EEL 5722C
Field-Programmable Gate Array Design

Lecture 15: Introduction to SystemC* (cont.)

Prof. Mingjie Lin

* SystemC Tutorial, Silvio Velo

Starting Example:Full Adder

FullAdder.h

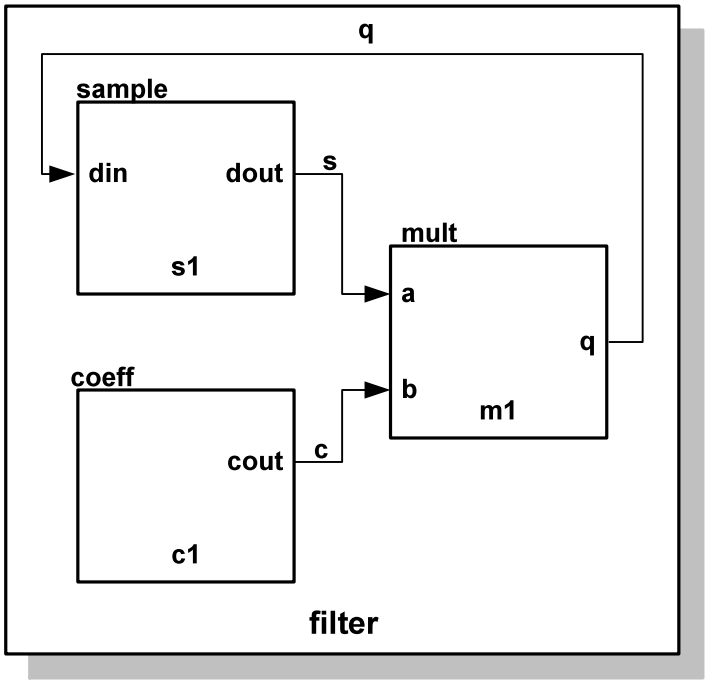
```
SC_MODULE( FullAdder ) {

   sc_in< sc_uint<16> >   A;
   sc_in< sc_uint<16> >   B;
   sc_out< sc_uint<17> > result;

   void doIt( void );

   SC_CTOR( FullAdder ) {

      SC_METHOD( doIt );
         sensitive << A;
         sensitive << B;
   }

};
```

FullAdder.cpp

```
void FullAdder::doIt( void ) {
   sc_int<16> tmp_A, tmp_B;
   sc_int<17> tmp_R;

   tmp_A = (sc_int<16>) A.read();
   tmp_B = (sc_int<16>) B.read();

   tmp_R = tmp_A + tmp_B;

   result.write( (sc_uint<16>) tmp_R.range(15,0) );
}
```

Modules

- Example:




```
SC_MODULE(filter) {
        // Sub-modules : "components"
        sample *s1;
        coeff *c1;
        mult *m1;

        sc_signal<sc_uint 32> > q, s, c; // Signals

        // Constructor : "architecture"
        SC_CTOR(filter) {

            // Sub-modules instantiation and
          mapping
            s1 = new sample ("s1");
            s1->din(q); // named mapping
            s1->dout(s);

            c1 = new coeff("c1");
            c1->out(c); // named mapping

            m1 = new mult ("m1");
            (*m1)(s, c, q); // Positional mapping

        }
}
```

Processes

- Processes are functions that are identified to the SystemC kernel. They are called if one signal of the sensitivity list changes its value.
- Processes implement the funcionality of modules
- Processes are very similar to a C++ function or method
- Processes can be Methods, Threads and CThreads

Processes

- Methods

When activated, executes and returns

- SC_METHOD(process_name)

- Threads

Can be suspended and reactivated

- wait() -> suspends
- one sensitivity list event -> activates
- SC_THREAD(process_name)

- CThreads

Are activated in the clock pulse

- SC_CTHREAD(process_name, clock value);

Processes

Type	SC_METHOD	SC_THREAD	SC_CTHREAD
Activates Exec.	Event in sensit. list	Event in sensit. List	Clock pulse
Suspends Exec.	NO	YES	YES
Infinite Loop	NO	YES	YES
suspended/ reactivated by	N.D.	wait()	wait() wait_until()

Constructor & Sensibility definition	SC_METHOD(*call_back*);
	sensitive(*signals*);
	sensitive_pos(*signals*);
	sensitive_neg(*signals*);
		SC_THREAD(*call_back*);
		sensitive(*signals*);
		sensitive_pos(*signals*);
		sensitive_neg(*signals*);
			SC_CTHREAD(
			call_back,
			clock.pos());
			SC_CTHREAD(
			call_back,
			clock.neg());

