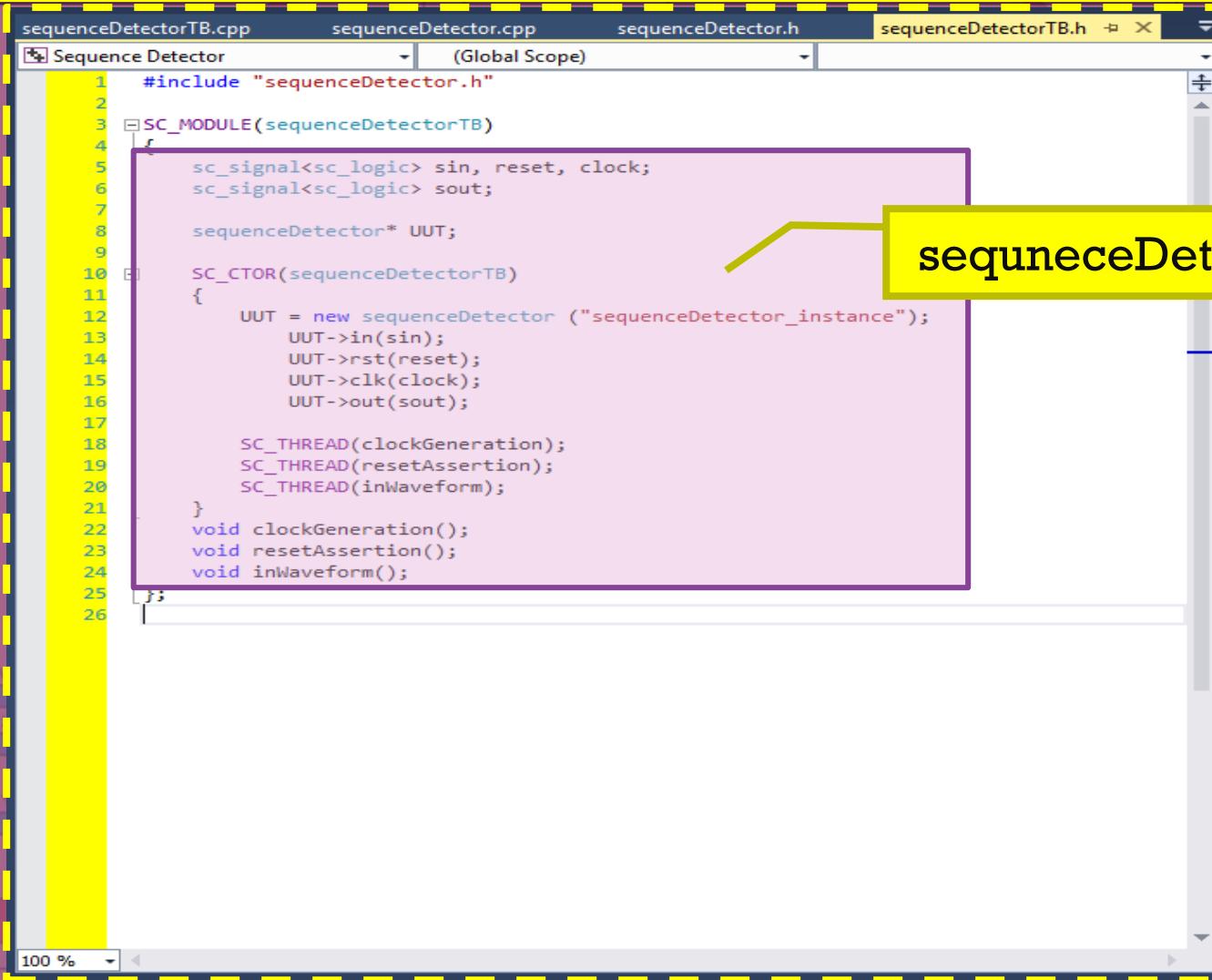
Set to their inactive value

Read is also an sc_signal method

sequenceDetector.cpp

```
sequenceDetectorTB.cpp sequenceDetector.cpp ✘ X sequenceDetector.h sequenceDetectorTB.h
Sequence Detector (Global Scope)
1 #include "sequenceDetector.h"
2
3 void sequenceDetector::evlCombinational()
4 {
5     out = SC_LOGIC_0;
6     Nstate = ST0;
7
8     switch (Pstate.read()){
9         case ST0:
10            if (in == SC_LOGIC_1) Nstate = ST1;
11            else Nstate = ST0; break;
12        case ST1:
13            if (in == SC_LOGIC_1) Nstate = ST2;
14            else Nstate = ST0; break;
15        case ST2:
16            if (in == SC_LOGIC_1) Nstate = ST3;
17            else Nstate = ST2; break;
18        case ST3:
19            if (in == SC_LOGIC_1) Nstate = ST4;
20            else Nstate = ST0; break;
21        case ST4:
22            if (in == SC_LOGIC_1) Nstate = ST5;
23            else Nstate = ST0; break;
24        case ST5:
25            if (in == SC_LOGIC_1) Nstate = ST5;
26            else Nstate = ST3; break;
27        }
28
29        if (Pstate == ST5) out = SC_LOGIC_1;
30    }
31
32 void sequenceDetector::evlSequential()
33 {
34     if (rst == SC_LOGIC_1) Pstate = ST0;
35     else if (clk->event() && clk == SC_LOGIC_1) Pstate = Nstate;
36 }
37
```

Sequence detector testbench



```
#include "sequenceDetector.h"

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

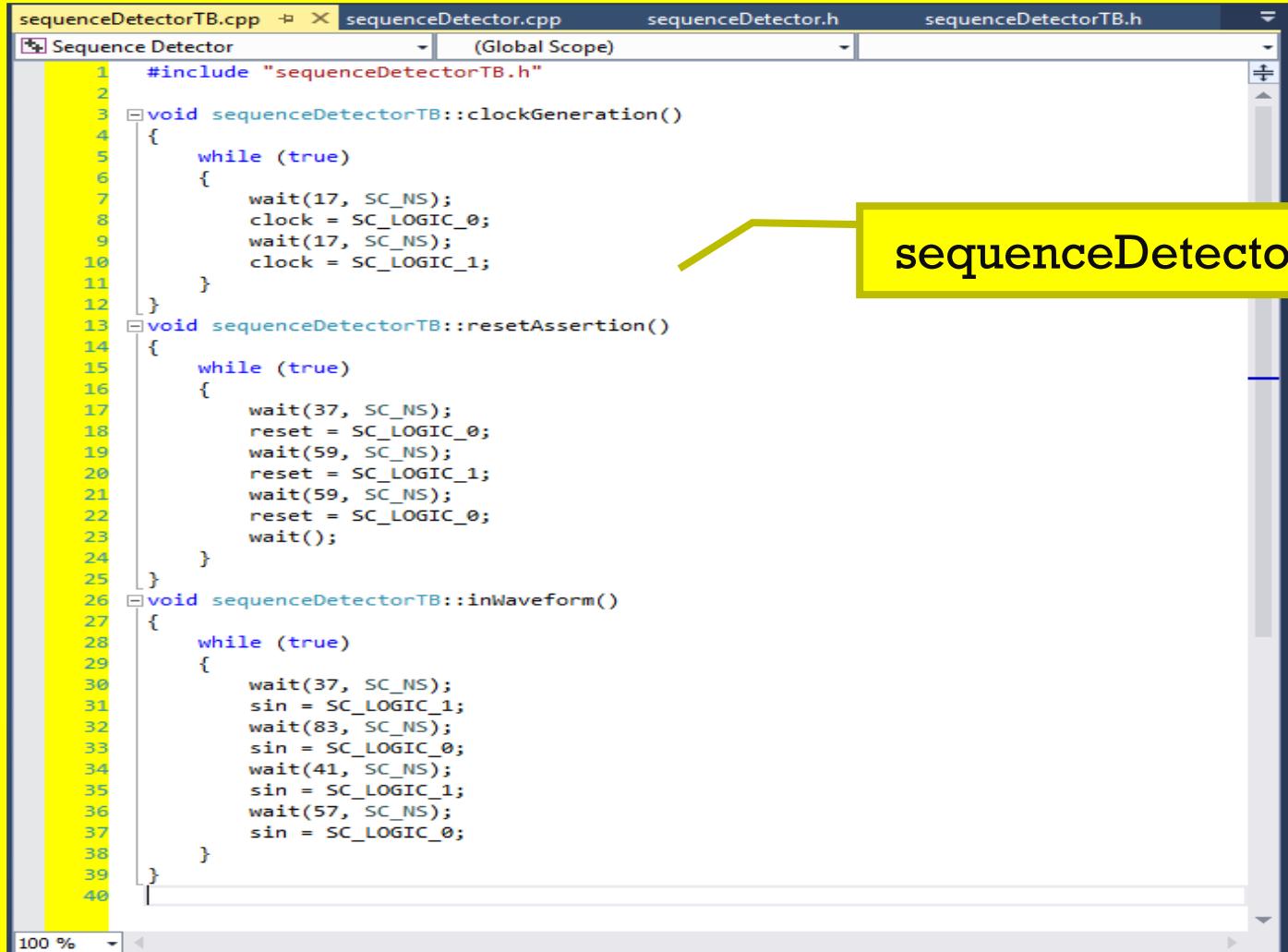
    sequenceDetector* UUT;

    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();
};
```

