

Chapter 4

RTL SystemC Modeling

Zainalabedin Navabi

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

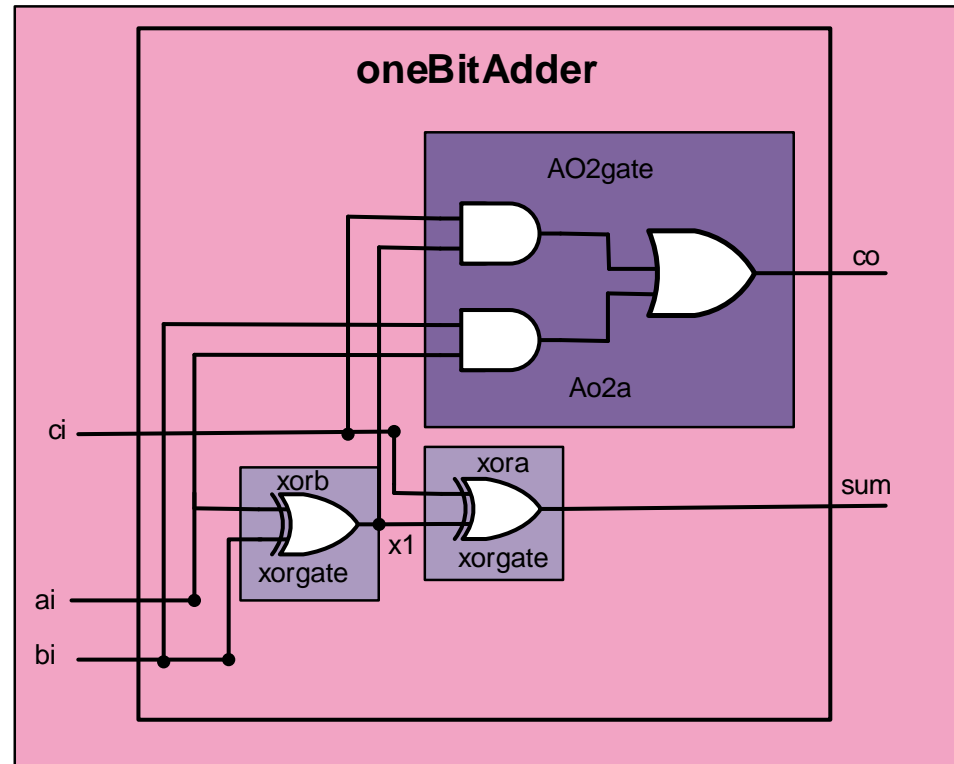
SystemC Modeling

- 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

- Taking Off From C++

C++ modeling of 1-bit Adder

- Hierarchical Structure Hardware



C++ modeling of 1-bit Adder

- Components

```
1 #include "classVectorPrimitives.h"
2 #include <string>
3 using namespace std;
4
5 class XORgate {
6     bus *i1, *i2, *o1;
7 public:
8     XORgate(bus& a, bus& b, bus& xo) :
9         i1(&a), i2(&b), o1(&xo)
10     {
11         o1->fill('X');
12     }
13     ~XORgate();
14     void evl();
15 };
16
17 class AO2gate { ... };
18
19 class oneBitAdder {
20     bus *i1, *i2, *i3,
21         *o1, *o2;
22
23     XORgate* XORA;
24     XORgate* XORB;
25     AO2gate* AO2a;
26
27     bus x1;
28
29 public:
30     oneBitAdder(bus& ai, bus& bi, bus& ci, bus& co, bus& su);
31     ~oneBitAdder();
32     void evl();
33 };
```

simpleHierarchical.h

All ports are bus pointers

constructor

evl() function

Resembles standard hardware description but lacks the concurrency

- Taking Off From C++

C++ modeling of 1-bit Adder

- Components

Use overloaded "=="

simpleHierarchical.cpp

```
classVectorPrimitives.h  simpleHierarchical.cpp  simpleHierarchical.h  simpleHierarchicalTB.cpp
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
```

- Taking Off From C++

C++ modeling of 1-bit Adder

- Top-Level

The screenshot shows a C++ code editor with the following code in `simpleHierarchical.h`:

```
1 #include "classVectorPrimitives.h"
2 #include <string>
3 using namespace std;
4
5 class XORgate {
6     bus *i1, *i2, *o1;
7 public:
8     XORgate(bus& a, bus& b, bus& xo) :
9         i1(&a), i2(&b), o1(&xo)
10    {
11        o1->fill('X');
12    }
13    ~XORgate();
14    void evl();
15 };
16
17 class AO2gate { ... };
18
19 class oneBitAdder {
20     bus *i1, *i2, *i3,
21         *o1, *o2;
22
23     XORgate* XORa;
24     XORgate* XORb;
25     AO2gate* AO2a;
26
27     bus x1;
28
29 public:
30     oneBitAdder(bus& ai, bus& bi, bus& ci, bus& co, bus& su);
31     ~oneBitAdder();
32     void evl();
33 };
34
35
```

A callout box labeled "Internal signal" points to the `bus x1;` declaration in the `oneBitAdder` class. Another callout box labeled `simpleHierarchical.h` points to the file name in the editor's title bar.

- Taking Off From C++

C++ modeling of 1-bit Adder

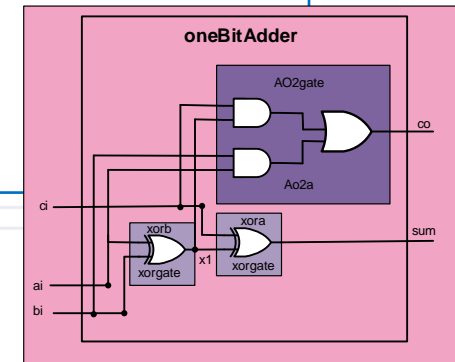
Top-Level

```
classVectorPrimitives.h  simpleHierarchical.cpp  simpleHierarchical.h  simpleHierarchicalTB.cpp
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 }
```

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



- Taking Off From C++

C++ modeling of 1-bit Adder

Testbench

The image shows a code editor window with the file `simpleHierarchicalTB.cpp` open. The code is a testbench for a 1-bit adder. It includes the header `simpleHierarchical.h` and defines a `main()` function. Inside `main()`, several local buses are declared: `aData`, `bData`, `cData`, `cOut`, and `sOut`. A `oneBitAdder` object is instantiated as `FA1`. The testbench enters a loop where it prompts the user to enter values for `a`, `b`, and `c`, then calls the `evl()` function on `FA1` to compute the sum and carry. The results are printed, and the user is prompted to continue or stop the simulation.

Annotations in the image highlight specific parts of the code:

- Local buses:** Points to the declarations of `aData`, `bData`, `cData`, `cOut`, and `sOut`.
- Top-Level instantiation:** Points to the line `oneBitAdder* FA1 = new oneBitAdder(aData, bData, cData, cOut, sOut);`.
- Calling evl() function:** Points to the line `FA1->evl();`.

```
1  #include "simpleHierarchical.h"
2
3  int main()
4  {
5      int ij;
6
7      bus aData;
8      bus bData;
9      bus cData;
10     bus cOut;
11     bus sOut;
12
13     oneBitAdder* FA1 = new oneBitAdder(aData, bData, cData, cOut, sOut);
14
15     do{
16         cout << "Enter a, b, c: ";
17         cin >> aData >> bData >> cData;
18
19         FA1->evl();
20
21         cout << "Carry and Sum: " << cOut << " " << sOut << "\n";
22
23         cout << "\n" << "Continue (0 or 1)?"; cin >> ij;
24     } while (ij >0);
25 }
26
```

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

The screenshot shows a code editor with three files: `simpleHierarchicalTB.cpp`, `simpleHierarchical.cpp`, and `simpleHierarchical.h`. The `simpleHierarchical.h` file is open, showing the following code:

```
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
```

Annotations and their corresponding code elements:

- All ports are `sc_port`**: Points to the `sc_port` declarations in the `XORgate` and `oneBitAdder` classes.
- `CS_CTOR` macro allows inline definition of constructor**: Points to the `SC_CTOR(XORgate)` macro in the `XORgate` class.
- `evl()` function is registered as `sc_method` that also run once at the beginning of the simulation**: Points to the `SC_METHOD(ev1);` line in the `XORgate` class.
- Is inherited from `sc_module` which contains ports, channels, concurrent processes**: Points to the `public sc_module {` line in the `XORgate` class.
- `sc_logic` is 4 value logic system: `sc_logic_0`, `sc_logic_1`, `sc_logic_z`, `sc_logic_x`**: Points to the `sc_logic` type in the `sc_signal` declarations.
- `evl()` wakes up when an event occurs on sensitivity list**: Points to the `sensitive << i1 << i2;` line in the `XORgate` class.

SystemC modeling of 1-bit Adder

Components

Operators are overloaded for
sc_logic

Other form: (*i1).read()

```
simpleHierarchicalTB.cpp  simpleHierarchical.cpp  × simpleHierarchical.h
SC Start (Global Scope)
1 #include "simpleHierarchical.h"
2
3 void XORgate::evl()
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::evl()
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 }
28
```

simpleHierarchical.cpp

SystemC modeling of 1-bit Adder

Top-Level

```
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
28 class oneBitAdder : public sc_module {
29 public:
30     sc_port<sc_signal_in_if<sc_logic>, 1> i1, i2, i3;
31     sc_port<sc_signal_out_if<sc_logic>, 1> o1, o2;
32
33     sc_signal<sc_logic> x1;
34
35     XORgate* XORa;
36     XORgate* XORb;
37     AO2gate* AO2a;
38
39     SC_HAS_PROCESS(oneBitAdder);
40     oneBitAdder(sc_module_name);
41 };
```

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

```
1 #include "simpleHierarchical.h"
2
3 void XORgate::evl()
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::evl()
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 }
28
```

simpleHierarchical.cpp

Submodule instantiation and binding by position

No need to handle ordering

oneBitAdder

Diagram illustrating the 1-bit adder circuit structure, showing inputs (ai, bi, ci), intermediate signals (x1, x2), and outputs (sum, co).

SystemC modeling of 1-bit Adder

Testbench

The image shows a screenshot of a SystemC testbench for a 1-bit adder. The main window displays the code for `simpleHierarchicalTB.cpp`. The code is annotated with several callouts:

- function `sc_main` in the global namespace**: Points to the `int sc_main(int argc, char **argv)` function signature.
- Local signals**: Points to the signal declarations: `sc_signal<sc_logic> aData;`, `sc_signal<sc_logic> bData;`, `sc_signal<sc_logic> cData;`, `sc_signal<sc_logic> cOut;`, and `sc_signal<sc_logic> sOut;`.
- Top-Level instantiation**: Points to the instantiation of the `oneBitAdder` component: `oneBitAdder* FA1 = new oneBitAdder("FA1_instance");` and `(*FA1) (aData, bData, cData, cOut, sOut);`.
- VCD file and tracing signals**: Points to the VCD file creation and tracing calls: `sc_trace_file* VCDFile;`, `VCDFile = sc_create_vcd_trace_file("simpleHierarchical");`, and `sc_trace(VCDFile, aData, "aIn");`, `sc_trace(VCDFile, bData, "bIn");`, `sc_trace(VCDFile, cData, "carryIn");`, `sc_trace(VCDFile, cOut, "carryOut");`, and `sc_trace(VCDFile, sOut, "sumOut");`.
- Data generation**: Points to the `return 0;` statement.

On the right side, a separate window shows the `simpleHierarchicalTB.cpp` file with a loop for data generation:

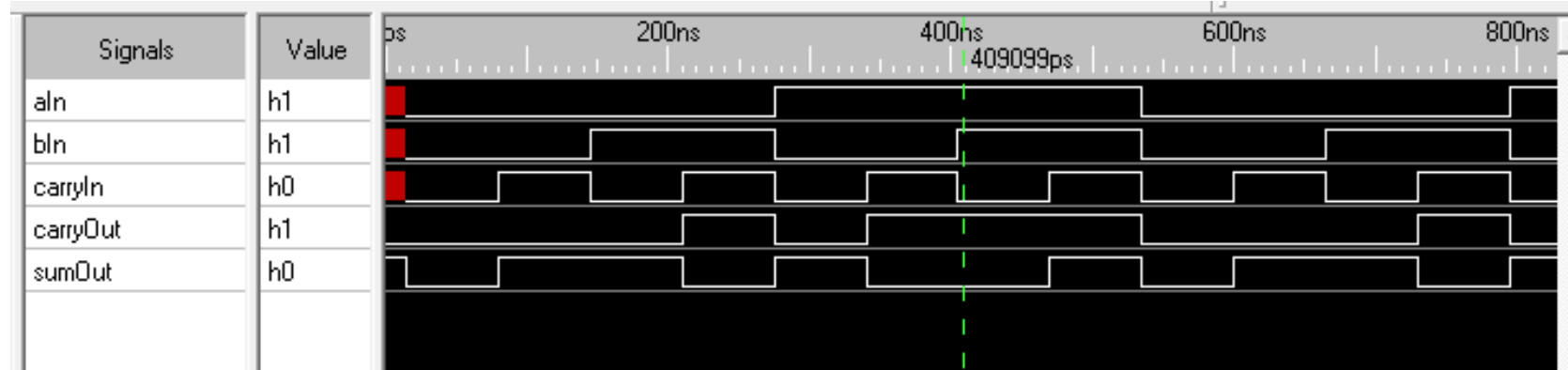
```
21 sc_int<3> intData;
22 sc_lv<3> abcData;
23
24 sc_start(15, SC_NS);
25
26 intData = 0;
27 int ij=0;
28 do {
29     abcData = intData;
30     aData = abcData[2];
31     bData = abcData[1];
32     cData = abcData[0];
33     sc_start(15, SC_NS);
34     intData = intData + 1;
35     sc_start(50, SC_NS);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

A callout box explains the `sc_start` function: **`sc_start` advances the simulation as much as specified time of its argument**. This callout points to the `sc_start(15, SC_NS);` and `sc_start(50, SC_NS);` calls in the loop.

SystemC modeling of 1-bit Adder

◉ Waveform in a VCD viewer

- Generated VCD file can be 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

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

```
simpleHierarchical.cpp  simpleHierarchical.h  (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
```

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

```
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(ev1);
14        sensitive << i1 << i2;
15    }
16    void ev1();
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 };
42
43 }
```

Gates with propagation delay

simpleHierarchical.h

Hierarchical timed design for Serial Adder

Timed Components

- XOR Gate

SC_THREAD is meant 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

```
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 }
```

simpleHierarchical.cpp

Hierarchical timed design for Serial Adder

Timed Components

- 1-bit Adder

The screenshot shows a SystemC code editor with the following code in `simpleHierarchical.h`:

```
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(ev1);
14        sensitive << i1 << i2;
15    }
16    void ev1();
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 };
```

Annotations in the image:

- simpleHierarchical.h**: Points to the header file name in the editor tab.
- Passing delay values**: Points to the `sc_time(0.5, SC_NS)` and `sc_time(0.4, SC_NS)` arguments in the submodule instantiation.
- Submodule instantiation and binding by position**: Points to the `new XORgate("xor_insta", ...)` and `new XORgate("xor_instb", ...)` lines.

Hierarchical timed design for Serial Adder

Timed Components

- D-Flip Flop

```
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
97 };
98
```

Sensitive to clock and reset

simpleHierarchical.h

Hierarchical timed design for Serial Adder

Timed Components

- D-Flip Flop

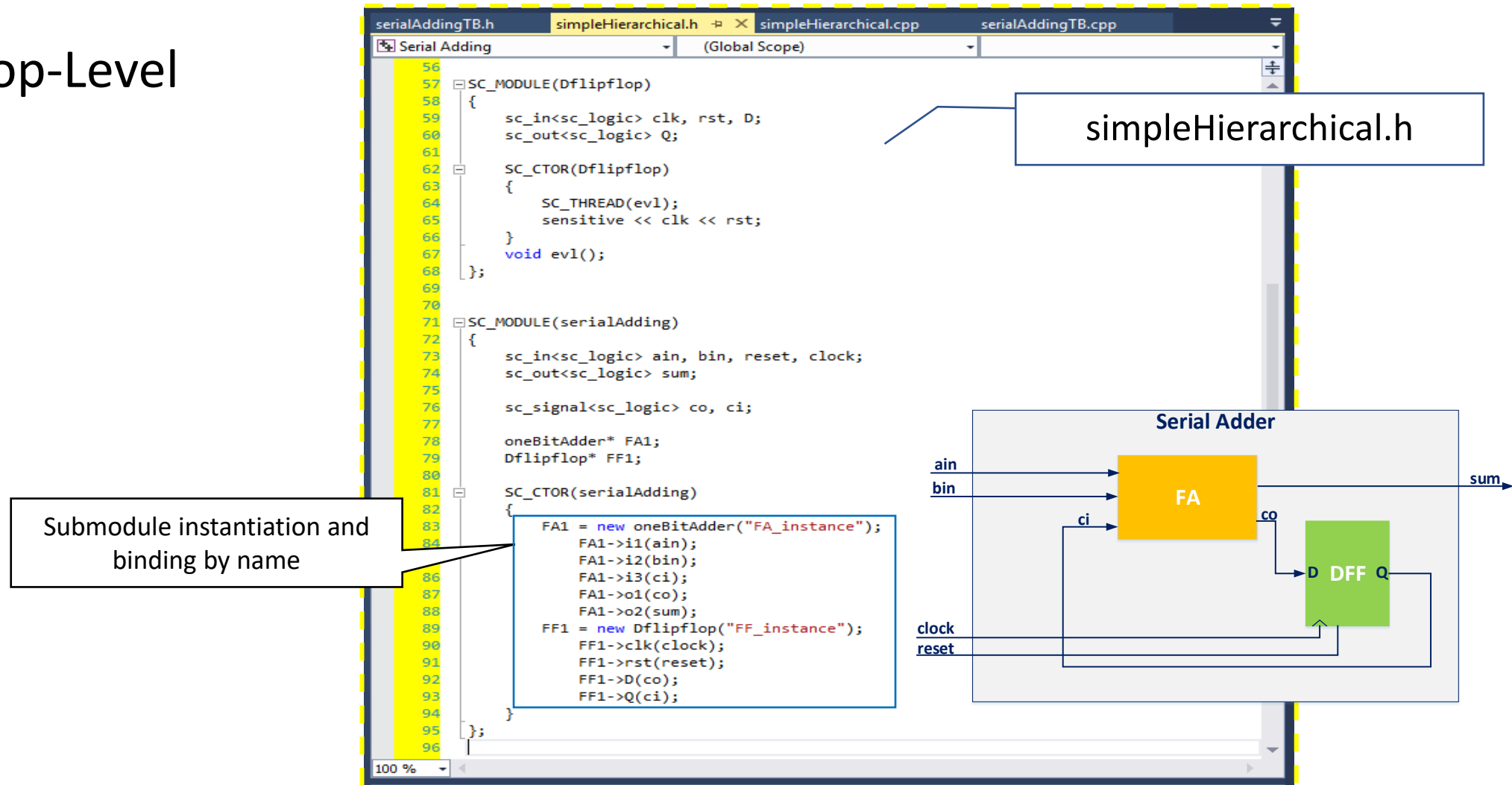
The screenshot shows the `simpleHierarchical.cpp` file in a code editor. The `Dflipflop::evl()` method is highlighted, showing a `while (true)` loop. Inside the loop, there is a conditional statement: `if (rst == SC_LOGIC_1) { wait(0.6, SC_NS); Q = SC_LOGIC_0; }`. This is followed by an `else if` clause: `else if ((clk->event() && (clk == '1')) { wait(0.6, SC_NS); Q = D; }`. The `wait()` method is used to suspend the process until a specified time or event occurs. The `event()` method is used to check for a signal event.

Annotations:

- 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



Hierarchical timed design for Serial Adder

● Testbench

Testbench as a **SC_MODULE**

Local signals

Top-Level instantiation

Data generation

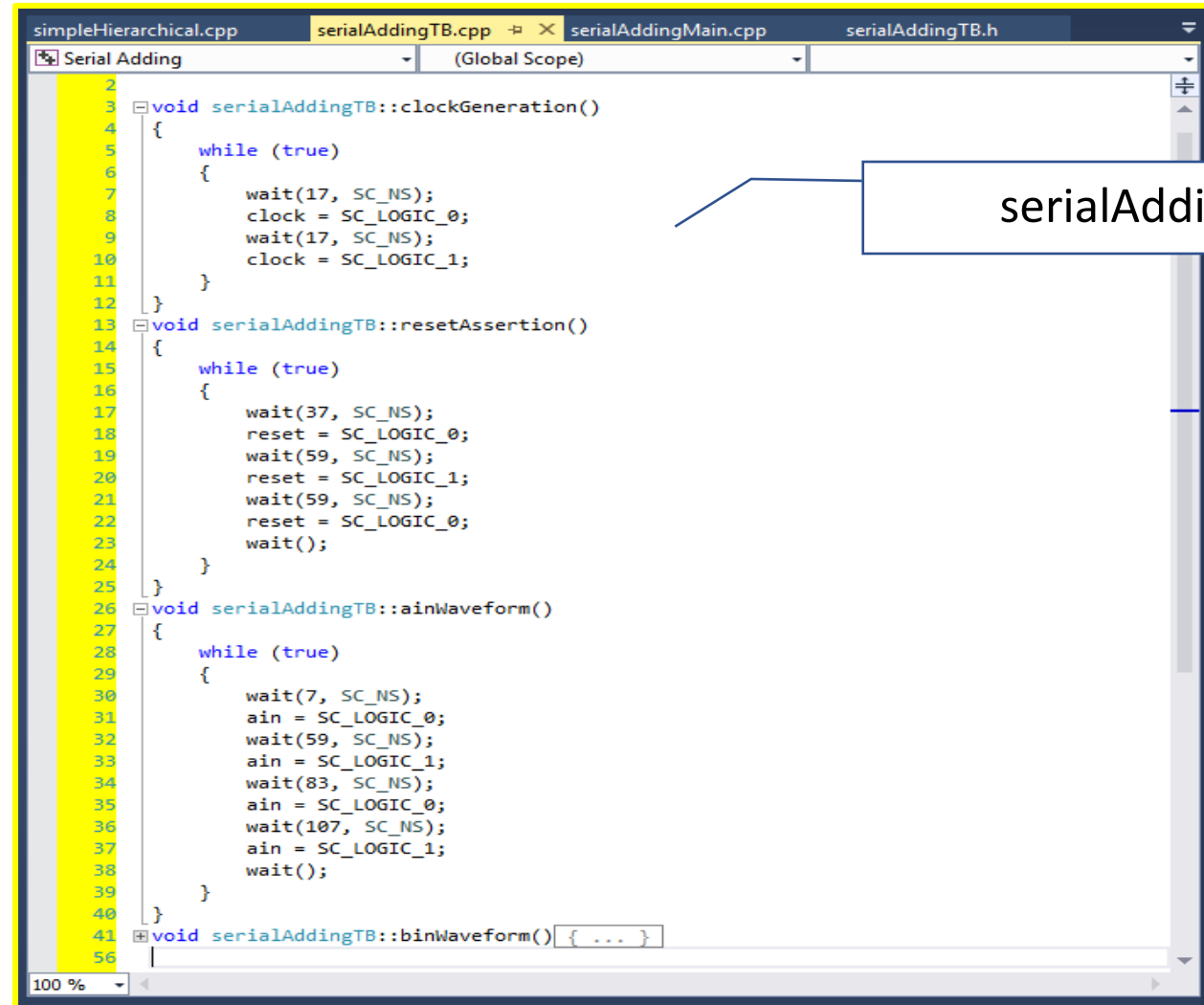
```
simpleHierarchical.cpp  serialAddingTB.cpp  serialAddingMain.cpp  serialAddingTB.h
Serial Adding (Global Scope)
1  #include "simpleHierarchical.h"
2
3  SC_MODULE(serialAddingTB)
4  {
5      sc_signal<sc_logic> ain, bin, reset, clock;
6      sc_signal<sc_logic> sum;
7
8      serialAdding* uut;
9
10     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

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



```
simpleHierarchical.cpp serialAddingTB.cpp 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() { ... }
56
```

serialAddingTB.cpp

Hierarchical timed design for Serial Adder

● sc_main

function *sc_main* in the
global namespace

Testbench instantiation

VCD file and tracing signals

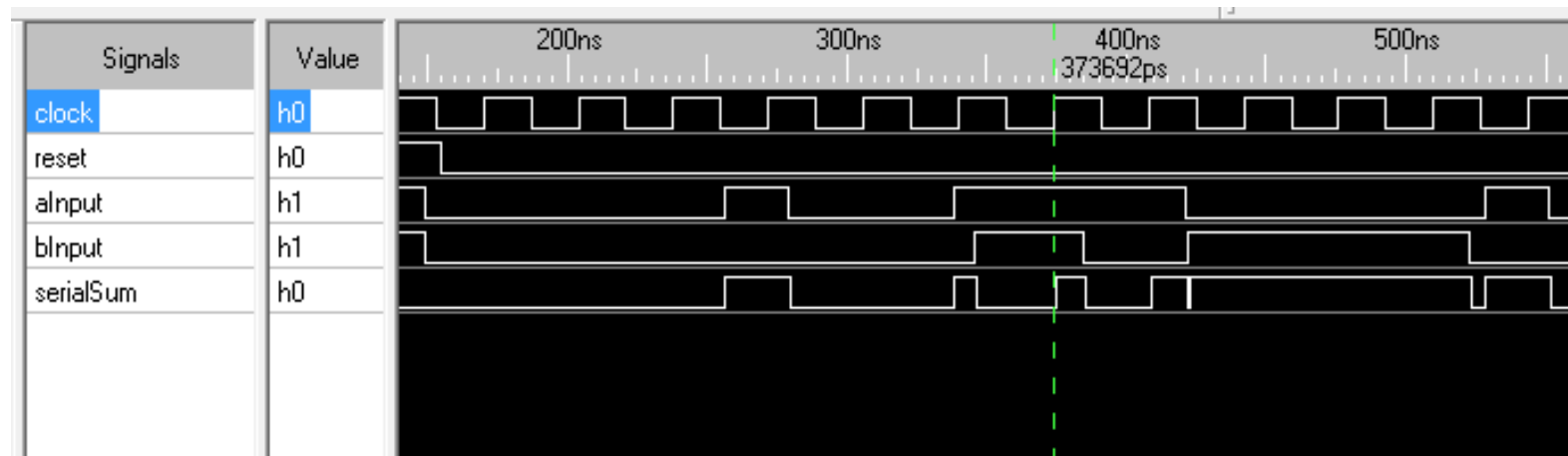
Access to internal signals

serialAddingMain.cpp

```
simpleHierarchical.cpp  serialAddingTB.cpp  serialAddingMain.cpp  serialAddingTB.h
Serial Adding (Global Scope)
1 #include "serialAddingTB.h"
2
3 int sc_main(int argc, char **argv)
4 {
5     serialAddingTB* TOP = new serialAddingTB("serialAddingTB_instance");
6
7     sc_trace_file* VCDFile;
8     VCDFile = sc_create_vcd_trace_file("serialAdding");
9     sc_trace(VCDFile, TOP->ain, "aInput");
10    sc_trace(VCDFile, TOP->bin, "bInput");
11    sc_trace(VCDFile, TOP->reset, "reset");
12    sc_trace(VCDFile, TOP->clock, "clock");
13    sc_trace(VCDFile, TOP->sum, "serialSum");
14
15    sc_trace(VCDFile, TOP->UUT->FA1->i1, "i1_fulladder");
16    sc_trace(VCDFile, TOP->UUT->FA1->i2, "i2_fulladder");
17    sc_trace(VCDFile, TOP->UUT->FA1->i3, "i3_fulladder");
18    sc_trace(VCDFile, TOP->UUT->FA1->o1, "o1_fulladder");
19    sc_trace(VCDFile, TOP->UUT->FA1->o2, "o2_fulladder");
20
21    sc_start(4000, SC_NS);
22    return 0;
23 }
24
25
```