Processes

- Process Example

Into the .H file

```
void doIt( void );

   SC_CTOR( Mux21 ) {

      SC_METHOD( doIt );
         sensitive << selection;
         sensitive << in1;
         sensitive << in2;

   }
```

Into the .CPP file

```
void Mux21::doIt( void ) {

   sc_uint<8> out_tmp;

   if( selection.read() ) {
      out_tmp = in2.read();
   } else {
      out_tmp = in1.read();
   }

   out.write( out_tmp );
}
```

Ports and Signals

- Ports of a module are the external interfaces that pass information to and from a module
- In SystemC one port can be *IN*, *OUT* or *INOUT*
- Signals are used to connect module ports allowing modules to communicate
- Very similar to ports and signals in VHDL

Ports and Signals

- Types of ports and signals:
  - All natives C/C++ types
  - All SystemC types
  - User defined types
- How to declare
  - IN : sc_in<***port_typ***>
  - OUT : sc_out<***port_type***>
  - Bi-Directional : sc_inout<***port_type***>

Ports and Signals

- How to read and write a port ?
  - Methods *read( );* and *write( );*
- Examples:
  - in_tmp = in.read( ); //reads the port in to in_tmp
  - out.write(out_temp); //writes out_temp in the out port

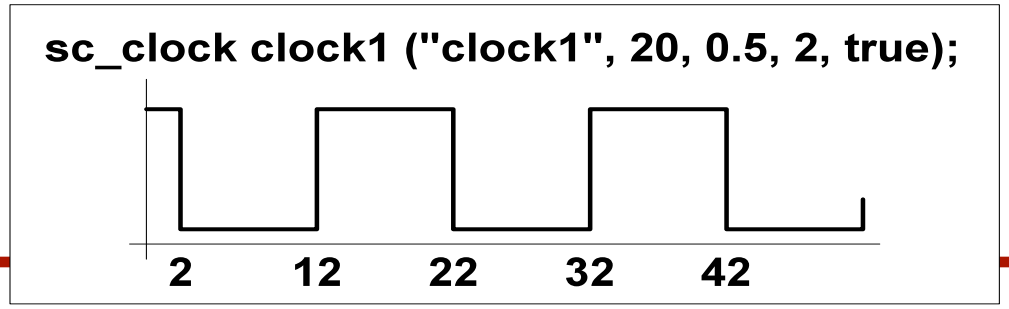
Clocks

- Special object
- How to create ?

  sc_clock ***clock_name*** (

  "***clock_label***", ***period***, ***duty_ratio***, ***offset***, ***initial_value***);
- Clock connection

  f1.clk(clk_signal); //where f1 is a
- Clock example:



Data Types

- SystemC supports:
  - C/C++ native types
  - SystemC types
- SystemC types
  - Types for systems modelling
  - 2 values **(‘0’,’1’)**
  - 4 values **(‘0’,’1’,’Z’,’X’)**
  - Arbitrary size integer (Signed/Unsigned)
  - Fixed point types

SystemC types

Type	Description
sc_logic	Simple bit with 4 values(0/1/X/Z)
sc_int	Signed Integer from 1-64 bits
sc_uint	Unsigned Integer from 1-64 bits
sc_bigint	Arbitrary size signed integer
sc_biguint	Arbitrary size unsigned integer
sc_bv	Arbitrary size 2-values vector
sc_lv	Arbitrary size 4-values vector
sc_fixed	templated signed fixed point
sc_ufixed	templated unsigned fixed point
sc_fix	untemplated signed fixed point
sc_ufix	untemplated unsigned fixed point

SystemC types

- ***Simple bit type***
- Assignment similar to *char*
  - my_bit = '1';
- Declaration
  - bool my_bit;

Operators				
Bitwise	& (and)	| (or)	^ (xor)	~ (not)
Assignment	=♪	&=♪	|=♪	^=♪
Equality	==♪	!=♪		

SystemC types

- ***SC_LOGIC type***

- More general than *bool*, 4 values :
  - (‘0’ (false), ‘1’ (true), ‘X’ (undefined) , ‘Z’(high-impedance))
- Assignment like *bool*
  - my_logic = ‘0’;
  - my_logic = ‘Z’;
- Simulation time bigger than *bool*
- Operators like *bool*
- Declaration
  - **sc_logic my_logic;**