sequenceDetectorTB.h

Sequence detector testbench

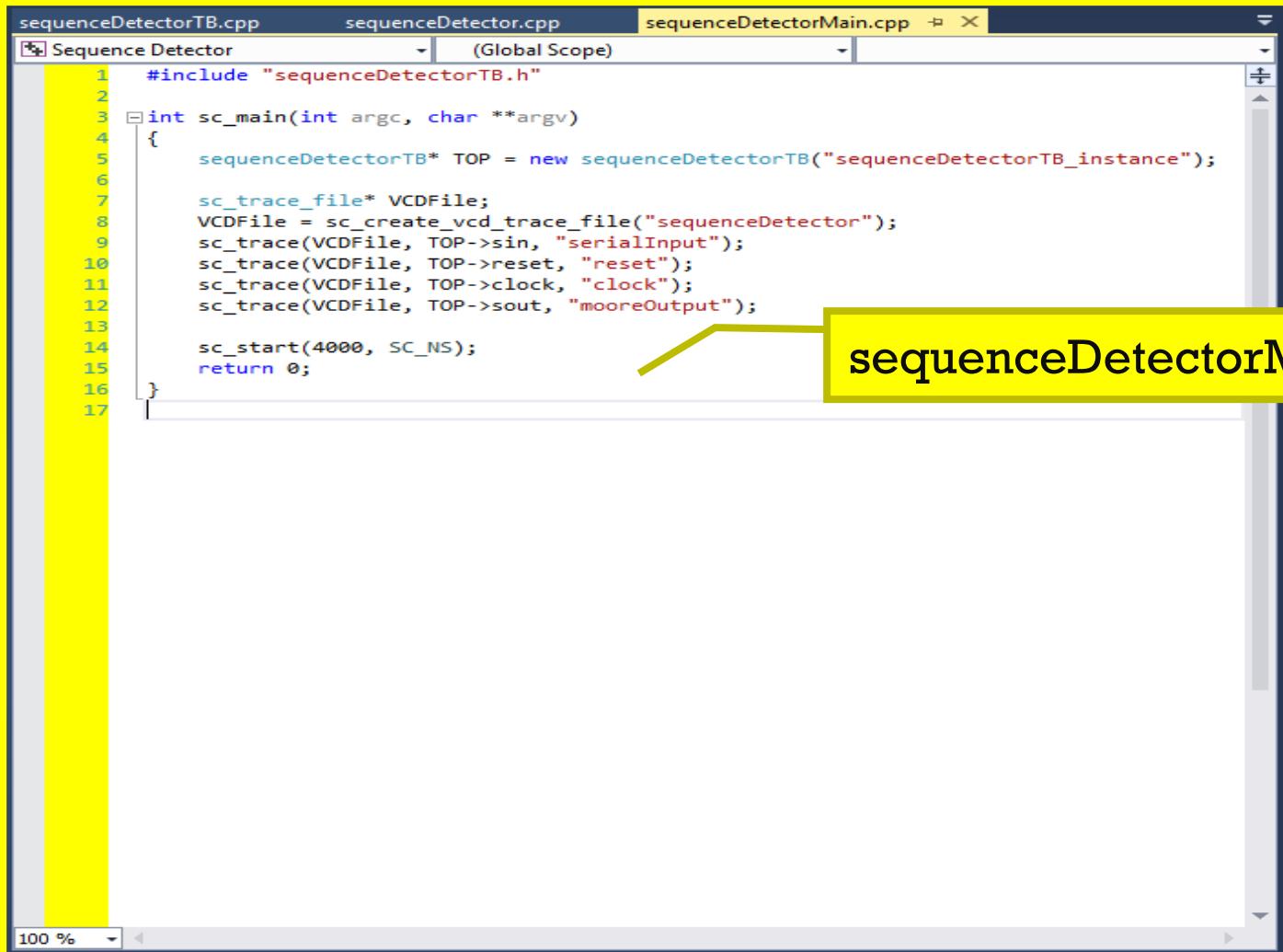


```
sequenceDetectorTB.cpp  X sequenceDetector.cpp      sequenceDetector.h      sequenceDetectorTB.h
Sequence Detector  (Global Scope)

1 #include "sequenceDetectorTB.h"
2
3 void sequenceDetectorTB::clockGeneration()
4 {
5     while (true)
6     {
7         wait(17, SC_NS);
8         clock = SC_LOGIC_0;
9         wait(17, SC_NS);
10        clock = SC_LOGIC_1;
11    }
12}
13 void sequenceDetectorTB::resetAssertion()
14 {
15     while (true)
16     {
17         wait(37, SC_NS);
18         reset = SC_LOGIC_0;
19         wait(59, SC_NS);
20         reset = SC_LOGIC_1;
21         wait(59, SC_NS);
22         reset = SC_LOGIC_0;
23         wait();
24     }
25}
26 void sequenceDetectorTB::inWaveform()
27 {
28     while (true)
29     {
30         wait(37, SC_NS);
31         sin = SC_LOGIC_1;
32         wait(83, SC_NS);
33         sin = SC_LOGIC_0;
34         wait(41, SC_NS);
35         sin = SC_LOGIC_1;
36         wait(57, SC_NS);
37         sin = SC_LOGIC_0;
38     }
39}
```

sequenceDetectorTB.cpp

sc_main



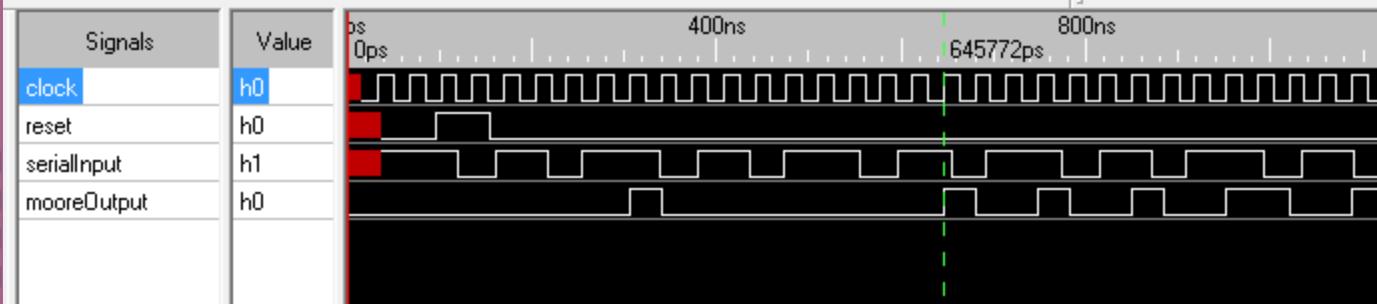
```
#include "sequenceDetectorTB.h"
int sc_main(int argc, char **argv)
{
    sequenceDetectorTB* TOP = new sequenceDetectorTB("sequenceDetectorTB_instance");

    VCDFile* 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

VCD waveform



RT Level SystemC

- Taking Off From c++
- SystemC Modeling
- Simulation Environment
- Utilities for HDL Orientation
- Sequential Modeling and Timing
- SystemC FSM Modeling
- Components for RT Level Design
- A configurable Memory

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>

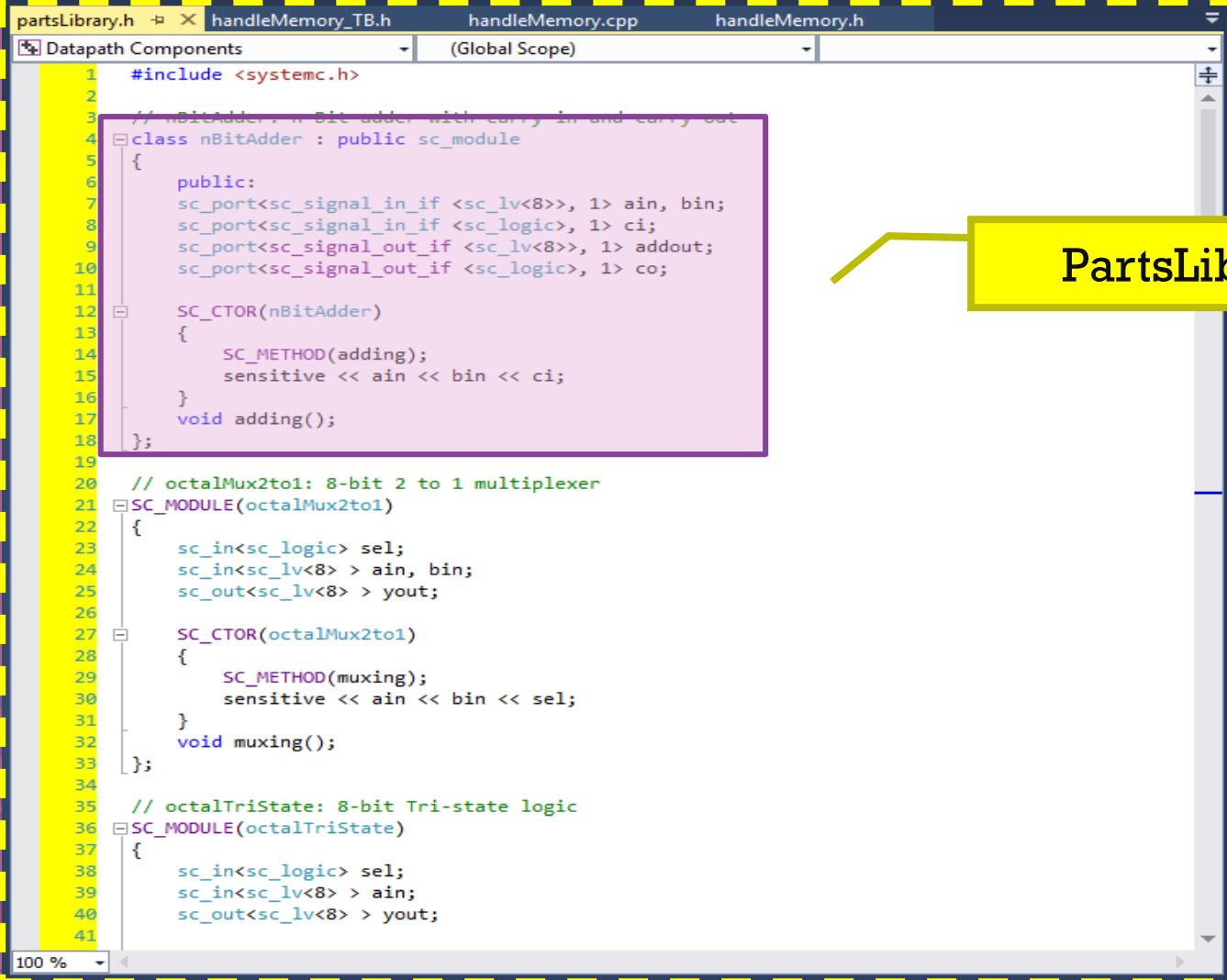
Sc_bit is two value logic

- 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

O	I	Z	X
O	0	X	0
I	1	1	1
Z	0	1	2
X	X	X	X

Components for RT Level Design



The image shows a software interface for SystemC modeling. At the top, there are tabs for "partsLibrary.h", "handleMemory_TB.h", "handleMemory.cpp", and "handleMemory.h". The "handleMemory.h" tab is active. Below the tabs, a tree view shows "Datapath Components" expanded, with "Global Scope" selected. The main area displays SystemC code for three components: nBitAdder, octalMux2to1, and octalTriState. The code uses sc_port<sc_signal_in_if>, sc_port<sc_signal_out_if>, SC_MODULE, SC_METHOD, and SC_CTOR. A yellow callout box with the text "PartsLibrary.h" points to the first component, nBitAdder.

```
#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(add);
        sensitive << ain << bin << ci;
    }
    void add();
};

// 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>> you;

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

// 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>> you;
}
```

Components for RT Level Design

```
partsLibrary.h          handleMemory_TB.h      handleMemory.cpp      handleMemory.h      partsLibrary.cpp
1 #include "partsLibrary.h"
2
3 void nBitAdder::adding()
4 {
5     sc_lv<9> res;
6     res = ain->read().to_uint() + bin->read().to_uint()
7         + ci->read().value();
8     addout->write(res.range(7, 0));
9     co->write(res[8]);
10 }
11
12 void octalMux2to1::muxing()
13 {
14     if (sel->read() == '1') yout->write(bin);
15     else yout->write(ain);
16 }
17
18 void octalTriState::selecting()
19 {
20     if (sel == '1') yout = ain;
21     else yout = "zzzzzzzz";
22 }
23
24 void dRegisterRaE::registering()
25 {
26     if (rst == '1')
27     {
28         regout = "00000000";
29     }
30     else if (clk->event() && (clk == '1'))
31     {
32         if (cen == '1') regout = regin;
33     }
34 }
35
36 void dRegisterRaEZ::registering()
37 {
38     if (rst == '1')
39     {
40         regout = "00000000";
41     }
42     else if (clk->event() && (clk == '1'))
43     {
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100 %
```

Add operation is not defined for the logic type

partsLibrary.cpp

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

always @ (a, b)
w <= ~
g = w & a;
end

Signals and variables

