

# Chapter 4

## RTL Level SystemC

Zainalabedin Navabi

Slides prepared by: Hanieh Hashemi

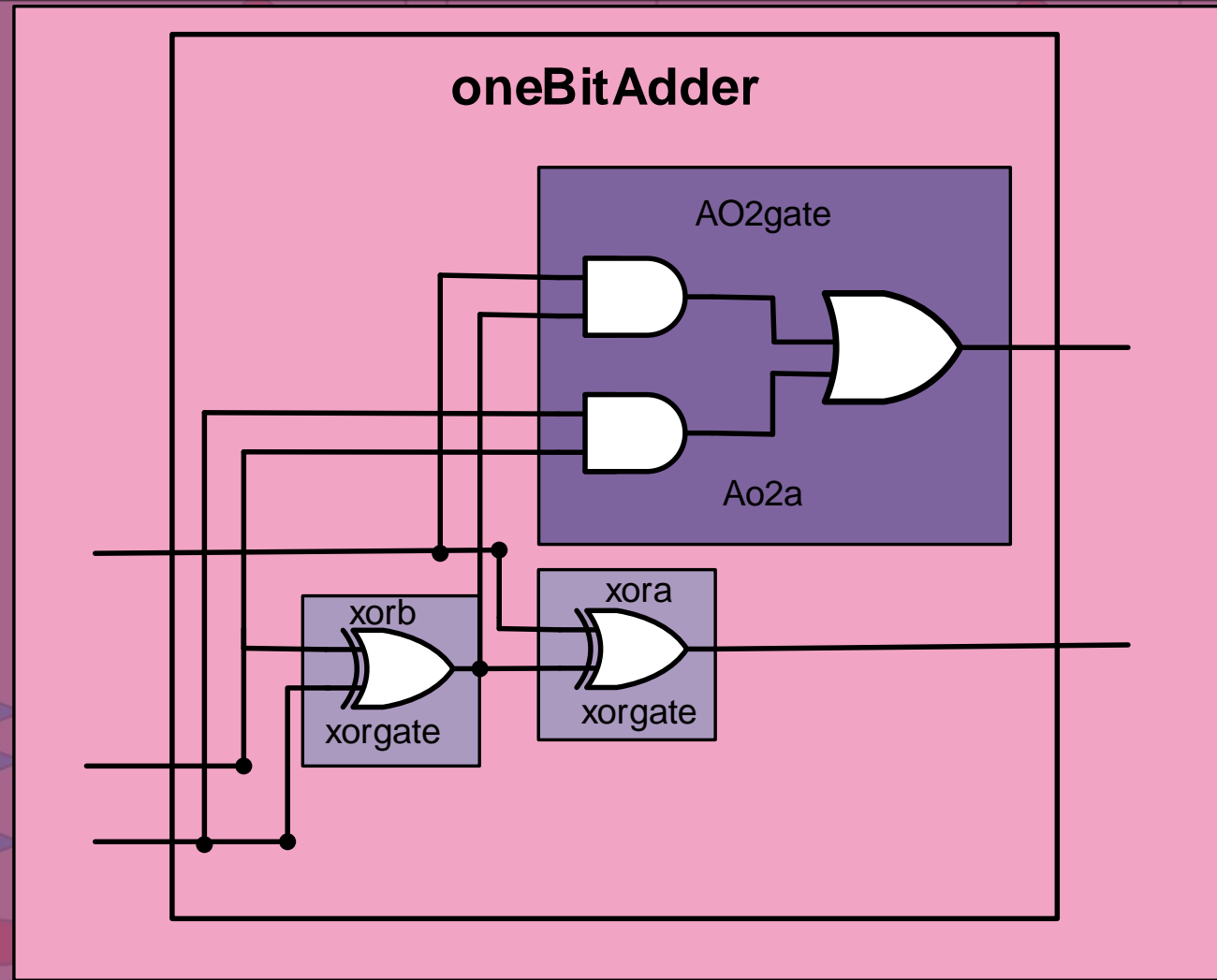
# RT Level SystemC

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

# RT Level SystemC

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

# Hierarchical Structure Hardware





# Low Level Component Functional models

All ports of our module are bus pointers.

```
classVectorPrimitives.h  simpleHierarchical.cpp  simpleHierarchical.h  simpleHierarchicalTB.cpp
SC Start (Global Scope)
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
36
37
38
39
40
41
42
43
44
```

simpleHierarchical.h

Resembles standard hardware description but lacks the concurrency

# Low Level Component Functional models

```
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      }
7      else
8          o1->fill('1');
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
```

Use overloaded "=="

simpleHierarchical.cpp

# Top Level Structural Module

```
classVectorPrimitives.h  simpleHierarchical.cpp  simpleHierarchical.h  simpleHierarchicalTB.cpp
SC Start (Global Scope)
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
36
37
38
39
40
41
42
43
44
```

simpleHierarchical.h

# Top Level Structural Module

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

simpleHierarchical.cpp

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

Calling evl() functions in  
an ordered form



# Model of Hierarchical Design

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

simpleHierarchicalTB.cpp

# RT Level SystemC

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

# sc\_module

```
simpleHierarchicalTB.cpp  simpleHierarchical.cpp  simpleHierarchical.h  -P X
SC Start  (Global Scope)
1  #include <systemc.h>
2
3  class XORgate : public sc_module
4  public:
5      sc_port<sc_signal_in_if<sc_logic>, 1> i1,
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 A02gate { ... };
17
18
19 class oneBitAdder : public sc_module {
20 public:
21     sc_port<sc_signal_in_if<sc_logic>, 1> i1, i2, i3;
22     sc_port<sc_signal_out_if<sc_logic>, 1> o1, o2;
23
24     sc_signal<sc_logic> x1;
25
26     XORgate* XORA;
27     XORgate* XORB;
28     A02gate* A02a;
29
30     SC_HAS_PROCESS(oneBitAdder);
31     oneBitAdder(sc_module_name);
32 };
33
34
35
36
37
38
39
40
41
42
43
```

