Merkblatt System-C

```cpp
#include <systemc.h>

SC_MODULE(example){
    sc_in<sc_logic> x, y;
    sc_out<sc_logic> z;

    void calculate_z(){
        z.write( // computation
            );
    }

    SC_CTOR(example){
        SC_METHOD(calculate_z);
        sensitive << x << y;
    }
};
```

• `sc_in` and `sc_out` are ports (specialized “signals”) that carry a type, in this example `sc_logic` (‘0’, ’1’, ‘Z’, ’X’). All C++ types and specific SystemC types are supported.
• The signals `x`, `y` and `z` represent the module’s input and output ports
• `calculate_z` is the module’s process, a special method that runs concurrently with all other processes; it should contain the module’s actual computation.
• `SC_METHOD(calculate_z)` specifies the process called when the module is instantiated and whenever the input value of a signal on the sensitivity list changes
• `sensitive << x << y` adds `x` and `y` to the sensitivity list

```cpp
SC_MODULE(composite_example){
    sc_in<sc_logic> in;
    sc_out<sc_logic> out;

    sc_signal<sc_logic> s;

    component * c_1, *c_2;

    SC_CTOR(composite_example){
        c_1 = new component("comp_1");
        c_2 = new component("comp_2");

        c_1->in_port(in);
        c_1->out_port(s);
        c_2->in_port(s);
        c_2->out_port(out);

        // ...
    }
}
```

Dr. Gordon Cichon (with contributions of Sascha Oberhuber)  
gordon.cichon@ifi.lmu.de  
https://www.tcs.ifi.lmu.de/teaching/summer-2016/modern
Merkblatt System-C

In this example, the module composite_example consists of an input port, an output port, and two SystemC modules, who are instances of component.

• “component” has one input port and one output port.
• c_1->in_port(in) wires composite_example's port in and c_1's port in_port.
• To do this, the ports' types need to match.
• c_1->out_port(s) wires c_1's out_port to the signal s, which is then linked to c_2 via c_2->in_port(s). Signals are used to link components and allow composition.

```c
Int sc_main(int argc, char ** argv){
    sc_signal<sc_logic> in1,in2;
    sc_signal<sc_logic> out;
    example ex("example");
    ex.x(in1);
    ex.y(in2);
    ex.z(out)

    sc_trace_file * tf = sc_create_vcd_trace_file ("example_trace");
    sc_trace (tf, ex.x, "x");
    sc_trace (tf, ex.y, "y");
    sc_trace (tf, ex.z, "z");

    y.write(sc_logic(0));

    for (int i=0; i<=10; i++) {
        x.write(sc_logic(i%2));
        sc_start(10, SC_NS);
    }

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

• `sc_main` is a wrapper for C/C++'s main method with additional debug functionality
• Connect signals to a module's ports, then stimulate ports manually and monitor output
• `sc_trace_file` is used to track input and output (use gtkWave to visualize)
• In this example, `y.write(sc_logic(0))` sets the first input port to zero
• `sc_start(10, SC_NS)` starts the simulation for 10 nanoseconds with every call.

Useful links

• SystemC download: http://www.accellera.org/downloads/standards/systemc
• SystemC installation guide: http://rkrara.blogspot.de/2012/12/install-systemc-230-on-linux.html
• Setting up SystemC for eclipse projects: http://geekwentfreak-

Dr. Gordon Cichon (with contributions of Sascha Oberhuber)  
gordon.cichon@ifi.lmu.de  
https://www.tcs.ifi.lmu.de/teaching/summer-2016/modern