```
#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'))
    {
```

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

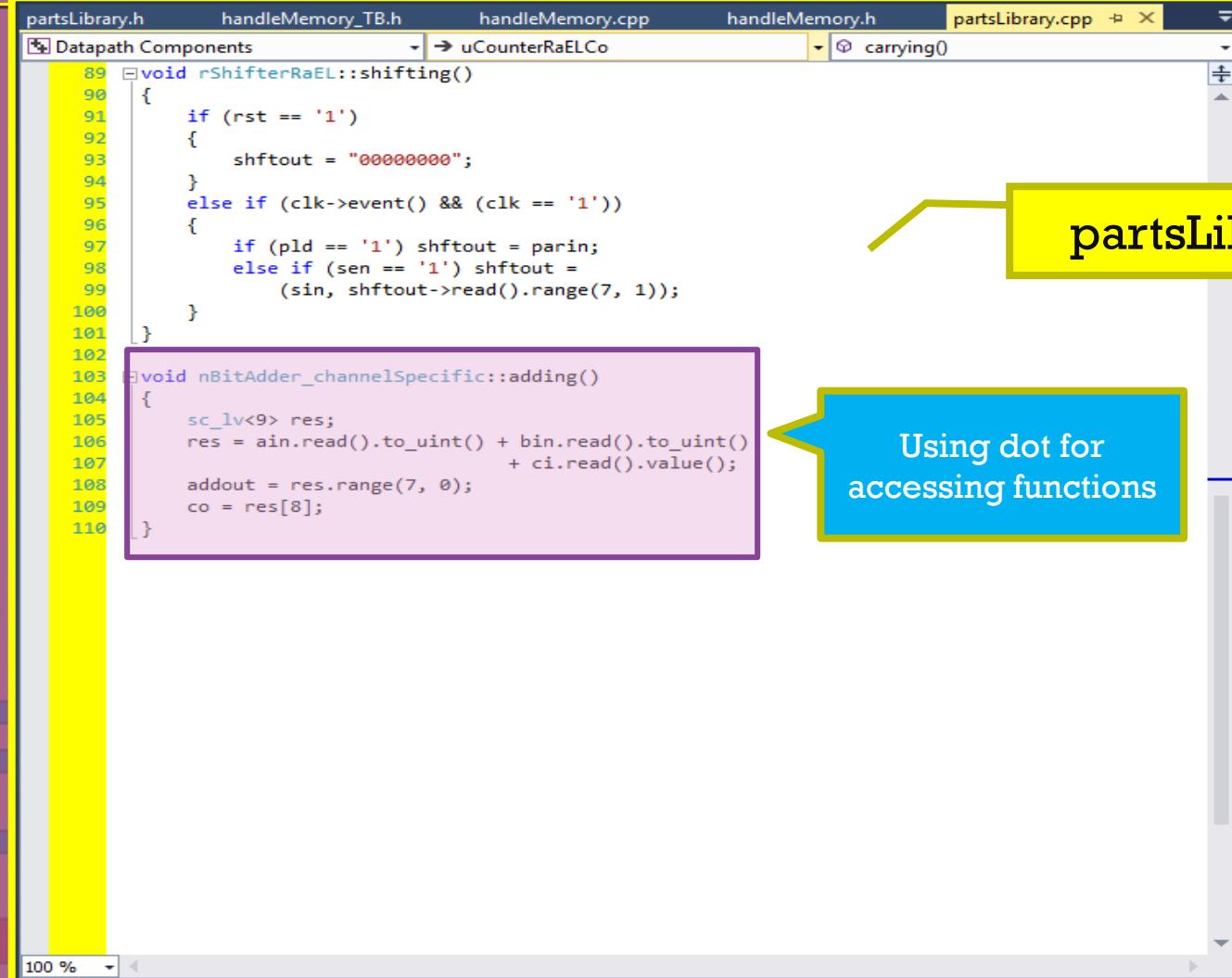
Using sc_in and sc_out

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h
Datapath Components  Memory<ADDRESS, WORD_LENGTH>  Memory(sc_module_name)

225 // nBitAdder: n Bit adder with carry in and carry out
226 SC_MODULE(nBitAdder_channelSpecific)
227 {
228     sc_in<sc_lv<8>> ain, bin;
229     sc_in<sc_logic> ci;
230     sc_out<sc_lv<8>> addout;
231     sc_out<sc_logic> co;
232
233     SC_CTOR(nBitAdder_channelSpecific)
234     {
235         SC_METHOD(add);
236         sensitive << ain << bin << ci;
237     }
238     void add();
239 }
```

partsLibrary.h

Using sc_in and sc_port

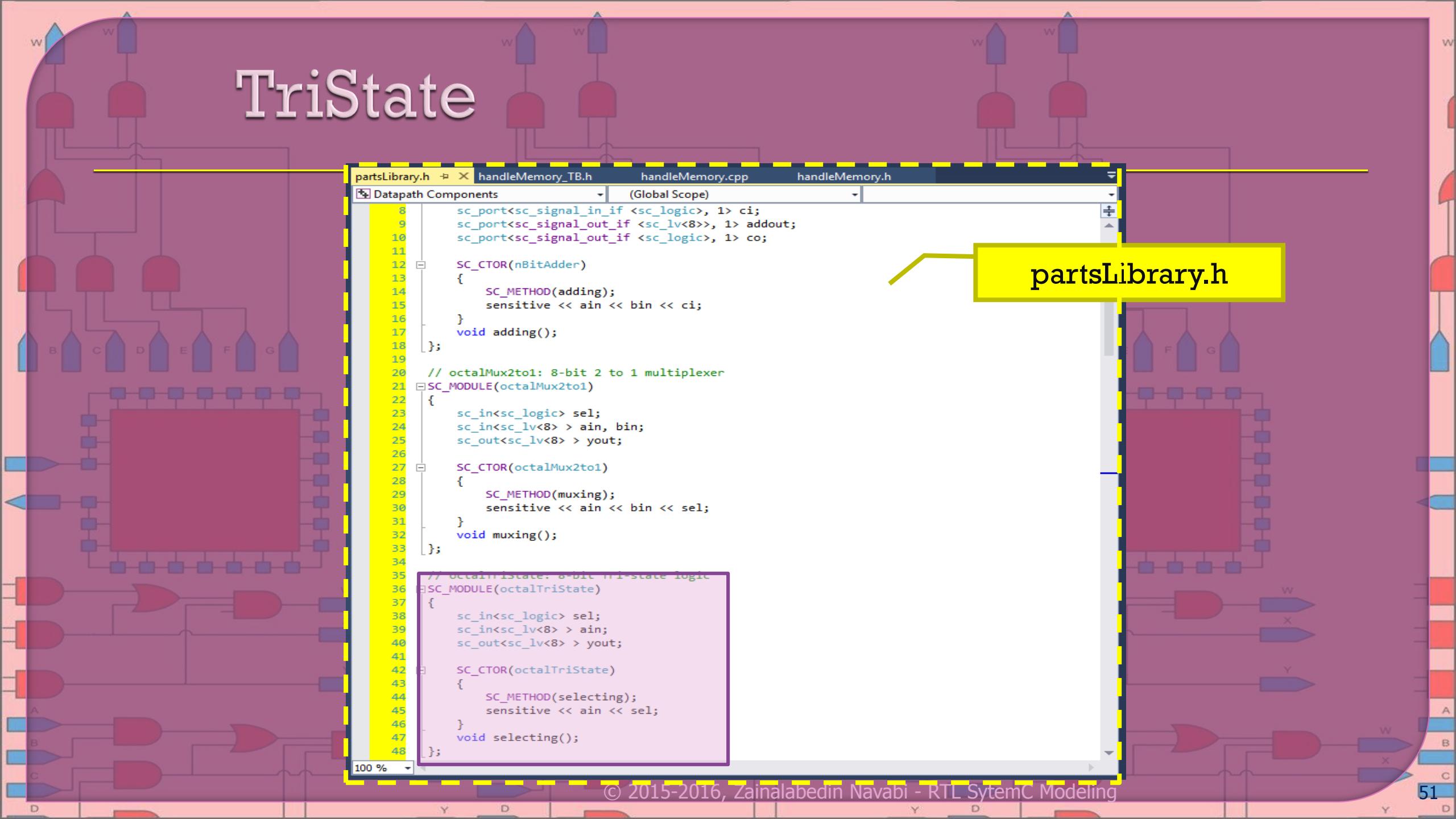


The screenshot shows a software interface with multiple tabs at the top: partsLibrary.h, handleMemory_TB.h, handleMemory.cpp, handleMemory.h, partsLibrary.cpp, and carrying(). The partsLibrary.cpp tab is active, displaying the following code:

```
89 void rShifterRaEL::shifting()
90 {
91     if (rst == '1')
92     {
93         shftout = "00000000";
94     }
95     else if (clk->event() && (clk == '1'))
96     {
97         if (pld == '1') shftout = parin;
98         else if (sen == '1') shftout =
99             (sin, shftout->read().range(7, 1));
100    }
101}
102
103 void nBitAdder_channelSpecific::adding()
104 {
105     sc_lv<9> res;
106     res = ain.read().to_uint() + bin.read().to_uint()
107         + ci.read().value();
108     addout = res.range(7, 0);
109     co = res[8];
110 }
```

A yellow callout box points to the word "partsLibrary.cpp" in the title bar, with the text "partsLibrary.cpp". Another yellow callout box points to the dot operator in the line "shftout = (sin, shftout->read().range(7, 1));", with the text "Using dot for accessing functions".