Hierarchical timed design for Serial Adder

- ◉ 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

— 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>*

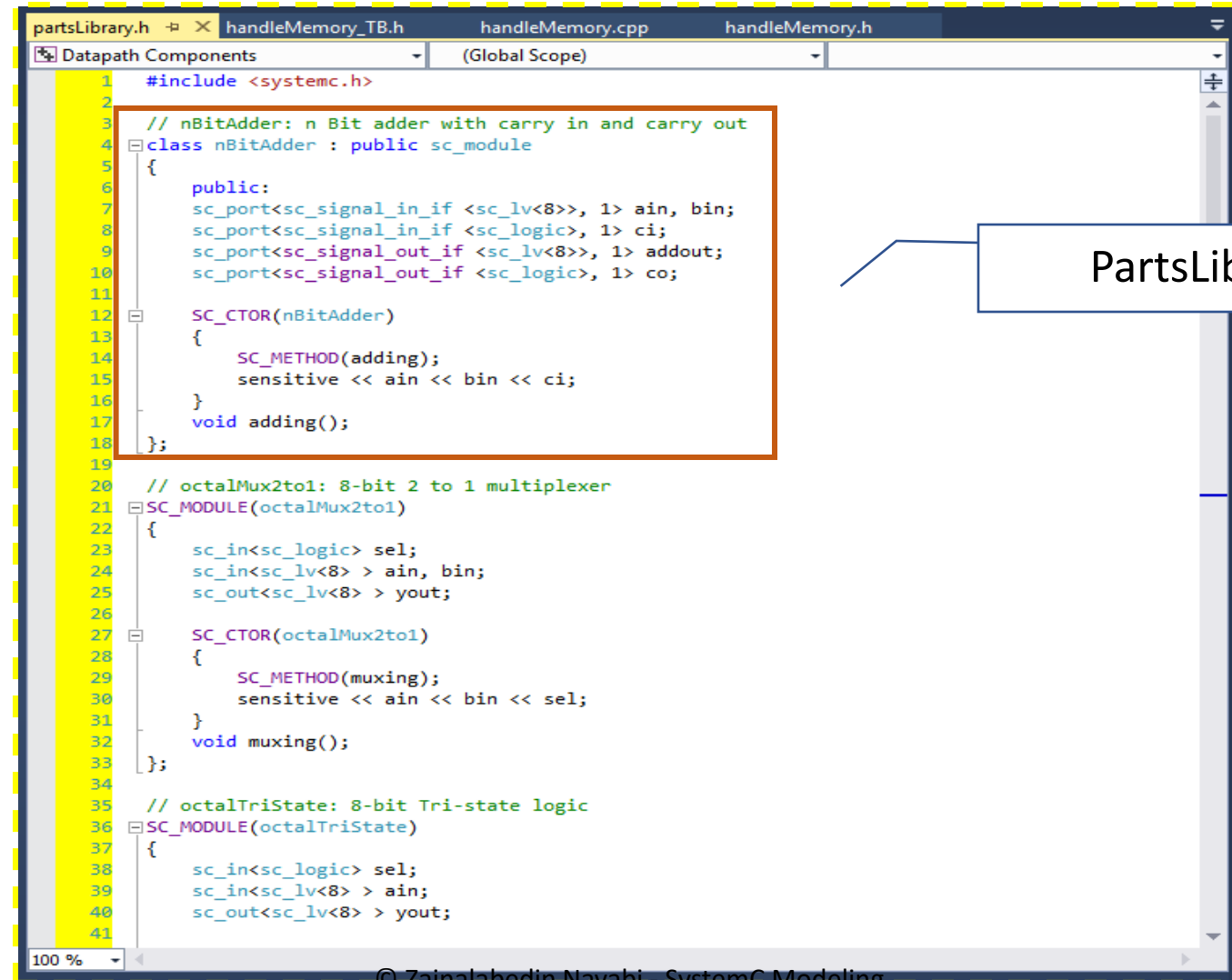
⦿ *sc_bit* is two value logic

- 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

Combinational

● Adder



```
1 #include <systemc.h>
2
3 // nBitAdder: n Bit adder with carry in and carry out
4 class nBitAdder : public sc_module
5 {
6     public:
7     sc_port<sc_signal_in_if<sc_lv<8>>, 1> ain, bin;
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 }
```

partsLibrary.h

Combinational

● Adder

The screenshot shows a C++ code editor with the following tabs: `partsLibrary.h`, `handleMemory_TB.h`, `handleMemory.cpp`, `handleMemory.h`, and `partsLibrary.cpp`. The `partsLibrary.cpp` file is open, showing the implementation of the `nBitAdder` class. The `adding()` method is highlighted with a red box. A callout box points to the `to_uint()` method call, stating: "Add operation is not defined for the logic type". Another callout box points to the `Value()` method call, stating: "To_uint() is only valid for vectors Value() is used for one bit". A third callout box points to the `partsLibrary.cpp` file name, stating: "partsLibrary.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 = "ZZZZZZZ";
}

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'))
    {
```

Combinational

● Adder

```
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     if (sel->read() == '1') yout->write(bin);
14     else yout->write(ain);
15 }
16
17 void octalTriState::selecting() {
18     if (sel == '1') yout = ain;
19     else yout = "ZZZZZZZZ";
20 }
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     {
```

variable

Use dereferencing -> because of using pointer ports

partsLibrary.cpp

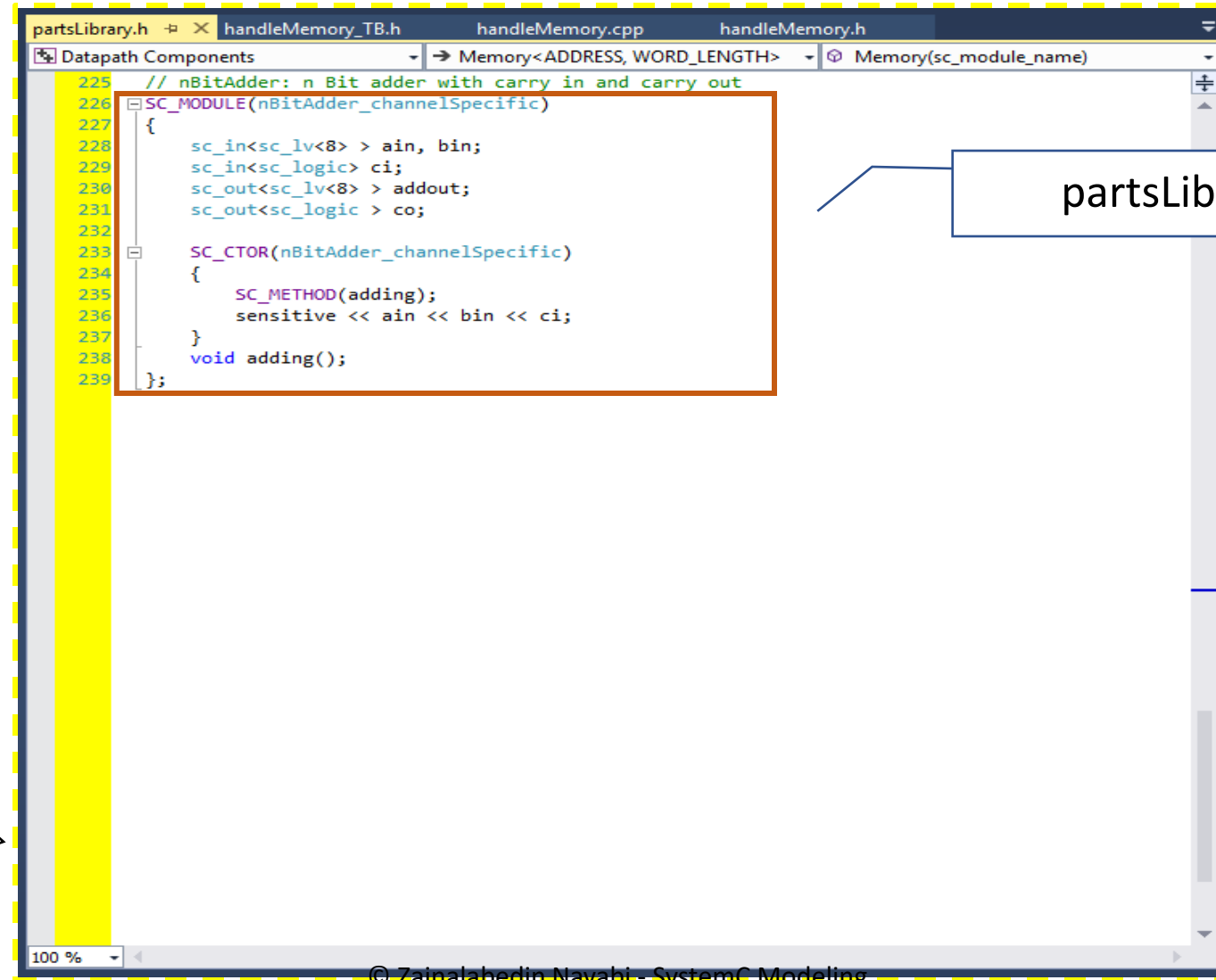
The assigned value to the signal is not available immediately. It will be updated after delta delay

Variables represent software-like variables with no timing

Signals and variables

Combinational

● Adder



```
225 // nBitAdder: n Bit adder with carry in and carry out
226
227 SC_MODULE(nBitAdder_channelSpecific)
228 {
229     sc_in<sc_lv<8> > ain, bin;
230     sc_in<sc_logic> ci;
231     sc_out<sc_lv<8> > addout;
232     sc_out<sc_logic> co;
233
234     SC_CTOR(nBitAdder_channelSpecific)
235     {
236         SC_METHOD(adding);
237         sensitive << ain << bin << ci;
238     }
239     void adding();
240 };
```

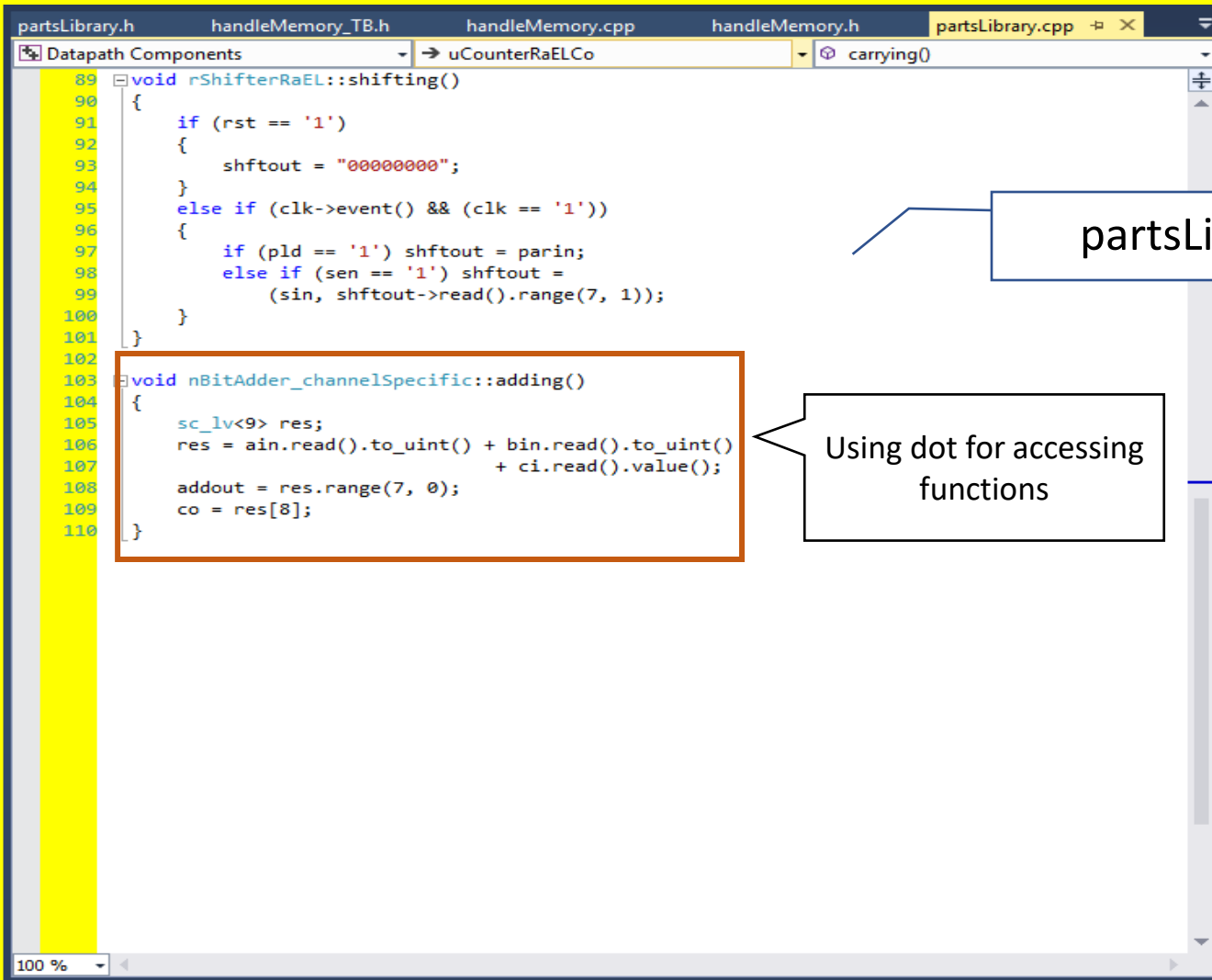
partsLibrary.h

Using sc_in
and sc_out

Components for RTL Design

Combinational

● Adder



The screenshot shows a code editor with several tabs: partsLibrary.h, handleMemory_TB.h, handleMemory.cpp, handleMemory.h, and partsLibrary.cpp. The 'Datapath Components' dropdown is set to 'uCounterRaELCo', and the 'carrying()' function is selected. The code in partsLibrary.cpp is as follows:

```
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 }
```

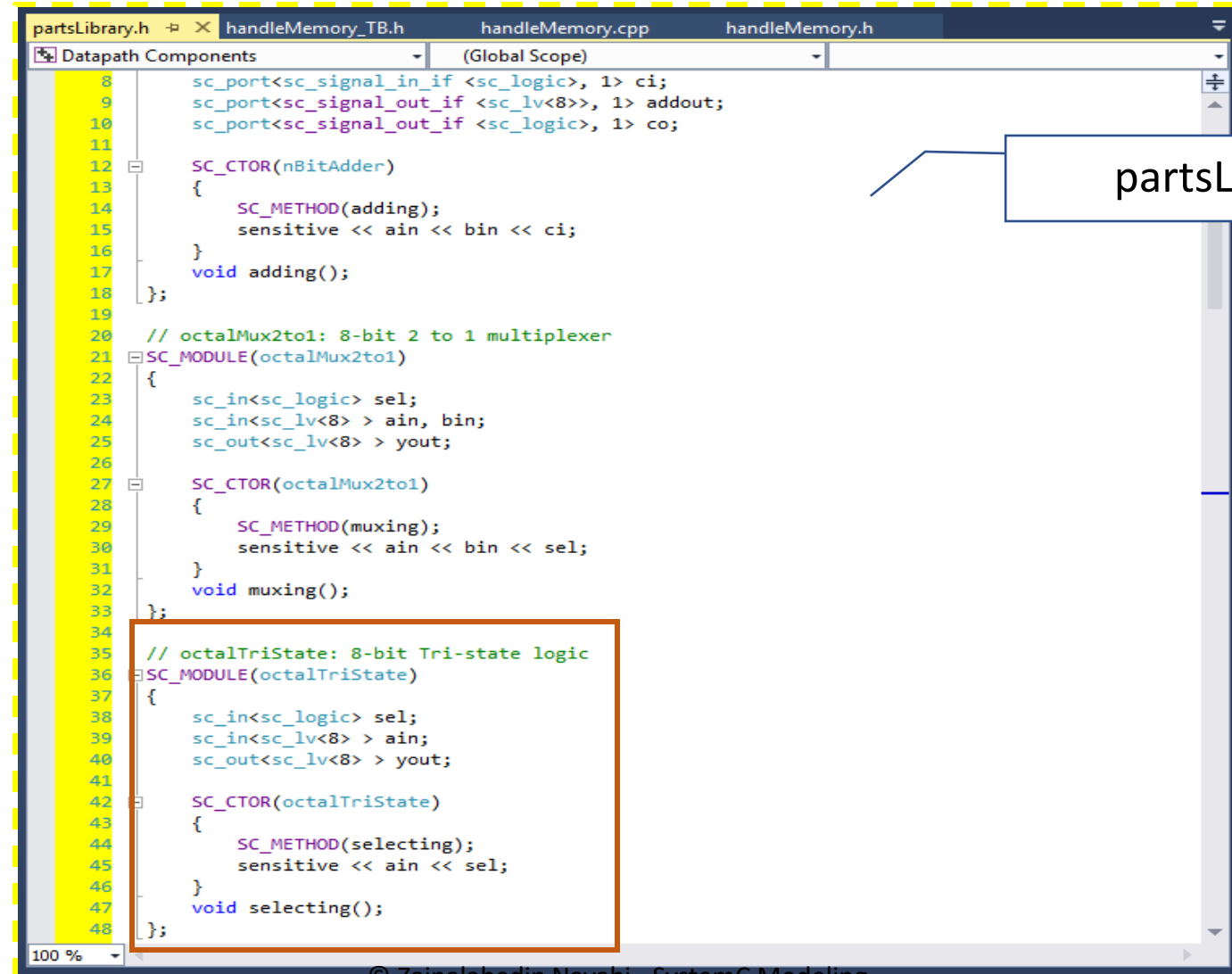
Annotations in the image:

- A box labeled "partsLibrary.cpp" points to the file name in the editor's title bar.
- A box labeled "Using dot for accessing functions" points to the `ain.read().to_uint()`, `bin.read().to_uint()`, and `ci.read().value()` expressions in the `adding()` function.

— Components for RTL Design

Combinational

● Tri-State

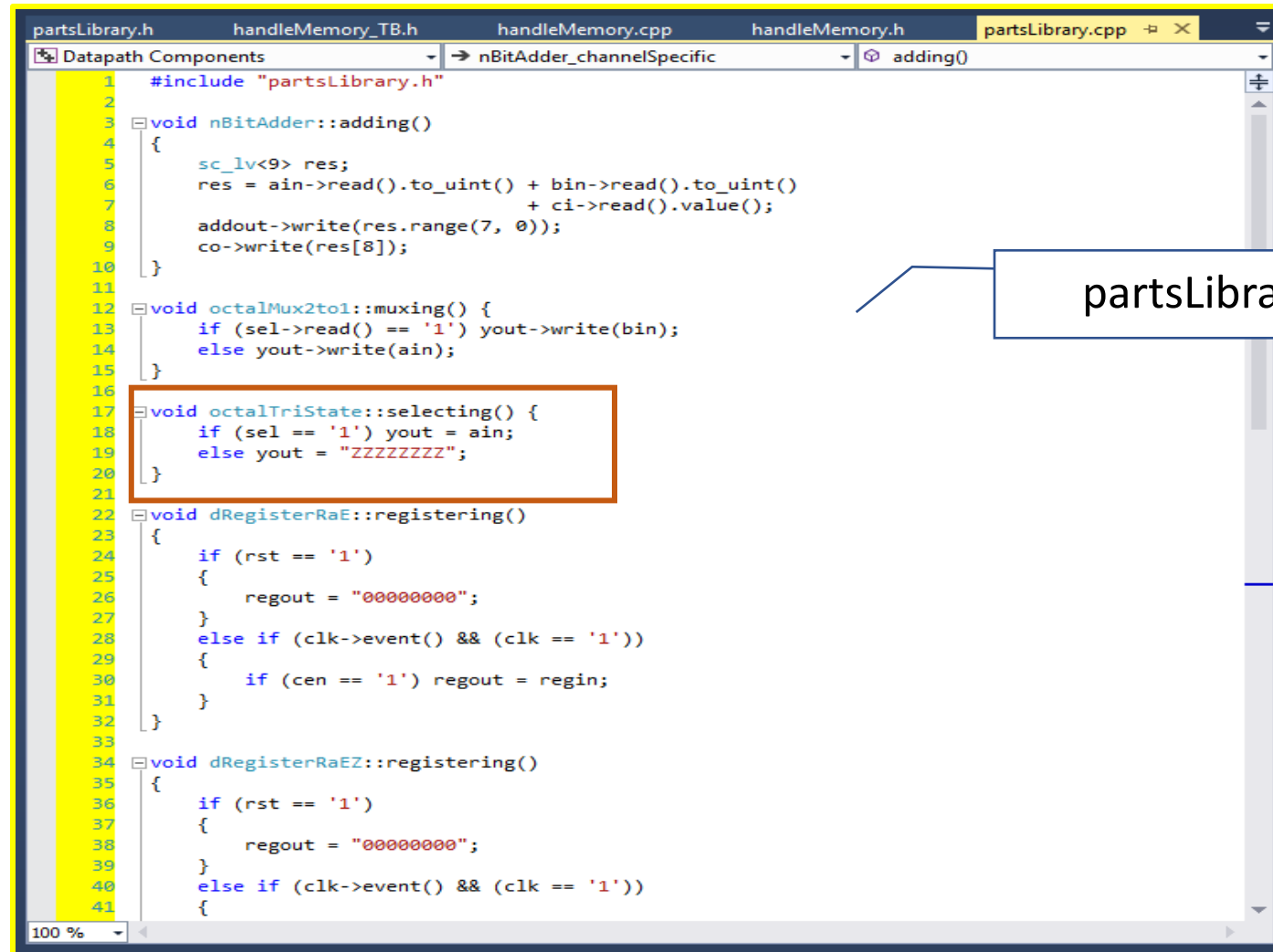


```
partsLibrary.h  X  handleMemory_TB.h  handleMemory.cpp  handleMemory.h
Datapath Components  (Global Scope)
8   sc_port<sc_signal_in_if<sc_logic>, 1> ci;
9   sc_port<sc_signal_out_if<sc_lv<8>>, 1> addout;
10  sc_port<sc_signal_out_if<sc_logic>, 1> co;
11
12  SC_CTOR(nBitAdder)
13  {
14      SC_METHOD(adding);
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  };
100 %
```

Components for RTL Design

Combinational

Tri-State



```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h  partsLibrary.cpp
Datapath Components  nBitAdder_channelSpecific  adding()

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     if (sel->read() == '1') yout->write(bin);
14     else yout->write(ain);
15 }
16
17 void octalTriState::selecting() {
18     if (sel == '1') yout = ain;
19     else yout = "ZZZZZZZ";
20 }
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     {
```

partsLibrary.cpp

Sequential

● Register

The image shows a code editor window with the file `partsLibrary.h` open. The editor displays three SystemC module definitions for registers. A callout box labeled "Static sensitivity" points to the `sensitive << rst << clk;` line in the `dRegisterRaE` module. Another callout box labeled "partsLibrary.h" points to the file name in the editor's title bar.