Is inherited from sc\_module which contains ports, channels, concurrent processes

simpleHierarchical.h

# sc\_port, sc\_signal, sc\_logic

Mechanism for accessing methods of the channel given by sc\_port

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

Channel communication and type of data

simpleHierarchical.h

Sc\_signal is similar to VHDL signals which has methods for handling hardware concurrency

Sc\_logic is 4 value logic system:  
Sc\_logic\_0, sc\_logic\_1,  
sc\_logic\_z, sc\_logic\_x



# Module Inline Constructor

Evl function is registered as sc\_method

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

Sc\_ctocvr macro allows inline definition of xorgate constructor

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

simpleHierarchical.h

Sc\_method also run once at the beginning of the simulation

```
simpleHierarchicalTB.cpp  simpleHierarchical.cpp  simpleHierarchical.h
SC Start (Global Scope)
1 #include <systemc.h>
2
3 class XORgate : public sc_module {
4 public:
5     sc_port<sc_signal_in_if<sc_logic>,
6         sc_port<sc_signal_out_if<sc_logic>,
7
14 };
15
16 class A02gate { ... };
28
29 class oneBitAdder : public sc_module {
30 public:
31     sc_port<sc_signal_in_if<sc_logic>, 1> i1, i2, i3;
32     sc_port<sc_signal_out_if<sc_logic>, 1> o1, o2;
33
34     sc_signal<sc_logic> x1;
35
36     XORgate* XORa;
37     XORgate* XORb;
38     A02gate* A02a;
39
40     SC_HAS_PROCESS(oneBitAdder);
41     oneBitAdder(sc_module_name);
42 };
43
```

# sc\_signal, sc\_has\_process

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

simpleHierarchical.h

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

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

# Submodule instantiation

```
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
```

Here operators are overloaded for sc\_logic

simpleHierarchical.cpp

No need to handle ordering

# RT Level SystemC

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



# sc\_main

```
simpleHierarchicalTB.cpp  simpleHierarchical.cpp  simpleHierarchical.h
SC Start (Global Scope)
1  #include "simpleHierarchical.h"
2
3  int sc_main(int argc, char **argv)
4  {
5      sc_signal<sc_logic> aData;
6      sc_signal<sc_logic> bData;
7      sc_signal<sc_logic> cData;
8      sc_signal<sc_logic> cOut;
9      sc_signal<sc_logic> sOut;
10
11     oneBitAdder* FA1 = new oneBitAdder("FA1_instance");
12     (*FA1) (aData, bData, cData, cOut, sOut);
13
14     sc_trace_file* VCDFile;
15     VCDFile = sc_create_vcd_trace_file("simpleHierarchical");
16     sc_trace(VCDFile, aData, "aIn");
17     sc_trace(VCDFile, bData, "bIn");
18     sc_trace(VCDFile, cData, "carryIn");
19     sc_trace(VCDFile, cOut, "carryOut");
20     sc_trace(VCDFile, sOut, "sumOut");
21     ...
40
41     return 0;
42 }
43
```

simpleHierarchicalTB.cpp

VCD file and tracing signals

# Simulation Run Timing

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

simpleHierarchicalTB.cpp

Sc\_start advances the simulation as much as specified time of its argument

# Waveform in VCD



# RT Level SystemC

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



# sc\_module, sc\_in, sc\_out

Only valid for  
sc\_signal

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

simpleHierarchical.h

Port  
association by  
position

# Ev10

```
simpleHierarchical.cpp  simpleHierarchical.h
SC Next (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     sc_logic w;
14     if (((i1->read() & i2->read()) | (i3->read() & i4->read())) == SC_LOGIC_1)
15         o1->write(SC_LOGIC_1);
16     else
17         o1->write(SC_LOGIC_0);
18 }
19
20
```

simpleHierarchical.cpp

# RT Level SystemC

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

# Gates with propagation delay

Use `sc_has_process` for passing delay argument

`simpleHierarchical.h`

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

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

```
serialAddingTB.h  simpleHierarchical.h  simpleHierarchical.cpp  serialAddingTB.cpp
Serial Adding (Global Scope)
1  #include <systemc.h>
2
3  SC_MODULE(XORgate)
4  {
5      sc_in<sc_logic> i1, i2;
6      sc_out<sc_logic> o1;
7
8      sc_time Td;
9      SC_HAS_PROCESS(XORgate);
10     XORgate::XORgate(sc_module_name, sc_time delay)
11     {
12         Td = delay;
13         SC_THREAD(ev1);
14         sensitive << i1 << i2;
15     }
16     void ev1();
17 };
18
19 SC_MODULE(AO2gate) { ... }
34
35 SC_MODULE(oneBitAdder)
36 {
37     sc_in<sc_logic> i1, i2, i3;
38     sc_out<sc_logic> o1, o2; // Carry, Sum
39
40     sc_signal<sc_logic> x1;
41
42     XORgate* XORa;
43     XORgate* XORb;
44     AO2gate* AO2a;
45
46     SC_CTOR(oneBitAdder)
47     {
48         XORa = new XORgate("xor_insta", sc_time(0.5, SC_NS));
49         (*XORa) (i1, i2, x1);
50         XORb = new XORgate("xor_instb", sc_time(0.5, SC_NS));
51         (*XORb) (x1, i3, o2);
52         AO2a = new AO2gate("ao2_insta", sc_time(0.4, SC_NS));
53         (*AO2a) (i1, i2, x1, i3, o1);
54     }
55 };
```