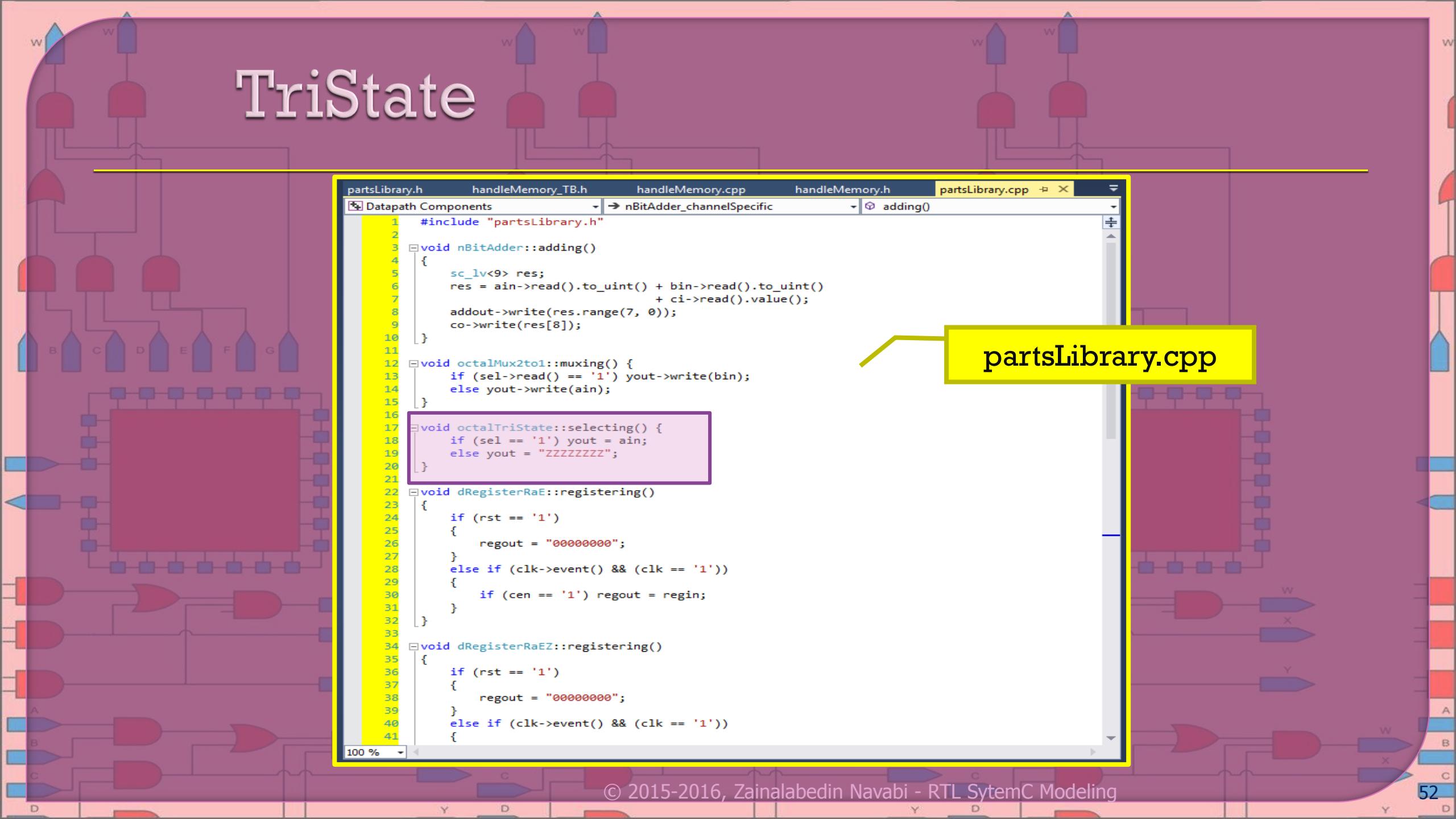
TriState



The image shows a complex SystemC model for a TriState component. The model consists of a large red rectangular block with many pins, connected to various logic gates (AND, OR, NOT) and tri-state buffers (indicated by blue triangles). A yellow box highlights the `partsLibrary.h` file in the code editor.

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h
partsLibrary.h
Datapath Components (Global Scope)
8     sc_port<sc_signal_in_if <sc_logic>, 1> ci;
9     sc_port<sc_signal_out_if <sc_lv<8>>, 1> addout;
10    sc_port<sc_signal_out_if <sc_logic>, 1> co;
11
12    SC_CTOR(nBitAdder)
13    {
14        SC_METHOD(adding);
15        sensitive << ain << bin << ci;
16    }
17    void adding();
18
19
20 // octalMux2to1: 8-bit 2 to 1 multiplexer
21 SC_MODULE(octalMux2to1)
22 {
23     sc_in<sc_logic> sel;
24     sc_in<sc_lv<8>> ain, bin;
25     sc_out<sc_lv<8>> yout;
26
27     SC_CTOR(octalMux2to1)
28     {
29         SC_METHOD(muxing);
30         sensitive << ain << bin << sel;
31     }
32     void muxing();
33
34
35 // octalTriState: 8-bit tri-state logic
36 SC_MODULE(octalTriState)
37 {
38     sc_in<sc_logic> sel;
39     sc_in<sc_lv<8>> ain;
40     sc_out<sc_lv<8>> yout;
41
42     SC_CTOR(octalTriState)
43     {
44         SC_METHOD(selecting);
45         sensitive << ain << sel;
46     }
47     void selecting();
48 }
```

TriState



The screenshot shows a SystemC IDE interface with several tabs open. The active tab is `partsLibrary.cpp`, which contains C++ code for a `TriState` component. The code includes declarations for `partsLibrary.h`, `handleMemory_TB.h`, `handleMemory.cpp`, and `handleMemory.h`. The `partsLibrary.cpp` file contains the implementation of the `TriState` class, including methods for adding, muxing, selecting, and registering.

```
#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

Register Modeling

Static sensitivity

partsLibrary.h

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h
Datapath Components (Global Scope)
50 // dRegisterRaE: D Register w/ asynch Reset, clock Enable
51 SC_MODULE(dRegisterRaE)
52 {
53     sc_in<sc_logic> rst, clk, cen;
54     sc_in<sc_lv<8>> regin;
55     sc_out<sc_lv<8>> regout;
56
57     SC_CTOR(dRegisterRaE)
58     {
59         SC_METHOD(registering);
60         sensitive << rst << clk;
61     }
62     void registering();
63 }
64
65 // dRegisterRaEZ: D Register w/ asynch Reset, clock Enable, load Zero
66 SC_MODULE(dRegisterRaEZ)
67 {
68     sc_in<sc_logic> rst, clk, cen, zer;
69     sc_in<sc_lv<8>> regin;
70     sc_out<sc_lv<8>> regout;
71
72     SC_CTOR(dRegisterRaEZ)
73     {
74         SC_METHOD(registering);
75         sensitive << rst << clk;
76     }
77     void registering();
78 }
79
80 // dRegisterRsE: D Register w/ sync Reset, clock Enable
81 SC_MODULE(dRegisterRsE)
82 {
83     sc_in<sc_logic> rst, clk, cen;
84     sc_in<sc_lv<8>> regin;
85     sc_out<sc_lv<8>> regout;
86
87     SC_CTOR(dRegisterRsE)
88     {
89         SC_METHOD(registering);
90         sensitive << clk.pos();
```

Register Modeling

Asynch reset

Dynamic sensitivity

partsLibrary.cpp

```
partsLibrary.h          handleMemory_TB.h      handleMemory.cpp      handleMemory.h      partsLibrary.cpp  X
Datapath Components      nBitAdder_channelSpecific
21
22 void dRegisterRaE::registering()
23 {
24     if (rst == '1')
25     {
26         regout = "00000000";
27     }
28     else if (clk->event() && (clk == '1'))
29     {
30         if (cen == '1') regout = regin;
31     }
32
33
34 void dRegisterRaEZ::registering()
35 {
36     if (rst == '1')
37     {
38         regout = "00000000";
39     }
40     else if (clk->event() && (clk == '1'))
41     {
42         if (cen == '1') {
43             if (zer == '1') regout = 0;
44             else regout = regin;
45         }
46     }
47
48
49 void dRegisterRsE::registering() {
50     if (rst == '1') {
51         regout = 0;
52     }
53     else if (cen == '1') {
54         regout = regin;
55     }
56 }
57
58 void uCounterRaEL::counting()
59 {
60     if (rst == '1')
61     {
```

Register Modeling

.pos() can
only be used
for sc_in

```
partsLibrary.h
Datapath Components (Global Scope) SC_CTOR(octalMux2to1)

80 // dRegisterRsE: D Register w/ sync Reset, clock Enable
81 SC_MODULE(dRegisterRsE)
82 {
83     sc_in<sc_logic> rst, clk, cen;
84     sc_in<sc_lv<8>> regin;
85     sc_out<sc_lv<8>> regout;
86
87 SC_CTOR(dRegisterRsE)
88 {
89     SC_METHOD(registering);
90     sensitive << clk.pos();
91 }
92 void registering();
93

94 // uCounterRaEL: Up-counter w/ asynch Reset, clock Enable, parallel Load
95 SC_MODULE(uCounterRaEL)
96 {
97     sc_in<sc_logic> rst, clk, cen, pld;
98     sc_in<sc_lv<8>> parin;
99     sc_out<sc_lv<8>> cntout;
100
101 SC_CTOR(uCounterRaEL)
102 {
103     SC_METHOD(counting);
104     sensitive << rst << clk;
105 }
106 void counting();
107
108