SystemC types

- ***Fixed precision integers***
- Used when arithmetic operations need fixed size arithmetic operands
- *INT* can be converted in *UINT* and vice-versa
- "int" in C++
  - The size depends on the machine
  - Faster in the simulation
- 1-64 bits integer
  - sc_int<***n***> -- signed integer with n-bits
  - sc_uint<***n***> -- unsigned integer with n-bits

SystemC types

- ***Simple bit type***
- Assignment similar to *char*
  - my_bit = '1';
- Declaration
  - bool my_bit;

Operators				
Bitwise	& (and)	| (or)	^ (xor)	~ (not)
Assignment	=♪	&♪	|=♪	^=♪
Equality	==♪	!=♪		

SystemC types

- ***Arbitrary precision integers***
- Integer bigger than 64 bits
  - sc_bigint<***n***>
  - sc_biguint<***n***>
- More precision, slow simulation
- Operators like *SC_LOGIC*
- Can be used together with:
  - Integer C++
  - sc_int, sc_uint

SystemC types

- Bit vector
  - sc_bv<***n***>
  - 2-value vector (0/1)
  - Not used in arithmetics operations
  - Faster simulation than *sc_lv*
- Logic Vector
  - sc_lv<***n***>
  - Vector to the *sc_logic* type
- Assignment operator ("=")
  - my_vector = "XZ01"
  - Conversion between vector and integer (int or uint)
  - Assignment between *sc_bv* and *sc_lv*
  - Additional Operators

Reduction Conversion	and_reduction() to_string()	or_reduction()	xor_reduction()

SystemC types

- Examples:
  - sc_bit y, sc_bv<8> x;
  - y = x[6];
  - sc_bv<16> x, sc_bv<8> y;
  - y = x.range(0,7);
  - sc_bv<64> databus, sc_logic result;
  - result = databus.or_reduce();
  - sc_lv<32> bus2;
  - cout << "bus = " << bus2.to_string();

User defined types

- Comparation operator
  - Operator “Built-in” “==“ can’t be used
  - function inline must be defined for user types

```
inline bool operator == (const packet_type& rhs) const
{♪
    return (rhs.info==info && rhs.seq==seq &&
      rhs.retry==retry);
}
```

Debugging

Text-based Debugging

- C++ “printf” debugging

```
printf(“Hello World”);

cout << “Hello World” << endl;
```

Text-based Debugging

- Constructor Debugging
  - Find out how your design is built up when the simulation starts.
  - Use the **name()** method to identify SystemC classes:

```
SC_CTOR(nand2) {
  cout << "Constructing nand2 " << name() << endl;
  ...
  ...
}

OUTPUT:
Constructing stim
Constructing nand2 exor2.N1
Constructung nand2 exor2.N2
```

Text-based Debugging

- Debugging methods available on all SystemC objects:
  - const char* name()
    - Returns the name of the object
  - const char* kind()
    - Returns the object's sub-class name
  - void print(ostream& out)
    - Prints the object's name to the output stream
  - void dump(ostream& out)
    - Prints the objects diagnostic data to the output stream.

25

Text-based Debugging

- Debugging threads and methods
  - All SystemC data types can be “printed” to **cout**.
    - e.g.: print inputs A, B, and F to **cout** in a table:

```
OUTPUT:

Time        A  B  F
10 ns       1  0  0
20 ns       1  1  0
30 ns       1  1  1
40 ns       0  0  1
```

