

---

# EEL 5722C

## Field-Programmable Gate Array Design

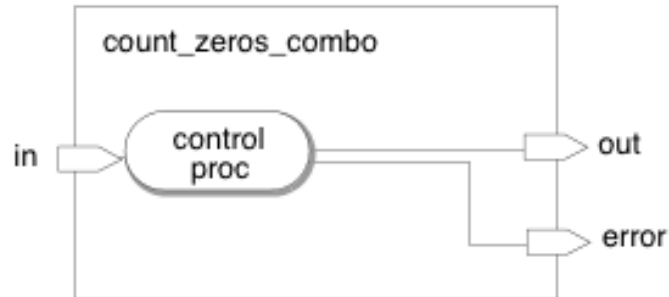
Lecture 18: Describing Synthesizable RTL in SystemC\*

Prof. Mingjie Lin



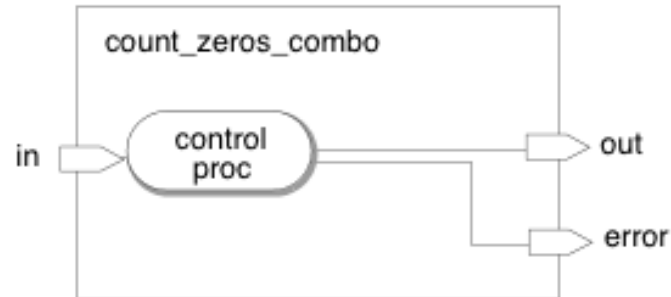
# Creating a Module With a Single SC\_METHOD Process

---



```
****count_zeros_comb.h file****  
#include "systemc.h"  
  
SC_MODULE(count_zeros_comb) {  
    sc_uint<8> in;  
    sc_uint<4> out;  
    sc_out<bool> error;  
  
    bool legal(sc_uint<8> x);  
    sc_uint<4> zeros(sc_uint<8> x);  
    void control_proc();  
  
    SC_CTOR(count_zeros_comb) {  
        SC_METHOD(control_proc);  
        sensitive << in;  
    }  
};
```

# Creating a Module With a Single SC\_METHOD Process



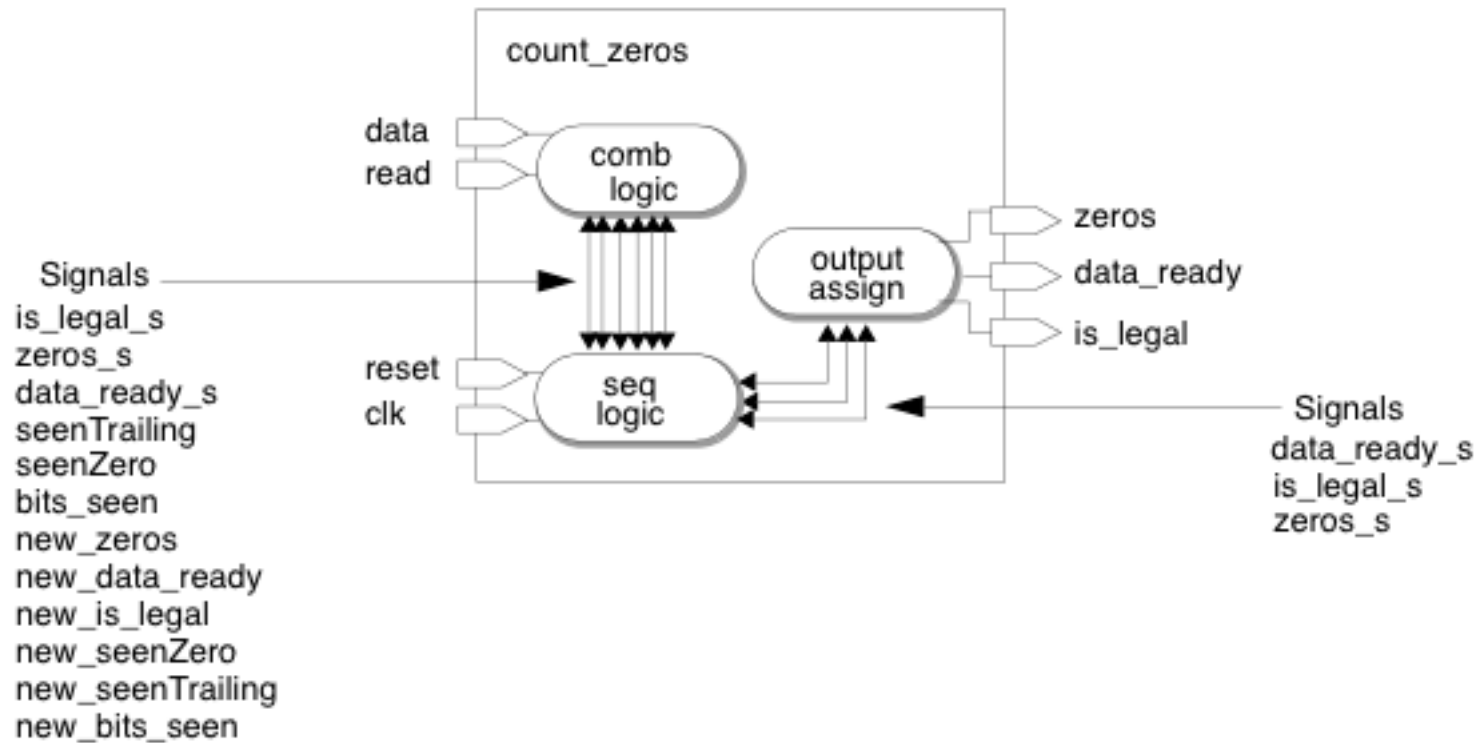
```
/*count_zeros_comb.cpp file*/
#include "count_zeros_comb.h"

void count_zeros_comb::control_proc() {
    sc_uint<4> tmp_out;
    bool is_legal = legal(in.read());
    error.write(! is_legal);
    is_legal ? tmp_out = zeros(in.read()) : tmp_out = 0;
    out.write(tmp_out);
}

bool count_zeros_comb::legal(sc_uint<8> x) {
    bool is_legal = 1;
    bool seenZero = 0;
    bool seenTrailing = 0;
    for (int i=0; i <=7; ++i) {
        if (seenTrailing && (x[i] == 0)) {
            is_legal = 0;
            break;
        } else if (seenZero && (x[i] == 1)) {
            seenTrailing = 1;
        } else if (x[i] == 0) {
            seenZero = 1;
        }
    }
    return is_legal;
}

sc_uint<4> count_zeros_comb::zeros(sc_uint<8> x) {
    int count = 0;
    for (int i=0; i <= 7; ++i) {
        if (x[i] == 0)
            ++count;
    }
    return count;
}
```