109 // uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
110 SC_MODULE(uCounterRaELCo)
111 {
112     sc_in<sc_logic> rst, clk, cen, pld;
113     sc_in<sc_logic> ci;
114     sc_out<sc_logic> co;
115     sc_in<sc_lv<8>> parin;
116     sc_out<sc_lv<8>> cntout;
117
118 SC_CTOR(uCounterRaELCo)
119 {
120 }
```

partsLibrary.h

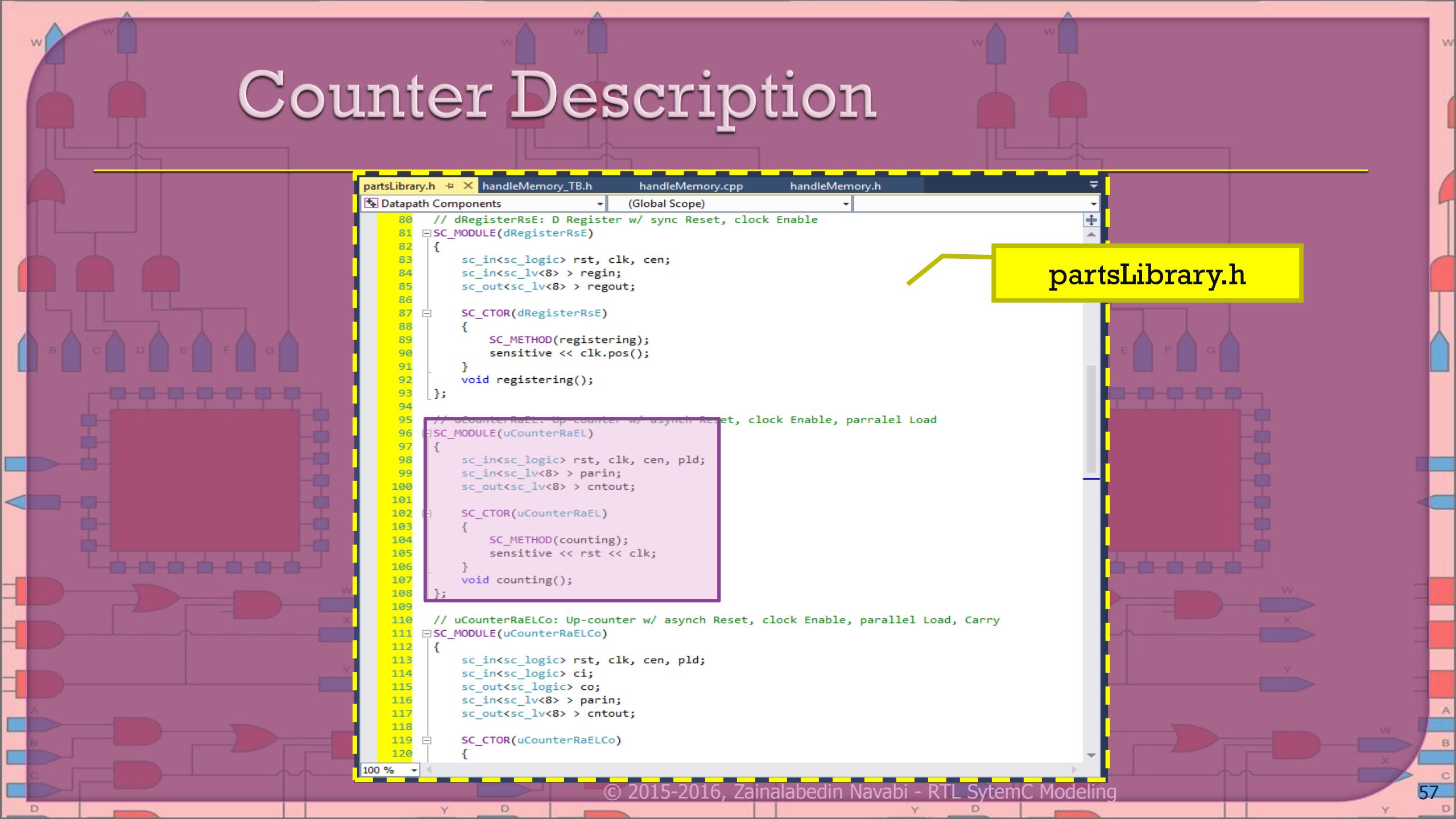
Register Modeling

Doesn't need
to check
clock edge

Synch reset

partsLibrary.cpp

Counter Description

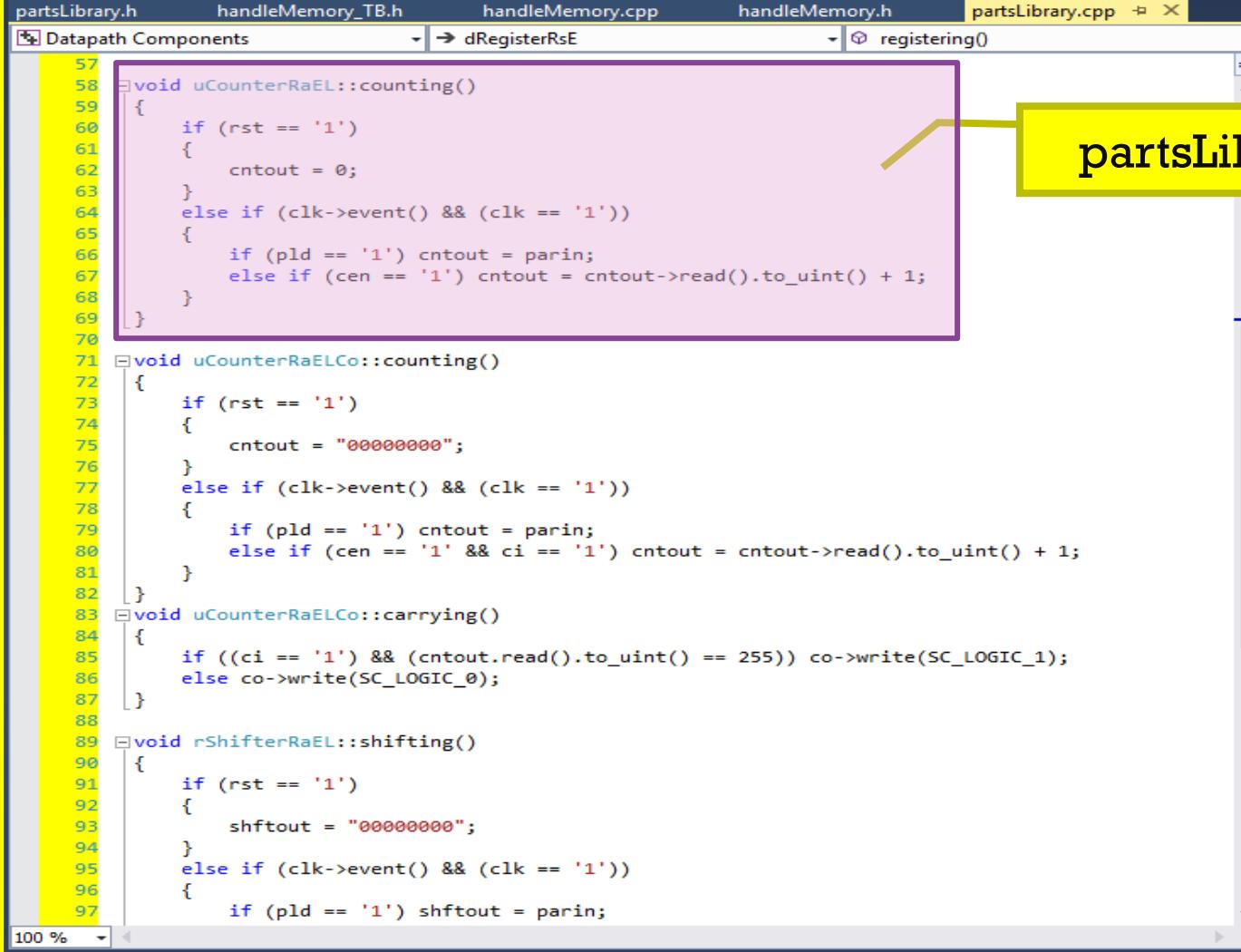


The diagram shows a complex digital logic circuit, likely a counter or memory controller, composed of various logic gates (AND, OR, NOT) and flip-flops. It has multiple input ports labeled A, B, C, D, E, F, G, X, Y, and Z, and output ports labeled W, X, Y, and Z. A large red rectangular component, possibly a memory or a complex logic block, is integrated into the design.

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h
Datapath Components (Global Scope)
80 // dRegisterRsE: D Register w/ sync Reset, clock Enable
81 SC_MODULE(dRegisterRsE)
82 {
83     sc_in<sc_logic> rst, clk, cen;
84     sc_in<sc_lv<8>> regin;
85     sc_out<sc_lv<8>> regout;
86
87     SC_CTOR(dRegisterRsE)
88     {
89         SC_METHOD(registering);
90         sensitive << clk.pos();
91     }
92     void registering();
93 };
94
95 // uCounterRaEL: Up Counter w/ asynch Reset, clock Enable, parallel Load
96 SC_MODULE(uCounterRaEL)
97 {
98     sc_in<sc_logic> rst, clk, cen, pld;
99     sc_in<sc_lv<8>> parin;
100    sc_out<sc_lv<8>> cntout;
101
102    SC_CTOR(uCounterRaEL)
103    {
104        SC_METHOD(counting);
105        sensitive << rst << clk;
106    }
107    void counting();
108 };
109
110 // uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
111 SC_MODULE(uCounterRaELCo)
112 {
113     sc_in<sc_logic> rst, clk, cen, pld;
114     sc_in<sc_logic> ci;
115     sc_out<sc_logic> co;
116     sc_in<sc_lv<8>> parin;
117     sc_out<sc_lv<8>> cntout;
118
119     SC_CTOR(uCounterRaELCo)
120     {
```

partsLibrary.h

Counter Description

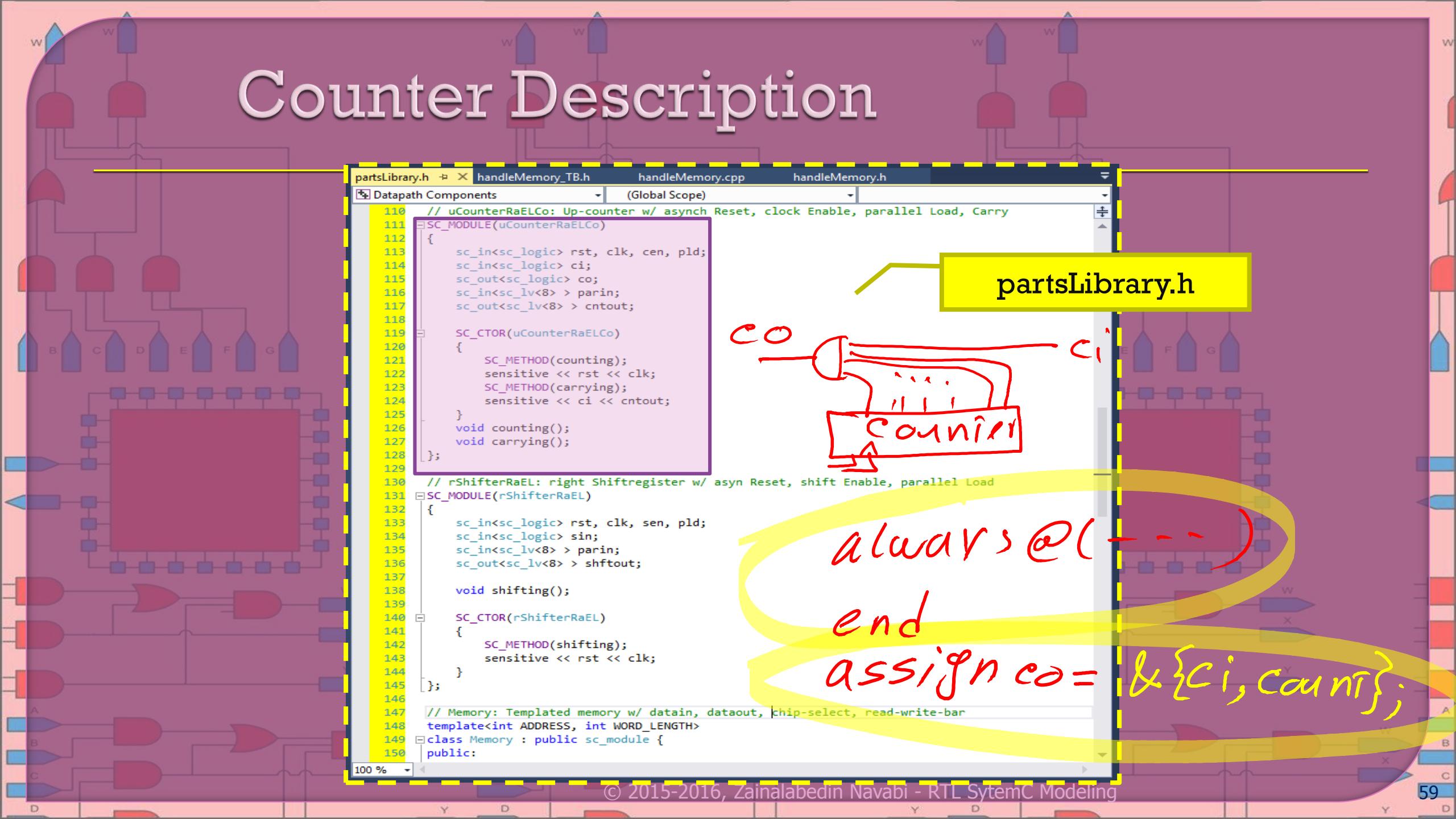


The screenshot shows a SystemC IDE interface with multiple tabs and files. The main code editor window displays C++ code for a counter component. A yellow box highlights the code for the `partsLibrary.cpp` file, which is labeled "registering0". The code implements counting logic for three different counter types: `uCounterRaEL`, `uCounterRaELCo`, and `rShifterRaEL`. It includes logic for reset, clock events, parallel load (pld), clear (cen), and carry-in (ci) operations, along with shifting logic for the `rShifterRaEL` type.

```
57 void uCounterRaEL::counting()
58 {
59     if (rst == '1')
60     {
61         cntout = 0;
62     }
63     else if (clk->event() && (clk == '1'))
64     {
65         if (pld == '1') cntout = parin;
66         else if (cen == '1') cntout = cntout->read().to_uint() + 1;
67     }
68 }
69
70 void uCounterRaELCo::counting()
71 {
72     if (rst == '1')
73     {
74         cntout = "00000000";
75     }
76     else if (clk->event() && (clk == '1'))
77     {
78         if (pld == '1') cntout = parin;
79         else if (cen == '1' && ci == '1') cntout = cntout->read().to_uint() + 1;
80     }
81 }
82
83 void uCounterRaELCo::carrying()
84 {
85     if ((ci == '1') && (cntout.read().to_uint() == 255)) co->write(SC_LOGIC_1);
86     else co->write(SC_LOGIC_0);
87 }
88
89 void rShifterRaEL::shifting()
90 {
91     if (rst == '1')
92     {
93         shftout = "00000000";
94     }
95     else if (clk->event() && (clk == '1'))
96     {
97         if (pld == '1') shftout = parin;
98     }
99 }
```