# sc\_thread

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

classVectorPrimitives.cpp

Suspend until the next event on i1 & i2

# Passing delay values

```
serialAddingTB.h  simpleHierarchical.h  simpleHierarchical.cpp  serialAddingTB.cpp
Serial Adding (Global Scope)
1  #include <systemc.h>
2
3  SC_MODULE(XORgate)
4  {
5      sc_in<sc_logic> i1, i2;
6      sc_out<sc_logic> o1;
7
8      sc_time Td;
9      SC_HAS_PROCESS(XORgate);
10     XORgate::XORgate(sc_module_name, sc_time delay)
11     {
12         Td = delay;
13         SC_THREAD(ev1);
14         sensitive << i1 << i2;
15     }
16     void ev1();
17 };
18
19 SC_MODULE(AO2gate) { ... }
34
35 SC_MODULE(oneBitAdder)
36 {
37     sc_in<sc_logic> i1, i2, i3;
38     sc_out<sc_logic> o1, o2; // Carry, Sum
39
40     sc_signal<sc_logic> x1;
41
42     XORgate* XORa;
43     XORgate* XORb;
44     AO2gate* AO2a;
45
46     SC_CTOR(oneBitAdder)
47     {
48         XORa = new XORgate("xor_insta", sc_time(0.5, SC_NS));
49         (*XORa) (i1, i2, x1);
50         XORb = new XORgate("xor_instb", sc_time(0.5, SC_NS));
51         (*XORb) (x1, i3, o2);
52         AO2a = new AO2gate("ao2_insta", sc_time(0.4, SC_NS));
53         (*AO2a) (i1, i2, x1, i3, o1);
54     }
55 };
```

simpleHierarchical.h

SC\_CTOR(oneBitAdder)

# Clocking and flip flop declaration

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

simpleHierarchical.h

Sensitive to clk and reset



# Clocking and flip flop

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

simpleHierarchical.cpp

Event() is an sc\_signal interface method



# Clocking and flip flop declaration

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

simpleHierarchical.h

Association by name

# Sequential test data application

```
simpleHierarchical.cpp  serialAddingTB.cpp  serialAddingMain.cpp  serialAddingTB.h  X
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

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

# Sequential test data application

```
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

# sc\_main

```
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_
6
7     sc_trace_file* VCDFfile;
8     VCDFfile = sc_create_vcd_trace_file("serialAdding");
9     sc_trace(VCDFfile, TOP->ain, "aInput");
10    sc_trace(VCDFfile, TOP->bin, "bInput");
11    sc_trace(VCDFfile, TOP->reset, "reset");
12    sc_trace(VCDFfile, TOP->clock, "clock");
13    sc_trace(VCDFfile, TOP->sum, "serialSum");
14    ...
21    sc_start(4000, SC_NS);
22    return 0;
23 }
24
25
```

serialAddingMain.cpp



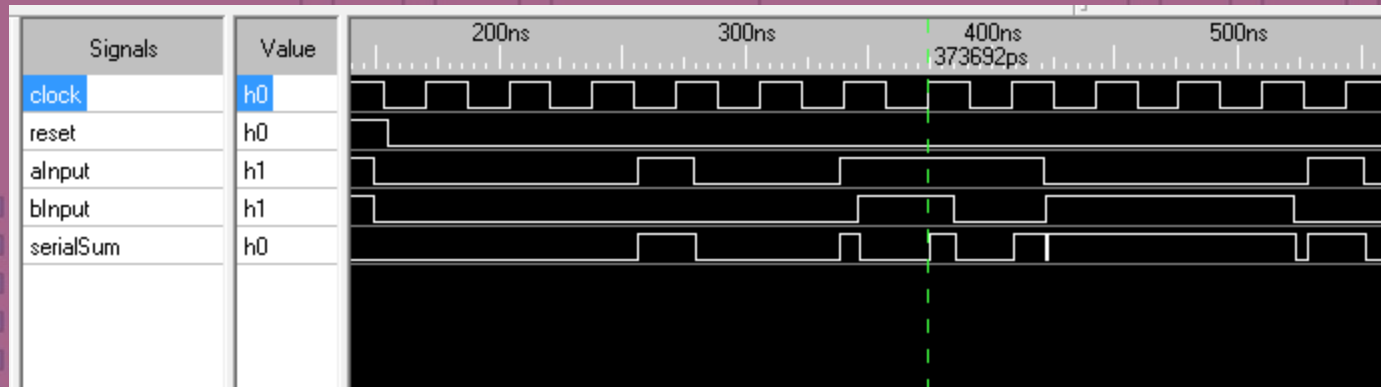
# sc\_main

```
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* VCDFFile;
8     VCDFFile = sc_create_vcd_trace_file("serialAdding");
9     sc_trace(VCDFFile, TOP->ain, "aInput");
10    sc_trace(VCDFFile, TOP->bin, "bInput");
11    sc_trace(VCDFFile, TOP->reset, "reset");
12    sc_trace(VCDFFile, TOP->clock, "clock");
13    sc_trace(VCDFFile, TOP->sum, "serialSum");
14
15    sc_trace(VCDFFile, TOP->UUT->FA1->i1, "i1_fulladder");
16    sc_trace(VCDFFile, TOP->UUT->FA1->i2, "i2_fulladder");
17    sc_trace(VCDFFile, TOP->UUT->FA1->i3, "i3_fulladder");
18    sc_trace(VCDFFile, TOP->UUT->FA1->o1, "o1_fulladder");
19    sc_trace(VCDFFile, TOP->UUT->FA1->o2, "o2_fulladder");
20
21    sc_start(4000, SC_NS);
22    return 0;
23 }
24
25
```