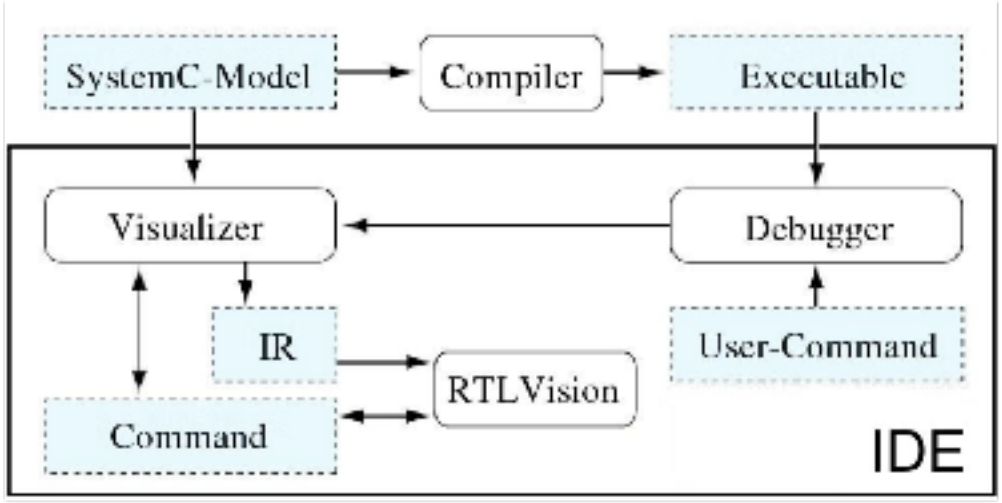
Text-based Debugging

```
SC_MODULE(mon)
{
  sc_in<bool> A,B,F;
  sc_in<bool> Clk;

  void monitor()
  {
    cout << "Time   A   B   F" << endl;
    while (true)
    {
      cout <<  sc_time_stamp() << ", ";
      cout <<  A.read() << ", ";
      cout <<  B.read() << ", ";
      cout <<  F.read() << endl;
      wait();    // wait for 1 clock cycle
    }
  }

  SC_CTOR(mon)
  {
    SC_THREAD(monitor);
    sensitive << Clk.pos();
  }
}
```