partsLibrary.cpp

Counter Description



```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h
Datapath Components  (Global Scope)

110 // uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
111 SC_MODULE(uCounterRaELCo)
112 {
113     sc_in<sc_logic> rst, clk, cen, pld;
114     sc_in<sc_logic> ci;
115     sc_out<sc_logic> co;
116     sc_in<sc_lv<8>> parin;
117     sc_out<sc_lv<8>> cntout;
118
119     SC_CTOR(uCounterRaELCo)
120     {
121         SC_METHOD(counting);
122         sensitive << rst << clk;
123         SC_METHOD(carrying);
124         sensitive << ci << cntout;
125     }
126     void counting();
127     void carrying();
128 };
129
130 // rShifterRaEL: right Shiftregister w/ asyn Reset, shift Enable, parallel Load
131 SC_MODULE(rShifterRaEL)
132 {
133     sc_in<sc_logic> rst, clk, sen, pld;
134     sc_in<sc_logic> sin;
135     sc_in<sc_lv<8>> parin;
136     sc_out<sc_lv<8>> shftout;
137
138     void shifting();
139
140     SC_CTOR(rShifterRaEL)
141     {
142         SC_METHOD(shifting);
143         sensitive << rst << clk;
144     }
145
146
147 // Memory: Templatized memory w/ datain, dataout, |chip-select, read-write-bar
148 template<int ADDRESS, int WORD_LENGTH>
149 class Memory : public sc_module {
150     public:
```

partsLibrary.h

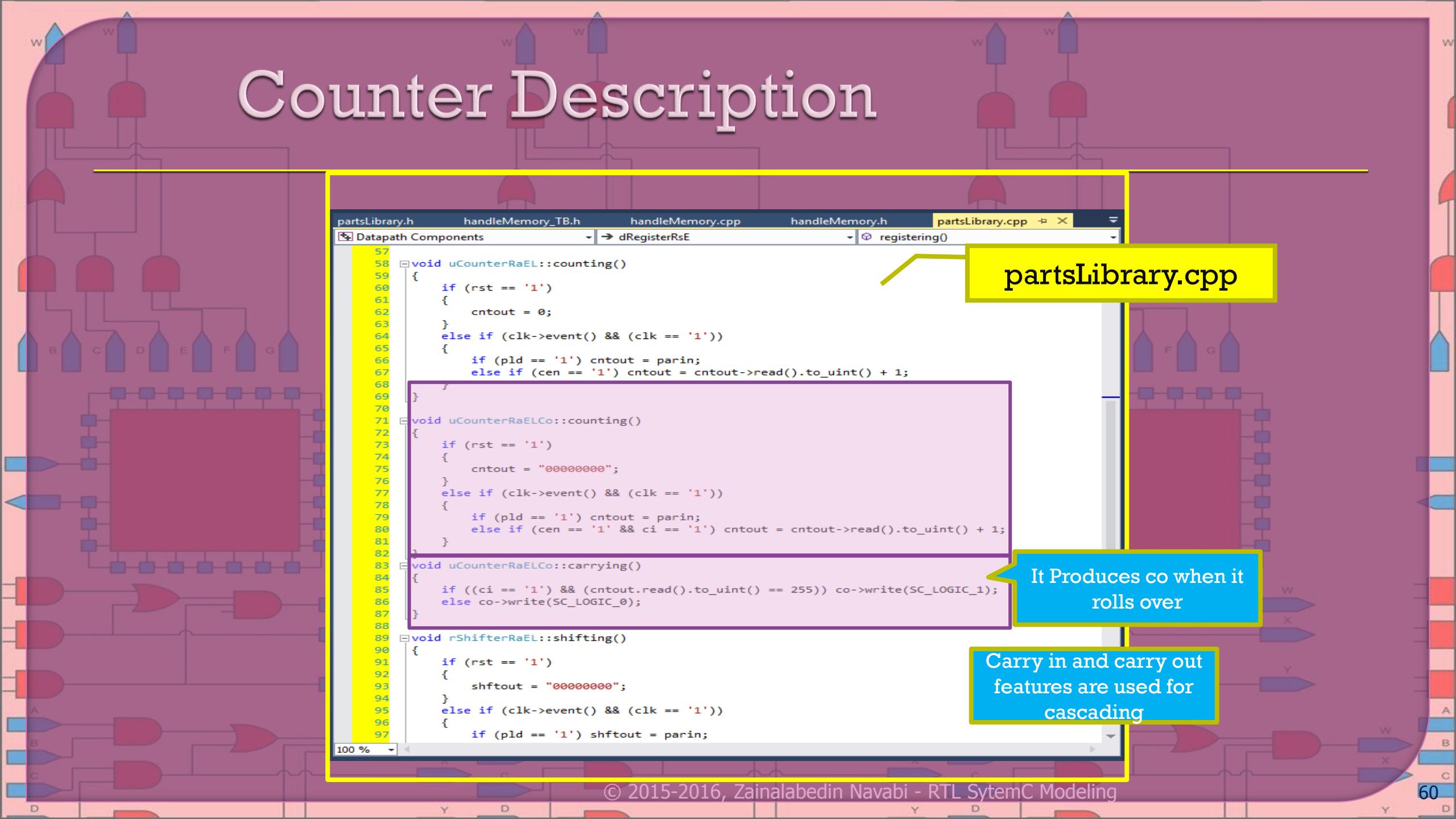


always @(--)

end

assign co = &{ci, count};

Counter Description



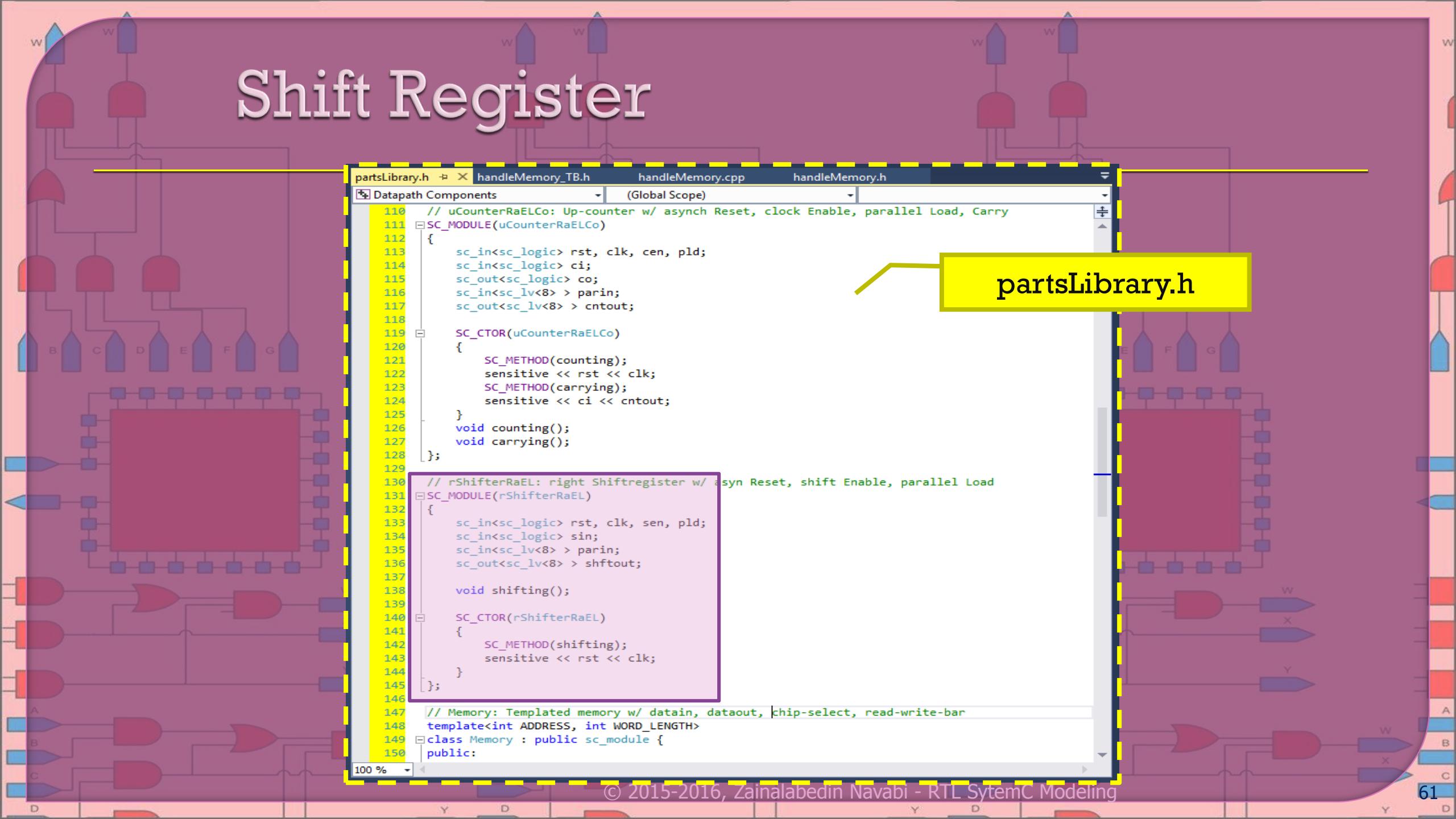
```
partsLibrary.h      handleMemory_TB.h      handleMemory.cpp      handleMemory.h      partsLibrary.cpp + X
Datapath Components
dRegisterRsE
registering0
57
58 void uCounterRaEL::counting()
59 {
60     if (rst == '1')
61     {
62         cntout = 0;
63     }
64     else if (clk->event() && (clk == '1'))
65     {
66         if (pld == '1') cntout = parin;
67         else if (cen == '1') cntout = cntout->read().to_uint() + 1;
68     }
69 }
70
71 void uCounterRaELCo::counting()
72 {
73     if (rst == '1')
74     {
75         cntout = "00000000";
76     }
77     else if (clk->event() && (clk == '1'))
78     {
79         if (pld == '1') cntout = parin;
80         else if (cen == '1' && ci == '1') cntout = cntout->read().to_uint() + 1;
81     }
82 }
83
84 void uCounterRaELCo::carrying()
85 {
86     if ((ci == '1') && (cntout.read().to_uint() == 255)) co->write(SC_LOGIC_1);
87     else co->write(SC_LOGIC_0);
88 }
89
90 void rShifterRaEL::shifting()
91 {
92     if (rst == '1')
93     {
94         shftout = "00000000";
95     }
96     else if (clk->event() && (clk == '1'))
97     {
98         if (pld == '1') shftout = parin;
99     }
100 }
```

partsLibrary.cpp

It Produces co when it rolls over

Carry in and carry out features are used for cascading

Shift Register



The diagram shows a complex digital logic circuit, specifically a Shift Register, composed of various logic gates like AND, OR, NOT, and XNOR, along with memory components represented by red rectangles. A yellow box highlights a specific part of the circuit, which corresponds to the code in the editor.