serailAddingMain.cpp

Access internal signals

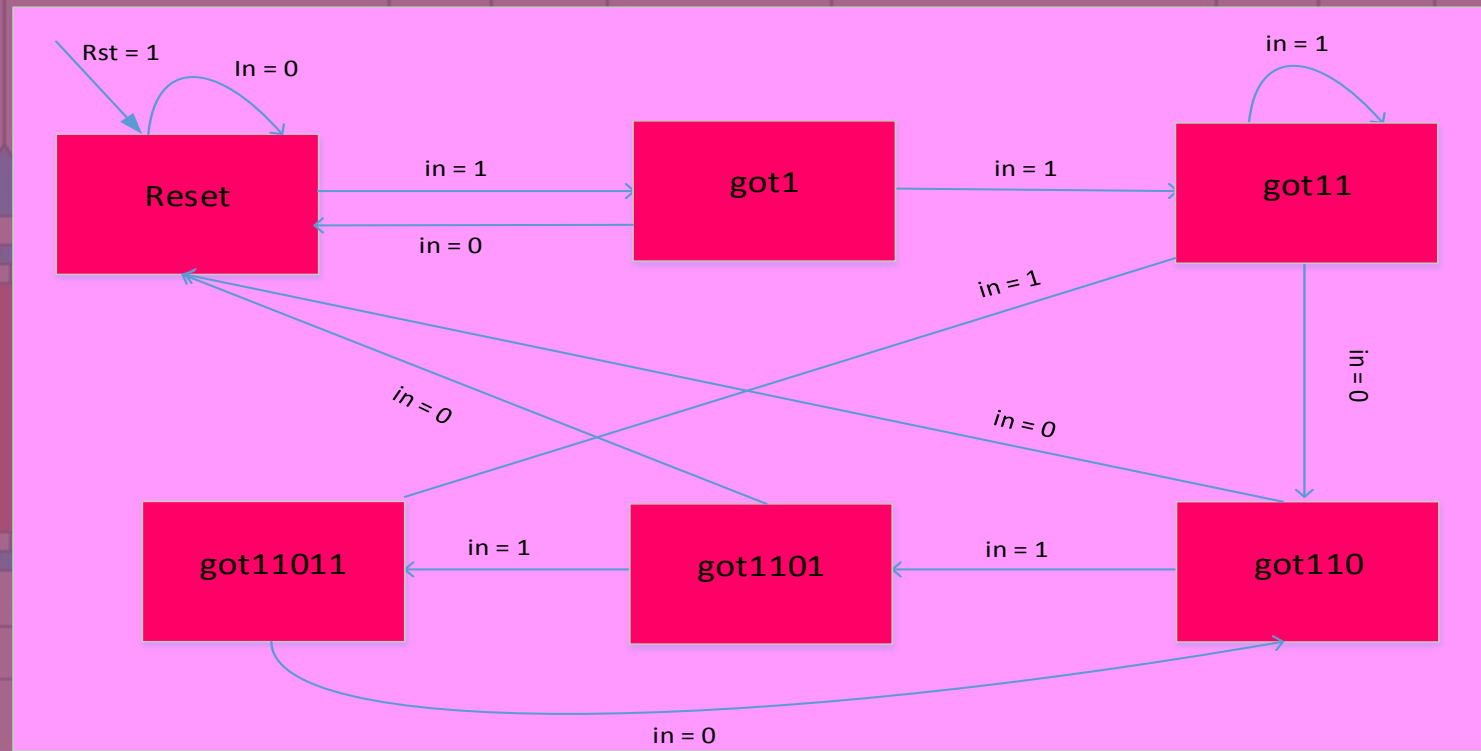
# VCD waveform



# RT Level SystemC

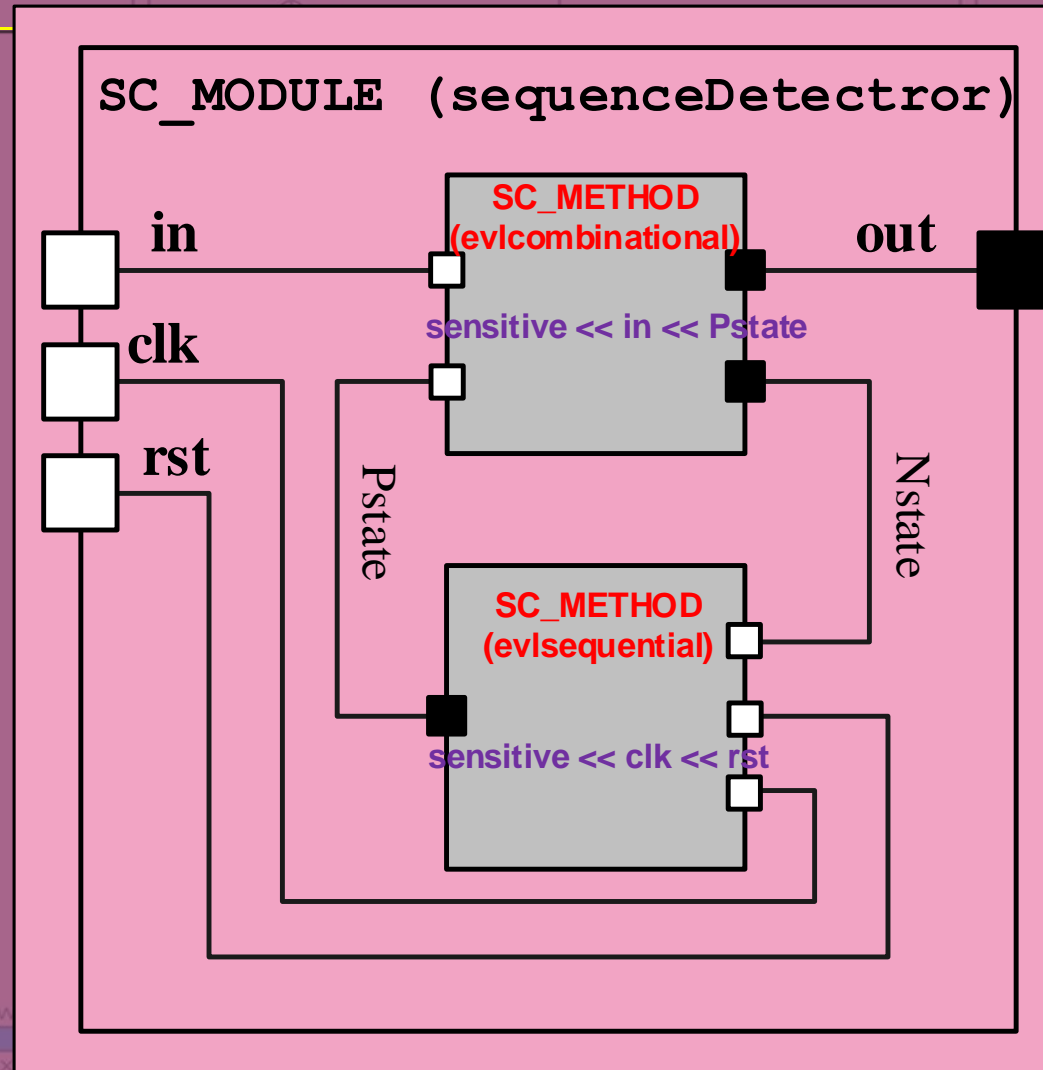
- Taking Off From c++
- SystemC Modeling
- Simulation Environment
- Utilities for HDL Orientation
- Sequential Modeling and Timing
- SystemC FSM Modeling
- Components for RT Level Design
- A configurable Memory

