Design & Co-design of Embedded Systems

Lecture 14:

Data Types in SystemC

Sharif University of Technology
Computer Engineering Dept.

Winter-Spring 2008

Mehdi Modarressi

Data Types

- SystemC data types
  - Single-bit Types
    - `sc_bit, sc_logic`
  - Integer Types
    - `sc_int, sc_uint, sc_bigint, sc_biguint`
  - Bit-Vector Types
    - `sc_bv, sc_lv`
  - Fixed-Point Types
    - `sc_fixed, sc_ufixed, sc_fix, sc_ufix`

Data Types (cont'd)

- Defined for all types
  - Streaming operator defined for all types
    - `ostream& operator << ( ostream&, T );`
    - Example

      ```
      sc_bit b;
      cout<<b;
      ```
  - Trace function
    - `sc_trace(<trace-file pointer>, <traced variable>, <string>)`
    - Example

      sc_bit b;

      sc_trace(tf, b, “message”);

Single-bit Types

- sc_bit
  - Two-valued logic: ‘0’ , ‘1’ (character literal)
  - Example:

```
sc_bit b;
b=‘1’; // the same as b=1 or b=true
b=‘0’; // the same as b=0 or b=false
```

  - Defined operators

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

Single-bit Types (cont'd)

- sc_logic
  - Four-valued logic: '0', '1','X','x','Z','z' (char. literal)
    - 'x' means unknown or indeterminate value
    - 'z' means high impedance or floating value
  - Example:

```
sc_logic l;
l='0'; // the same as l=0, false, sc_logic_0
l='1'; // the same as l=1, true, sc_logic_1
l='z'; // the same as l=2, sc_logic_Z
l='x'; // the same as l=3, sc_logic_X
```

Single-bit Types (cont’d)

- sc_logic (cont’d)
  - Defined operators

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

Single-bit Types (cont'd)

- Any mixture of **sc_bit**, **sc_logic**, and **bool** types in comparison and assignments is possible, except:

```
sc_logic k;
sc_bit b;      // or bool b
k = ‘z’;       // or k = ‘x’
b = k;         // Result is undefined.
               // Run-time warning is issued
```

Example: 3-state buffer

```
SC_MODULE(tristate_buf){
    sc_in< sc_bit > input;
    sc_out< sc_logic > output;
    sc_in< sc_bit > enable;

     void process() {
          sc_bit in, en;
          sc_logic out;

          in=input; en=enable;  // reading inputs to temporary variables
          if(en)
             out = in;
          else
             out = 'z';
          output = out;      // writing a temporary variable to output
    }

    SC_CTOR(tristate_buf) {
          SC_METHOD(process);
          sensitive<<enable<<input;
    }
};
```

Integer Types: Fixed Precision

- Fixed Precision Unsigned and Signed Integers
  - Integer variables with fixed width (number of bits)
  - `sc_int<n>, sc_uint<n>`
  - Signed representation uses 2’s complement
  - Underlying implementation: 64 bit integer

  - Can freely be mixed with C++ integers (`int, short, long`)

Fixed Precision Integers (cont’d)

- Defined operators

Bitwise	~	&	|	^	>>	<<
Arithemtic	+	-	*	/	%	
Assignment		+=	-=	*=	/=	%=
		&=	|=	^=	=	
Equality		==	!=			
Relational	<	<=	>	>=		
Auto-inc/dec		++	--			
Bit Select		[]				
Part Select		.range(,)				
Concatenation		(,)				

Fixed Precision Integers (cont'd)

- **Examples:**

```
sc_logic mybit;
sc_uint<8> myuint;
mybit = myuint[7];

sc_uint<4> myrange;
myrange = myuint.range(5,2);

sc_uint<12> my12int;
my12int = (myuint, myrange);
```

Fixed Precision Integers (cont'd)

```
sc_uint<8> u1, u2;
sc_int<16> i1, i2;

u1 = i1;      // convert int to uint
i2 = u2;      // convert uint to int
   // BUG! in SystemC 2.0 User's Guide
```

Integer Types: Arbitrary Precision Integers

- Integers with (virtually) no width limit
  - MAX_NBITS defined in sc_constants.h
  - `sc_bigint<n>, sc_biguint<n>`
  - Signed representation uses 2's complement
  - Operators are the same as `sc_int`, `sc_uint`
  - Can be intermixed with `sc_int`, `sc_uint`, and C++ integer types

Bit-Vector Types: Arbitrary Length Bit Vector

- Arbitrary Length Bit Vector
  - Vector of 2-valued bits
  - `sc_bv<n>`
  - No arithmetic operation
  - Assignment: String of ‘0’ and ‘1’ chars
  - Reduction operations