```
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();
```


Sequential

Register

Asynch reset

Dynamic sensitivity

partsLibrary.cpp

```
21 void dRegisterRaE::registering()
22 {
23     if (rst == '1')
24     {
25         regout = "00000000";
26     }
27     else if (clk->event() && (clk == '1'))
28     {
29         if (cen == '1') regout = regin;
30     }
31 }
32
33 void dRegisterRaEZ::registering()
34 {
35     if (rst == '1')
36     {
37         regout = "00000000";
38     }
39     else if (clk->event() && (clk == '1'))
40     {
41         if (cen == '1') {
42             if (zer == '1') regout = 0;
43             else regout = regin;
44         }
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     {
```

Components for RTL Design

Sequential

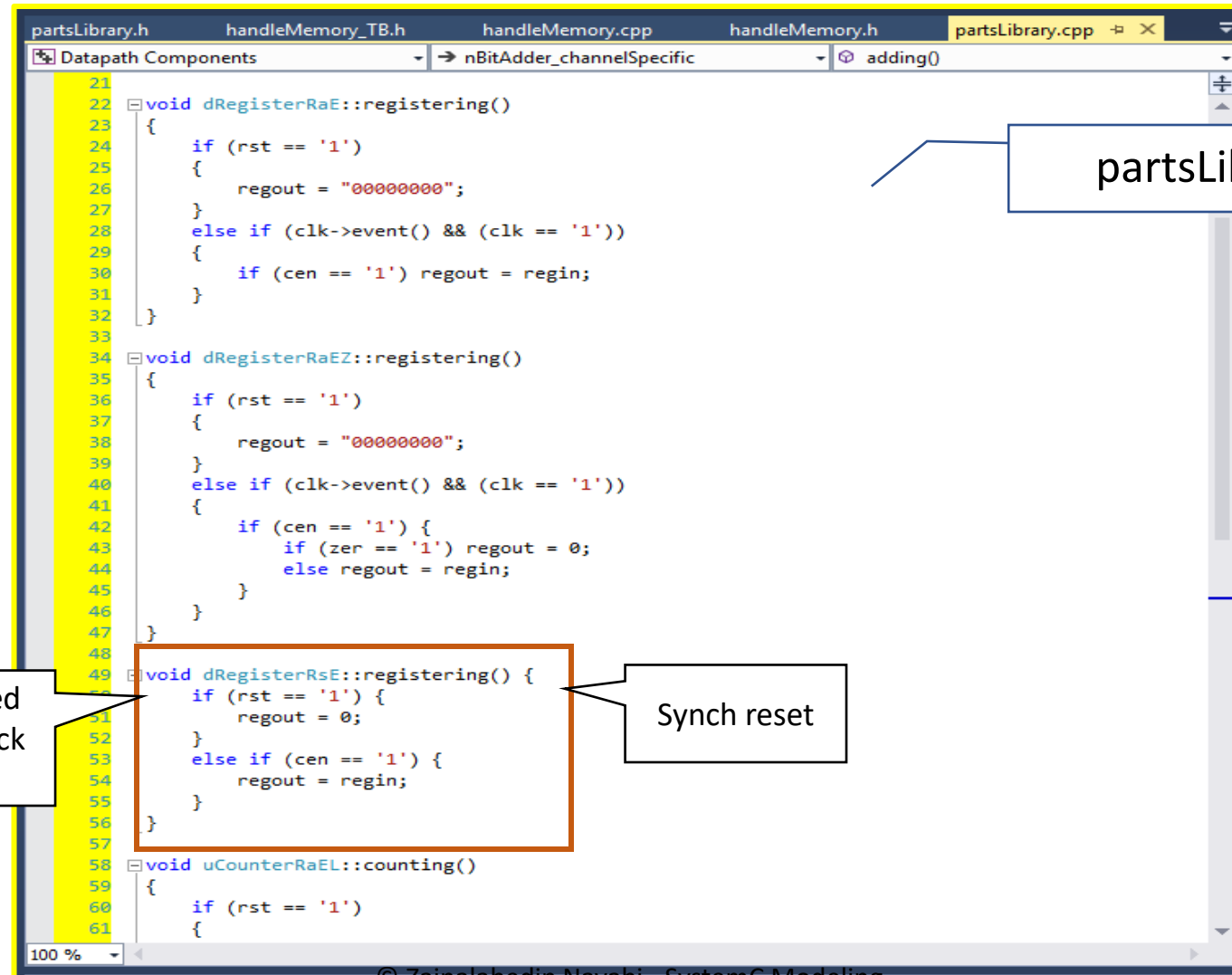
Register

.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
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     {
```

Sequential

Register



The screenshot shows a code editor with the file `partsLibrary.cpp` open. The editor has tabs for `partsLibrary.h`, `handleMemory_TB.h`, `handleMemory.cpp`, `handleMemory.h`, and `partsLibrary.cpp`. The `Datapath Components` dropdown is set to `nBitAdder_channelSpecific`, and the `adding()` function is selected. The code is as follows:

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

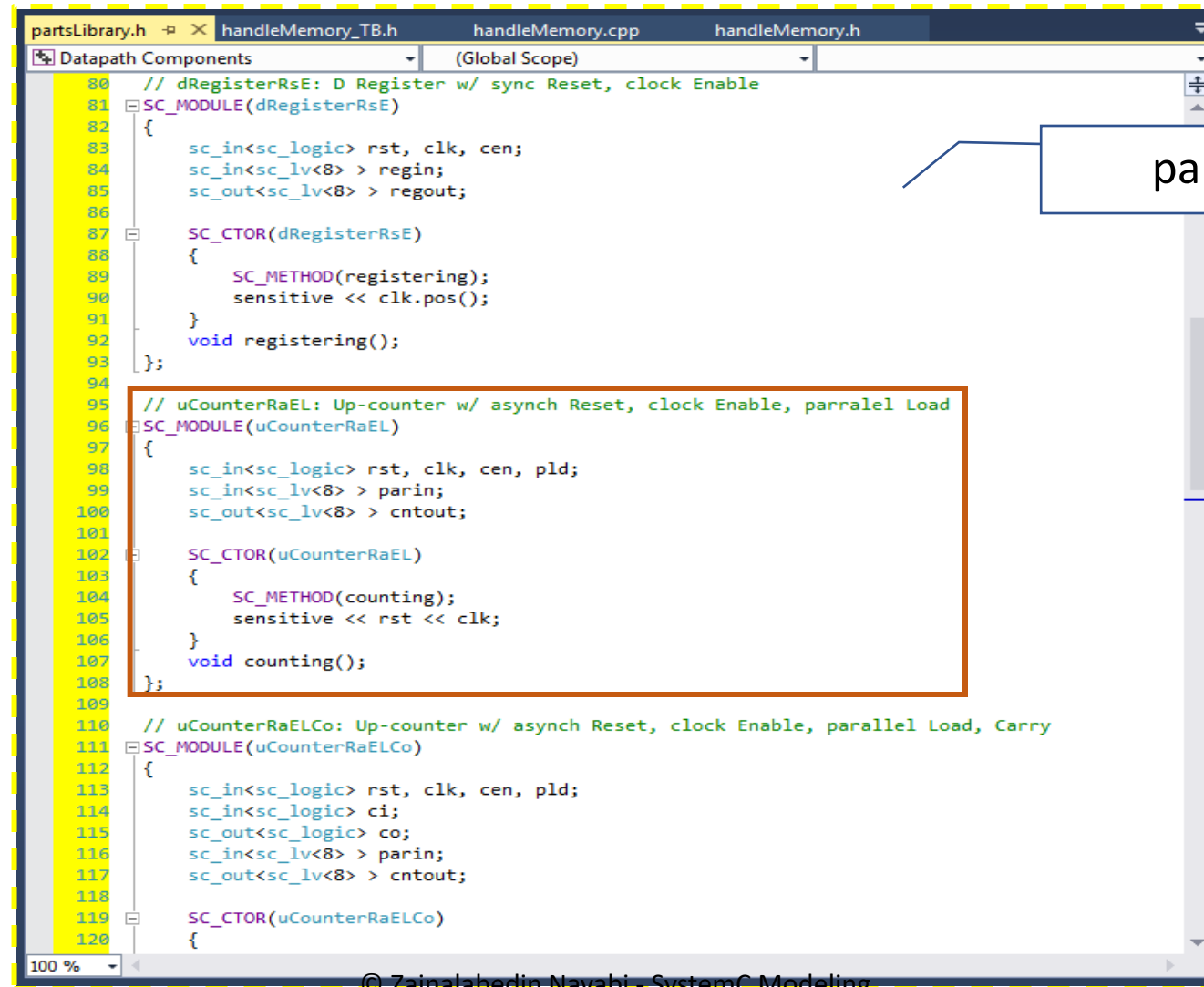
Annotations:

- A box labeled `partsLibrary.cpp` points to the file name in the editor's title bar.
- A box labeled "Doesn't need to check clock edge" points to the `dRegisterRsE::registering()` function.
- A box labeled "Synch reset" points to the `if (rst == '1')` condition in the `dRegisterRsE::registering()` function.

Components for RTL Design

Sequential

Counter



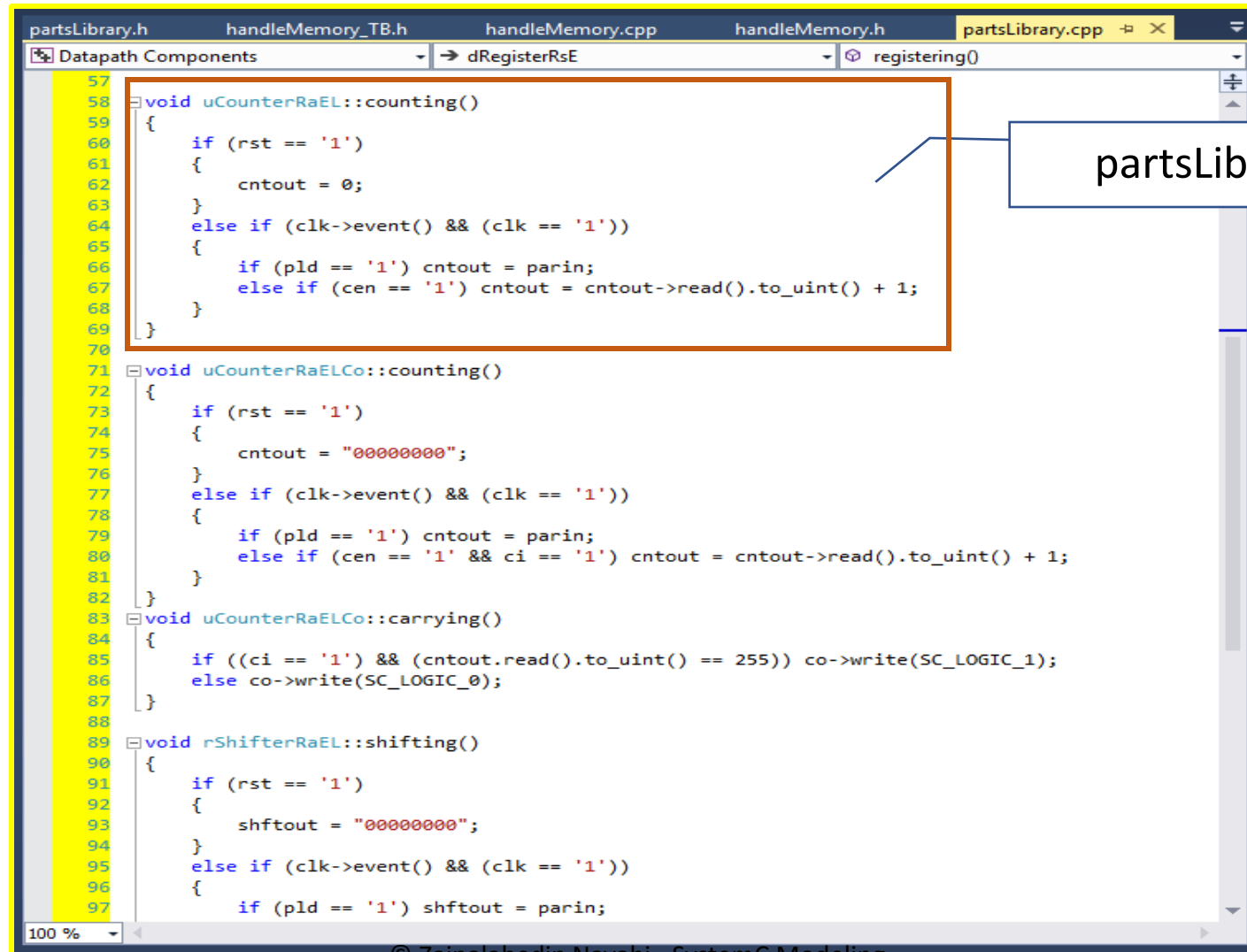
The screenshot shows a code editor with the file 'partsLibrary.h' open. The code defines three SystemC modules: 'dRegisterRsE', 'uCounterRaEL', and 'uCounterRaELCo'. The 'uCounterRaEL' module, which is an up-counter with asynchronous reset, clock enable, and parallel load, is highlighted with a red rectangle. It includes inputs for reset, clock, enable, and parallel load, and an output for the count. The 'uCounterRaELCo' module is a carry version of the counter. The editor's interface includes a tab bar at the top, a dropdown menu for 'Datapath Components', and a status bar at the bottom showing '100 %'.

```
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

Sequential

Counter



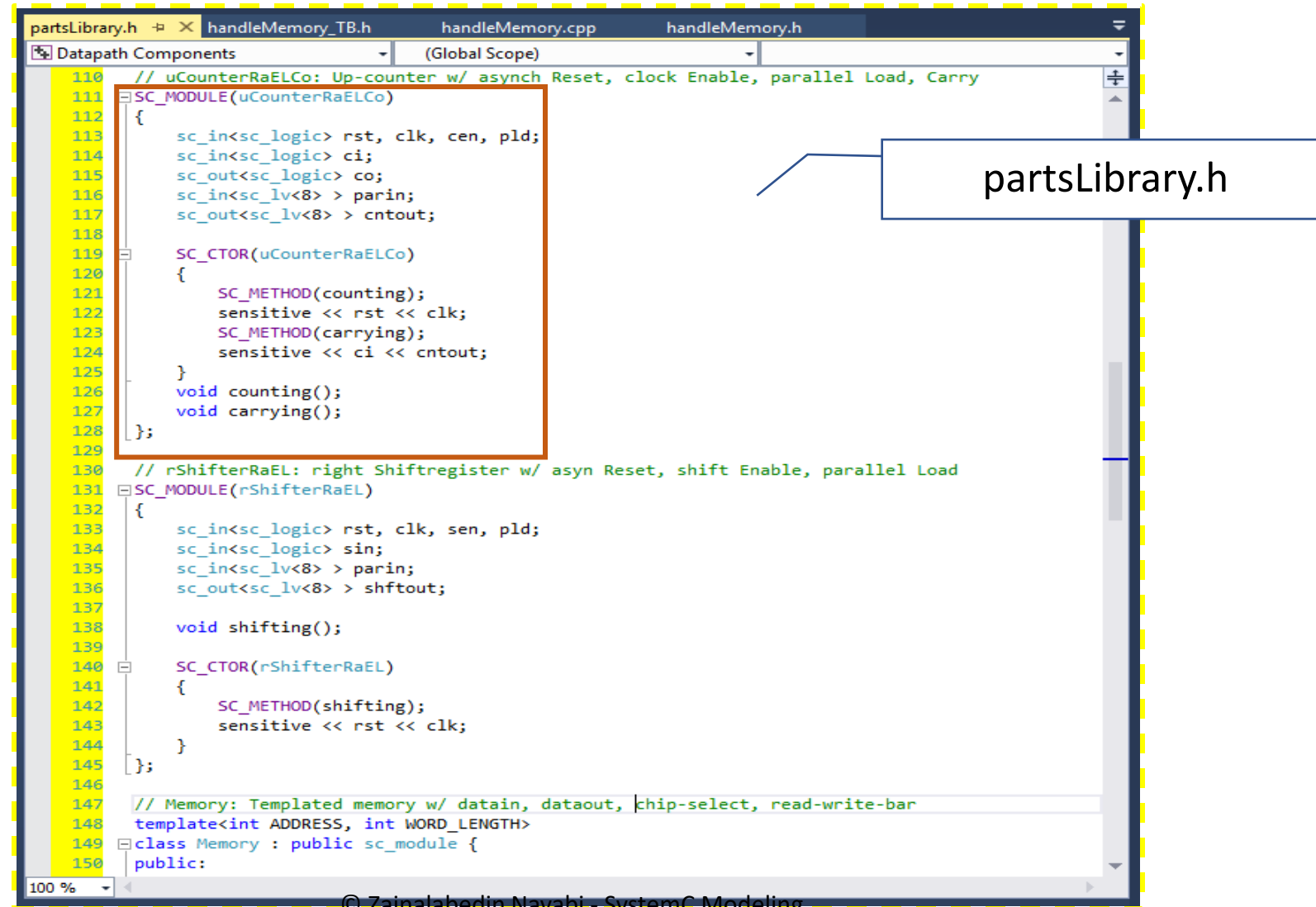
The screenshot shows a code editor with several tabs: partsLibrary.h, handleMemory_TB.h, handleMemory.cpp, handleMemory.h, and partsLibrary.cpp. The active tab is partsLibrary.cpp, showing the implementation of the uCounterRaEL component. A callout box labeled 'partsLibrary.cpp' points to the uCounterRaEL::counting() function, which is highlighted with a red border. The code is as follows:

```
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
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 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;
```

Components for RTL Design

Sequential

Counter



The screenshot shows a code editor with the following tabs: partsLibrary.h, handleMemory_TB.h, handleMemory.cpp, and handleMemory.h. The 'Datapath Components' pane is open, showing '(Global Scope)'. The code in partsLibrary.h is as follows:

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

A callout box labeled 'partsLibrary.h' points to the `SC_MODULE(uCounterRaELCo)` definition.

Sequential

Counter

The image shows a screenshot of a SystemC code editor with the file `partsLibrary.cpp` open. The editor displays the implementation of a counter component. A blue box highlights the `uCounterRaEL::counting()` function, and a red box highlights the `uCounterRaELCo::counting()` function. A callout box points to the `partsLibrary.cpp` tab. Another callout box points to the `co->write(SC_LOGIC_1);` line in the `uCounterRaELCo::counting()` function, with the text "It Produces co when it rolls over". A third callout box points to the `co->write(SC_LOGIC_0);` line in the same function, with the text "Carry in and carry out features are used for cascading".

```
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;
```

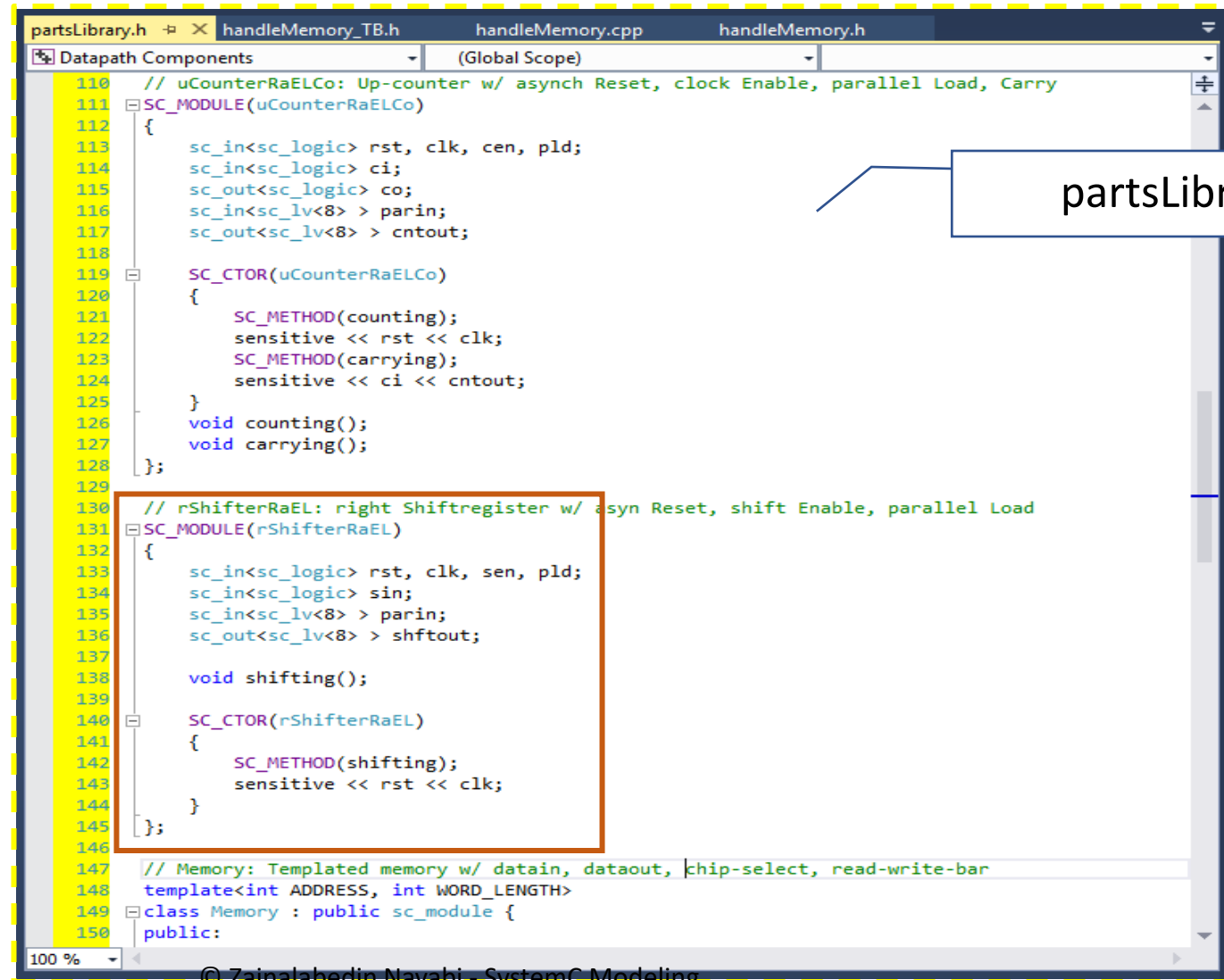
partsLibrary.cpp

It Produces co when it rolls over

Carry in and carry out features are used for cascading

Sequential

● Shift-Register



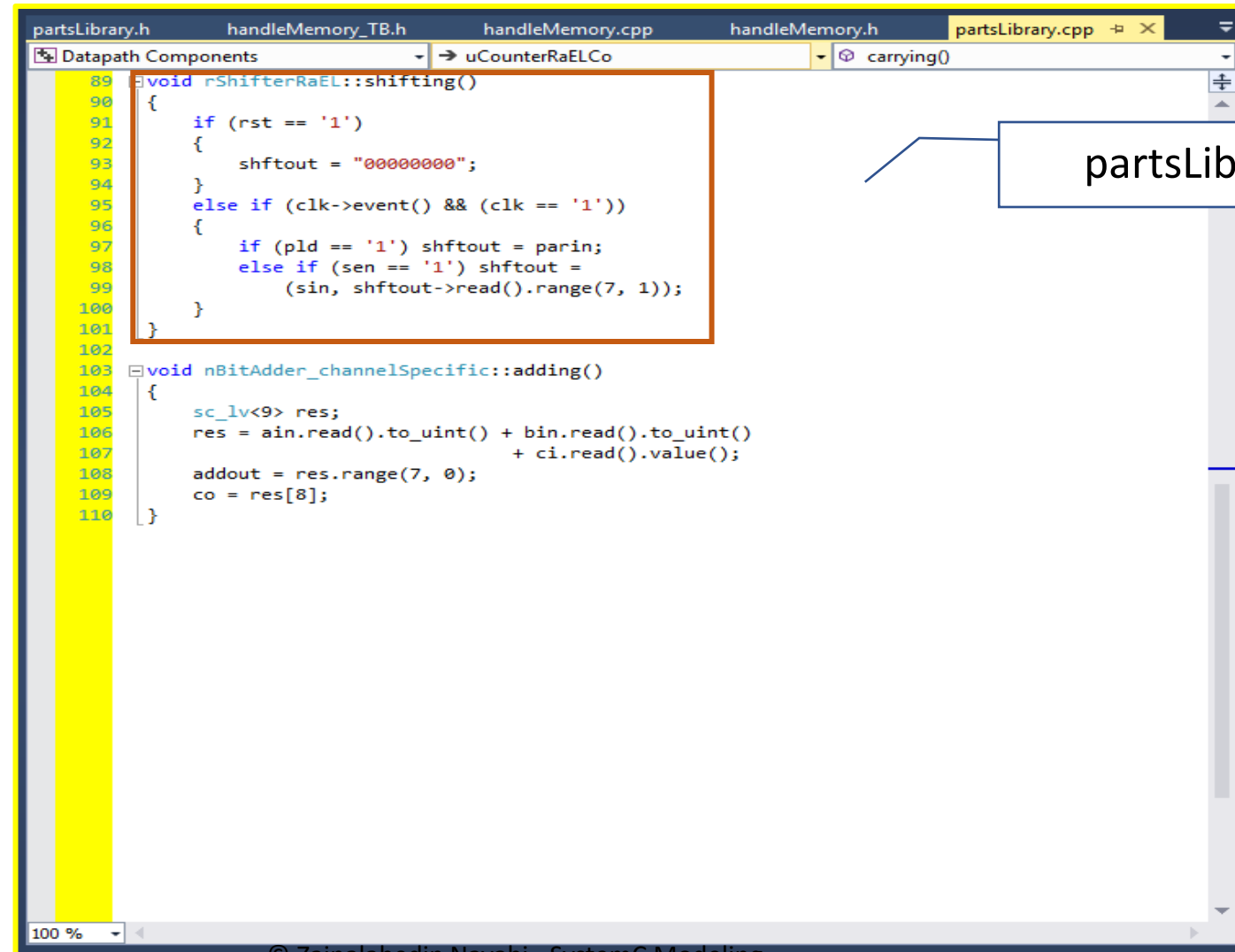
```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h
Datapath Components  (Global Scope)
110 // uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
111 SC_MODULE(uCounterRaELCo)
112 {
113     sc_in<sc_logic> rst, clk, cen, pld;
114     sc_in<sc_logic> ci;
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: Templated 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

— Components for RTL Design

Sequential

● Shift-Register



The screenshot shows a code editor with several tabs: partsLibrary.h, handleMemory_TB.h, handleMemory.cpp, handleMemory.h, and partsLibrary.cpp. The partsLibrary.cpp tab is active, showing 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 red rectangular box highlights the `rShifterRaEL::shifting()` function, which is located between lines 89 and 101. The function implements a shift register logic where it resets to zero on a reset signal, shifts left on a clock edge, and inserts a new value on a parallel load signal.