# SystemC FSM Modeling (sequence detector 11011)





# Huffman Model



# Sequence detector

```
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(ev1Combinational);
18             sensitive << in << Pstate;
19
20         SC_METHOD(ev1Sequential);
21             sensitive << clk << rst;
22     }
23
24     void ev1Combinational();
25     void ev1Sequential();
26 };
27
28
```

sequenceDetector.h

# Sequence detector

```
sequenceDetectorTB.cpp  sequenceDetector.cpp  X  sequenceDetector.h  sequenceDetectorTB.h
Sequence Detector (Global Scope)
1  #include "sequenceDetector.h"
2
3  void sequenceDetector::evlCombinational()
4  {
5      out = SC_LOGIC_0;
6      Nstate = ST0;
7
8      switch (Pstate.read()){
9      case ST0:
10         if (in == SC_LOGIC_1) Nstate = ST1;
11         else Nstate = ST0; break;
12     case ST1:
13         if (in == SC_LOGIC_1) Nstate = ST2;
14         else Nstate = ST0; break;
15     case ST2:
16         if (in == SC_LOGIC_1) Nstate = 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 void sequenceDetector::evlSequential()
32 {
33     if (rst == SC_LOGIC_1) Pstate = ST0;
34     else if (clk->event() && clk == SC_LOGIC_1) Pstate = Nstate;
35 }
36
37
```

Set to their inactive value

Read is also an sc\_signal method

sequenceDetector.cpp

# Sequence detector testbench

```
sequenceDetectorTB.cpp  sequenceDetector.cpp  sequenceDetector.h  sequenceDetectorTB.h  X
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 }
```

sequenceDetectorTB.h



# Sequence detector testbench

```
sequenceDetectorTB.cpp  x  sequenceDetector.cpp  sequenceDetector.h  sequenceDetectorTB.h
Sequence Detector      (Global Scope)
1  #include "sequenceDetectorTB.h"
2
3  void sequenceDetectorTB::clockGeneration()
4  {
5      while (true)
6      {
7          wait(17, SC_NS);
8          clock = SC_LOGIC_0;
9          wait(17, SC_NS);
10         clock = SC_LOGIC_1;
11     }
12 }
13 void sequenceDetectorTB::resetAssertion()
14 {
15     while (true)
16     {
17         wait(37, SC_NS);
18         reset = SC_LOGIC_0;
19         wait(59, SC_NS);
20         reset = SC_LOGIC_1;
21         wait(59, SC_NS);
22         reset = SC_LOGIC_0;
23         wait();
24     }
25 }
26 void sequenceDetectorTB::inWaveform()
27 {
28     while (true)
29     {
30         wait(37, SC_NS);
31         sin = SC_LOGIC_1;
32         wait(83, SC_NS);
33         sin = SC_LOGIC_0;
34         wait(41, SC_NS);
35         sin = SC_LOGIC_1;
36         wait(57, SC_NS);
37         sin = SC_LOGIC_0;
38     }
39 }
40
```

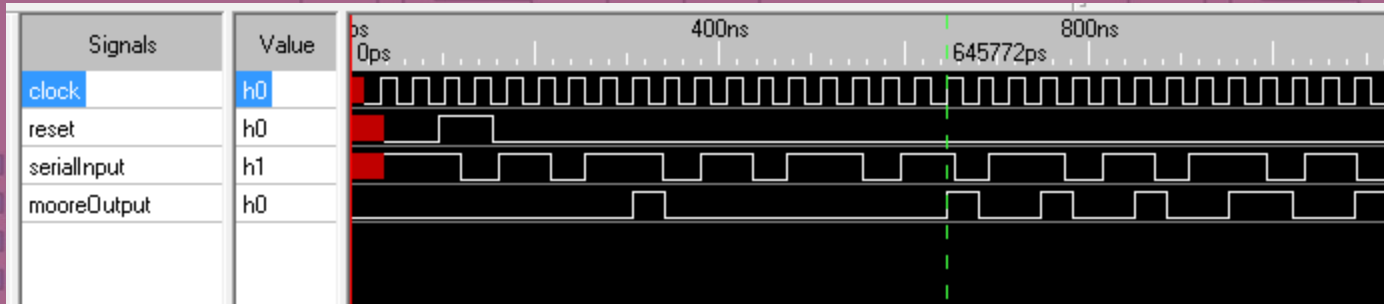
sequenceDetectorTB.cpp

# sc\_main

```
sequenceDetectorTB.cpp  sequenceDetector.cpp  sequenceDetectorMain.cpp  x
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* VCDFfile;
8      VCDFfile = sc_create_vcd_trace_file("sequenceDetector");
9      sc_trace(VCDFfile, TOP->sin, "serialInput");
10     sc_trace(VCDFfile, TOP->reset, "reset");
11     sc_trace(VCDFfile, TOP->clock, "clock");
12     sc_trace(VCDFfile, TOP->sout, "mooreOutput");
13
14     sc_start(4000, SC_NS);
15     return 0;
16 }
17
```

sequenceDetectorMain.cpp

# VCD waveform



# RT Level SystemC

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



# Four value logic

Sc\_logic is four value logic

- Logic value 0 and 1 are for standard logic values
- Z represents high impedance
- X represents unknown value
- Vector version is `sc_lv<n>`