    `and_reduce(), and, or`

```
sc_bv<6> databus;
databus = “100100”;
sc_logic result;
result = databus.or_reduce();
```

Arbitrary Length Bit Vector (cont’d)

- Defined operators

Bitwise	~	&	|	^	>>	<<
Assignment		=	&=	|=	^=	
Equality		==	!=			
Bit Select		[]				
Part Select		.range(,)				
Concatenation		(,)				
Reduction	and_reduce()		or_reduce()			
	xor_reduce()					

Arbitrary Length Bit Vector (cont'd)

- Examples:

```
sc_bv<16> data16;
sc_bv<32> data32;
data32.range(15,0) = data16;
data16 = (data32.range(7,0), data32.range(23,16));
(data16.range(3,0),data16.range(15,12)) =
       data32.range(7,0);

sc_bit y;
sc_bv<8> x;
y = x[6];

sc_bv<16> x;
sc_bv<8> y;
y = x.range(0,7);
```

Arbitrary Length Bit Vector (cont’d)

- Arithmetic operations not directly supported

```
sc_bv<8> b8=“11001010”;
sc_uint<8> u8;
u8=b8;
u8+=5;  // or any other arithmetic
b8=u8;
```

Bit-Vector Types: Arbitrary Length Logic Vector

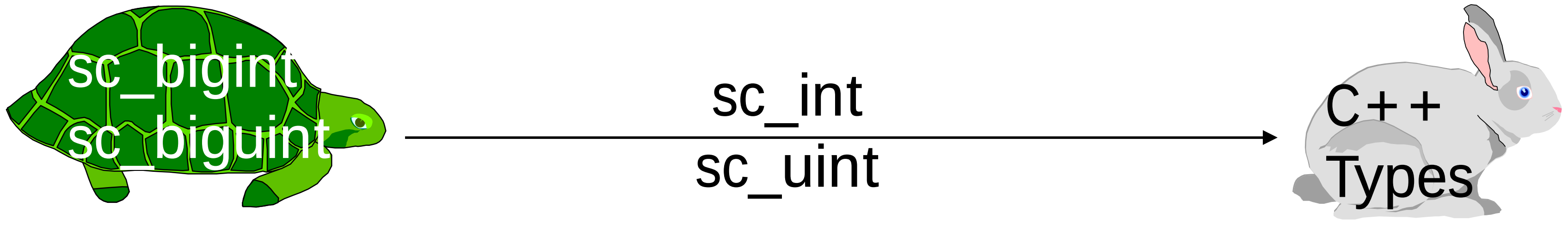
- Arbitrary Length Logic Vector
  - Vector of 4-valued bits
  - `sc_lv<n>`
  - No arithmetic operation
    - Any ‘x’ or ‘z’ in the RHS: runtime warning + undefined result
  - Assignment: String of ‘0’, ‘1’, ‘x’, ‘z’ chars
  - Operators are the same as `sc_bv`

Arbitrary Length Logic Vector (cont’d)

- Example:

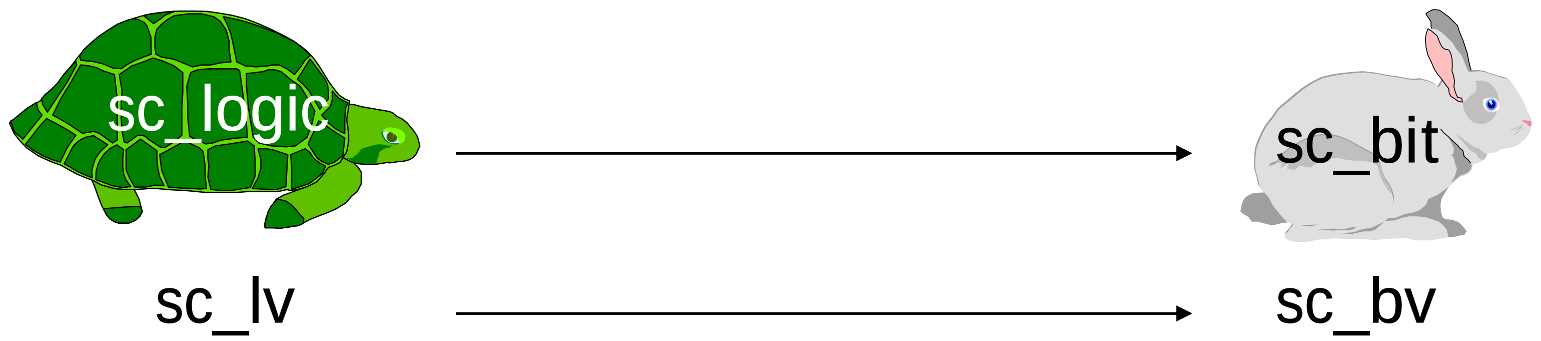
```
sc_lv<8> bus1;
if(enable)
  bus1 = “01xz10xx”;
else
  bus1 = “zzzzzzzz”;
cout<<bus1.to_string(); // OR: cout<<bus1
```

