Reading Assignment

- This lecture: Project I, SystemC User Guide (see Blackboard)
- Next lecture: Notes
Basic SystemC Constructs

Project I
### Example I: DFF

#### VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity dff is
    port(clock : in std_logic;
         din : in std_logic;
         dout : out std_logic);
end dff;

architecture behavior of dff is
begin
    process begin
        wait until clock'event and clock = '1';
        dout <= din;
    end process;
end behavior;
```

#### Verilog

```verilog
module dff(din, clock, dout);

    input din;
    input clock;
    output dout;
    reg dout;

    always @(posedge clock)
        dout <= din;

endmodule
```
// dff.h
#include "systemc.h"

class dff : public sc_module
{
public:
    sc_in<bool> din;
    sc_in<bool> clock;
    sc_out<bool> dout;

    void doit() {dout = din;}

    SC_HAS_PROCESS(dff);

dff(sc_module_name name) {
    SC_METHOD(doit);
    sensitive << clock.pos();
}
};

Module=ports+processes
class dff : public sc_module
{
public:
  ...
};

Or simply

SC_MODULE(dff)
{
  ...
};

- Module: use a class derived from sc_module to represent a piece of the system.
// dff.h
#include "systemc.h"

class dff : public sc_module
{
public:
    sc_in<bool> din;
    sc_in<bool> clock;
    sc_out<bool> dout;

    ...
};

▶ A module has ports as its public members.
  ▶ So other modules can communicate with this module through those ports.

▶ A value-based port can communicate a value.
▶ The value associated with the port can be accessed using the most obvious way, i.e. assignment and read/write.
Ports

// dff.h
#include "systemc.h"

class dff : public sc_module
{
 public:
   sc_in<bool> din;
   sc_in<bool> clock;
   sc_out<bool> dout;

   ...
};

- A module has ports as its public members.
  - So other modules can communicate with this module through those ports.
- A value-based port can communicate a value.
  - The value associated with the port can be accessed using the most obvious way, i.e. assignment and read/write.
A module has ports as its public members.
- So other modules can communicate with this module through those ports.
- A value-based port can communicate a value.
- The value associated with the port can be accessed using the most obvious way, i.e. assignment and read/write.
A module’s behavior can be defined by its processes.

A process is defined as a public member function.

You need to inform the SystemC library which member functions are used as processes, e.g. by the SC_METHOD macro.

The processes defined by SC_METHOD are executed every time some event in the sensitivity list happens. i.e. they are actually actors.
Processes

A module’s behavior can be defined by its processes.

A process is defined as a public member function.

You need to inform the SystemC library which member functions are used as processes, e.g. by the SC_METHOD macro.

The processes defined by SC_METHOD are executed every time some event in the sensitivity list happens.

i.e. they are actually actors.

```
class dff : public sc_module
{
public:
  ...
  void doit() {dout = din;}

  SC_HAS_PROCESS(dff);
  dff(sc_module_name name) {
    SC_METHOD(doit);
    sensitive << clock.pos();
  }
};
```