Sc\_bit is two value logic

- Vector version is `sc_bv<n>`

Sc\_resolved is four value logic which is used for the signals which can be simultaneously driven by several sources

	0	1	Z	X
0	0	1	0	X
1	X	1	1	X
Z	0	1	Z	X
X	X	1	X	X

# Components for RT Level Design

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

# Components for RT Level Design

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h  partsLibrary.cpp  X
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     {
```

Add operation is not defined for the logic type

partsLibrary.cpp

To\_uint() is only valid for vectors  
Value() is used for one bit

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

# Signals and variables

variable

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h  partsLibrary.cpp  X
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     {
```

Use dereferencing -> because of using pointer ports

partsLibrary.cpp

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

Variables represent software-like variables with no timing



# Using sc\_in and sc\_out

```
partsLibrary.h  x handleMemory_TB.h  handleMemory.cpp  handleMemory.h
Datapath Components  Memory<ADDRESS, WORD_LENGTH>  Memory(sc_module_name)
225 // nBitAdder: n Bit adder with carry in and carry out
226 SC_MODULE(nBitAdder_channelSpecific)
227 {
228     sc_in<sc_lv<8> > ain, bin;
229     sc_in<sc_logic> ci;
230     sc_out<sc_lv<8> > addout;
231     sc_out<sc_logic> co;
232
233     SC_CTOR(nBitAdder_channelSpecific)
234     {
235         SC_METHOD(adding);
236         sensitive << ain << bin << ci;
237     }
238     void adding();
239 };
```

partsLibrary.h

# Using `sc_in` and `sc_port`

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

partsLibrary.cpp

Using dot for accessing functions

# TriState

```
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  };
```

partsLibrary.h

# TriState

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h  partsLibrary.cpp  X
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 = "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     {
```

partsLibrary.cpp



# Register Modeling

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

partsLibrary.h

Static sensitivity

# Register Modeling

Asynch reset

Dynamic sensitivity

partsLibrary.cpp

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h  partsLibrary.cpp  x
Datapath Components  nBitAdder_channelSpecific  adding0
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
34 void dRegisterRaEZ::registering()
35 {
36     if (rst == '1')
37     {
38         regout = "00000000";
39     }
40     else if (clk->event() && (clk == '1'))
41     {
42         if (cen == '1') {
43             if (zer == '1') regout = 0;
44             else regout = regin;
45         }
46     }
47 }
48
49 void dRegisterRsE::registering() {
50     if (rst == '1') {
51         regout = 0;
52     }
53     else if (cen == '1') {
54         regout = regin;
55     }
56 }
57
58 void uCounterRaEL::counting()
59 {
60     if (rst == '1')
61     {
```

# Register Modeling

.pos() can only be used for sc\_in

partsLibrary.h

```
partsLibrary.h  X
Datapath Components (Global Scope) SC_CTOR(octalMux2to1)
80 // dRegisterRsE: D Register w/ async 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, parralel 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     {
```

# Register Modeling

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h  partsLibrary.cpp  x
Datapath Components  nBitAdder_channelSpecific  adding0
21
22 void dRegisterRaE::registering()
23 {
24     if (rst == '1')
25     {
26         regout = "00000000";
27     }
28     else if (clk->event() && (clk == '1'))
29     {
30         if (cen == '1') regout = regin;
31     }
32 }
33
34 void dRegisterRaEZ::registering()
35 {
36     if (rst == '1')
37     {
38         regout = "00000000";
39     }
40     else if (clk->event() && (clk == '1'))
41     {
42         if (cen == '1') {
43             if (zer == '1') regout = 0;
44             else regout = regin;
45         }
46     }
47 }
48
49 void dRegisterRsE::registering() {
50     if (rst == '1') {
51         regout = 0;
52     }
53     else if (cen == '1') {
54         regout = regin;
55     }
56 }
57
58 void uCounterRaEL::counting()
59 {
60     if (rst == '1')
61     {
```

partsLibrary.cpp

Doesn't need to check clock edge

Synch reset



# Counter Description

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

partsLibrary.h

# Counter Description

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h  partsLibrary.cpp  x
Datapath Components  dRegisterRsE  registering0
57
58 void uCounterRaEL::counting()
59 {
60     if (rst == '1')
61     {
62         cntout = 0;
63     }
64     else if (clk->event() && (clk == '1'))
65     {
66         if (pld == '1') cntout = parin;
67         else if (cen == '1') cntout = cntout->read().to_uint() + 1;
68     }
69 }
70
71 void uCounterRaELCo::counting()
72 {
73     if (rst == '1')
74     {
75         cntout = "00000000";
76     }
77     else if (clk->event() && (clk == '1'))
78     {
79         if (pld == '1') cntout = parin;
80         else if (cen == '1' && ci == '1') cntout = cntout->read().to_uint() + 1;
81     }
82 }
83 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

# Counter Description

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h
Datapath Components  (Global Scope)
110 // uCounterRaELCo: Up-counter w/ asynch Reset, clock Enable, parallel Load, Carry
111 SC_MODULE(uCounterRaELCo)
112 {
113     sc_in<sc_logic> rst, clk, cen, pld;
114     sc_in<sc_logic> ci;
115     sc_out<sc_logic> co;
116     sc_in<sc_lv<8>> parin;
117     sc_out<sc_lv<8>> cntout;
118
119     SC_CTOR(uCounterRaELCo)
120     {
121         SC_METHOD(counting);
122         sensitive << rst << clk;
123         SC_METHOD(carrying);
124         sensitive << ci << cntout;
125     }
126     void counting();
127     void carrying();
128 };
129
130 // rShifterRaEL: right Shiftregister w/ asyn Reset, shift Enable, parallel Load
131 SC_MODULE(rShifterRaEL)
132 {
133     sc_in<sc_logic> rst, clk, sen, pld;
134     sc_in<sc_logic> sin;
135     sc_in<sc_lv<8>> parin;
136     sc_out<sc_lv<8>> shftout;
137
138     void shifting();
139
140     SC_CTOR(rShifterRaEL)
141     {
142         SC_METHOD(shifting);
143         sensitive << rst << clk;
144     }
145 };
146
147 // Memory: 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



always @(---)

end

assign co = &{ci, count};

# Counter Description

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h  partsLibrary.cpp  ↵ ×
Datapath Components  → dRegisterRsE  registering()
57
58 void uCounterRaEL::counting()
59 {
60     if (rst == '1')
61     {
62         cntout = 0;
63     }
64     else if (clk->event() && (clk == '1'))
65     {
66         if (pld == '1') cntout = parin;
67         else if (cen == '1') cntout = cntout->read().to_uint() + 1;
68     }
69 }
70
71 void uCounterRaELCo::counting()
72 {
73     if (rst == '1')
74     {
75         cntout = "00000000";
76     }
77     else if (clk->event() && (clk == '1'))
78     {
79         if (pld == '1') cntout = parin;
80         else if (cen == '1' && ci == '1') cntout = cntout->read().to_uint() + 1;
81     }
82 }
83 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



# Shift Register