Speed Issues: Integer Types



with
MAX_NBITS
not defined

Speed Issues: Bit and Logic Types



User-Defined Type Issues

- Required for every type used as signal or port

1. Equality-Check Operator

```
struct packet_type {
  int info, seq, retry;
  bool operator == (const packet_type& rhs) const
      { return (rhs.info == info &&
               rhs.seq == seq &&
               rhs.retry == retry);
      }
     };
```

User-Defined Type Issues (cont'd)

2. A special trace function

```
void sc_trace(sc_trace_file *tf, const packet_type& v, const
     sc_string& NAME) {
       sc_trace(tf, v.info, NAME + ".info");
       sc_trace(tf, v.seq, NAME + ".seq");
       sc_trace(tf, v.retry, NAME + ".retry");
     }
```

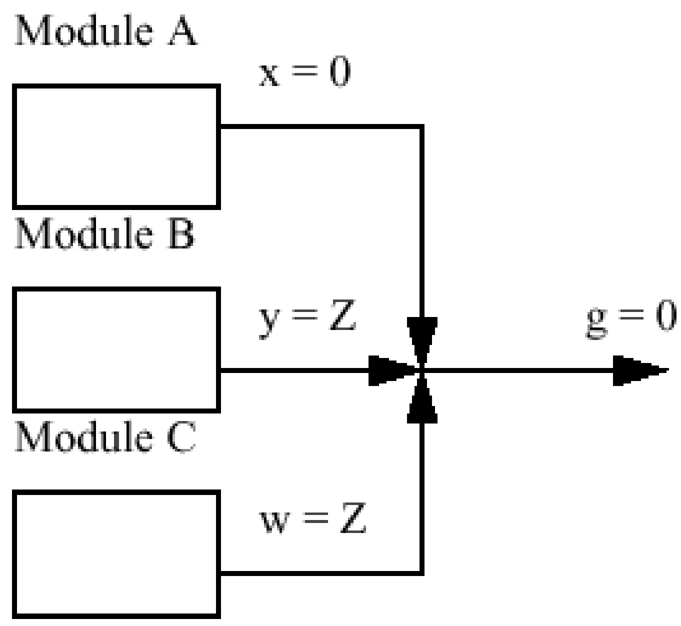
User-Defined Type Issues (cont'd)

3. A special streaming operator

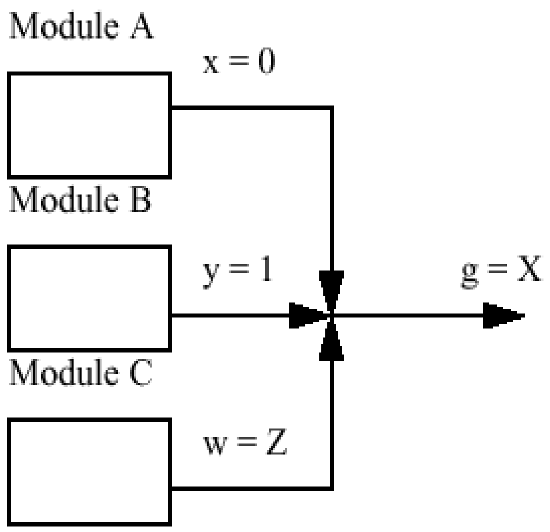
```
ostream & operator <<(ostream &os, packet_type p) {
    os << p.info << p.seq << p.retry << endl;
    return os;
}
```

Resolved Logic Vectors

- More than one driver on a single signal => *resolution* is required



Resolved Logic Vectors (cont'd)



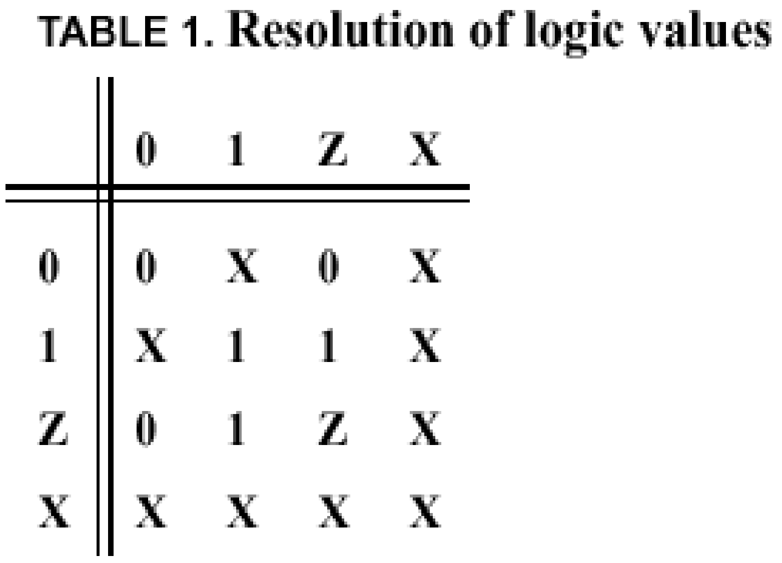


TABLE 1. Resolution of logic values

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

Resolved Logic Vectors (cont'd)