```
Or simply:

```
class dff : public sc_module
{

  void doit() {dout = din;}

  SC_CTOR(dff) {
    SC_METHOD(doit);
    sensitive << clock.pos();
  }
};
```
```
Processes

- A module's behavior can be defined by its processes.
- A process is defined as a public member function.
- You need to inform the SystemC library which member functions are used as processes, e.g. by the SC_METHOD macro.
- The processes defined by SC_METHOD are executed every time some event in the sensitivity list happens.
  - i.e. they are actually actors.
A module’s behavior can be defined by its processes.

A process is defined as a public member function.

You need to inform the SystemC library which member functions are used as processes, e.g. by the SC_METHOD macro.

The processes defined by SC_METHOD are executed every time some event in the sensitivity list happens.

i.e. they are actually actors.
Value-based ports provide events to the sensitivity list of the preceding process.

- sensitive is a member of the base class sc_module that maintains sensitivity lists for processes.
- Default event: change of value.
- If the value is bool, events for positive/negative edges can be derived.
SC_MODULE(dff) {
  ...
  SC_CTOR(dff) {
    SC_METHOD(doit);
    sensitive << clock.pos();
  }
};

▶ Value-based ports provide events to the sensitivity list of the preceding process.
  ▶ sensitive is a member of the base class sc_module that maintains sensitivity lists for processes.

▶ Default event: change of value.
  ▶ If the value is bool, events for positive/negative edges can be derived.
SC_MODULE(dff) {
  ...
  SC_CTOR(dff) {
    SC_METHOD(doit);
    sensitive << clock.pos();
  }
};

- Value-based ports provide events to the sensitivity list of the preceding process.
  - sensitive is a member of the base class sc_module that maintains sensitivity lists for processes.
- Default event: change of value.
- If the value is bool, events for positive/negative edges can be derived.
Example II: Counter

SC_MODULE(counter) {
    sc_in<bool> load, clock;
    sc_in<int> din;
    sc_out<int> dout;

    void count_up() {
        if (load) count_val = din;
        else {++count_val; dout = count_val;}
    }

    SC_CTOR(counter) {
        SC_METHOD(count_up);
        sensitive << clock.pos();
    }

    protected:
    int count_val;
};

The module class can hold additional members/member functions to help defining module behavior.
Example III: RAM

```c
SC_MODULE(ram) {
    sc_in<int> addr;
    sc_in<bool> wrt; // true for write, false for read
    sc_inout<int> data;

    void ramread() {if (!wrt) data = memdata[addr.read()];}
    void ramwrite() {if (wrt) memdata[addr.read()] = data;}

    SC_CTOR(ram){
        SC_METHOD(ramread);
        sensitive << addr << wrt;

        SC_METHOD(ramwrite)
        sensitive << addr << data << wrt;
    }

    protected:
        int memdata[64];
};
```

- Multiple processes and multiple events for the sensitivity lists are supported.
typedef sc_uint<32> sc_uint_32;

SC_MODULE(filter) {
    sample s1;
    coeff c1;
    multi m1;

    sc_signal<sc_uint_32> q, s, c;

    SC_CTOR(filter) {
        s1.din(q); s1.dout(s);
        c1.out(c);
        m1.a(s); m1.b(c); m1.q(q);
    }
};

SC_MODULE(sample) {
    sc_in<sc_uint_32> din;
    sc_out<sc_uint_32> dout;
    ...
};

SC_MODULE(coeff) {
    sc_out<sc_uint_32> out;
    ...
};

SC_MODULE(multi) {
    sc_in<sc_uint_32> a;
    sc_in<sc_uint_32> b;
    sc_out<sc_uint_32> q;
    ...
};
Instead of specifying the module behavior as processes, you can specify it in a hierarchical way.

Need to instantiate sub-modules by creating corresponding objects.

Need to instantiate communications.
  - For value-based ports, use signals as the links, just like a wire/bus.

Don’t forget to bind ports to signals.
Example V: Stimulation Generator for Counter

A process can also be defined by the SC_THREAD macro
- With the same lifetime as the module.
- The process runs from the beginning of the simulation until reaching wait().
- wait() returns and the process continue when some event in the sensitivity list happens.
Example V: Stimulation Generator for Counter

A process can also be defined by the SC_THREAD macro

- With the same lifetime as the module.

- The process runs from the beginning of the simulation until reaching wait().

- wait() returns and the process continues when some event in the sensitivity list happens.
Example V: Stimulation Generator for Counter

A process can also be defined by the SC_THREAD macro

- With the same lifetime as the module.
- The process runs from the beginning of the simulation until reaching wait().
- wait() returns and the process continue when some event in the sensitivity list happens.