```
partsLibrary.h  x  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

# Shift Register

```
partsLibrary.h  handleMemory_TB.h  handleMemory.cpp  handleMemory.h  partsLibrary.cpp  X
Datapath Components  uCounterRaELCo  carrying0
89 void rShifterRaEL::shifting()
90 {
91     if (rst == '1')
92     {
93         shftout = "00000000";
94     }
95     else if (clk->event() && (clk == '1'))
96     {
97         if (pld == '1') shftout = parin;
98         else if (sen == '1') shftout =
99             (sin, shftout->read().range(7, 1));
100     }
101 }
102
103 void nBitAdder_channelSpecific::adding()
104 {
105     sc_lv<9> res;
106     res = ain.read().to_uint() + bin.read().to_uint()
107         + ci.read().value();
108     addout = res.range(7, 0);
109     co = res[8];
110 }
```

partsLibrary.cpp

# RT Level SystemC

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

# A Configurable Memory

Use Template for Configurability

Use resolved to have multiple sources

Memory Array

```
handleMemory.cpp  handleMemory_TB.h  partsLibrary.cpp  partsLibrary.h  handleMemory.h
Datapath Components  Memory<ADDRESS, WORD_LENGTH>  memdump0
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 }
```

partsLibrary.h

These parameters should be defined at completeime



# Resolved value

Z is the weakest and  
X is the strongest

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

# A Configurable Memory

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

Type Conversion from  
sc\_rv to sc\_uint

Type Conversion from  
sc\_uint to sc\_rv

partsLibrary.h

# A Configurable Memory

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

partsLibrary.h

# Exponential Circuit Design

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

HandleMemory\_TB.h

SC\_CTOR(handleMemory\_TB) { EXC1 = new handleMemory("EXC\_Instance"); (\*EXC1) (addrbus, databusin, databusout, cs, rwbar); MEM = new Memory<10, 8>("MEM\_Instance", sc\_time(2000, SC\_NS), "memout.txt"); (\*MEM) (addrbus, databusin, databusout, cs, rwbar); }

Controller and datapath class pointers



# RT SystemC Example

- RTL design example 2: Exponential Circuit
  - Exponential Circuit Datapath
  - Exponential Circuit Controller