partsLibrary.h

```
110 // uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
111 SC_MODULE(uCounterRaELCo)
112 {
113     sc_in<sc_logic> rst, clk, cen, pld;
114     sc_in<sc_logic> ci;
115     sc_out<sc_logic> co;
116     sc_in<sc_lv<8>> parin;
117     sc_out<sc_lv<8>> cntout;
118
119     SC_CTOR(uCounterRaELCo)
120     {
121         SC_METHOD(counting);
122         sensitive << rst << clk;
123         SC_METHOD(carrying);
124         sensitive << ci << cntout;
125     }
126     void counting();
127     void carrying();
128 };
129
130 // rShifterRaEL: right Shiftregister w/ asyn Reset, shift Enable, parallel Load
131 SC_MODULE(rShifterRaEL)
132 {
133     sc_in<sc_logic> rst, clk, sen, pld;
134     sc_in<sc_logic> sin;
135     sc_in<sc_lv<8>> parin;
136     sc_out<sc_lv<8>> shftout;
137
138     void shifting();
139
140     SC_CTOR(rShifterRaEL)
141     {
142         SC_METHOD(shifting);
143         sensitive << rst << clk;
144     }
145 };
146
147 // Memory: Templatized memory w/ datain, dataout, |chip-select, read-write-bar
148 template<int ADDRESS, int WORD_LENGTH>
149 class Memory : public sc_module {
150 public:
```

Shift Register

The screenshot shows a SystemC IDE interface with multiple tabs open. The tabs include `partsLibrary.h`, `handleMemory_TB.h`, `handleMemory.cpp`, `handleMemory.h`, and `partsLibrary.cpp`. A yellow callout box points to the `partsLibrary.cpp` tab, which contains the following code:

```
89 void rShifterRaEL::shifting()
90 {
91     if (rst == '1')
92     {
93         shftout = "00000000";
94     }
95     else if (clk->event() && (clk == '1'))
96     {
97         if (pld == '1') shftout = parin;
98         else if (sen == '1') shftout =
99             (sin, shftout->read().range(7, 1));
100    }
101}
102
103 void nBitAdder_channelSpecific::adding()
104 {
105     sc_lv<9> res;
106     res = ain.read().to_uint() + bin.read().to_uint()
107     + ci.read().value();
108     addout = res.range(7, 0);
109     co = res[8];
110 }
```

partsLibrary.cpp

RT Level SystemC

- Taking Off From c++
- SystemC Modeling
- Simulation Environment
- Utilities for HDL Orientation
- Sequential Modeling and Timing
- SystemC FSM Modeling
- Components for RT Level Design
- A Configurable Memory

A Configurable Memory

Use Template for Configurability

Use resolved to have multiple sources

Memory Array

partsLibrary.h

These parameters should be defined at comipletime

```
handleMemory.cpp handleMemory_TB.h partsLibrary.cpp partsLibrary.h ✘ handleMemory.h
Datapath Components → Memory<ADDRESS, WORD_LENGTH> → memdump()
148 // Memory: Templatized memory w/ datain, dataout, chip-select, read-write-bar
149 template<int ADDRESS, int WORD_LENGTH>
150 class Memory : public sc_module {
151     public:
152         sc_in<rv><ADDRESS> addr;
153         sc_in<rv><WORD_LENGTH> datain;
154         sc_out<rv><WORD_LENGTH> dataout;
155         sc_in<resolved> cs, rwbar;
156
157         sc_time dumpTime;
158         char* dumpFile;
159
160         int addrSpace;
161         sc_uint<WORD_LENGTH> *mem;
162
163         void meminit();
164         void memread();
165         void memwrite();
166         void memdump();
167
168         SC_HAS_PROCESS(Memory);
169         Memory(sc_module_name, sc_time, char*);
170     };
171
172     template<int ADDRESS, int WORD_LENGTH>
173     Memory<ADDRESS, WORD_LENGTH>::Memory(sc_module_name, sc_time dt, char* df)
174     {
175         dumpTime = dt;
176         dumpFile = df;
177
178         addrSpace = int(pow(2, ADDRESS));
179         mem = new sc_uint<WORD_LENGTH>[addrSpace];
180
181         SC_THREAD(meminit);
182         SC_METHOD(memread);
183         sensitive << addr << cs << rwbar;
184         SC_METHOD(memwrite);
185         sensitive << addr << datain << cs << rwbar;
186         SC_THREAD(memdump);
187     }
188 }
```

Resolved value

Z is the weakest and
X is the strongest

		X	0	1	Z
Driving Value 1	X	X	X	X	X
	0	X	0	X	0
1	X	X	1	1	
Z	X	0	1	Z	

A Configurable Memory

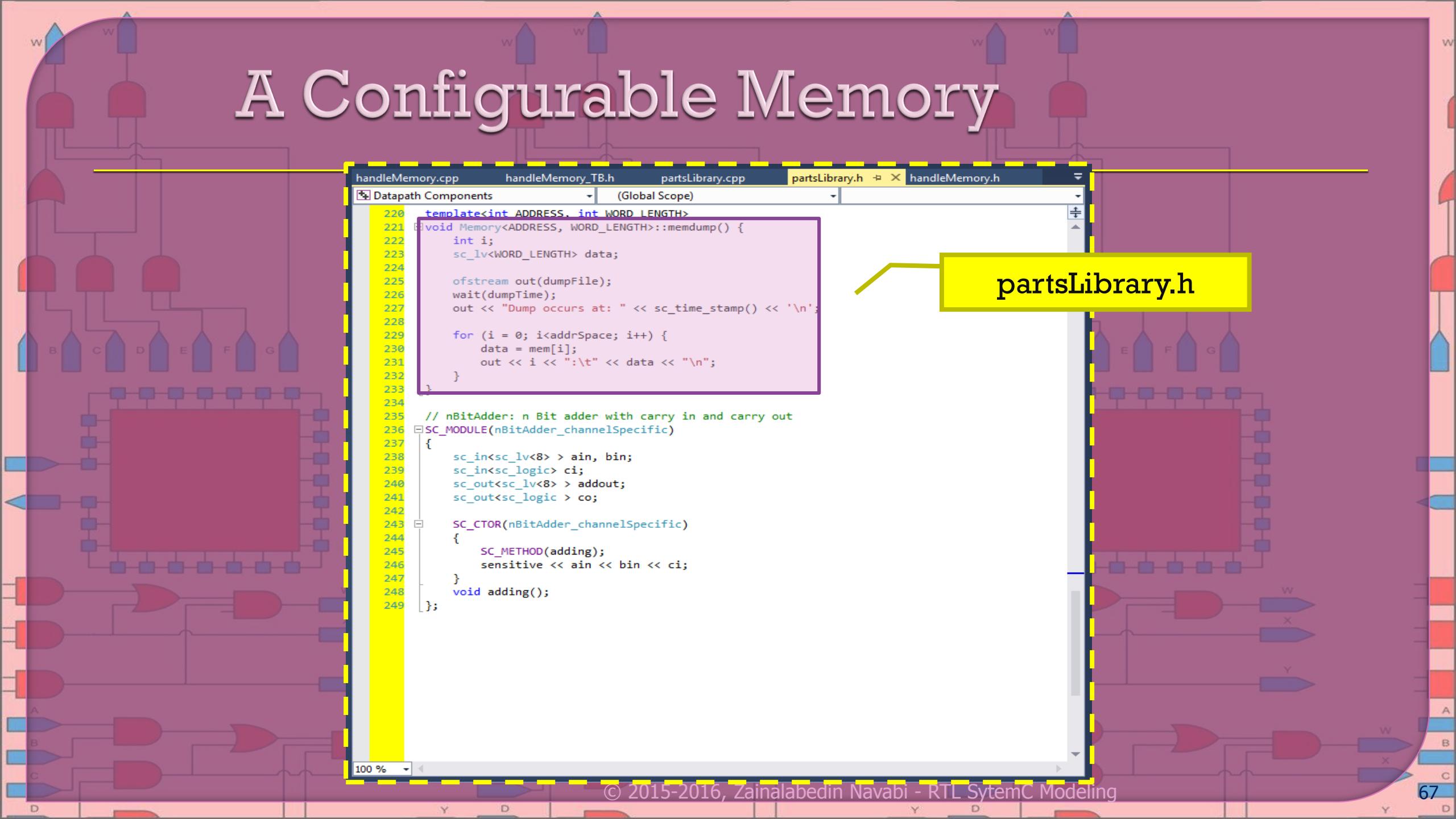
Type Conversion from
sc_rv to sc_uint

partsLibrary.h

Type Conversion from
sc_uint to sc_rv

```
189 template<int ADDRESS, int WORD_LENGTH>
190 void Memory<ADDRESS, WORD_LENGTH>::meminit() {
191     int i;
192     for (i = 0; i<addrSpace; i++) {
193         mem[i] = i;
194         cout << "Init at: " << i << " writes: " << i << '\n';
195     }
196 }
197
198 template<int ADDRESS, int WORD_LENGTH>
199 void Memory<ADDRESS, WORD_LENGTH>::memwrite() {
200     sc_uint<ADDRESS> ad;
201     if (cs->read() == '1') {
202         if (rwbar->read() == '0') {
203             ad = addr;
204             mem[ad] = datain;
205         }
206     }
207 }
208
209 template<int ADDRESS, int WORD_LENGTH>
210 void Memory<ADDRESS, WORD_LENGTH>::memread() {
211     sc_uint<ADDRESS> ad;
212     if (cs->read() == '1') {
213         if (rwbar->read() == '1') {
214             ad = addr;
215             dataout = mem[ad];
216         }
217     }
218 }
219
220 template<int ADDRESS, int WORD_LENGTH>
221 void Memory<ADDRESS, WORD_LENGTH>::memdump() {
222     int i;
223     sc_lv<WORD_LENGTH> data;
224
225     ofstream out(dumpFile);
226     wait(dumpTime);
227     out << "Dump occurs at: " << sc_time_stamp() << '\n';
228
229     for (i = 0; i<addrSpace; i++) {
```