```cpp
SC_MODULE(counter_stim) {
    sc_out<bool> load;
    sc_out<int> din;
    sc_in<bool> clock;
    sc_in<int> dout;

    void stimgen();

    SC_CTOR(counter_stim) {
        SC_THREAD(stimgen);
        sensitive << clock.pos();
    }
};

void counter_stim::stimgen() {
    while (true) {
        load = true; // load 0
        din = 0;
        wait(); // count up, value = 1
        load = false;
        wait(); // count up, value = 2
        wait(); // count up, value = 3
        wait(); // count up, value = 4
        wait(); // count up, value = 5
        wait(); // count up, value = 6
    }
}
```
A process can also be defined by the SC_THREAD macro
- With the same lifetime as the module.
- The process runs from the beginning of the simulation until reaching wait().
- wait() returns and the process continue when some event in the sensitivity list happens.
Summary

- A module is defined as a class derived from `sc_module`.
  - Ports as members, processes as member functions.
  - Allow additional members/member functions
  - Initialize everything in constructor.

- Value-based ports
  - Have directions.
  - Cause events to happen if values change.

- To support hierarchy,
  - Instantiate sub-modules and communication channels.
  - Bind ports to signals.

- Processes can be defined by either `SC_METHOD` or `SC_THREAD`.
  - Different life times.
  - Similar activation mechanisms.

- Read User Guide for more details, including available data types.
Summary

- A module is defined as a class derived from `sc_module`.
  - Ports as members, processes as member functions.
  - Allow additional members/member functions
  - Initialize everything in constructor.

- Value-based ports
  - Have directions.
  - Cause events to happen if values change.

- To support hierarchy,
  - Instantiate sub-modules and communication channels.
  - Bind ports to signals.

- Processes can be defined by either `SC_METHOD` or `SC_THREAD`.
  - Different life times.
  - Similar activation mechanisms.

- Read User Guide for more details, including available data types.
A module is defined as a class derived from sc_module.
- Ports as members, processes as member functions.
- Allow additional members/member functions
- Initialize everything in constructor.

Value-based ports
- Have directions.
- Cause events to happen if values change.

To support hierarchy,
- Instantiate sub-modules and communication channels.
- Bind ports to signals.

Processes can be defined by either SC_METHOD or SC_THREAD.
- Different life times.
- Similar activation mechanisms.

Read User Guide for more details, including available data types.
A module is defined as a class derived from sc_module.
- Ports as members, processes as member functions.
- Allow additional members/member functions
- Initialize everything in constructor.

Value-based ports
- Have directions.
- Cause events to happen if values change.

To support hierarchy,
- Instantiate sub-modules and communication channels.
- Bind ports to signals.

Processes can be defined by either SC_METHOD or SC_THREAD.
- Different life times.
- Similar activation mechanisms.

Read User Guide for more details, including available data types.
Summary

- A module is defined as a class derived from `sc_module`.
  - Ports as members, processes as member functions.
  - Allow additional members/member functions
  - Initialize everything in constructor.

- Value-based ports
  - Have directions.
  - Cause events to happen if values change.

- To support hierarchy,
  - Instantiate sub-modules and communication channels.
  - Bind ports to signals.

- Processes can be defined by either `SC_METHOD` or `SC_THREAD`.
  - Different life times.
  - Similar activation mechanisms.

- Read User Guide for more details, including available data types.
Basic SystemC Constructs

Project I
A Small NoC System

Goal for Project I

- Understand how the NoC system works
- Extend the NoC system
- We will reuse the code for Project II
A Small NoC System

Goal for Project I

- Understand how the NoC system works
- Extend the NoC system

We will reuse the code for Project II
A Small NoC System

Goal for Project I

- Understand how the NoC system works
- Extend the NoC system

- We will reuse the code for Project II
Descriptions of Source Files

- Routers are defined in ece587-src/router.h and ece587-src/router.cpp.
- PEs are defined in ece587-src/pe.h and ece587-src/pe.cpp.
- The system is assembled as a top module and simulated in ece587-src/top.cpp.
Descriptions of Source Files

- Routers are defined in ece587-src/router.h and ece587-src/router.cpp.