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

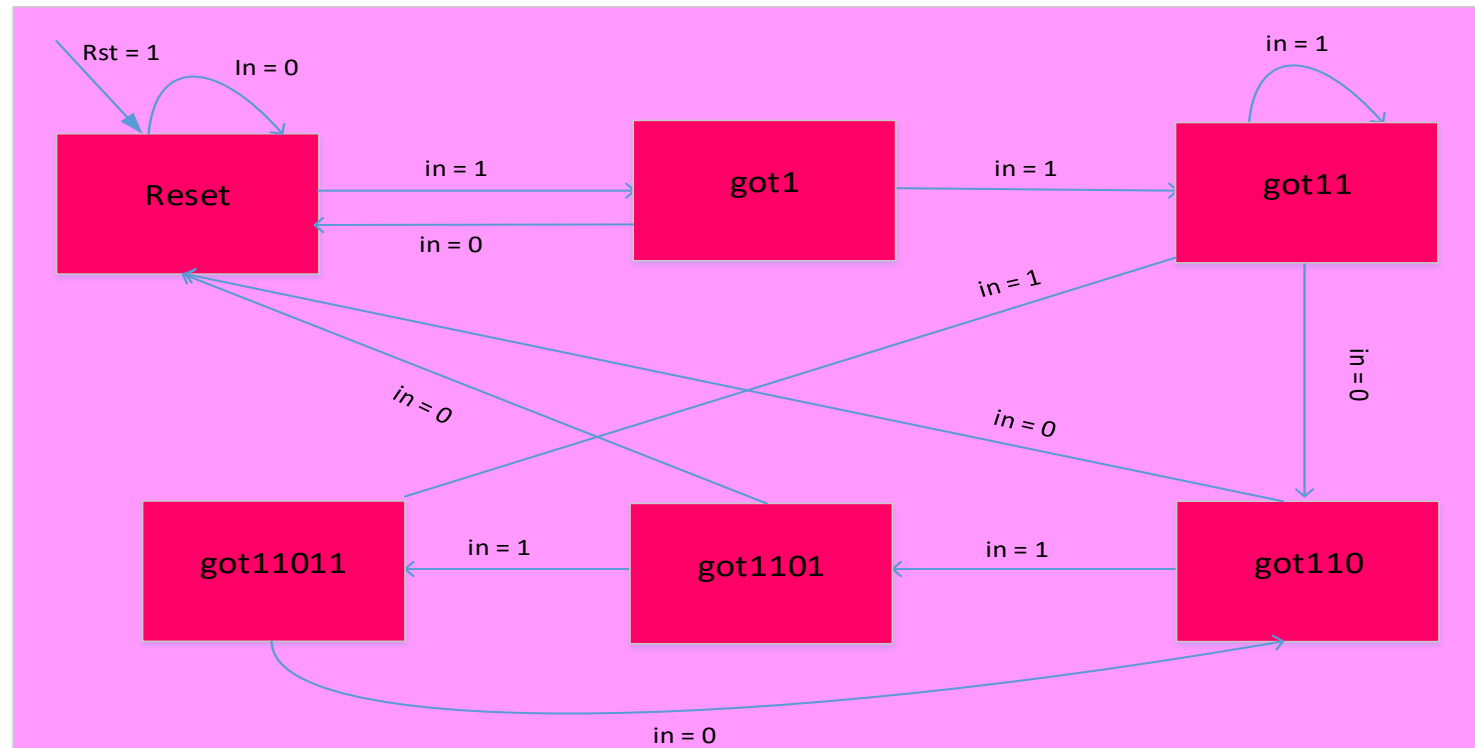
Exponential Circuit

SystemC Functional Modeling

- + SystemC Functional Design - Example
- Summary

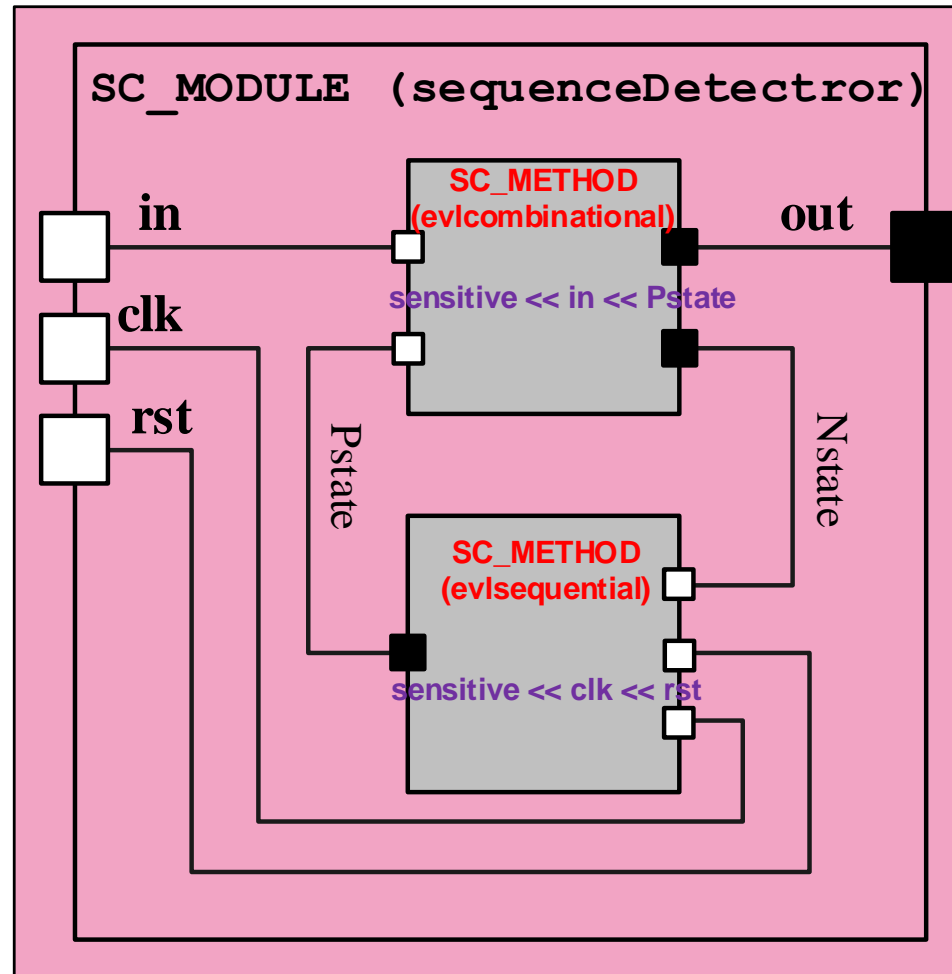
Sequence Detector 11011

◉ State Diagram



Sequence Detector 11011

⦿ Huffman Model



Sequence Detector 11011

```
sequenceDetectorTB.cpp  sequenceDetector.cpp  sequenceDetector.h  sequenceDetectorTB.h
Sequence Detector  (Global Scope)
1  #include <systemc.h>
2
3  SC_MODULE(sequenceDetector)
4  {
5      sc_in<sc_logic> in, rst, clk;
6      sc_out<sc_logic> out;
7
8      enum states { ST0, ST1, ST2, ST3, ST4, ST5 };
9
10     sc_signal<states> Nstate, Pstate;
11
12     SC_CTOR(sequenceDetector)
13     {
14         Pstate.write(ST0);
15         Nstate.write(ST0);
16
17         SC_METHOD(evlCombinational);
18         sensitive << in << Pstate;
19
20         SC_METHOD(evlSequential);
21         sensitive << clk << rst;
22     }
23
24     void evlCombinational();
25     void evlSequential();
26 };
27
28
```

sequenceDetector.h

Sequence Detector 11011

sequenceDetectorTB.cpp sequenceDetector.cpp 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 = ST2;

17 else Nstate = ST3; 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 if (Pstate == ST5) out = SC_LOGIC_1;

29 }

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 }

Set to their inactive value

Read is also an sc_signal method

sequeceDetector.cpp

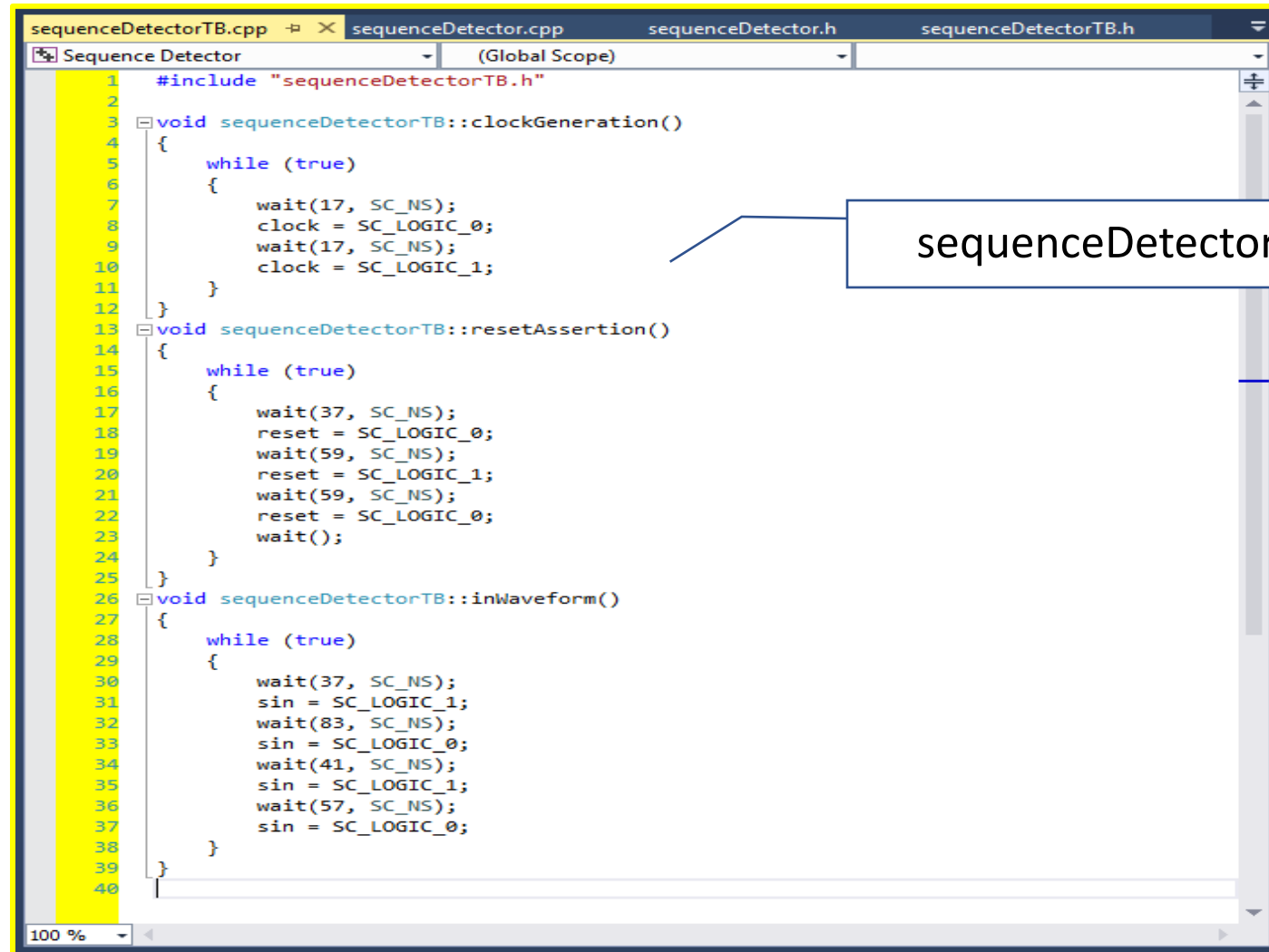
Sequence Detector 11011

● Testbench

```
sequenceDetectorTB.cpp  sequenceDetector.cpp  sequenceDetector.h  sequenceDetectorTB.h
Sequence Detector (Global Scope)
1  #include "sequenceDetector.h"
2
3  SC_MODULE(sequenceDetectorTB)
4  {
5      sc_signal<sc_logic> sin, reset, clock;
6      sc_signal<sc_logic> sout;
7
8      sequenceDetector* UUT;
9
10     SC_CTOR(sequenceDetectorTB)
11     {
12         UUT = new sequenceDetector ("sequenceDetector_instance");
13         UUT->in(sin);
14         UUT->rst(reset);
15         UUT->clk(clock);
16         UUT->out(sout);
17
18         SC_THREAD(clockGeneration);
19         SC_THREAD(resetAssertion);
20         SC_THREAD(inWaveform);
21     }
22     void clockGeneration();
23     void resetAssertion();
24     void inWaveform();
25 }
26
```

Sequence Detector 11011

● Testbench

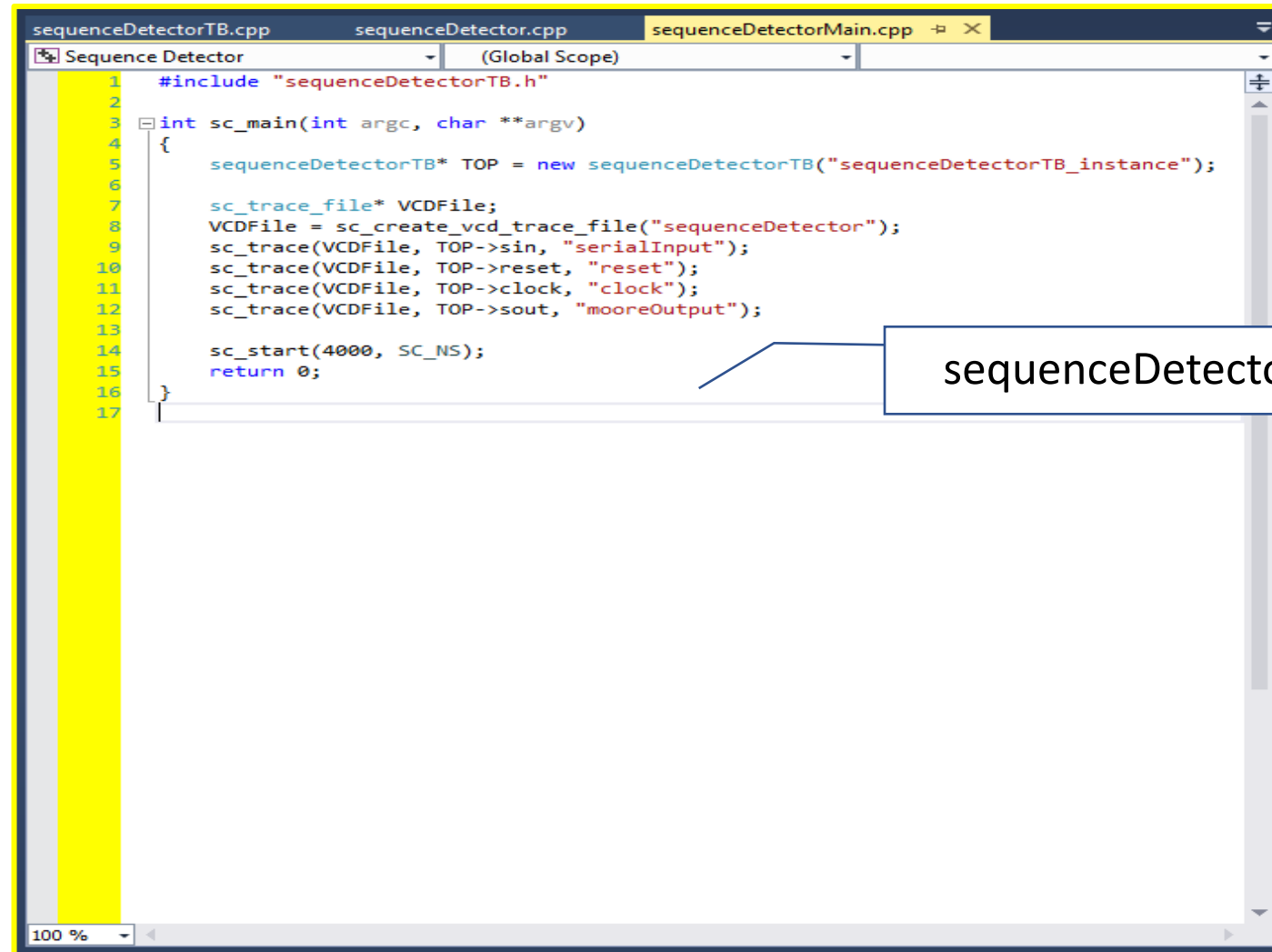


```
sequenceDetectorTB.cpp  sequenceDetector.cpp  sequenceDetector.h  sequenceDetectorTB.h
Sequence Detector  (Global Scope)
1  #include "sequenceDetectorTB.h"
2
3  void sequenceDetectorTB::clockGeneration()
4  {
5      while (true)
6      {
7          wait(17, SC_NS);
8          clock = SC_LOGIC_0;
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 }
40
```

sequenceDetectorTB.cpp

Sequence Detector 11011

● sc_main



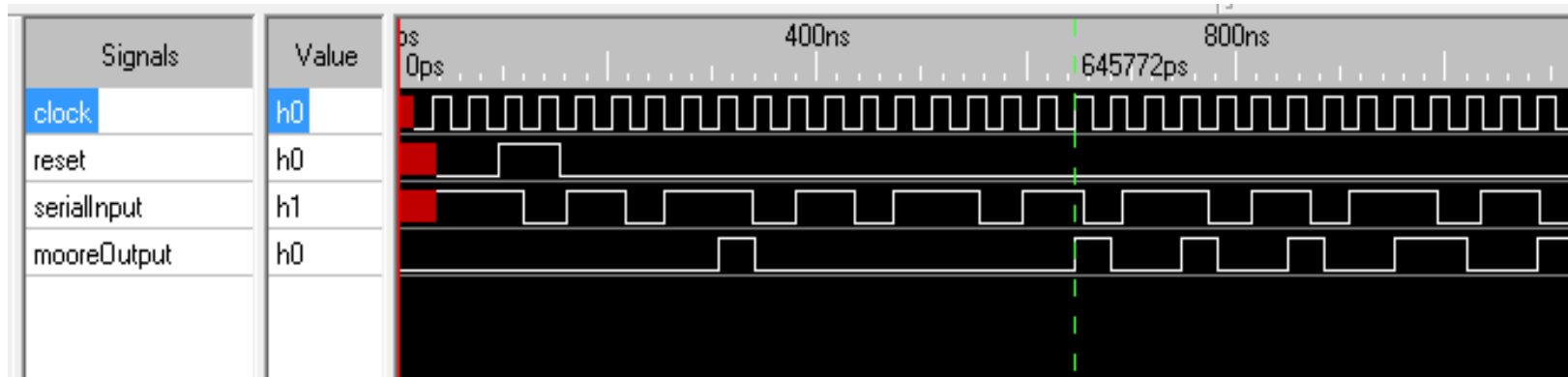
```
sequenceDetectorTB.cpp  sequenceDetector.cpp  sequenceDetectorMain.cpp
Sequence Detector      (Global Scope)
1  #include "sequenceDetectorTB.h"
2
3  int sc_main(int argc, char **argv)
4  {
5      sequenceDetectorTB* TOP = new sequenceDetectorTB("sequenceDetectorTB_instance");
6
7      sc_trace_file* VCDFile;
8      VCDFile = sc_create_vcd_trace_file("sequenceDetector");
9      sc_trace(VCDFile, TOP->sin, "serialInput");
10     sc_trace(VCDFile, TOP->reset, "reset");
11     sc_trace(VCDFile, TOP->clock, "clock");
12     sc_trace(VCDFile, TOP->sout, "mooreOutput");
13
14     sc_start(4000, SC_NS);
15     return 0;
16 }
17
```

sequenceDetectorMain.cpp

SystemC RTL Design - Example

Sequence Detector 11011

● VCD waveform



A Configurable Memory

The image shows a screenshot of a code editor with several tabs: `handleMemory.cpp`, `handleMemory_TB.h`, `partsLibrary.cpp`, `partsLibrary.h`, and `handleMemory.h`. The `partsLibrary.h` tab is active, displaying the following code:

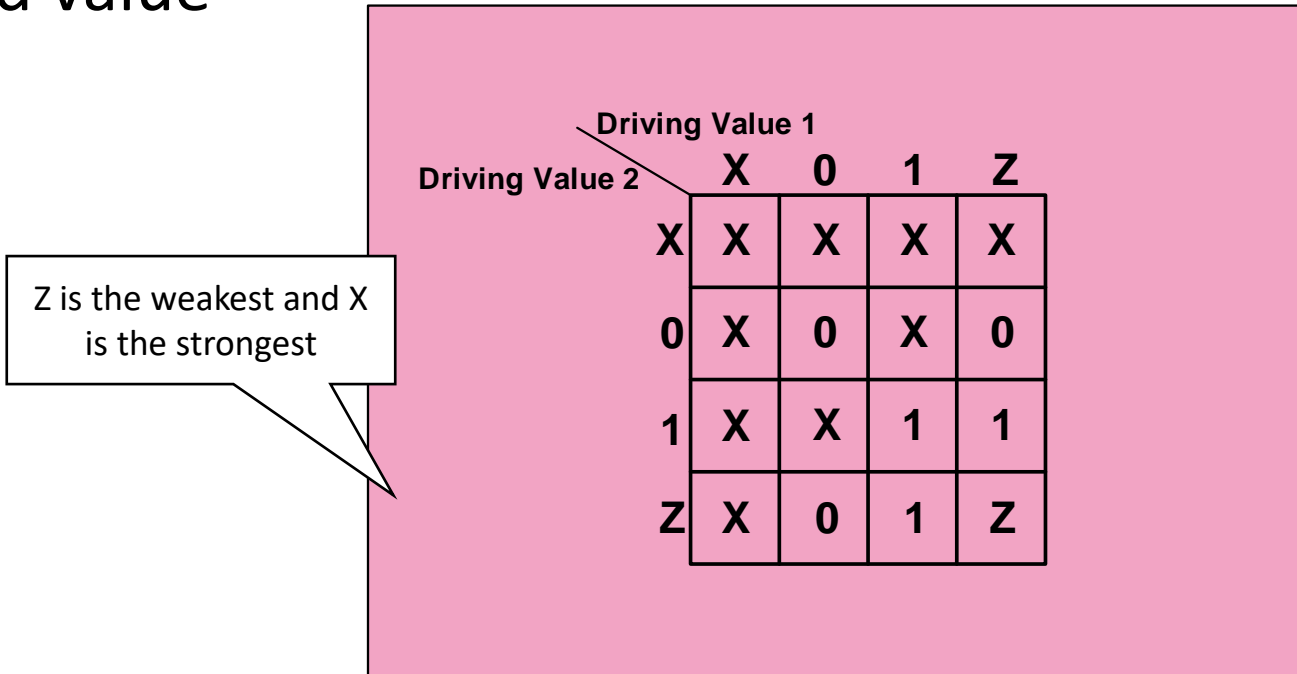
```
148 // Memory: Templated memory w/ datain, dataout, chip-select, read-write-bar
149
150 template<int ADDRESS, int WORD_LENGTH>
151 class Memory : public sc_module {
152 public:
153     sc_in_rv<ADDRESS> addr;
154     sc_in_rv<WORD_LENGTH> datain;
155     sc_out_rv<WORD_LENGTH> dataout;
156     sc_in_resolved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

Annotations and callouts:

- Use Template for Configurability**: Points to the `template<int ADDRESS, int WORD_LENGTH>` line.
- Use resolved to have multiple sources**: Points to the `sc_in_resolved cs, rwbar;` line.
- Memory Array**: Points to the `mem = new sc_uint<WORD_LENGTH>[addrSpace];` line.
- These parameters should be defined at compile time**: Points to the `ADDRESS` and `WORD_LENGTH` template parameters.
- partsLibrary.h**: A box containing the text `partsLibrary.h`, with a line pointing to the `partsLibrary.h` tab.

A Configurable Memory

⦿ Resolved value



Driving Value 1

Driving Value 2

| | 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

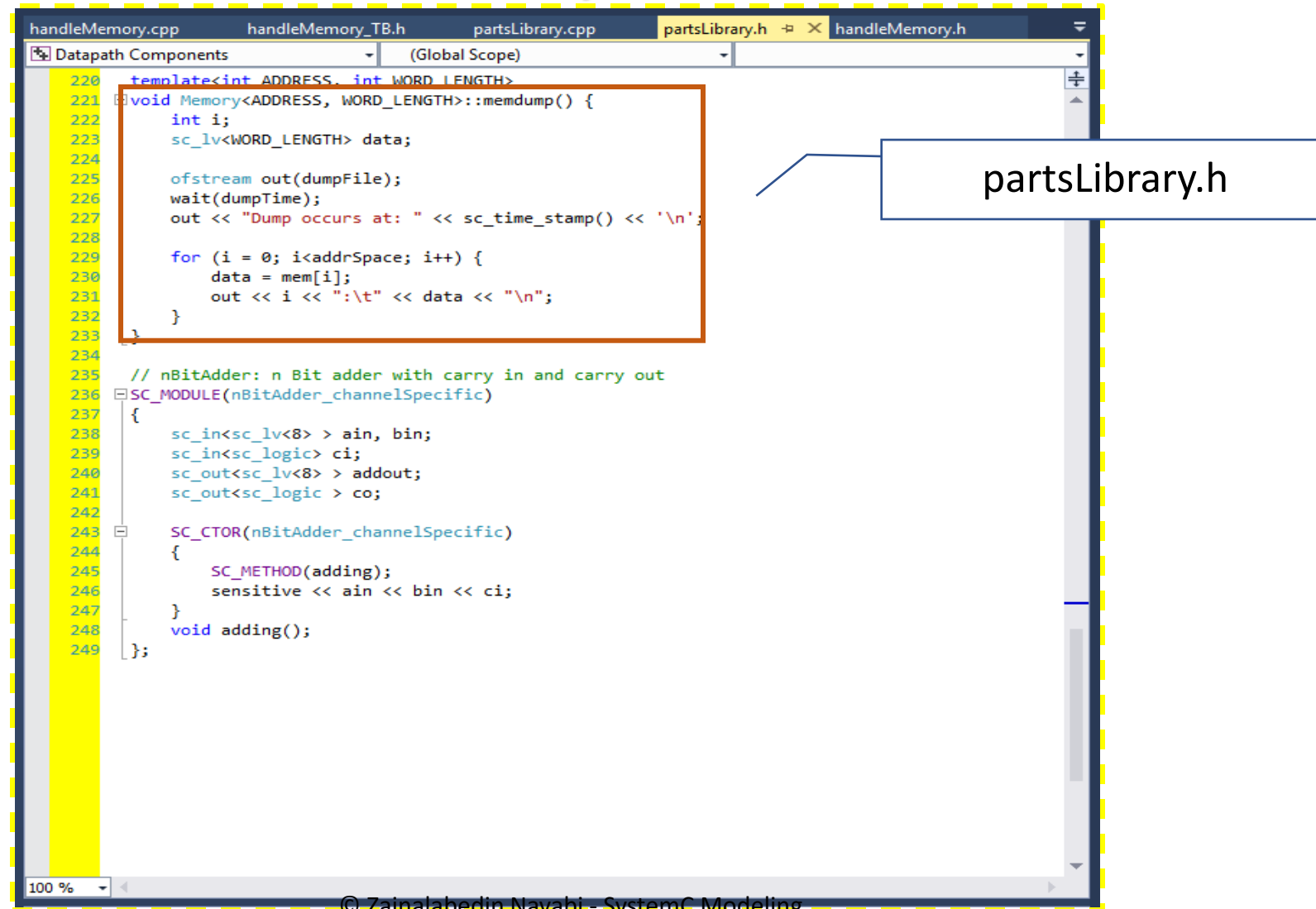
The image shows a code editor window with the following tabs: `handleMemory.cpp`, `handleMemory_TB.h`, `partsLibrary.cpp`, `partsLibrary.h`, and `handleMemory.h`. The `partsLibrary.h` tab is active, showing the following code:

```
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++) {
```

Annotations:

- A callout box labeled "Type Conversion from sc_rv to sc_uint" points to the `sc_uint<ADDRESS> ad;` line in the `memwrite()` function.
- A callout box labeled "Type Conversion from sc_uint to sc_rv" points to the `dataout = mem[ad];` line in the `memread()` function.
- A callout box labeled "partsLibrary.h" points to the header file tab.

A Configurable Memory



The screenshot shows a SystemC IDE with a code editor displaying the contents of `partsLibrary.h`. The editor has tabs for `handleMemory.cpp`, `handleMemory_TB.h`, `partsLibrary.cpp`, `partsLibrary.h` (selected), and `handleMemory.h`. The left sidebar shows a project tree with 'Datapath Components' and '(Global Scope)'. The code in the editor is as follows:

```
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++) {
230         data = mem[i];
231         out << i << ":\t" << data << "\n";
232     }
233 }
234
235 // nBitAdder: n Bit adder with carry in and carry out
236 SC_MODULE(nBitAdder_channelSpecific)
237 {
238     sc_in<sc_lv<8> > ain, bin;
239     sc_in<sc_logic> ci;
240     sc_out<sc_lv<8> > addout;
241     sc_out<sc_logic> co;
242
243     SC_CTOR(nBitAdder_channelSpecific)
244     {
245         SC_METHOD(adding);
246         sensitive << ain << bin << ci;
247     }
248     void adding();
249 };
```

A callout box labeled `partsLibrary.h` points to the `memdump()` function definition.

A Configurable Memory

The screenshot shows a code editor with the following content:

```
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)
5 {
6     sc_signal_rv<8> databusin, databusout;
7     sc_signal_rv<10> addrbus;
8     sc_signal_resolved cs, rwbar;
9
10    handleMemory* EXC1;
11    Memory<10, 8>* MEM;
12
13    SC_CTOR(handleMemory_TB)
14    {
15        EXC1 = new handleMemory("EXC_Instance");
16        (*EXC1) (addrbus, databusin, databusout, cs, rwbar);
17        MEM = new Memory<10, 8>("MEM_Instance", sc_time(2000, SC_NS), "memout.txt");
18        (*MEM) (addrbus, databusin, databusout, cs, rwbar);
19    }
20
21    void resetting();
22    void clocking();
23    void displaying();
24 };
25
```

Annotations:

- A box labeled "HandleMemory_TB.h" points to the header file in the editor's title bar.
- A box labeled "Controller and datapath class pointers" points to the declarations of `EXC1` and `MEM` in the `SC_MODULE` block.

Exponential Circuit

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

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


Exponential Circuit

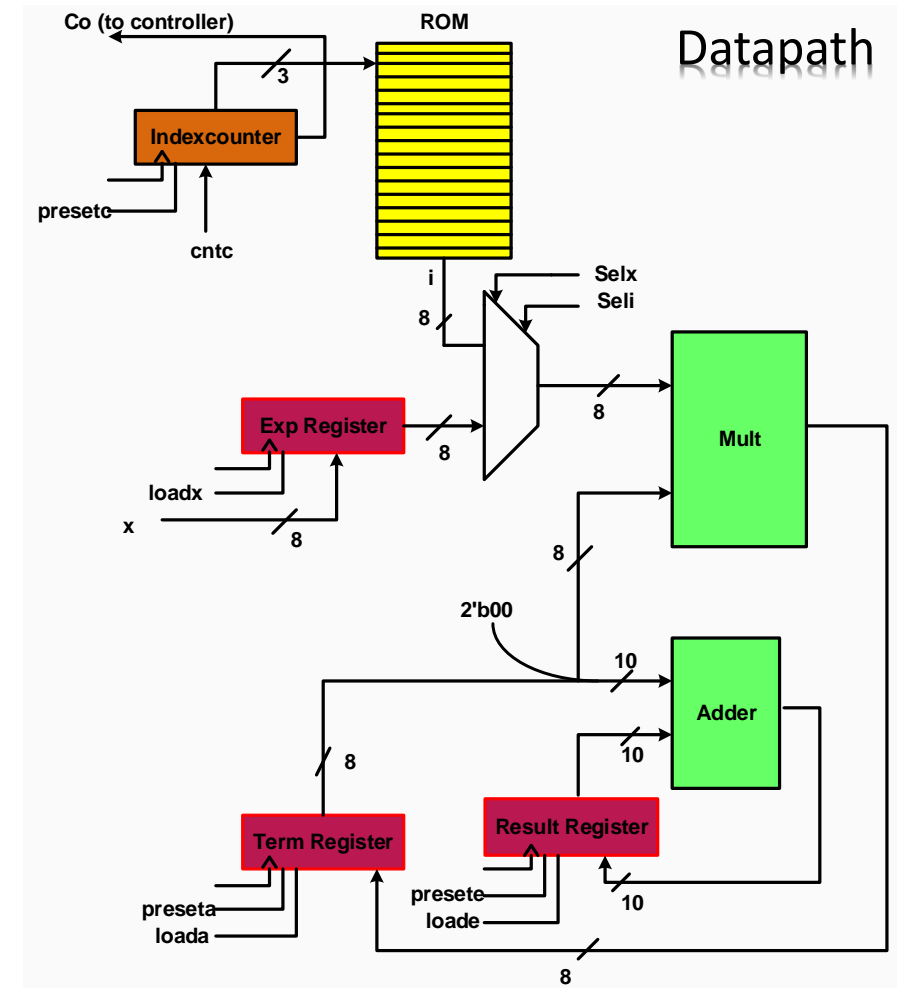
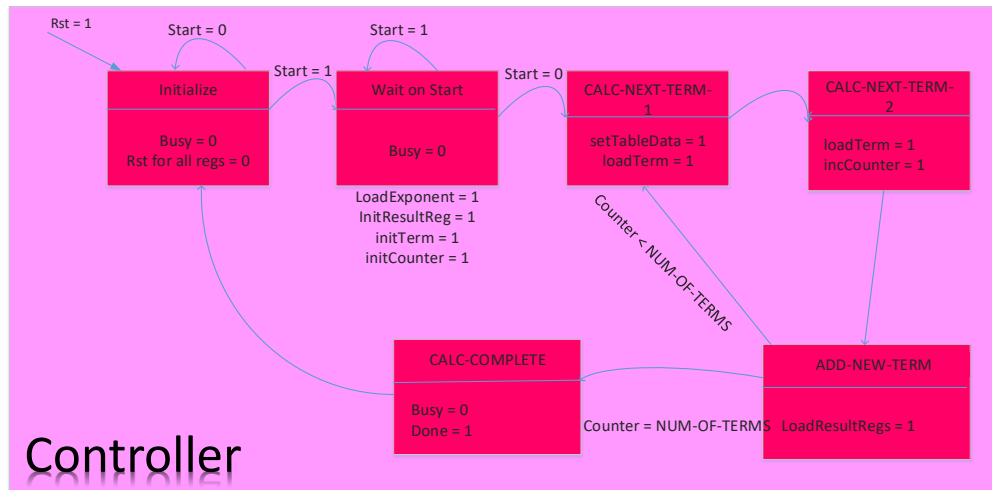
$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Using only
one
Multiplier

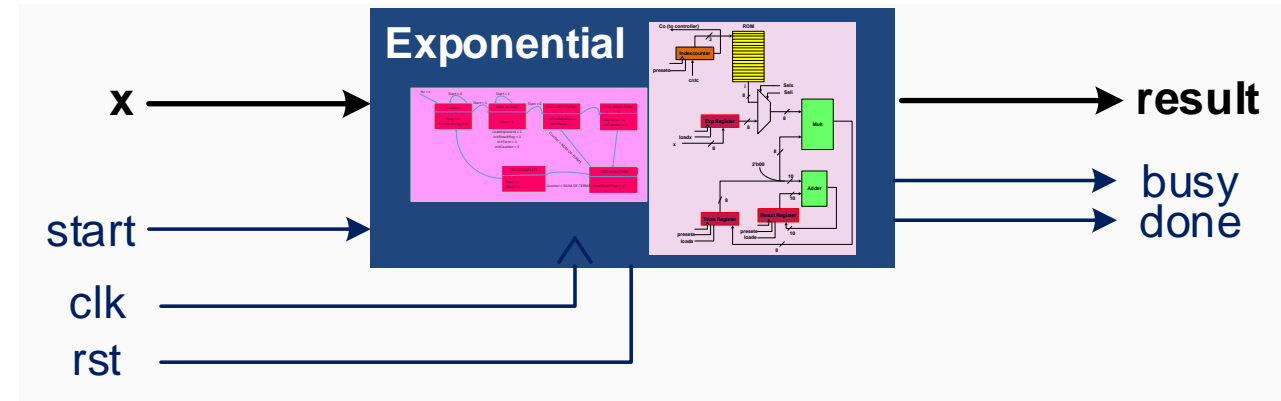
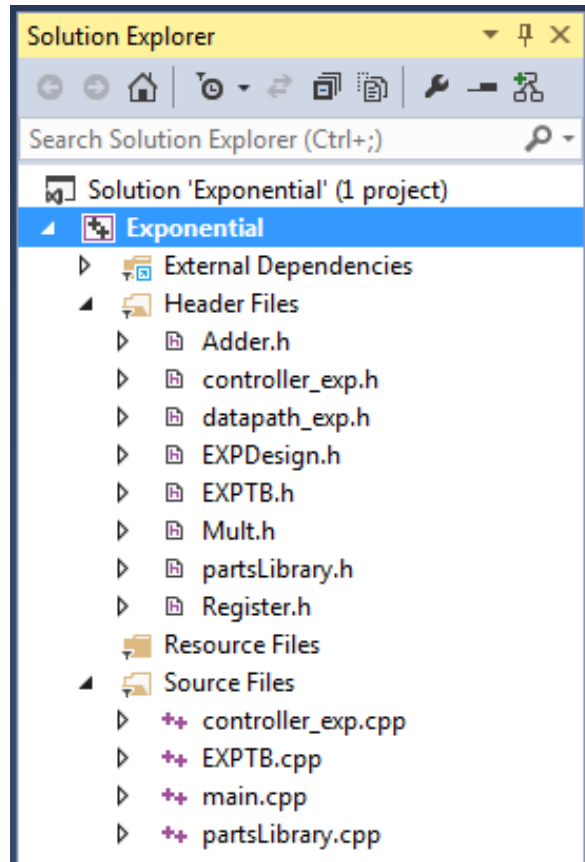
```
term = 1;  
exp = 1;  
for( i = 1; i < n; i++ ) {  
    term = term × ( 1 / i );  
    term = term × x;  
    exp = exp + term;  
}
```

Exponential Circuit

⦿ Datapath & Controller



Exponential Circuit



Exponential Circuit

EXPDesign.h datapath_exp.h controller_exp.h controller_exp.cpp EXPTB.h EXPTB.cpp main.cpp Register.h partsLibrary.h

(Global Scope)

```
1 #include "datapath_exp.h"
2 #include "controller_exp.h"
3
4 SC_MODULE(EXP){
5     sc_in<sc_logic> clk, rst, start;
6     sc_in<sc_lv<8>> x;
7     sc_out<sc_lv<10>> result;
8     sc_out<sc_logic> busy, done;
9     sc_signal<sc_logic> loadExponent, rstExponent;
10    sc_signal<sc_logic> loadTerm, initTerm, rstTerm;
11    sc_signal<sc_logic> selTableData;
12    sc_signal<sc_logic> rstResultReg, initResultReg, loadResultReg;
13    sc_signal<sc_logic> enCounter, rstCounter, initCounter;
14    sc_signal<sc_logic> co, memSel, rwbar;
15
16    datapath* DP;
17    controller* CNTRL;
18
19    SC_CTOR(EXP){
20        DP = new datapath("Datapath");
21        (*DP) (clk, rst, rwbar, memSel, selTableData, loadExponent, rstExponent, loadTerm, initTerm, rstTerm,
22              rstResultReg, initResultReg, loadResultReg, enCounter, rstCounter, initCounter, x, co, result);
23
24        CNTRL = new controller("Controller");
25        (*CNTRL) (rst, clk, start, co, rwbar, memSel, selTableData, loadExponent, rstExponent, loadTerm, initTerm,
26                 rstTerm, rstResultReg, initResultReg, loadResultReg, enCounter, rstCounter, initCounter, busy, done);
27    }
28 }
```

EXPDesign.h

Exponential

x → start → clk → rst → result → busy → done

Datapath instantiation

Controller instantiation

Exponential Circuit

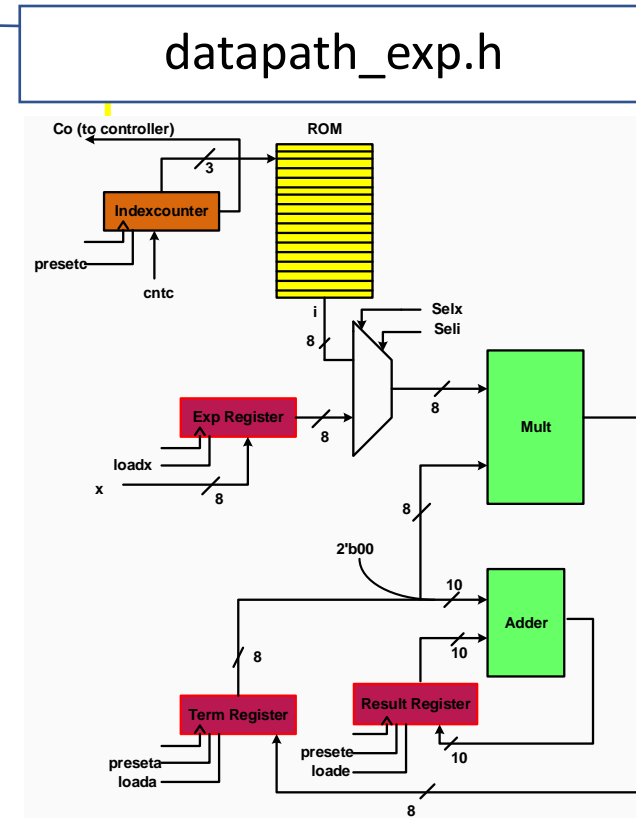
The diagram illustrates the ALU architecture. It features an **Indexcounter** (orange box) with inputs **preseto** and **cntc**, and outputs **Co (to controller)** and a 3-bit bus. This bus connects to a **ROM** (yellow stack of 16 horizontal bars). The ROM outputs an 8-bit bus to a multiplexer. The multiplexer also receives a 1-bit input **i** and an 8-bit input from the **Exp Register** (red box). The multiplexer's output (8 bits) goes to the **Mult** (green box). The **Mult** also receives a 2'b00 constant and an 8-bit input from the **Term Register** (red box). The **Mult**'s output (10 bits) goes to the **Adder** (green box). The **Adder** also receives a 10-bit input from the **Result Register** (red box) and a 10-bit input from the **Term Register**. The **Adder**'s output (10 bits) goes to the **Result Register**. The **Result Register** also receives a 10-bit input from the **Term Register**. The **Term Register** has inputs **preseta** and **loada**. The **Result Register** has inputs **preseto** and **loada**. The **Exp Register** has inputs **loadx** and **x**. The **Exp Register**'s output (8 bits) goes to the multiplexer. The **Term Register**'s output (8 bits) goes to the **Mult**. The **Result Register**'s output (8 bits) goes to the **Term Register**.

Exponential Circuit

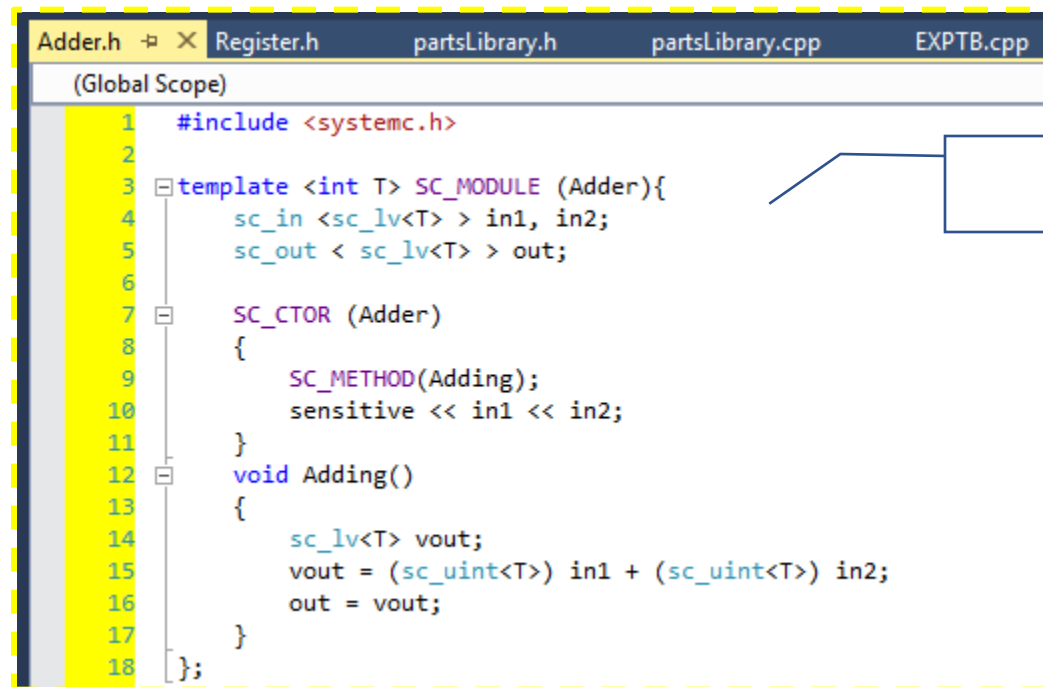
```

EXPDesign.h  datapath_exp.h  controller_exp.h  controller_exp.cpp  EXPTB.h  EXPTB.cpp
(Global Scope)
89  SC_METHOD(datapath_assign);
90  sensitive << loadExponent << rstExponent
91          << loadTerm << initTerm << rstTerm << term
92          << loadResultReg << initResultReg << rstResultReg
93          << multResult << addResult
94          << countValue;
95  }
96
97  // Function declaration & definition
98  void datapath_assign(){
99
100     enableExponent = (loadExponent->read() | rstExponent->read());
101     exponentInput = (loadExponent->read() == '1') ? x->read() :
102                    (rstExponent->read() == '1') ? "00000000" : "XXXXXXXX";
103
104     co = (sc_uint<4> (countValue.read()) < 8) ? SC_LOGIC_0 : SC_LOGIC_1;
105
106     sc_lv<8> temp;
107     temp = (multResult.read().range(15, 8));
108     termInput = ((loadTerm->read() == '1') ? temp :
109                (initTerm->read() == '1') ? "11111111" :
110                (rstTerm->read() == '1') ? "00000000" : "XXXXXXXX");
111     enableTermReg = (loadTerm->read() | initTerm->read() | rstTerm->read());
112
113     addUpperInput = (sc_lv<2>("00"), term.read());
114
115     resultInput = ( (loadResultReg->read() == '1') ? addResult.read() :
116                   (initResultReg->read() == '1') ? "0100000000" :
117                   (rstResultReg->read() == '1') ? "0000000000" : "XXXXXXXXXXXX");
118
119     enableResultReg = (loadResultReg->read() |
120                       initResultReg->read() | rstResultReg->read());
121 }
122 }

```



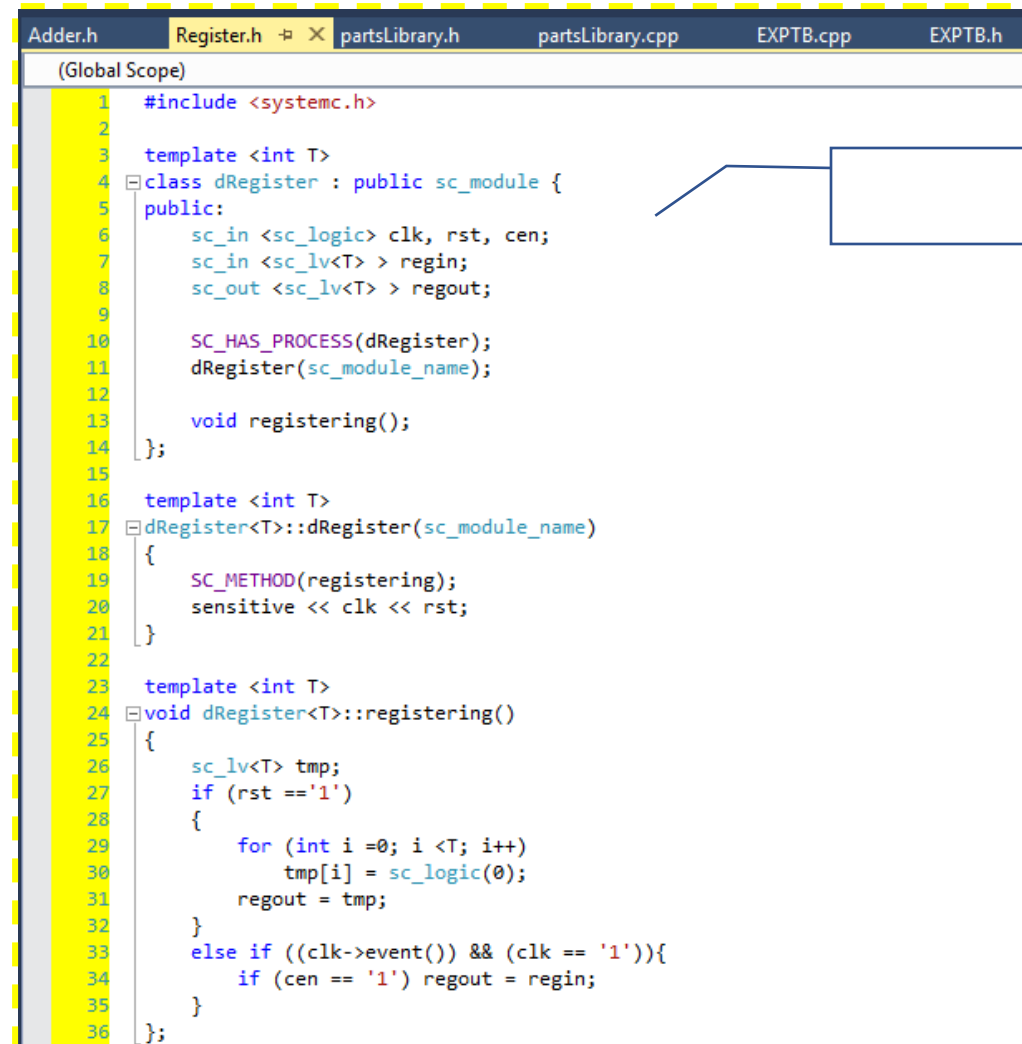
Exponential Circuit



```
1  #include <systemc.h>
2
3  template <int T> SC_MODULE (Adder){
4      sc_in <sc_lv<T> > in1, in2;
5      sc_out < sc_lv<T> > out;
6
7      SC_CTOR (Adder)
8      {
9          SC_METHOD(Adding);
10         sensitive << in1 << in2;
11     }
12     void Adding()
13     {
14         sc_lv<T> vout;
15         vout = (sc_uint<T>) in1 + (sc_uint<T>) in2;
16         out = vout;
17     }
18 };
```

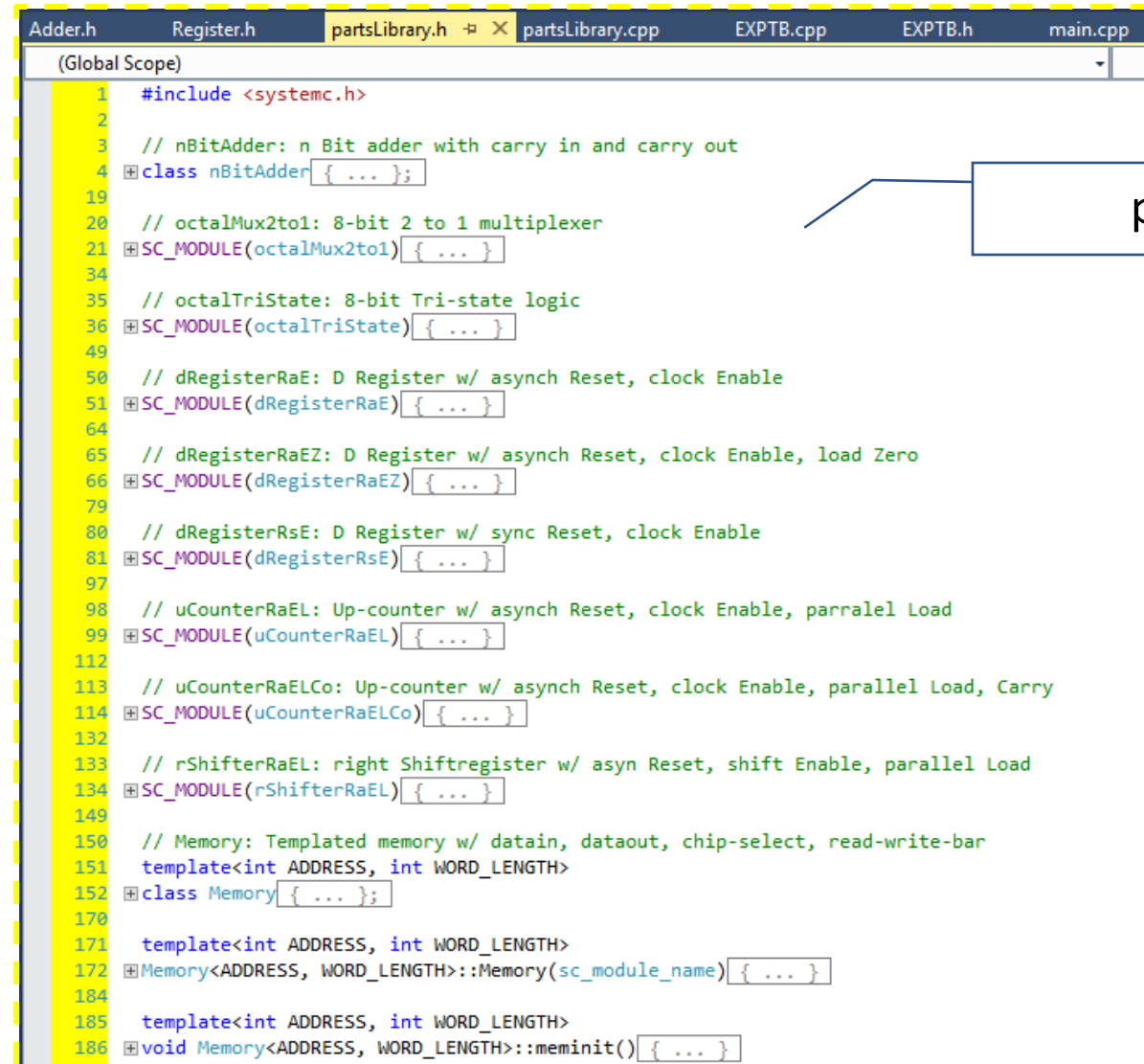
Adder.h

Exponential Circuit