- Syntax for resolved-logic ports

```
sc_in_rv<n> port_name;
sc_out_rv<n> port_name;
sc_inout_rv<n> port_name;
```

- Notes
  - Imposes extra simulation overhead
  - Virtually no limit on the vector size (n)
- Corresponding resolved logic vector signal

```
sc_signal_rv<n> signal_name;
```

  - Used to connect resolved logic ports

Tracing Variable Values

- Tracing Waveforms
  - Creating The Trace File
  - Tracing Scalar Variable and Signals
  - Tracing Variable and Signal Arrays

Tracing Variable Values (cont’d)

- Supported waveform formats
  - VCD (Value Change Dump)
  - ASCII WIF (Wave Intermediate Format)
  - ISDB (Integrated Signal DataBase)
- Waveform viewers
  - VCD
    - Commercial tools: ModelSim®
    - Free tools:
      - GTKWave http://www.cs.man.ac.uk/amulet/tools/gtkwave/
      - SystemC_Win http://www.geocities.com/systemc_win/

Tracing Variable Values (cont’d)

- Notes
  - Only global variables (signals, ports) can be traced
    - Variables local to a function cannot be traced
  - scalar, array, and aggregate types can be traced
  - Different types of trace files can be created during the same simulation run
  - A signal or variable can be traced any number of times in different trace formats

Creating/Closing Trace Files

- Tracing wave forms
  - Creating the trace file:

    ```
    sc_trace_file *tf;
    tf = sc_create_vcd_trace_file(“trace_file”);
    ```
  - For other trace file formats

    ```
    sc_create_wif_trace_file(<filename>);
    sc_create_isdb_trace_file(<filename>);
    ```
  - To close the trace file

    ```
    sc_close_vcd_trace_file(<trace-file pointer>);
    sc_close_wif_trace_file(<trace-file pointer>);
    sc_close_isdb_trace_file(<trace-file pointer>);
    ```

Tracing scalar variables

- Syntax

  sc_trace(<*trace-file pointer*>**,** <*traced variable*>**,**
  <*string*>**);**

- Example

```
sc_signal<int> a;
float b;
sc_trace(trace_file, a, “MyA”);
sc_trace(trace_file, b, “B”);
```

Example: Tristate Buffer

```
SC_MODULE(tristate_buf){
    sc_in< sc_bit > input;
    sc_out< sc_logic > output;
    sc_in< sc_bit > enable;

     void process() {
          sc_bit in, en;
          sc_logic out;

          in=input;
          en=enable;
          if(en)
                    out = in;
          else
                    out = 'z';
          output = out;
    }

    SC_CTOR(tristate_buf) {
          SC_METHOD(process);
          sensitive<<enable<<input;
    }
};
```

Example: Tristate Buffer (cont'd)

```
SC_MODULE(test_bench) {
    sc_out< sc_bit > input;
    sc_in< sc_logic > output;
    sc_out< sc_bit > enable;
    sc_in_clk clk;

    void process()
    {
           while(1) {
                      enable= (sc_bit) 0;
                      input = (sc_bit) 0;
                      wait();
                      input = (sc_bit) 1;
                      wait();
                      enable =(sc_bit) 1;
                      input = (sc_bit) 0;
                      wait();
                      input = (sc bit) 1;
                      wait();
           }
    }

    SC_CTOR(test_bench) {
           SC_CTHREAD(process, clk.pos() );
    }
};
```

Example: Tristate Buffer (cont'd)

```
int sc_main(int, char*[])
{
    tristate_buf buf("tristateBuffer");
    test_bench tb("testBench");

    sc_signal< sc_bit > in, en;
    sc_signal< sc_logic > out;
    sc_clock clk;

    buf(in, out, en);
    tb(in, out, en, clk);

    sc_trace_file *tf;
    tf = sc_create_vcd_trace_file("BufferTraceFile"); // file extension defaults to ".vcd"
    sc_trace(tf, in, "input signal");
    sc_trace(tf, en, "enable signal");
    sc_trace(tf, out, "output signal");

    sc_start(10);

    sc_close_vcd_trace_file(tf);
    return 0;
}
```