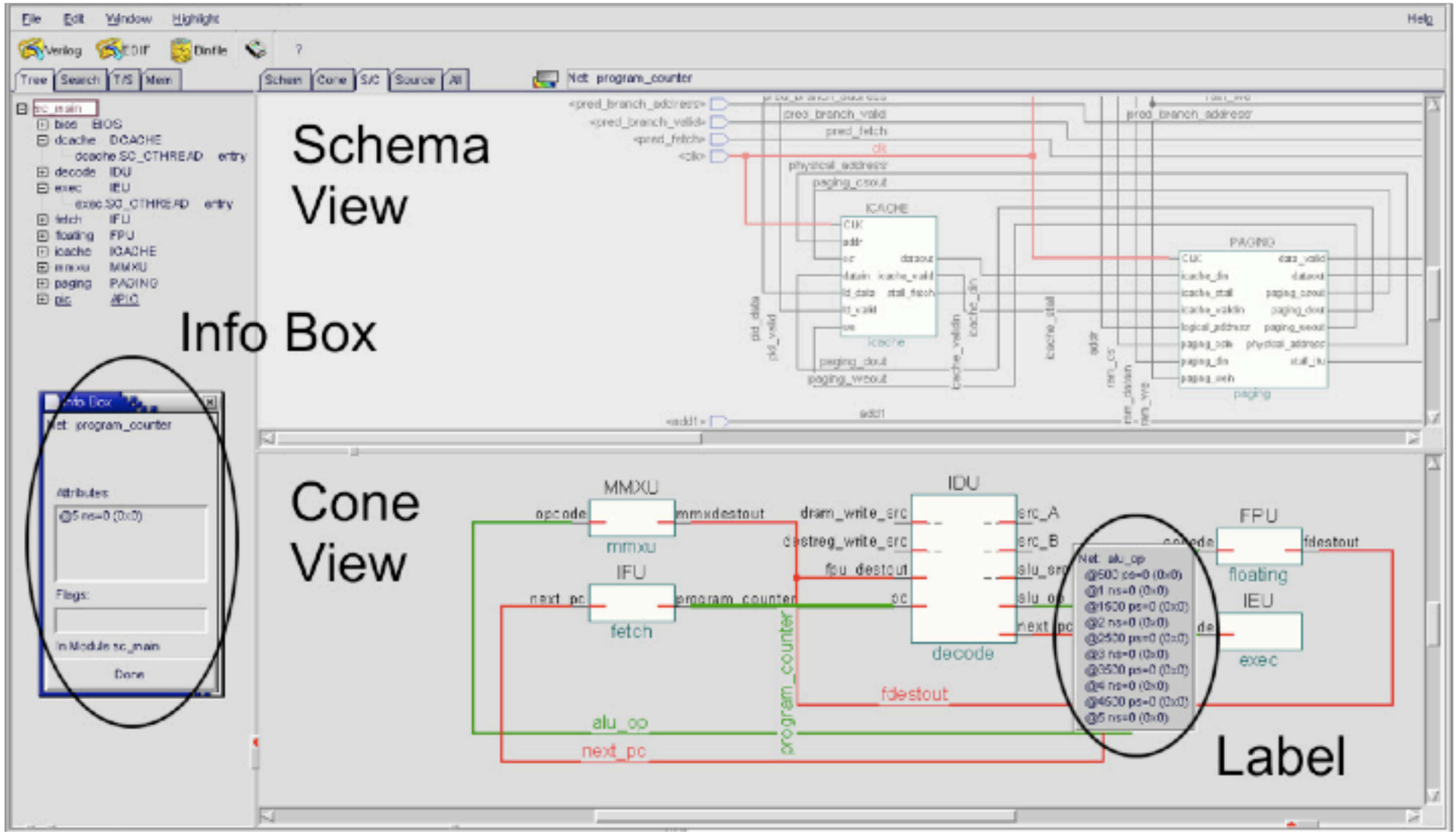
Advanced Debugging

- Standard C++ debugging tools
  - GDB, etc...

- SystemC-specific debuggers and visualizers.



Advanced Debugging



Wave-form Debugging

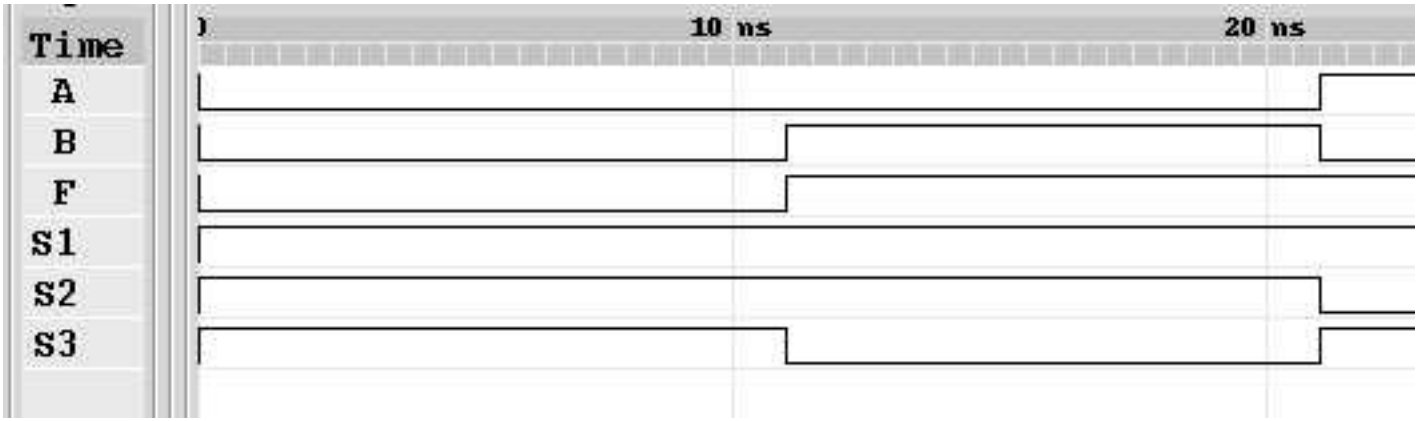
- Requires adding additional SystemC statements to **sc_main()**
  - Wave-form data written to file as simulation runs.
  - Sequence of operations:
    - Declare and create the trace file
    - Register signals or events for tracing
    - Run the simulation
    - Close the trace file

Wave-form Tracing

```
int sc_main(int argc, char* argv[])
{
  sc_signal<bool> ASig, BSig, FSig;
  sc_clock TestClk("TestClock", 10, SC_NS,0.5, 1, SC_NS);

  // Set up simulation
  ...

  // Set up trace file
  sc_trace_file* Tf;
  Tf = sc_create_vcd_trace_file("traces");          // Create Trace File
  ((vcd_trace_file*)Tf)->sc_set_vcd_time_unit(-9);  // Set time unit
  sc_trace(Tf, ASig  , "A" );                       // Register signals
  sc_trace(Tf, BSig  , "B" );                       // and variables.
  sc_trace(Tf, FSig  , "F" );
  sc_trace(Tf, DUT.S1, "S1");
  sc_trace(Tf, DUT.S2, "S2");
  sc_trace(Tf, DUT.S3, "S3");

  sc_start();  // run forever                       // Start the simulation
  sc_close_vcd_trace_file(Tf);                      // Close the trace file
  return 0;
}
```

Wave-form Tracing

- Sample Output



Final issues

- Come by my office hours (right after class)
- Any questions or concerns?