```
1  #include <systemc.h>
2
3  template <int T>
4  class dRegister : public sc_module {
5  public:
6      sc_in <sc_logic> clk, rst, cen;
7      sc_in <sc_lv<T> > regin;
8      sc_out <sc_lv<T> > regout;
9
10     SC_HAS_PROCESS(dRegister);
11     dRegister(sc_module_name);
12
13     void registering();
14 };
15
16 template <int T>
17 dRegister<T>::dRegister(sc_module_name)
18 {
19     SC_METHOD(registering);
20     sensitive << clk << rst;
21 }
22
23 template <int T>
24 void dRegister<T>::registering()
25 {
26     sc_lv<T> tmp;
27     if (rst == '1')
28     {
29         for (int i = 0; i < T; i++)
30             tmp[i] = sc_logic(0);
31         regout = tmp;
32     }
33     else if ((clk->event()) && (clk == '1')){
34         if (cen == '1') regout = regin;
35     }
36 };
```


Exponential Circuit

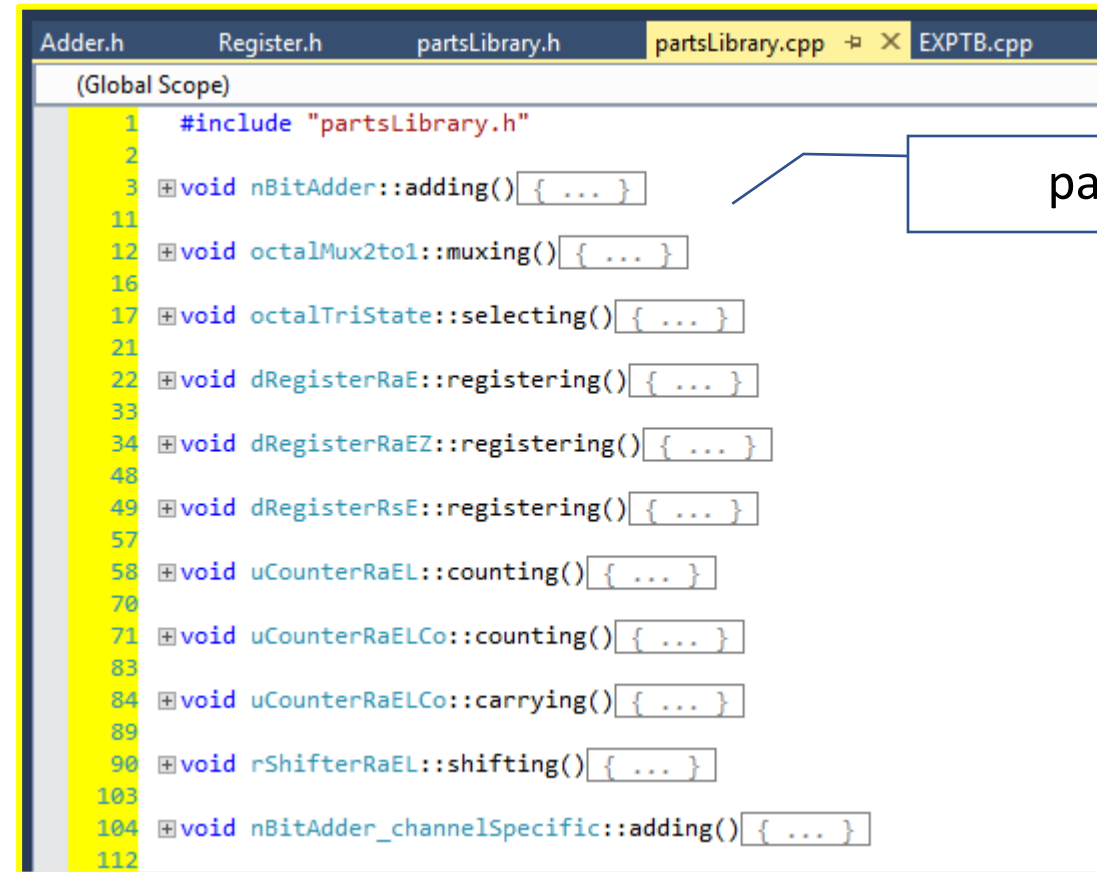


The image shows a code editor window with several tabs: Adder.h, Register.h, partsLibrary.h (active), partsLibrary.cpp, EXPTB.cpp, EXPTB.h, and main.cpp. The active tab, partsLibrary.h, contains the following code:

```
(Global Scope)
1  #include <systemc.h>
2
3  // nBitAdder: n Bit adder with carry in and carry out
4  class nBitAdder { ... };
19
20 // octalMux2to1: 8-bit 2 to 1 multiplexer
21 SC_MODULE(octalMux2to1) { ... }
34
35 // octalTriState: 8-bit Tri-state logic
36 SC_MODULE(octalTriState) { ... }
49
50 // dRegisterRaE: D Register w/ asynch Reset, clock Enable
51 SC_MODULE(dRegisterRaE) { ... }
64
65 // dRegisterRaEZ: D Register w/ asynch Reset, clock Enable, load Zero
66 SC_MODULE(dRegisterRaEZ) { ... }
79
80 // dRegisterRsE: D Register w/ sync Reset, clock Enable
81 SC_MODULE(dRegisterRsE) { ... }
97
98 // uCounterRaEL: Up-counter w/ asynch Reset, clock Enable, parallel Load
99 SC_MODULE(uCounterRaEL) { ... }
112
113 // uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
114 SC_MODULE(uCounterRaELCo) { ... }
132
133 // rShifterRaEL: right Shiftregister w/ asyn Reset, shift Enable, parallel Load
134 SC_MODULE(rShifterRaEL) { ... }
149
150 // Memory: Templated memory w/ datain, dataout, chip-select, read-write-bar
151 template<int ADDRESS, int WORD_LENGTH>
152 class Memory { ... };
170
171 template<int ADDRESS, int WORD_LENGTH>
172 Memory<ADDRESS, WORD_LENGTH>::Memory(sc_module_name) { ... }
184
185 template<int ADDRESS, int WORD_LENGTH>
186 void Memory<ADDRESS, WORD_LENGTH>::meminit() { ... }
```

partsLibrary.h

Exponential Circuit



The screenshot shows a code editor with a tab bar at the top containing 'Adder.h', 'Register.h', 'partsLibrary.h', 'partsLibrary.cpp' (selected), and 'EXPTB.cpp'. The main window displays the contents of 'partsLibrary.cpp' under the '(Global Scope)' header. The code includes a preprocessor directive and several function declarations, each followed by a placeholder '{ ... }'. A yellow highlight is applied to the first 10 lines of the code. A blue callout box with the text 'partsLibrary.cpp' points to the code area.

```
(Global Scope)
1  #include "partsLibrary.h"
2
3  void nBitAdder::adding() { ... }
11
12 void octalMux2to1::muxing() { ... }
16
17 void octalTriState::selecting() { ... }
21
22 void dRegisterRaE::registering() { ... }
33
34 void dRegisterRaEZ::registering() { ... }
48
49 void dRegisterRsE::registering() { ... }
57
58 void uCounterRaEL::counting() { ... }
70
71 void uCounterRaELCo::counting() { ... }
83
84 void uCounterRaELCo::carrying() { ... }
89
90 void rShifterRaEL::shifting() { ... }
103
104 void nBitAdder_channelSpecific::adding() { ... }
112
```

Exponential Circuit

```
EXPDesign.h  datapath_exp.h  controller_exp.h  controller_exp.cpp  EXPTB.h
(Global Scope)
1  #include <systemc.h>
2
3  SC_MODULE (controller){
4
5      sc_in <sc_logic> rst, clk, start, co;
6      sc_out <sc_logic> rwbar, memSel, selTableData;
7      sc_out <sc_logic> loadExponent, rstExponent;
8      sc_out <sc_logic> loadTerm, initTerm, rstTerm;
9      sc_out <sc_logic> rstResultReg, initResultReg, loadResultReg;
10     sc_out <sc_logic> enCounter, rstCounter, initCounter;
11     sc_out <sc_logic> busy;
12     sc_out <sc_logic> done;
13
14     enum states {INITIALIZE, WAIT_ON_START, CALC_NEXT_TERM_1,
15                  CALC NEXT TERM 2, ADD NEW TERM, CALC COMPLETE};
16     sc_signal <states> p_state, n_state;
17
18     SC_CTOR(controller)
19     {
20         SC_METHOD (comb_S_function);
21         sensitive << start << co << p_state;
22         SC_METHOD (comb_O_function);
23         sensitive << p_state;
24         SC_THREAD (seq_function);
25         sensitive << clk << rst;
26     };
27
28     void comb_S_function();
29     void comb_O_function();
30     void seq_function();
31 };
```

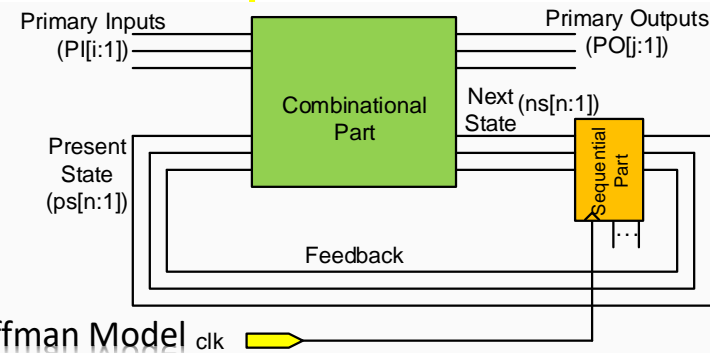
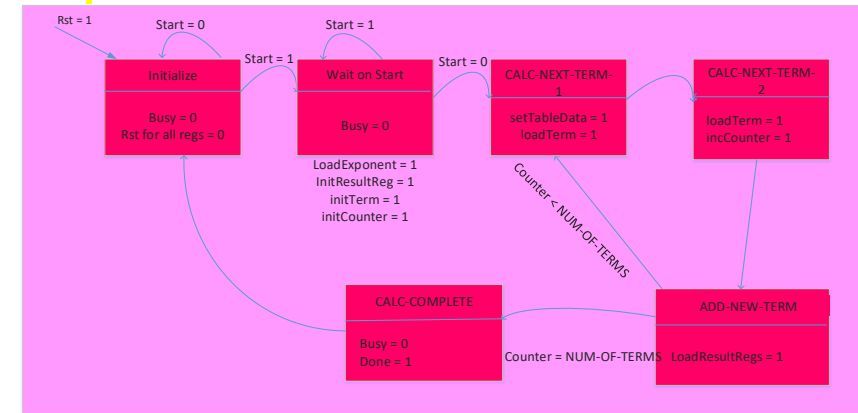
enum definition for states

Signal definition for next state and present state

Combinational Part

Sequential Part

controller_exp.h



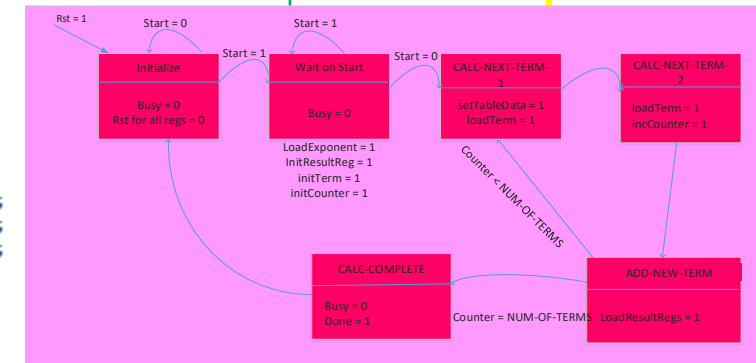
Huffman Model

Exponential Circuit

```
EXPDesign.h    datapath_exp.h    controller_exp.h    controller_exp.cpp  EXP
(Global Scope)
1  #include "controller_exp.h"
2
3  void controller::comb_0_function()
4  {
5      // Inactive output values
6      rwbar = SC_LOGIC_0;
7      memSel = SC_LOGIC_0;
8      selTableData = SC_LOGIC_0;
9      loadExponent = SC_LOGIC_0;
10     rstExponent = SC_LOGIC_0;
11     loadTerm = SC_LOGIC_0;
12     initTerm = SC_LOGIC_0;
13     rstTerm = SC_LOGIC_0;
14     rstResultReg = SC_LOGIC_0;
15     initResultReg = SC_LOGIC_0;
16     loadResultReg = SC_LOGIC_0;
17     enCounter = SC_LOGIC_0;
18     rstCounter = SC_LOGIC_0;
19     initCounter = SC_LOGIC_0;
20     busy = SC_LOGIC_0;
21     done = SC_LOGIC_0;
```

```
EXPDesign.h    datapath_exp.h    controller_exp.h    controller_exp.cpp  X
(Global Scope)
23  switch( p_state ){
24      case INITIALIZE:
25          rstExponent = SC_LOGIC_1; rstTerm = SC_LOGIC_1;
26          rstResultReg = SC_LOGIC_1; rstCounter = SC_LOGIC_1;
27          break;
28      case WAIT_ON_START:
29          loadExponent = SC_LOGIC_1;
30          initResultReg = SC_LOGIC_1;
31          initTerm = SC_LOGIC_1;
32          initCounter = SC_LOGIC_1;
33          enCounter = SC_LOGIC_1;
34          break;
35      case CALC_NEXT_TERM_1:
36          busy = SC_LOGIC_1;
37          selTableData = SC_LOGIC_1;
38          loadTerm = SC_LOGIC_1;
39          break;
40      case CALC_NEXT_TERM_2:
41          rwbar = SC_LOGIC_1;
42          memSel = SC_LOGIC_1;
43          selTableData = SC_LOGIC_0;
44          busy = SC_LOGIC_1;
45          loadTerm = SC_LOGIC_1;
46          enCounter = SC_LOGIC_1;
47          break;
48      case ADD_NEW_TERM:
49          loadResultReg = SC_LOGIC_1;
50          busy = SC_LOGIC_1;
51          break;
52      case CALC_COMPLETE:
53          done = SC_LOGIC_1;
54          busy = SC_LOGIC_0;
55          break;
56      default:
57          rwbar = SC_LOGIC_0;
58          memSel = SC_LOGIC_0;
59          selTableData = SC_LOGIC_0;
60          loadExponent = SC_LOGIC_0;
61          rstExponent = SC_LOGIC_0;
62          loadTerm = SC_LOGIC_0;
63          rstTerm = SC_LOGIC_0;
64          initTerm = SC_LOGIC_0;
65          rstResultReg = SC_LOGIC_0;
66          initResultReg = SC_LOGIC_0;
67          loadResultReg = SC_LOGIC_0;
68          initResultReg = SC_LOGIC_0;
69          enCounter = SC_LOGIC_0;
70          rstCounter = SC_LOGIC_0;
71          initCounter = SC_LOGIC_0;
72          busy = SC_LOGIC_0;
73          done = SC_LOGIC_0;
74          break;
75  }
```

controller_exp.cpp



Exponential Circuit

EXPDesign.h datapath_exp.h controller_exp.h controller_exp.cpp EXPTB.h

(Global Scope)

```
77 void controller::comb_S_function(){
78     n_state = INITIALIZE;
79     switch( p_state ){
80     case INITIALIZE:
81         if (start == '0')
82             n_state = INITIALIZE;
83         else
84             n_state = WAIT_ON_START;
85         break;
86     case WAIT_ON_START:
87         if( start == '0')
88             n_state = CALC_NEXT_TERM_1;
89         else
90             n_state = WAIT_ON_START;
91         break;
92     case CALC_NEXT_TERM_1:
93         n_state = CALC_NEXT_TERM_2;
94         break;
95     case CALC_NEXT_TERM_2:
96         n_state = ADD_NEW_TERM;
97         break;
98     case ADD_NEW_TERM:
99         if( co == '0' )
100             n_state = CALC_NEXT_TERM_1;
101         else
102             n_state = CALC_COMPLETE;
103         break;
104     case CALC_COMPLETE:
105         if ( start == '0')
106             n_state = INITIALIZE;
107         else
108             n_state = WAIT_ON_START;
109         break;
110     default:
111         n_state = INITIALIZE;
112         break;
113     }
114 }
```

controller_exp.cpp

```
graph LR
    Rst1[Rst = 1] --> Init[INITIALIZE  
Busy = 0  
Rst for all regs = 0]
    Init -- Start = 1 --> Wait[WAIT_ON_START  
Busy = 0  
LoadExponent = 1  
InitResultReg = 1  
initTerm = 1  
incCounter = 1]
    Wait -- Start = 0 --> Calc1[CALC_NEXT_TERM_1  
setTableData = 1  
loadTerm = 1]
    Calc1 -- Start = 0 --> Calc2[CALC_NEXT_TERM_2  
loadTerm = 1  
incCounter = 1]
    Calc2 -- Counter < NUM-OF-TERMS --> Add[ADD_NEW_TERM  
LoadResultRegs = 1]
    Add -- Counter = NUM-OF-TERMS --> CalcComp[CALC_COMPLETE  
Busy = 0  
Done = 1]
    CalcComp -- start = 0 --> Init
```

EXPDesign.h datapath_exp.h controller_exp.h controller_exp.cpp EXPTB.h

(Global Scope)

```
116 void controller::seq_function(){
117     while (1){
118         if (rst == '1')
119             p_state = INITIALIZE;
120         else if (clk->event() &&(clk == '1'))
121             p_state = n_state;
122         wait();
123     }
124 }
```

Exponential Circuit

The image shows a code editor window with the following tabs: EXPDesign.h, EXPTB.h (selected), datapath_exp.h, and controller_exp.h. The code is in the Global Scope and is as follows:

```
1  #include "EXPDesign.h"
2
3  SC_MODULE(EXPTB){
4      sc_signal<sc_logic> clk, rst, start;
5      sc_signal<sc_lv<8>> x;
6      sc_signal<sc_lv<10>> result;
7      sc_signal<sc_logic> busy;
8      sc_signal<sc_logic> done;
9
10     EXP* Exp1;
11
12     SC_CTOR (EXPTB){
13         Exp1= new EXP ("Exponential_1");
14         Exp1->clk(clk);
15         Exp1->rst(rst);
16         Exp1->start(start);
17         Exp1->x(x);
18         Exp1->result(result);
19         Exp1->busy(busy);
20         Exp1->done(done);
21
22         //
23         SC_THREAD(inputing);
24         SC_THREAD(reseting);
25         SC_THREAD(clocking);
26         SC_METHOD(displaying);
27         sensitive <<done.posedge_event();
28     }
29     void inputing();
30     void resetting();
31     void clocking();
32     void displaying();
33 }
```

Annotations in the image:

- A box labeled **EXPTB.h** points to the file name in the tab bar.
- A box labeled **Exponential instantiation** points to the instantiation of the EXP module in the SC_CTOR (lines 13-20).
- A box labeled **Input generation** points to the thread definitions (lines 23-32).

Exponential Circuit

The image displays two windows from a SystemC IDE, showing the implementation of an exponential circuit. The left window shows the `EXPTB.cpp` file, and the right window shows the `datapath_exp.h` file. A callout box labeled `EXPTB.cpp` points to the `EXPTB::clocking()` function in the right window.

EXPTB.cpp (Left Window):

```
1  #include "EXPTB.h"
2
3  void EXPTB::inputing(){
4      start = SC_LOGIC_0;
5      wait(10, SC_NS);
6      start = SC_LOGIC_1;
7      x = "1111111";
8      wait(60, SC_NS);
9      start = SC_LOGIC_0;
10     wait(2400, SC_NS);
11     start = SC_LOGIC_1;
12     x = "10000000";
13     wait(100, SC_NS);
14     start = SC_LOGIC_0;
15     wait(2400, SC_NS);
16     start = SC_LOGIC_1;
17     x = "00000000";
18     wait(100, SC_NS);
19     start = SC_LOGIC_0;
20     wait(2400, SC_NS);
21     start = SC_LOGIC_1;
22     x = "00000001";
23     wait(100, SC_NS);
24     start = SC_LOGIC_0;
25 }
26
27 void EXPTB::clocking(){ ... }
28
29 void EXPTB::reseting(){ ... }
30
31 void EXPTB::displaying(){ ... }
```

datapath_exp.h (Right Window):

```
27 void EXPTB::clocking(){
28     int i;
29     clk = sc_logic('1');
30     for (i=0; i <=50; i++)
31     {
32         clk = sc_logic('0');
33         wait (50, SC_NS);
34         clk = sc_logic('1');
35         wait (50, SC_NS);
36     }
37 }
38
39 void EXPTB::reseting(){
40     rst = (sc_logic)'0';
41     wait (5, SC_NS);
42     rst = (sc_logic)'1';
43     wait (5, SC_NS);
44     rst = (sc_logic)'0';
45 }
46
47 void EXPTB::displaying(){
48     cout << " x = " << x.read() << " result = " << result.read() << endl;
49 }
50
```

Exponential Circuit

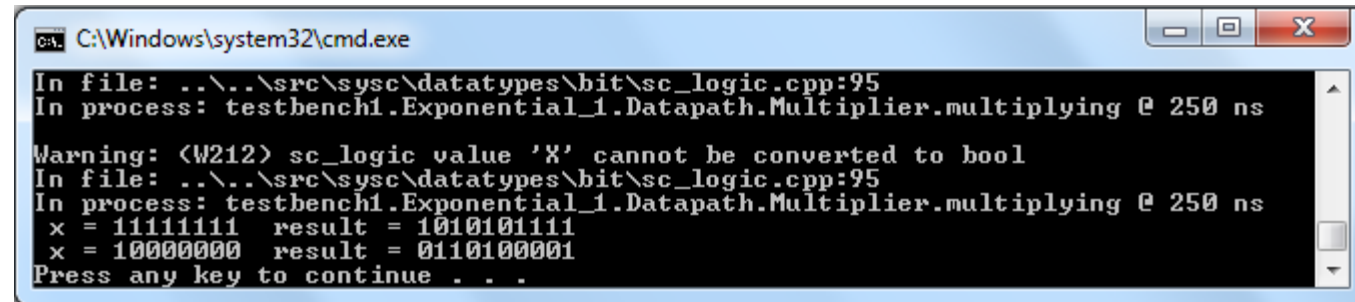
The image shows a code editor window with the following tabs: EXPTB.h, EXPTB.cpp, main.cpp (active), Adder.h, Register.h, and datapath_exp.h. The code in main.cpp is as follows:

```
(Global Scope)
1  #include "EXPTB.h"
2
3  int sc_main (int argc, char **argv){
4
5      EXPTB TB1 ("testbench1");
6
7      sc_trace_file* vcdfile;
8      vcdfile = sc_create_vcd_trace_file("Exponential_test");
9      sc_trace(vcdfile, TB1.clk, "clk");
10     sc_trace(vcdfile, TB1.rst, "rst");
11     sc_trace(vcdfile, TB1.start, "start");
12     sc_trace(vcdfile, TB1.busy, "busy");
13     sc_trace(vcdfile, TB1.done, "done");
14     sc_trace(vcdfile, TB1.x, "x");
15     sc_trace(vcdfile, TB1.result, "result");
16
17     sc_start (12000, SC_NS);
18     return 0;
19 }
```

Annotations in the image:

- A box labeled "Main.cpp" points to the main.cpp tab.
- A box labeled "Top-Level instantiation" points to line 5: `EXPTB TB1 ("testbench1");`.
- A box labeled "VCD file and tracing signals" points to the block of code from line 7 to line 15, which sets up the VCD file and traces various signals.

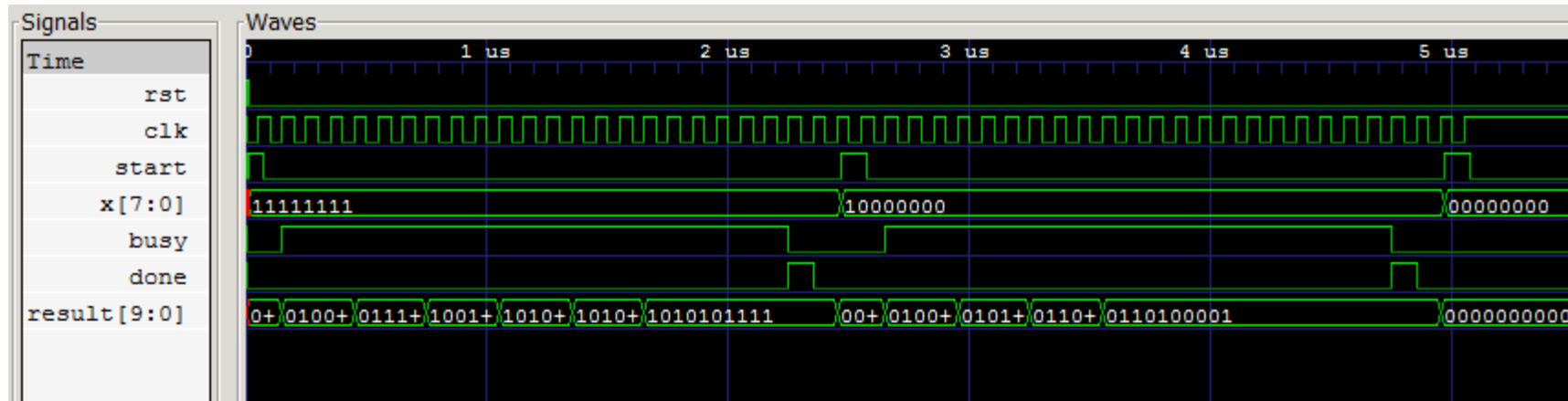
Exponential Circuit