- PEs are defined in ece587-src/pe.h and ece587-src/pe.cpp.
- The system is assembled as a top module and simulated in ece587-src/top.cpp.
 Descriptions of Source Files

- Routers are defined in ece587-src/router.h and ece587-src/router.cpp.
- PEs are defined in ece587-src/pe.h and ece587-src/pe.cpp.
- The system is assembled as a top module and simulated ece587-src/top.cpp.
System Description

- The system is synchronous – there is a clock.
- Processes for all routers and PEs are defined by SC_METHOD and are executed at the positive edge of the clock.
- Use value-based ports and signals for communication.
- Each communication link sends at most one packet per clock cycle.
System Description

- The system is synchronous – there is a clock.
- Processes for all routers and PEs are defined by \texttt{SC\_METHOD} and are executed at the positive edge of the clock.
- Use value-based ports and signals for communication.
- Each communication link sends at most one packet per clock cycle.
System Description

- The system is synchronous – there is a clock.
- Processes for all routers and PEs are defined by SC_METHOD and are executed at the positive edge of the clock.
- Use value-based ports and signals for communication.
- Each communication link sends at most one packet per clock cycle.
System Description

- The system is synchronous – there is a clock.
- Processes for all routers and PEs are defined by `SC_METHOD` and are executed at the positive edge of the clock.
- Use value-based ports and signals for communication.
- Each communication link sends at most one packet per clock cycle.
- 5 pairs of ports: 4 for connection to other routers, 1 for attached PE

- Use queues at output ports to store packets that should be sent but not sent yet.
Router

- 5 pairs of ports: 4 for connection to other routers, 1 for attached PE

- Use queues at output ports to store packets that should be sent but not sent yet.
void router::main() {
    for (int iport = 0; iport < PORTS; ++iport) read_packet(iport);
    for (int iport = 0; iport < PORTS; ++iport) write_packet(iport);
}

void router::read_packet(int iport) {
    packet p = port_in[iport].read();
    if ((p.src_x == -1) && (p.src_y == -1)) return; // empty packet
    route_packet_xy(p);
}

void router::write_packet(int iport) {
    if (out_queue_[iport].empty()) {
        port_out[iport].write(packet()); // write an empty packet
    } else {
        port_out[iport].write(out_queue_[iport].front());
        out_queue_[iport].pop_front();
    }
}
void router::route_packet_xy(packet p) {
    // ignore dest_y for now
    if (p.dest_x == x_) {// to PE
        out_queue_[PE].push_back(p);
    }
    else if (p.dest_x < x_) {// left to WEST
        out_queue_[WEST].push_back(p);
    }
    else {// (p.dest_x > x_) right to EAST
        out_queue_[EAST].push_back(p);
    }
}

XY-routing

Missing part should be implemented in Project II
void router::route_packet_xy(packet p) {
   // ignore dest_y for now
   if (p.dest_x == x_) {// to PE
      out_queue_[PE].push_back(p);
   }
   else if (p.dest_x < x_){// left to WEST
      out_queue_[WEST].push_back(p);
   }
   else {// (p.dest_x > x_) right to EAST
      out_queue_[EAST].push_back(p);
   }
}

- XY-routing
- Missing part should be implemented in Project II
Processes

- Start with a SDF model
- Decide the mapping from processes to PEs.
- Implement the processes.
Processes

- Start with a SDF model
- Decide the mapping from processes to PEs.
- Implement the processes.
Processes

- Start with a SDF model
- Decide the mapping from processes to PEs.
- Implement the processes.
The Top Module

SC_MODULE(top) {
    public:
        enum {N = 2};

        router *routers[N];
        PE_base *pes[N];

        sc_signal<packet> router_to_pe[N], pe_to_router[N];
        sc_signal<packet> router_to_router_east[N-1], router_to_router_west[N-1];
        sc_signal<packet> terminal_loop_north[N], terminal_loop_south[N];
        sc_signal<packet> terminal_loop_east, terminal_loop_west;
        sc_signal<bool> clock;

        SC_CTOR(top) {
            create_pes();
            create_network();
        }

        ...
};
Simulation

```
int sc_main(int argc, char *argv[]) {
    srand(0);

    top top_module("top");

    printf("cycle 0 ================\n");
    sc_start(0, SC_NS);
    for(int i = 1; i < 20; i++){
        printf("cycle %2d ================\n", i);
        top_module.clock.write(1); sc_start(10, SC_NS);
        top_module.clock.write(0); sc_start(10, SC_NS);
    }
    return 0;
}
```

▶ SystemC programs start from sc_main instead of main.