# RTL Example 2: Exponential Circuit

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

# Input-output Range

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

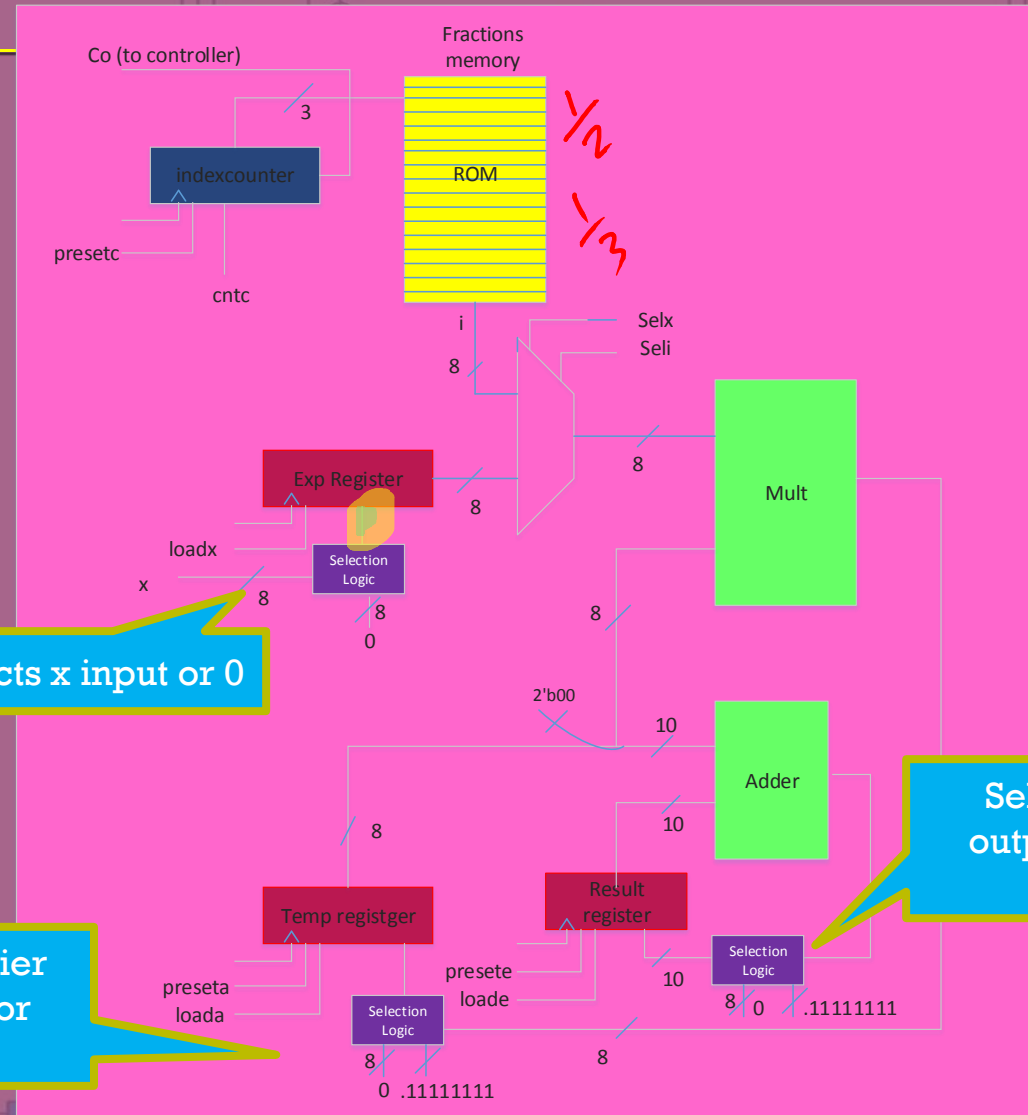
# Taylor Series

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

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



# Exponential Circuit Datapath



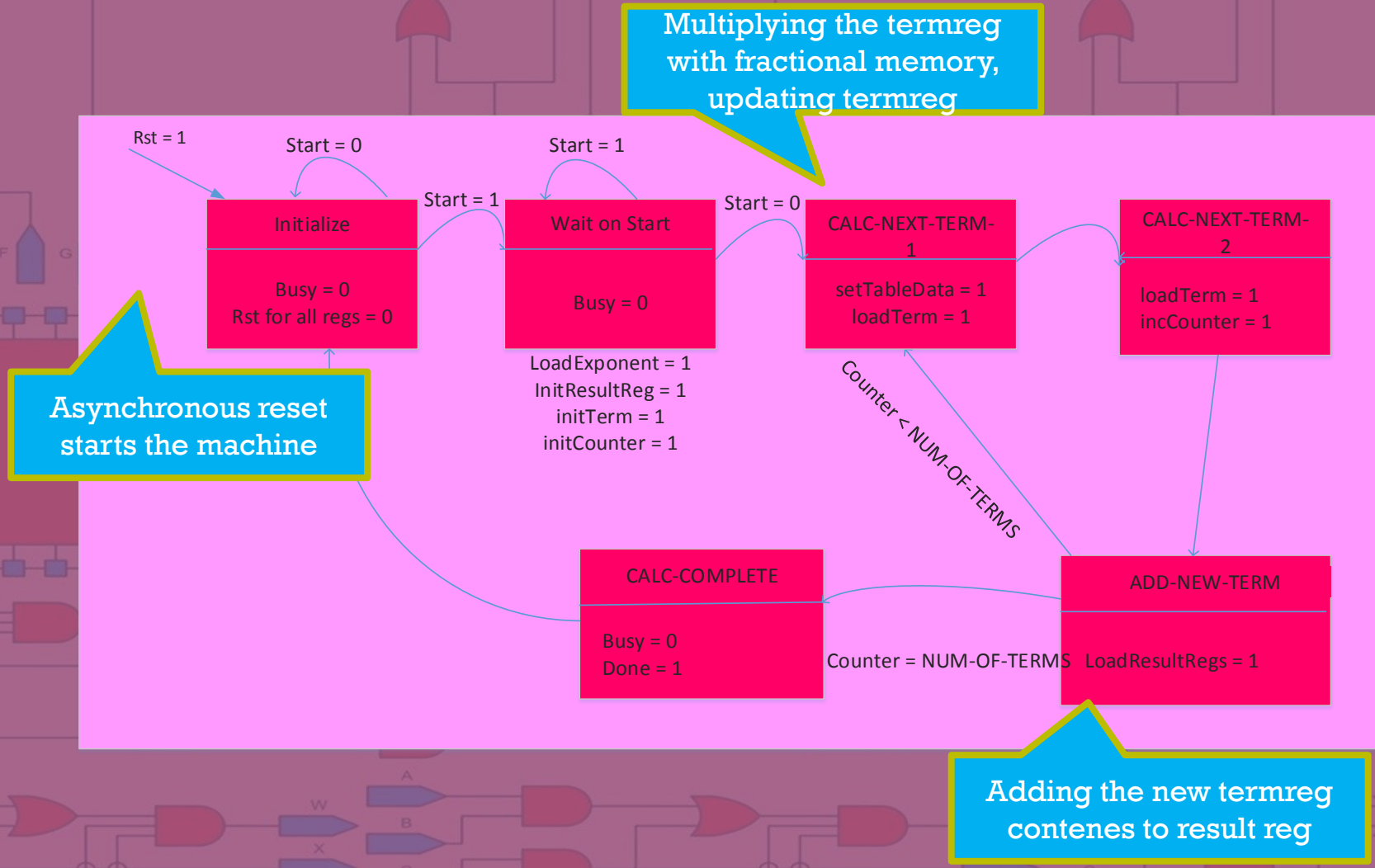
Selects x input or 0

Selects multiplier output, Reset or preset

Selects adder output, Reset or preset

# Exponential Circuit Controller

Exponential Circuit Controller



# Summary

---

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