A Configurable Memory



The image shows a complex digital circuit diagram in the background, featuring numerous logic gates (AND, OR, NOT), multiplexers, and a large central RAM-like component. A yellow dashed rectangle highlights a specific area of the code editor, which contains SystemC code for memory dump functionality and a nBitAdder module.

```
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';

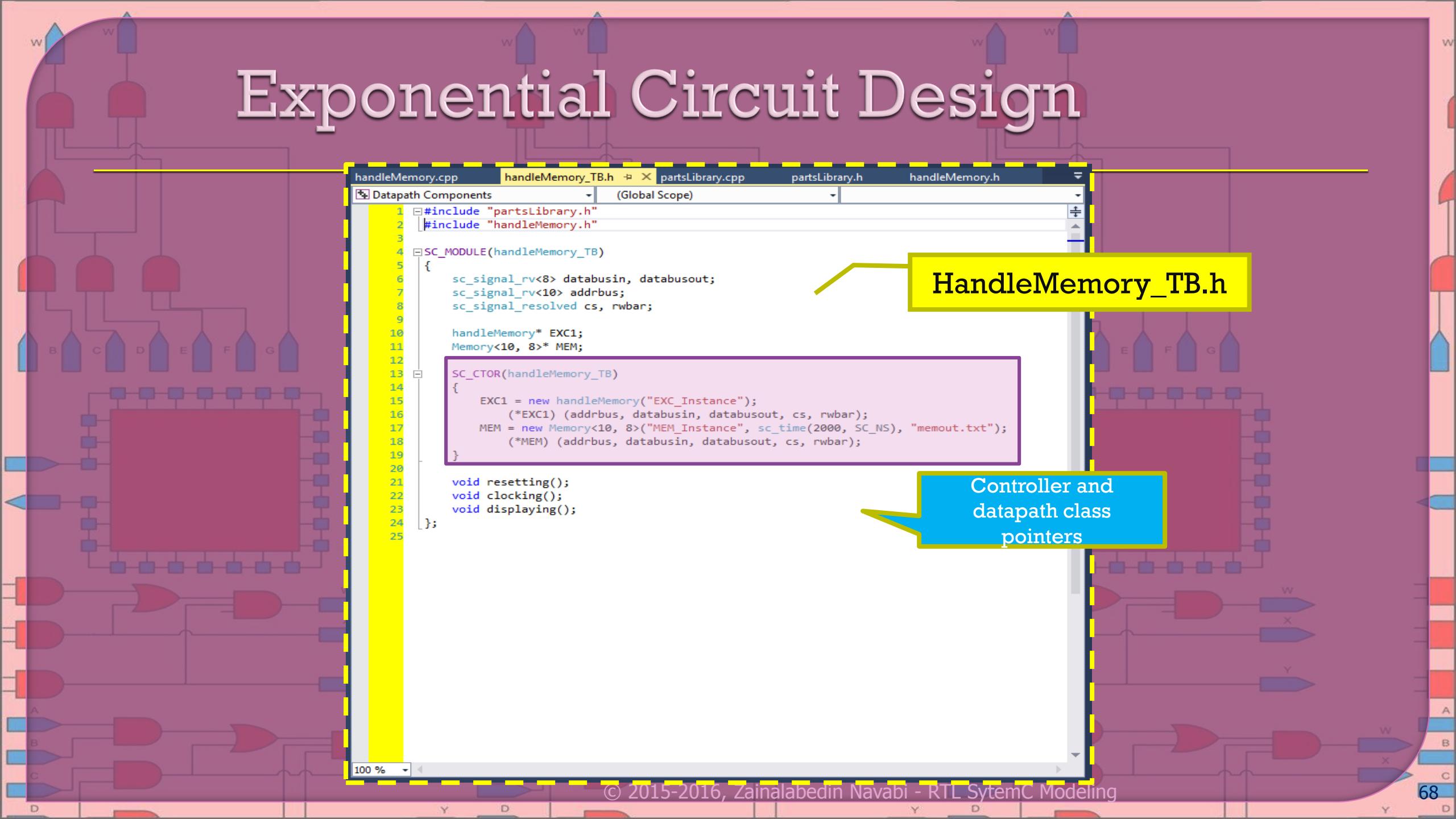
    for (i = 0; i<addrSpace; i++) {
        data = mem[i];
        out << i << ":" << data << "\n";
    }
}

// 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(add);
        sensitive << ain << bin << ci;
    }
    void add();
};
```

partsLibrary.h

Exponential Circuit Design



```
handleMemory.cpp handleMemory_TB.h partsLibrary.cpp partsLibrary.h handleMemory.h
Datapath Components (Global Scope)
1 #include "partsLibrary.h"
2 #include "handleMemory.h"
3
4 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

RT SystemC Example

- RTL design example 2: Exponential Circuit

- Exponential Circuit Datapath
- Exponential Circuit Controller

RTL Example 2: Exponential Circuit

- The circuit calculates e^x using Taylor expansion.
- The input is an 8-bit fixed-point number.
- The output is a 10-bit fixed-point number including 2 integer bits and 8 fractional bits.
- The circuit receives x as the input with the pulse on the start signal.
- The calculation continues for 8 iterations.
- When the result becomes ready, done signal will be issued.

Input-output Range

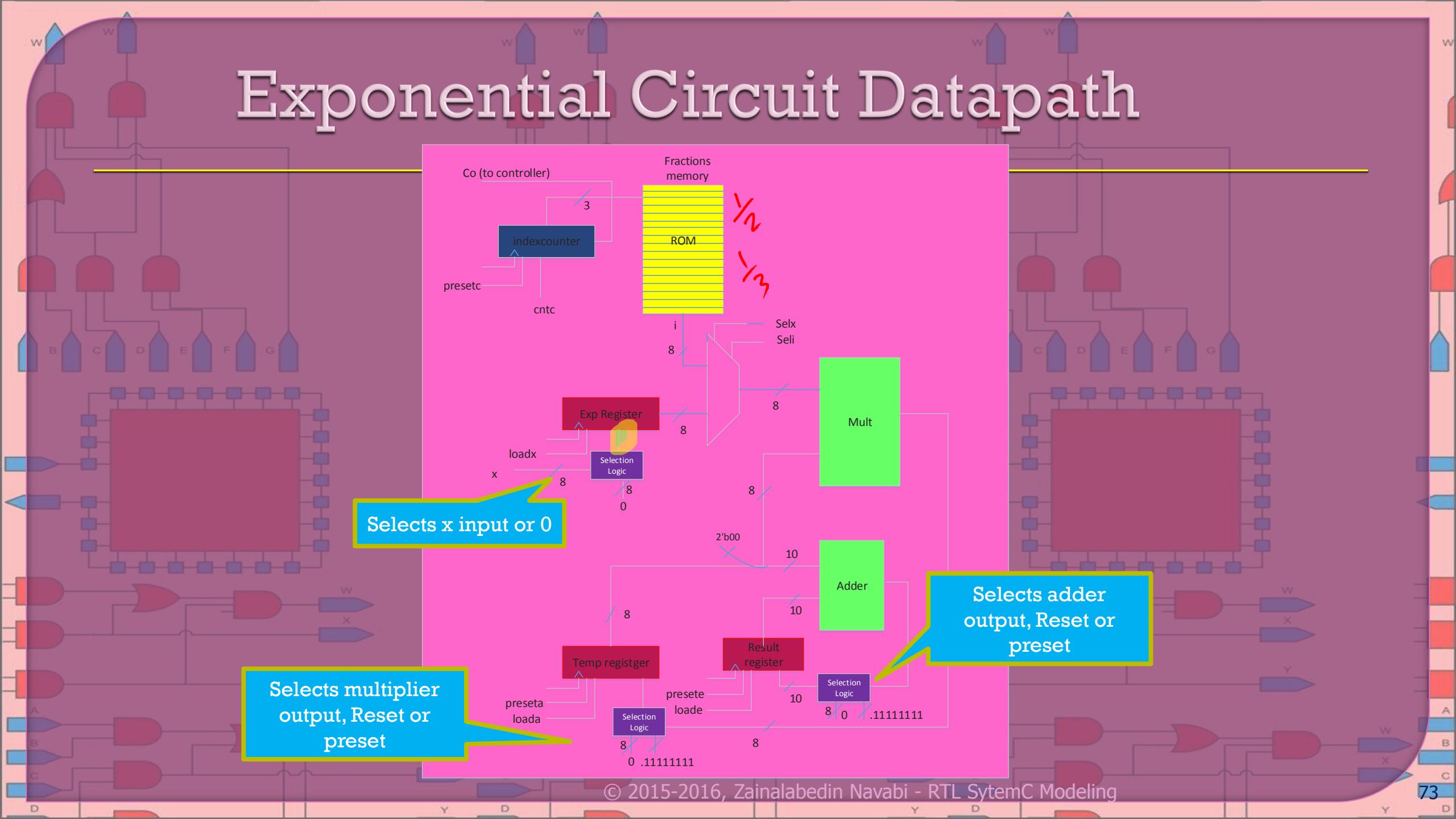
- With 8 bit input size, x is in the range between 0.00000000 for the smallest and 0.11111111 for the largest value.
- Smallest output = 1 when e^0
- Largest output = 10.1010111 (2.68357) when e^1

Taylor Series

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

```
e = 1;  
a = 1;  
for( i = 1; i < n; i++ )  
    a = a * x * x * ( 1 / i );  
    e = e + a;  
}
```

Exponential Circuit Datapath



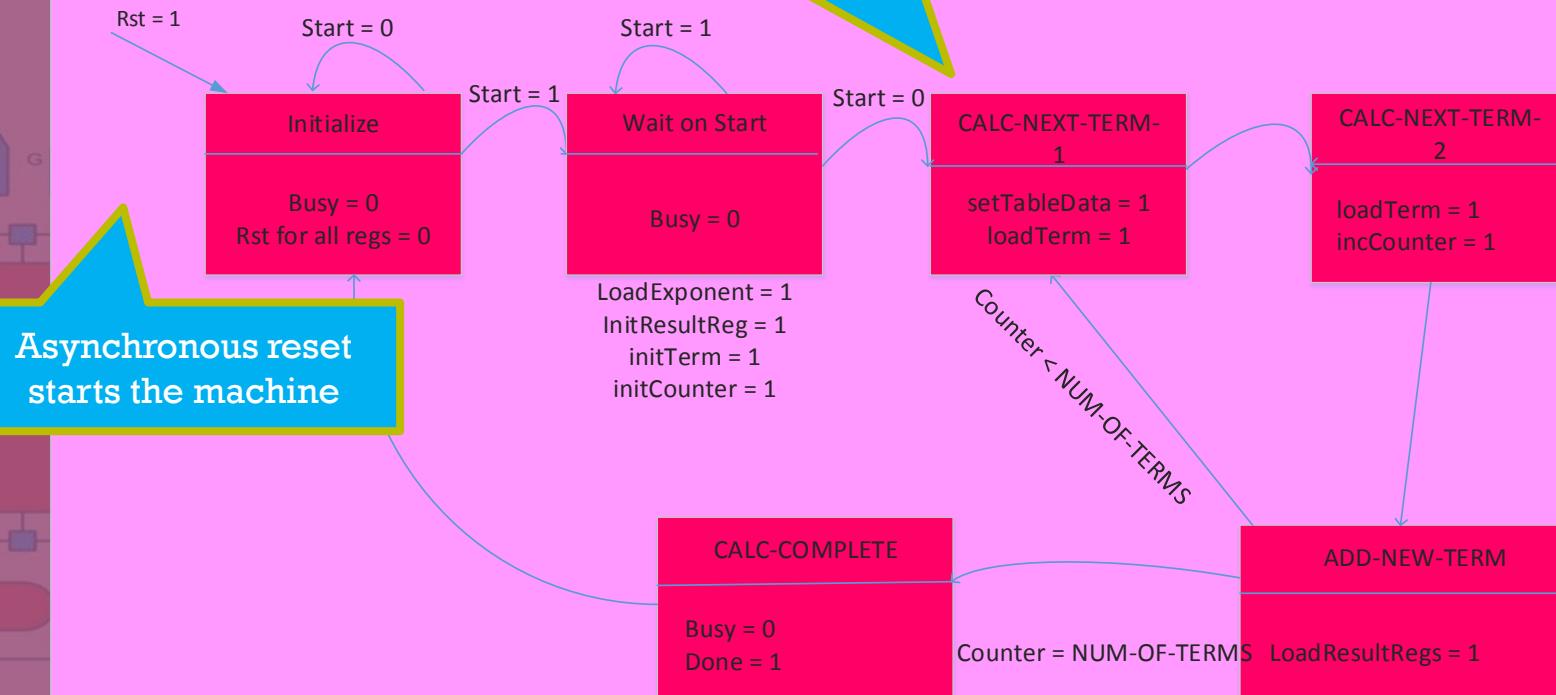
Exponential Circuit Controller

Exponential Circuit Controller

Asynchronous reset
starts the machine

Multiplying the termreg
with fractional memory,
updating termreg

Adding the new termreg
contents to result reg



Summary

- Taking Off From c++
- SystemC Modeling
- Simulation Environment
- Utilities for HDL Orientation
- Sequential Modeling and Timing
- SystemC FSM Modeling
- Components for RT Level Design
- A configurable Memory