```
C:\Windows\system32\cmd.exe
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 = 101010111
x = 100000000 result = 0110100001
Press any key to continue . . .
```

Exponential Circuit

- ◉ Waveform in a VCD viewer



SystemC Modeling

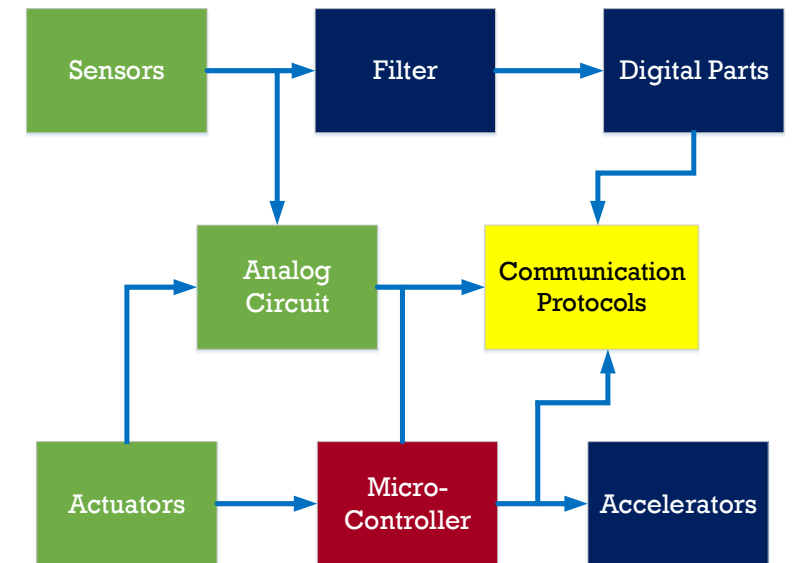
- + Taking Off From C++
- + SystemC Gate-Level Modeling
 - Utilities for HDL Orientation
- + Timing & SystemC RT-Level Modeling
- + Components for RTL Design
- + SystemC RTL Design - Example
- SystemC Functional Modeling
- + SystemC Functional Design - Example
- Summary

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



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

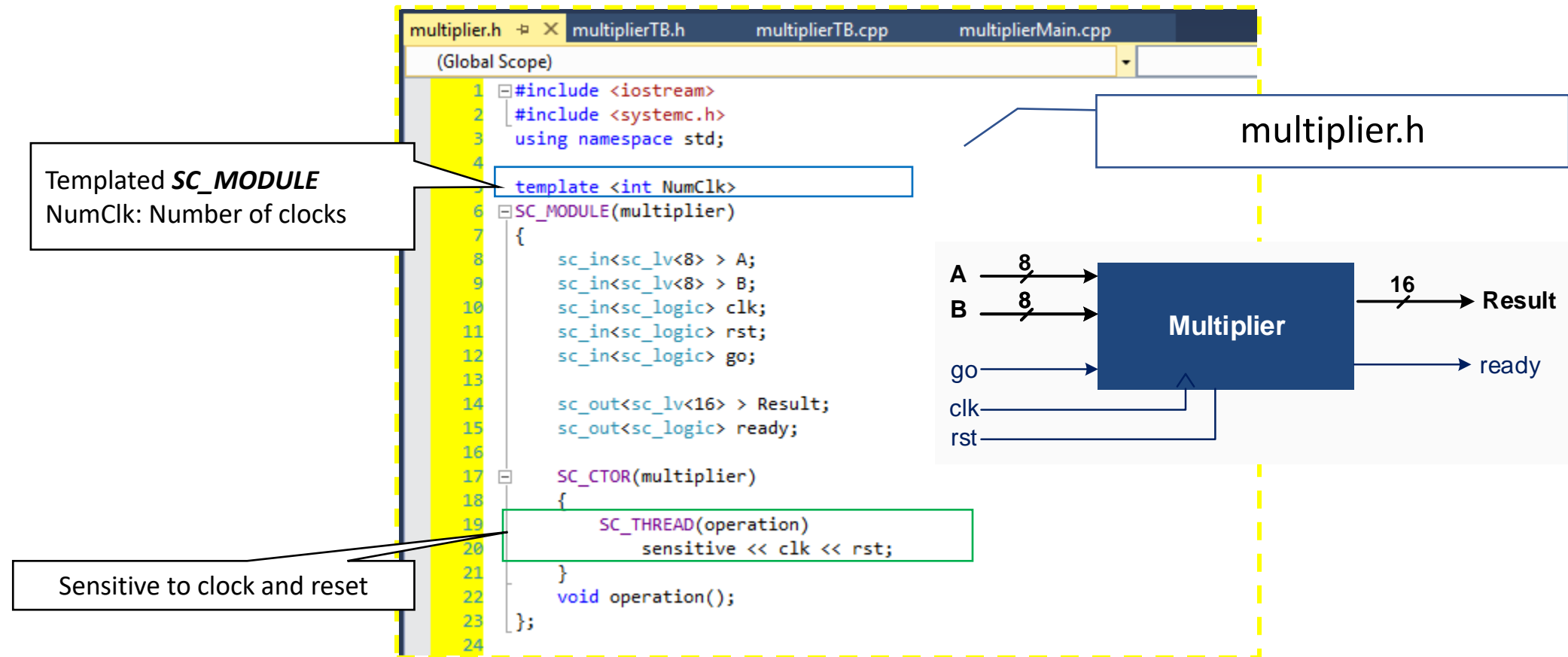
Multiplier

Divider

Exponential Circuit

Summary

Multiplier



Multiplier

The image shows a code editor window with the file `multiplier.h` selected. The code is a SystemC template for a multiplier. Three callout boxes highlight specific parts of the code:

- Asynchronous reset:** Points to the `if (rst == '1')` block where `ready` is set to `SC_LOGIC_0` and `Result` is initialized to `"0000000000000000"`.
- Mimicking clock:** Points to the `for(int i = 0; i < NumClk; i++) wait(clk->posedge_event());` loop, which simulates a clock by waiting for a series of clock edges.
- Performing multiplication function:** Points to the final calculation `Result = A->read().to_uint() * B->read().to_uint();` inside the `while` loop.

```
25 template <int NumClk>
26 void multiplier<NumClk>::operation()
27 {
28     while (true)
29     {
30         if (rst == '1')
31         {
32             ready = SC_LOGIC_0;
33             Result = "0000000000000000";
34         }
35
36         else if ((clk == '1') && (clk->event()))
37         {
38             if (go == '1')
39             {
40                 ready = SC_LOGIC_0;
41                 Result = "XXXXXXXXXXXXXXXXXX";
42
43                 for(int i = 0; i < NumClk; i++)
44                     wait(clk->posedge_event());
45
46                 ready = SC_LOGIC_1;
47                 Result = A->read().to_uint() * B->read().to_uint();
48             }
49         }
50         wait();
51     }
52 }
```


Multiplier

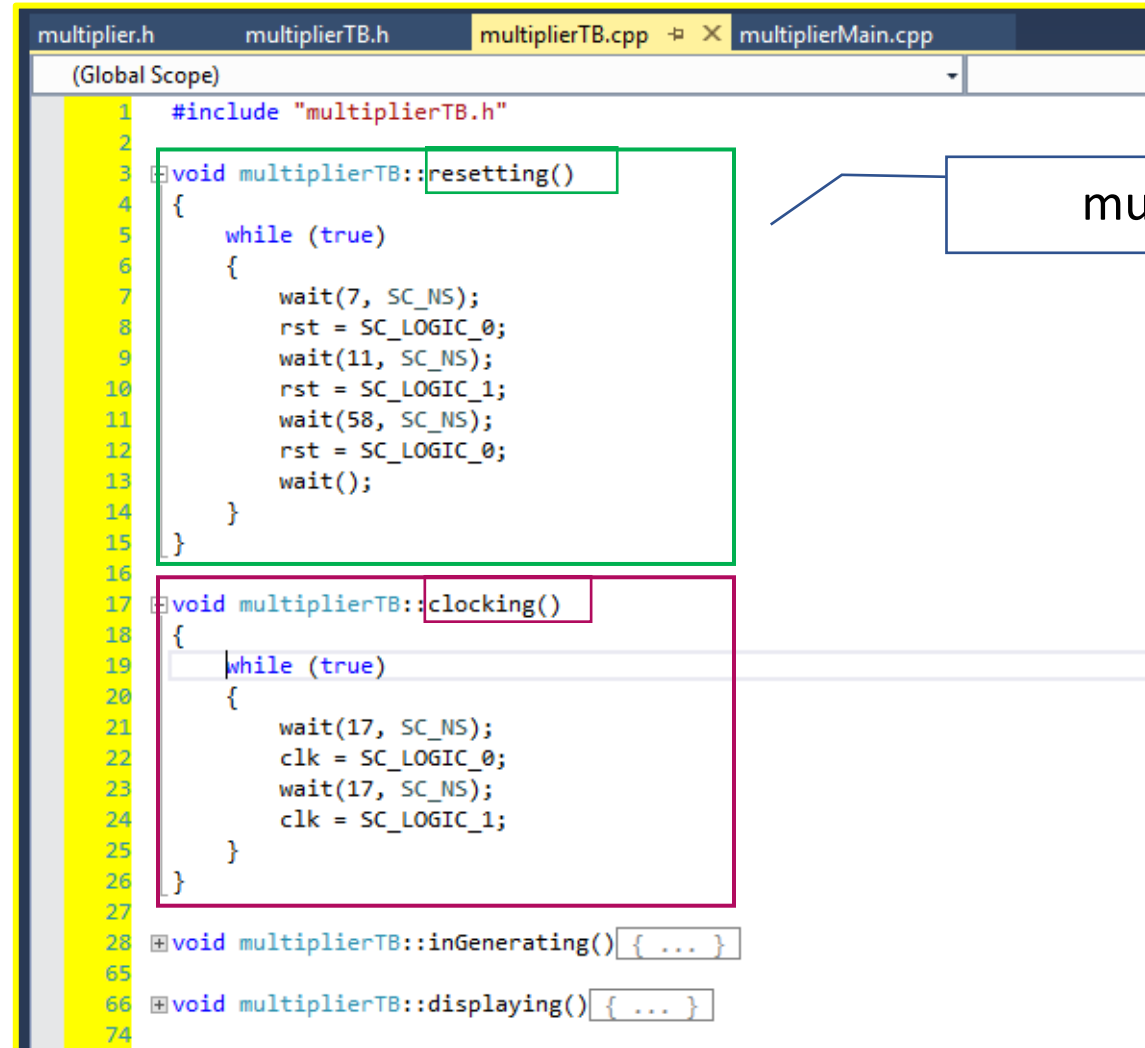
The image shows a code editor window with four tabs: multiplier.h, multiplierTB.h (active), multiplierTB.cpp, and multiplierMain.cpp. The active tab displays the following C++ code:

```
(Global Scope)
1  #include "multiplier.h"
2
3  SC_MODULE(multiplierTB)
4  {
5      sc_signal<sc_logic> clk, rst, go, ready;
6      sc_signal<sc_lv<8> > A, B;
7      sc_signal<sc_lv<16> > Result;
8
9      multiplier<12>* MUL;
10
11  SC_CTOR(multiplierTB)
12  {
13      MUL = new multiplier<12> ("MUL_Instance");
14          (*MUL) (A, B, clk, rst, go, Result, ready);
15
16      SC_THREAD(resetting);
17      SC_THREAD(clocking);
18      SC_THREAD(inGenerating);
19      SC_THREAD(displaying);
20      sensitive << ready;
21
22  }
23  void resetting();
24  void clocking();
25  void inGenerating();
26  void displaying();
27  };
```

Annotations in the image include:

- A box labeled "multiplierTB.h" pointing to the file tab.
- A box labeled "Multiplier instantiation" pointing to line 13: `MUL = new multiplier<12> ("MUL_Instance");`
- A box labeled "Input generation" pointing to lines 16-20, which define the threads for generating test inputs.

Multiplier



```
multiplier.h  multiplierTB.h  multiplierTB.cpp  multiplierMain.cpp
(Global Scope)
1  #include "multiplierTB.h"
2
3  void multiplierTB::resetting()
4  {
5      while (true)
6      {
7          wait(7, SC_NS);
8          rst = SC_LOGIC_0;
9          wait(11, SC_NS);
10         rst = SC_LOGIC_1;
11         wait(58, SC_NS);
12         rst = SC_LOGIC_0;
13         wait();
14     }
15 }
16
17 void multiplierTB::clocking()
18 {
19     while (true)
20     {
21         wait(17, SC_NS);
22         clk = SC_LOGIC_0;
23         wait(17, SC_NS);
24         clk = SC_LOGIC_1;
25     }
26 }
27
28 void multiplierTB::inGenerating() { ... }
65
66 void multiplierTB::displaying() { ... }
74
```

multiplierTB.cpp

Multiplier

The image shows two windows of a SystemC testbench for a multiplier. The top window, titled 'multiplierTB.cpp', shows the 'inGenerating()' function. The bottom window, titled 'multiplierTB.cpp', shows the 'displaying()' function. Annotations highlight specific parts of the code.

multiplierTB.cpp (inGenerating())

```
28 void multiplierTB::inGenerating()
29 {
30     while (true)
31     {
32         A = "00000000";
33         B = "00000000";
34         go = SC_LOGIC_0;
35         wait(60, SC_NS);
36
37         A = "00000011";
38         B = "00000010";
39         go = SC_LOGIC_1;
40         wait(60, SC_NS);
41         go = SC_LOGIC_0;
42
43         wait(500, SC_NS);
44         A = "00001010";
45         B = "01000000";
46         go = SC_LOGIC_1;
47         wait(60, SC_NS);
48         go = SC_LOGIC_0;
49
50         wait(500, SC_NS);
51         A = "00000110";
52         B = "00010100";
53         go = SC_LOGIC_0;
54
55         wait(120, SC_NS);
56         A = "00000110";
57         B = "01010100";
58         go = SC_LOGIC_1;
59         wait(60, SC_NS);
60         go = SC_LOGIC_0;
61
62         wait();
63     }
64 }
```

Annotations:

- Input initialization:** Points to the initialization of A, B, and go in the first iteration (lines 32-35).
- go signal has not been issued:** Points to the state where go is SC_LOGIC_0 and waiting (lines 40-41).

multiplierTB.cpp (displaying())

```
66 void multiplierTB::displaying()
67 {
68     while (true){
69         if (ready == '1')
70             cout << A << " * " << B << " = " << Result << " - Time : " << sc_time_stamp() << endl;
71         wait();
72     }
73 }
```

Multiplier

The image shows a code editor window with four tabs: `multiplier.h`, `multiplierTB.h`, `multiplierTB.cpp`, and `multiplierMain.cpp`. The `multiplierMain.cpp` tab is active, showing the following code:

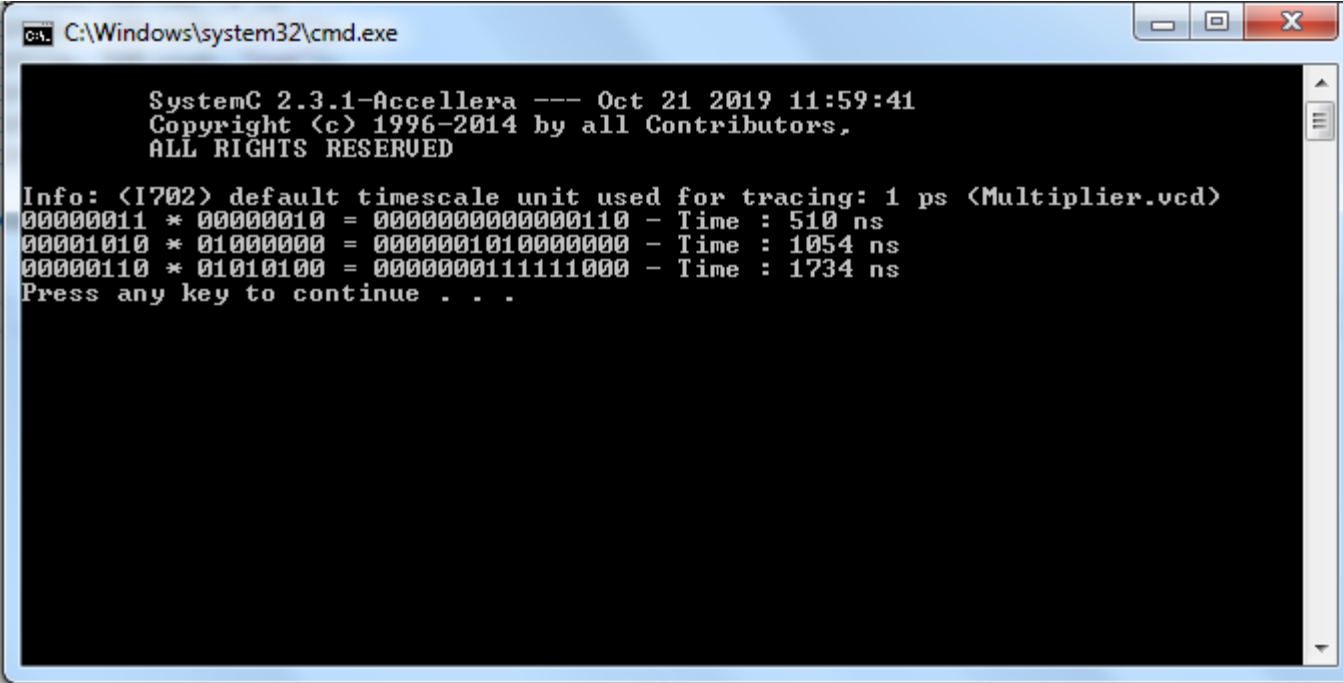
```
(Global Scope)
1  #include "multiplierTB.h"
2
3  int sc_main (int argc , char *argv[])
4  {
5      multiplierTB* TOP = new multiplierTB ("multiplierTB_Instance");
6
7      sc_trace_file* VCDFile;
8      VCDFile = sc_create_vcd_trace_file("Multiplier");
9      sc_trace(VCDFile, TOP->A, "A");
10     sc_trace(VCDFile, TOP->B, "B");
11     sc_trace(VCDFile, TOP->clk, "clk");
12     sc_trace(VCDFile, TOP->rst, "rst");
13     sc_trace(VCDFile, TOP->go, "go");
14     sc_trace(VCDFile, TOP->ready, "ready");
15     sc_trace(VCDFile, TOP->Result, "Result");
16
17     sc_start(2000, SC_NS);
18     return 0;
19 }
```

Annotations with callout boxes:

- Top-Level instantiation**: Points to line 5, `multiplierTB* TOP = new multiplierTB ("multiplierTB_Instance");`
- VCD file and tracing signals**: Points to lines 8-15, which define the VCD file and trace various signals.
- multiplierMain.cpp**: A box pointing to the active tab.

– SystemC Functional Design - Example

Multiplier



```
C:\Windows\system32\cmd.exe

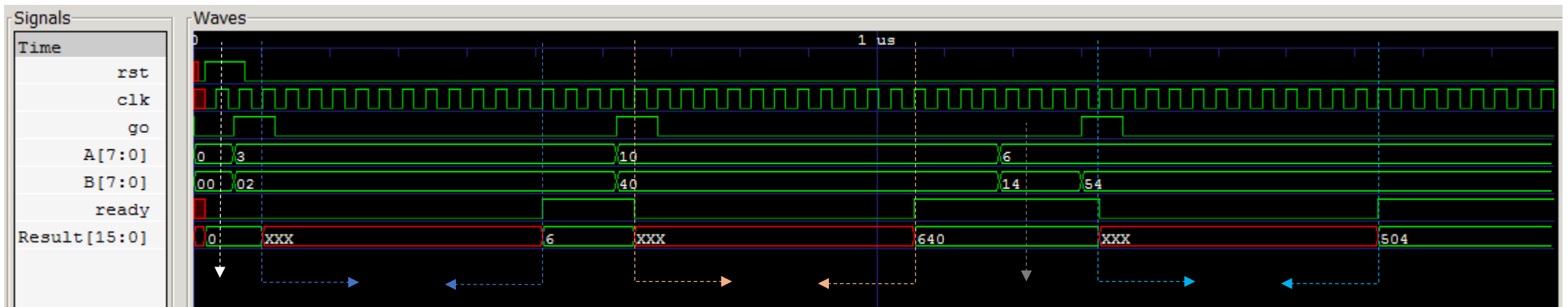
SystemC 2.3.1-Accellera --- Oct 21 2019 11:59:41
Copyright (c) 1996-2014 by all Contributors,
ALL RIGHTS RESERVED

Info: <I702> default timescale unit used for tracing: 1 ps <Multiplier.vcd>
00000011 * 00000010 = 00000000000000110 - Time : 510 ns
00001010 * 01000000 = 0000001010000000 - Time : 1054 ns
00000110 * 01010100 = 0000000111111000 - Time : 1734 ns
Press any key to continue . . .
```

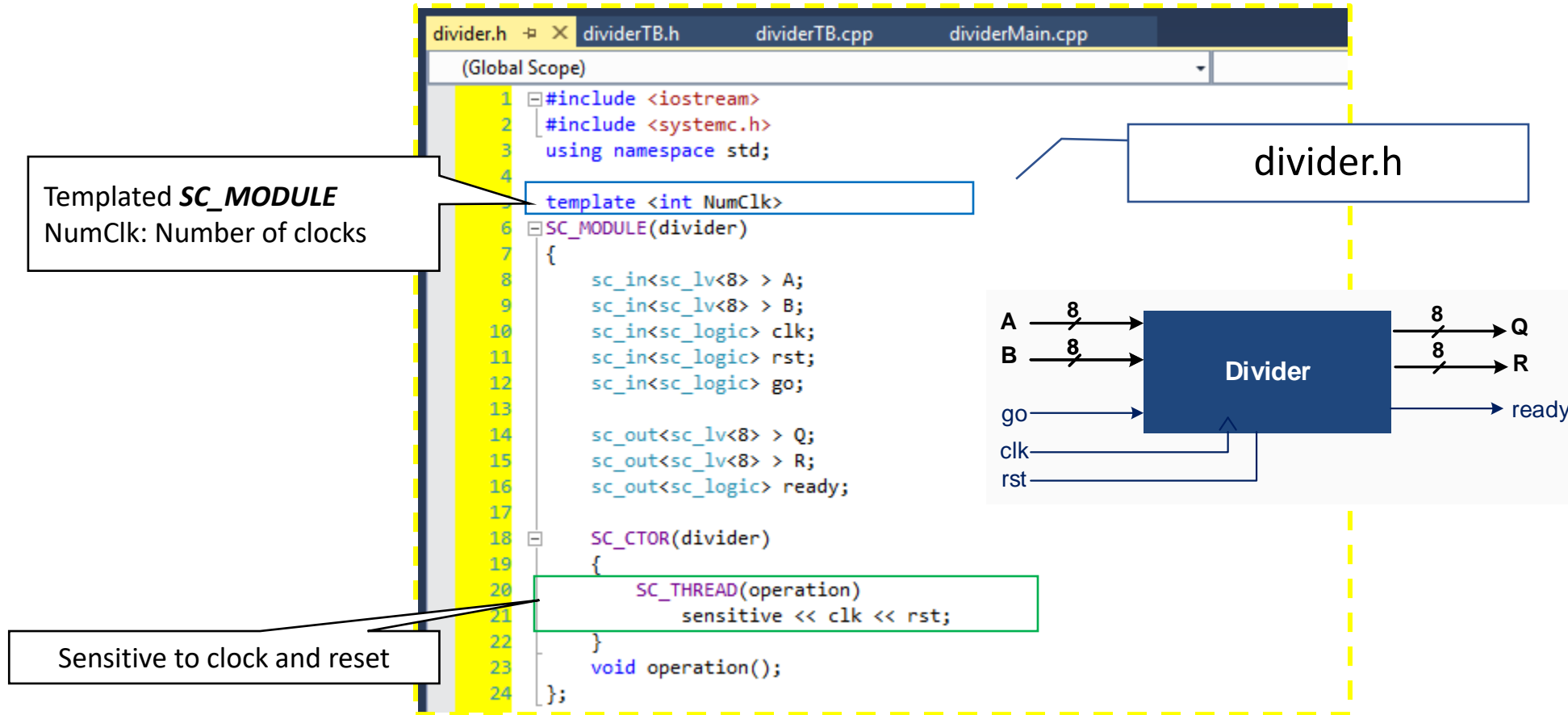
– SystemC Functional Design - Example

Multiplier

⦿ Waveform in a VCD viewer



Divider



Divider

The image shows a code editor window with the following tabs: `divider.h`, `dividerTB.h`, `dividerTB.cpp`, and `dividerMain.cpp`. The `divider.h` file is selected, showing the following code:

```
26 template <int NumClk>
27 void divider<NumClk>::operation()
28 {
29     sc_lv<8> Q_var;
30
31     while (true)
32     {
33         if (rst == '1')
34         {
35             ready = SC_LOGIC_0;
36             Q = "00000000";
37             R = "00000000";
38         }
39
40         else if ((clk == '1') && (clk->event()))
41         {
42             if (go == '1')
43             {
44                 ready = SC_LOGIC_0;
45                 Q = "XXXXXXXX";
46                 R = "XXXXXXXX";
47
48                 for(int i = 0; i < NumClk; i++)
49                     wait(clk->posedge_event());
50
51                 ready = SC_LOGIC_1;
52                 Q_var = A->read().to_uint() / B->read().to_uint();
53                 Q = Q_var;
54                 R = (A->read().to_uint()) - (B->read().to_uint() * Q_var.to_uint());
55             }
56         }
57         wait();
58     }
59 }
```

Annotations on the left side of the code editor:

- Asynchronous reset**: Points to the `if (rst == '1')` block (lines 33-38).
- Mimicking clock**: Points to the `for(int i = 0; i < NumClk; i++) wait(clk->posedge_event());` loop (lines 48-49).
- Calculating Quotient and Remainder**: Points to the calculation of `Q_var` and `R` (lines 52-54).

A box on the right side of the code editor is labeled `divider.h`.

© Zainalabedin Navabi - SystemC Modeling

Divider

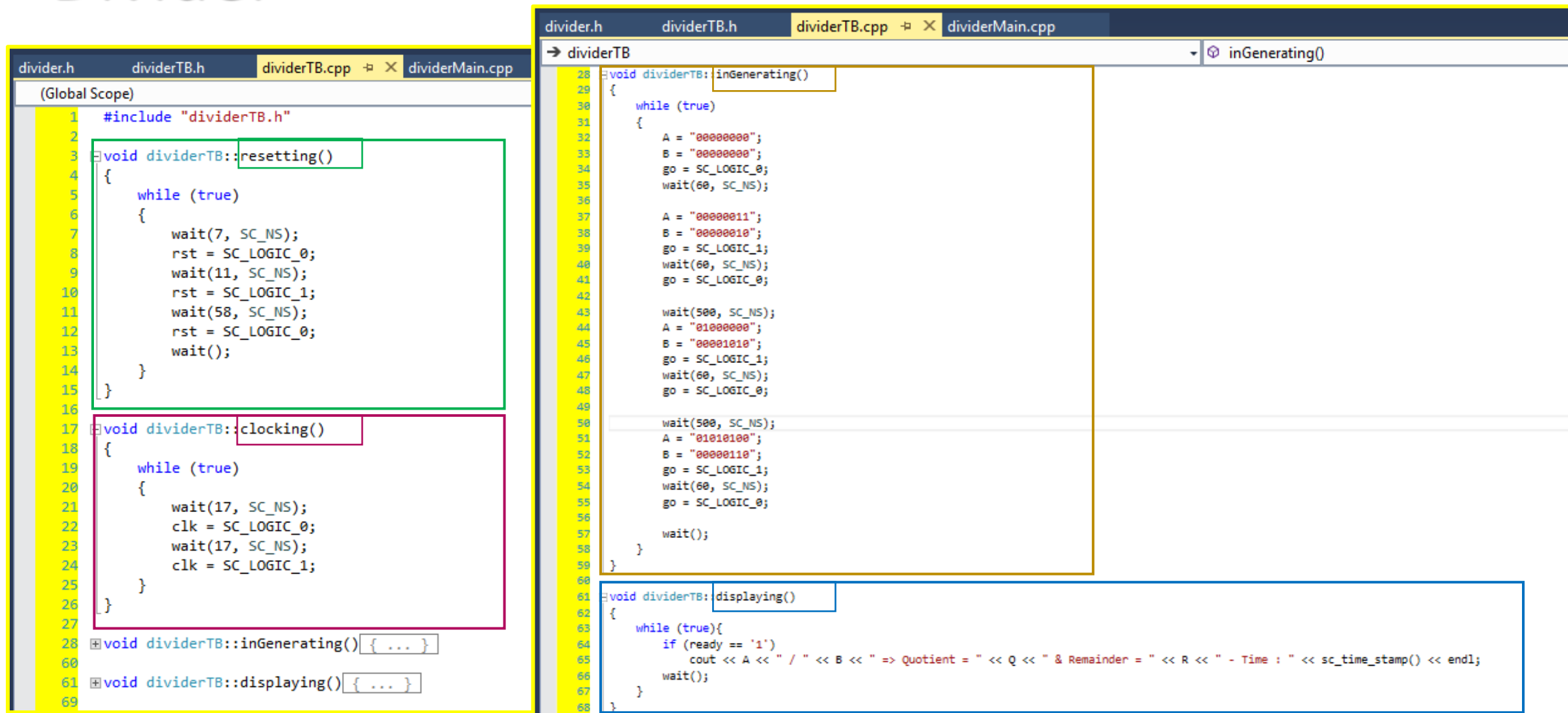
The image shows a code editor window with the following tabs: `divider.h`, `dividerTB.h` (active), `dividerTB.cpp`, and `dividerMain.cpp`. The editor displays the content of `dividerTB.h` in Global Scope. The code is as follows:

```
1  #include "divider.h"
2
3  SC_MODULE(dividerTB)
4  {
5      sc_signal<sc_logic> clk, rst, go, ready;
6      sc_signal<sc_lv<8> > A, B, Q, R;
7
8      divider<9>* DIV;
9
10     SC_CTOR(dividerTB)
11     {
12         DIV = new divider<9> ("DIV_Instance");
13         (*DIV) (A, B, clk, rst, go, Q, R, ready);
14
15         SC_THREAD(resetting);
16         SC_THREAD(clocking);
17         SC_THREAD(inGenerating);
18         SC_THREAD(displaying);
19         sensitive << ready;
20
21     }
22     void resetting();
23     void clocking();
24     void inGenerating();
25     void displaying();
26 };
```

Annotations in the image:

- A box labeled `dividerTB.h` points to the `dividerTB` module definition.
- A box labeled "Divider instantiation" points to line 12: `DIV = new divider<9> ("DIV_Instance");`
- A box labeled "Input generation" points to lines 15-18, which define the threads for generating inputs: `SC_THREAD(resetting);`, `SC_THREAD(clocking);`, `SC_THREAD(inGenerating);`, and `SC_THREAD(displaying);`.

Divider



```
divider.h

dividerTB.hdividerTB.cppdividerMain.cpp



(Global Scope)



1#include "dividerTB.h"



2



3void dividerTB::reseting()



4{



5while (true)



6{



7wait(7, SC_NS);



8rst = SC_LOGIC_0;



9wait(11, SC_NS);



10rst = SC_LOGIC_1;



11wait(58, SC_NS);



12rst = SC_LOGIC_0;



13wait();



14}



15}



16



17void dividerTB::clocking()



18{



19while (true)



20{



21wait(17, SC_NS);



22clk = SC_LOGIC_0;



23wait(17, SC_NS);



24clk = SC_LOGIC_1;



25}



26}



27



28void dividerTB::inGenerating() { ... }



60



61void dividerTB::displaying() { ... }



69


```

```
dividerTB.cpp

divider.hdividerTB.hdividerTB.cppdividerMain.cpp



→ dividerTB



28void dividerTB::inGenerating()



29{



30while (true)



31{



32A = "00000000";



33B = "00000000";



34go = SC_LOGIC_0;



35wait(60, SC_NS);



36



37A = "00000011";



38B = "00000010";



39go = SC_LOGIC_1;



40wait(60, SC_NS);



41go = SC_LOGIC_0;



42



43wait(500, SC_NS);



44A = "01000000";



45B = "00001010";



46go = SC_LOGIC_1;



47wait(60, SC_NS);



48go = SC_LOGIC_0;



49



50wait(500, SC_NS);



51A = "01010100";



52B = "00000110";



53go = SC_LOGIC_1;



54wait(60, SC_NS);



55go = SC_LOGIC_0;



56



57wait();



58}



59}



60



61void dividerTB::displaying()



62{



63while (true){



64if (ready == '1')



65cout << A << " / " << B << " => Quotient = " << Q << " & Remainder = " << R << " - Time : " << sc_time_stamp() << endl;



66wait();



67}



68}


```

Divider

The image shows a code editor window with the file `dividerMain.cpp` selected. The code is in the Global Scope and contains the following lines:

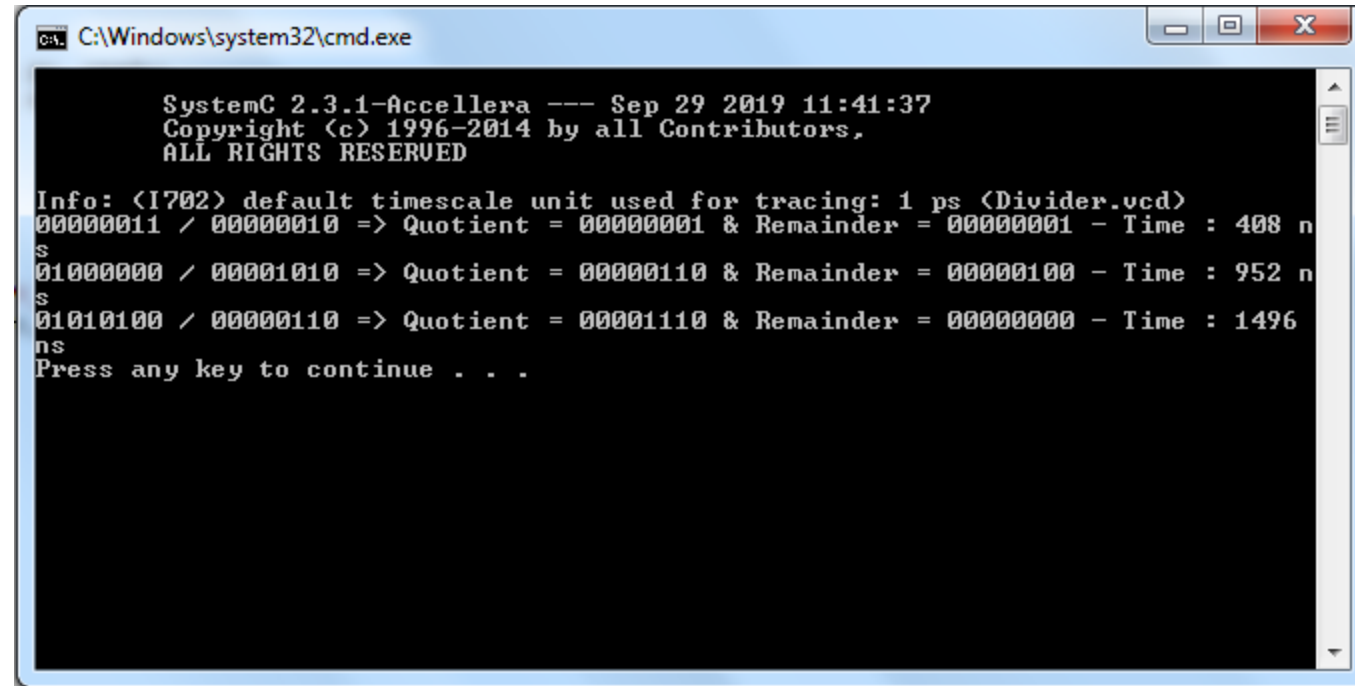
```
1  #include "dividerTB.h"
2
3  int sc_main (int argc , char *argv[])
4  {
5      dividerTB* TOP = new dividerTB ("dividerTB_Instance");
6
7      sc_trace_file* VCDFile;
8      VCDFile = sc_create_vcd_trace_file("Divider");
9      sc_trace(VCDFile, TOP->A, "A");
10     sc_trace(VCDFile, TOP->B, "B");
11     sc_trace(VCDFile, TOP->clk, "clk");
12     sc_trace(VCDFile, TOP->rst, "rst");
13     sc_trace(VCDFile, TOP->go, "go");
14     sc_trace(VCDFile, TOP->ready, "ready");
15     sc_trace(VCDFile, TOP->Q, "Q");
16     sc_trace(VCDFile, TOP->R, "R");
17
18     sc_start(2000, SC_NS);
19     return 0;
20 }
```

Annotations in the image:

- A callout box labeled "Top-Level instantiation" points to line 5: `dividerTB* TOP = new dividerTB ("dividerTB_Instance");`
- A callout box labeled "VCD file and tracing signals" points to lines 7-16, which define the VCD file and trace various signals.
- A callout box labeled "dividerMain.cpp" points to the file name in the editor's tab bar.

– SystemC Functional Design - Example

Divider



```
C:\Windows\system32\cmd.exe

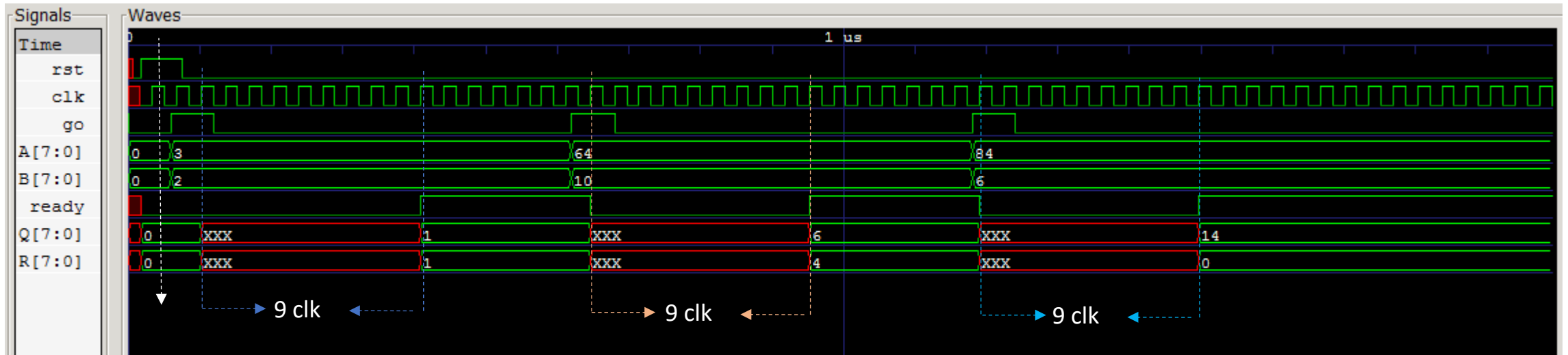
SystemC 2.3.1-Accellera --- Sep 29 2019 11:41:37
Copyright (c) 1996-2014 by all Contributors,
ALL RIGHTS RESERVED

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

– SystemC Functional Design - Example

Divider

◉ Waveform in a VCD viewer

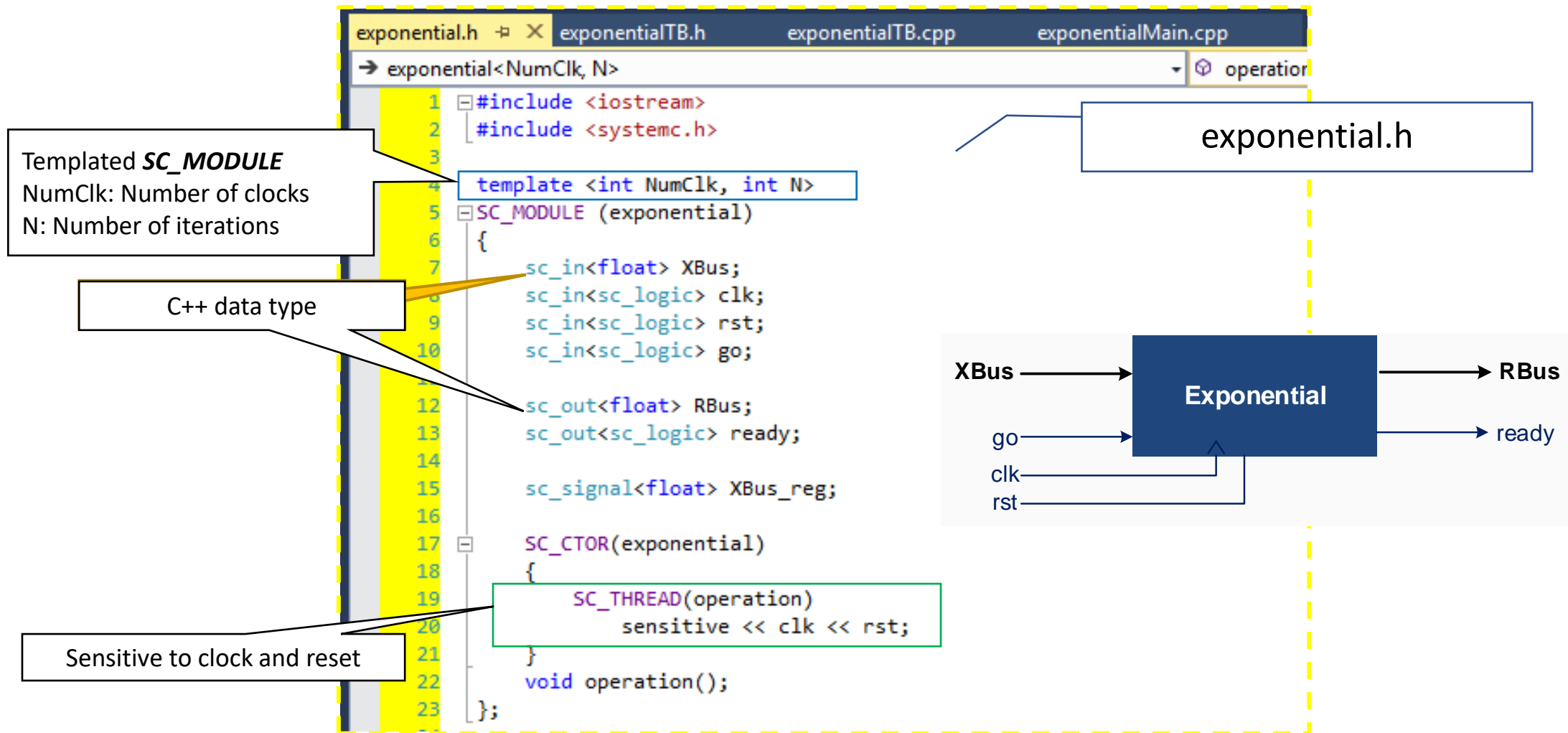


Exponential

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

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

Exponential



Exponential

The image shows a code editor with four tabs: `exponential.h`, `exponentialTB.h`, `exponentialTB.cpp`, and `exponentialMain.cpp`. The `exponential.h` tab is active, showing the following code:

```
(Global Scope)
26 template <int NumClk, int N>
27 void exponential<NumClk, N>::operation()
28 {
29     while (true)
30     {
31         float term = 1;
32         float exp = 1;
33
34         if (rst == '1')
35         {
36             ready = SC_LOGIC_0;
37             RBus = 0;
38         }
39
40         else if ((clk == '1') && (clk->event()))
41         {
42             if (go == '1')
43             {
44                 ready = SC_LOGIC_0;
45                 RBus = 0;
46
47                 XBus_reg = XBus->read();
48                 wait(clk->posedge_event());
49
50                 for (int i = 1; i < N; i++) {
51                     term = term * XBus_reg.read() * (1 / float(i));
52                     exp = exp + term;
53                 }
54
55                 for (int i = 0; i < NumClk - 1; i++)
56                     wait(clk->posedge_event());
57
58                 ready = SC_LOGIC_1;
59                 RBus = exp;
60             }
61         }
62         wait();
63     }
64 }
```

Annotations and their corresponding code sections:

- exponential.h**: Points to the header file tab.
- Variable declaration and initialization**: Points to lines 31-32 (`float term = 1;` and `float exp = 1;`).
- Input data registration**: Points to lines 47-48 (`XBus_reg = XBus->read();` and `wait(clk->posedge_event());`).
- Mimicking clock**: Points to lines 55-56 (`for (int i = 0; i < NumClk - 1; i++) wait(clk->posedge_event());`).
- Performing e^x function**: Points to lines 50-53 (the inner loop calculating the exponential term).

Exponential

The image shows a code editor with four tabs: `exponential.h`, `exponentialTB.h`, `exponentialTB.cpp` (active), and `exponentialMain.cpp`. The editor displays the implementation of the `exponentialTB` module in `exponentialTB.cpp`. The code is as follows:

```
1  #include "exponential.h"
2
3  SC_MODULE(exponentialTB)
4  {
5      sc_signal<sc_logic> clk, rst, go, ready;
6      sc_signal<float> XBus, RBus;
7
8      exponential<12, 10>* EXP;
9
10     SC_CTOR(exponentialTB)
11     {
12         EXP = new exponential<12, 10> ("EXP_Instance");
13         (*EXP) (XBus, clk, rst, go, RBus, ready);
14
15         SC_THREAD(resetting);
16         SC_THREAD(clocking);
17         SC_THREAD(inGenerating);
18         SC_THREAD(displaying);
19         sensitive << ready;
20
21     }
22     void resetting();
23     void clocking();
24     void inGenerating();
25     void displaying();
26 };
```

Annotations in the image:

- A box labeled `exponentialTB.h` points to the `#include "exponential.h"` line.
- A box labeled "Exponential instantiation" points to the lines `EXP = new exponential<12, 10> ("EXP_Instance");` and `(*EXP) (XBus, clk, rst, go, RBus, ready);`.
- A box labeled "Input generation" points to the thread declarations: `SC_THREAD(resetting);`, `SC_THREAD(clocking);`, `SC_THREAD(inGenerating);`, and `SC_THREAD(displaying);`.

Exponential

```
exponential.h    exponentialTB.h    exponentialTB.cpp    exponentialMain.cpp
(Global Scope)
1  #include "exponentialTB.h"
2
3  void exponentialTB::resetting()
4  {
5      while (true)
6      {
7          wait(7, SC_NS);
8          rst = SC_LOGIC_0;
9          wait(11, SC_NS);
10         rst = SC_LOGIC_1;
11         wait(58, SC_NS);
12         rst = SC_LOGIC_0;
13         wait();
14     }
15 }
16
17 void exponentialTB::clocking()
18 {
19     while (true)
20     {
21         wait(17, SC_NS);
22         clk = SC_LOGIC_0;
23         wait(17, SC_NS);
24         clk = SC_LOGIC_1;
25     }
26 }
27
28 void exponentialTB::inGenerating() { ... }
56
57 void exponentialTB::displaying() { ... }
65
```

```
exponential.h    exponentialTB.h    exponentialTB.cpp    exponentialMain.cpp
(Global Scope)
28 void exponentialTB::inGenerating()
29 {
30     while (true)
31     {
32         XBus = 0;
33         go = SC_LOGIC_0;
34         wait(60, SC_NS);
35
36         XBus = 0.5f;
37         go = SC_LOGIC_1;
38         wait(60, SC_NS);
39         go = SC_LOGIC_0;
40
41         wait(500, SC_NS);
42         XBus = 0.3f;
43         go = SC_LOGIC_1;
44         wait(60, SC_NS);
45         go = SC_LOGIC_0;
46
47         wait(500, SC_NS);
48         XBus = 0.1f;
49         go = SC_LOGIC_1;
50         wait(60, SC_NS);
51         go = SC_LOGIC_0;
52
53         wait();
54     }
55 }
56
57 void exponentialTB::displaying()
58 {
59     while (true){
60         if (ready == '1')
61             cout << " e^ " << XBus << " = " << RBus << " - Time : " << sc_time_stamp() << endl;
62         wait();
63     }
64 }
```

exponentialTB.cpp

Exponential

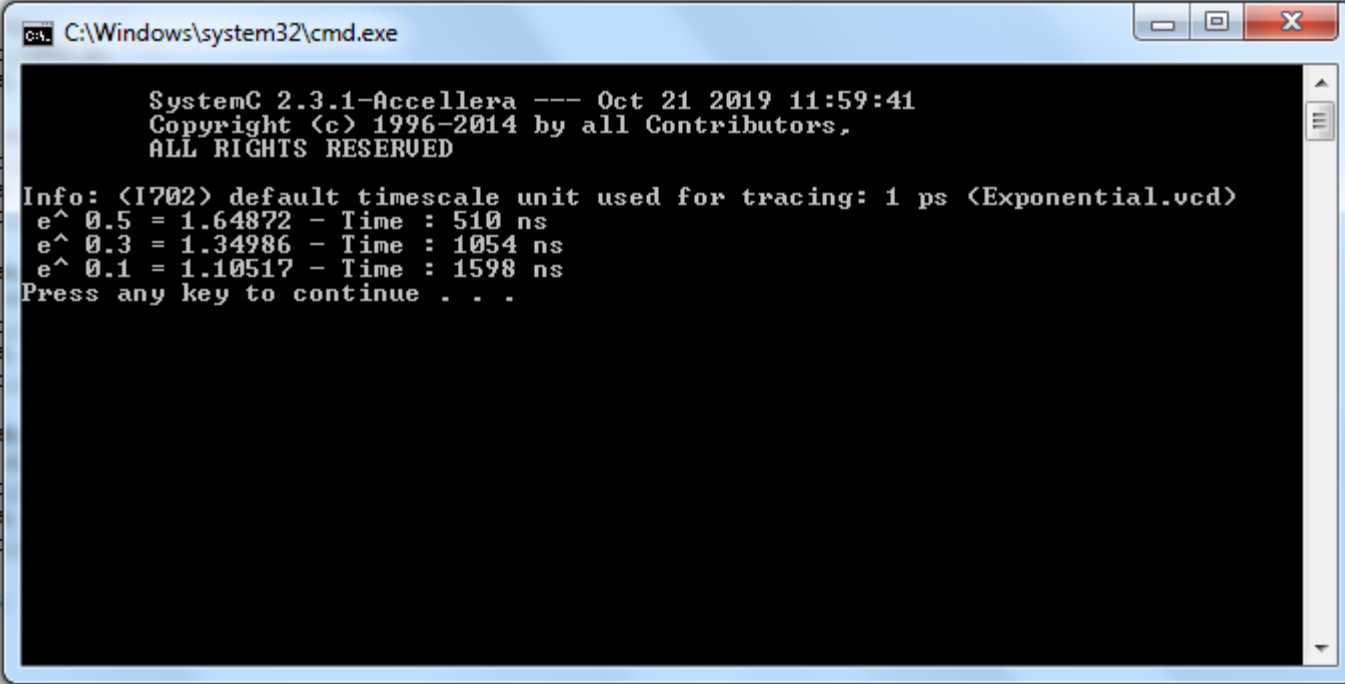
The image shows a code editor window with four tabs: `exponential.h`, `exponentialTB.h`, `exponentialTB.cpp`, and `exponentialMain.cpp`. The `exponentialMain.cpp` tab is active, showing the following code:

```
(Global Scope)
1  #include "exponentialTB.h"
2
3  int sc_main (int argc , char *argv[])
4  {
5      exponentialTB* TOP = new exponentialTB("exponentialTB_Instance");
6
7      sc_trace_file* VCDFile;
8      VCDFile = sc_create_vcd_trace_file("Exponential");
9      sc_trace(VCDFile, TOP->XBus, "XBus");
10     sc_trace(VCDFile, TOP->clk, "clk");
11     sc_trace(VCDFile, TOP->rst, "rst");
12     sc_trace(VCDFile, TOP->go, "go");
13     sc_trace(VCDFile, TOP->ready, "ready");
14     sc_trace(VCDFile, TOP->RBus, "RBus");
15
16     sc_start(2000, SC_NS);
17     return 0;
18 }
```

Annotations with callout boxes:

- Top-Level instantiation**: Points to line 5, `exponentialTB* TOP = new exponentialTB("exponentialTB_Instance");`
- VCD file and tracing signals**: Points to line 8, `VCDFile = sc_create_vcd_trace_file("Exponential");`
- exponentialMain.cpp**: Points to the `exponentialMain.cpp` tab.

Exponential



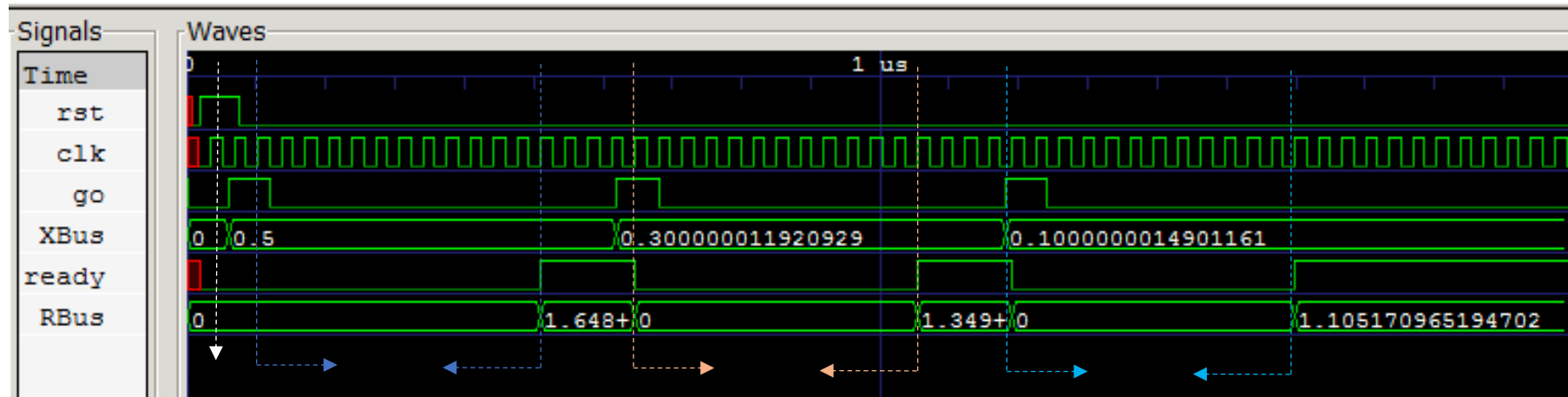
```
C:\Windows\system32\cmd.exe

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