# Creating a Module With Multiple SC\_METHOD Processes



# Creating a Module With Multiple SC\_METHOD Processes

---

```

/****count_zeros_seq.h file****/
#include "systemc.h"

#define ZEROS_WIDTH 4
#define MAX_BIT_READ 7

SC_MODULE(count_zeros_seq) {
    sc_in<bool> data, reset, read, clk;
    sc_out<bool> is_legal, data_ready;
    sc_out<sc_uint<ZEROS_WIDTH> > zeros;

    sc_signal<bool> new_data_ready, new_is_legal, new_seenZero, new_seenTrailing;
    sc_signal<bool> seenZero, seenTrailing;
    sc_signal<bool> is_legal_s, data_ready_s;
    sc_signal<sc_uint<ZEROS_WIDTH> > new_zeros, zeros_s;
    sc_signal<sc_uint<ZEROS_WIDTH - 1> > bits_seen, new_bits_seen;

    // Processes
    void comb_logic();
    void seq_logic();
    void assign_outputs();

    // Helper functions
    void set_defaults();

    SC_CTOR(count_zeros_seq) {
        SC_METHOD(comb_logic);
        sensitive << data << read << is_legal_s << data_ready_s;
        sensitive << seenTrailing << seenZero << zeros_s << bits_seen;

        SC_METHOD(seq_logic);
        sensitive_pos << clk << reset;

        SC_METHOD(assign_outputs);
        sensitive << is_legal_s << data_ready_s << zeros_s;
    }
};
```

---

## Creating a Module With Multiple SC\_METHOD Processes

```

/****count_zeros_seq.cpp file****/
#include "count_zeros_seq.h"

/*
 * SC_METHOD: comb_logic()
 * finds a singular run of zeros and counts them
 */
void count_zeros_seq::comb_logic() {
    set_defaults();
    if (read.read()) {
        if (seenTrailing && (data.read() == 0)) {
            new_is_legal = false;
            new_zeros = 0;
            new_data_ready = true;
        } else if (seenZero && (data.read() == 1)) {
            new_seenTrailing = true;
        } else if (data.read() == 0) {
            new_seenZero = true;
            new_zeros = zeros_s.read() + 1;
        }

        if (bits_seen.read() == MAX_BIT_READ)
            new_data_ready = true;
        else
            new_bits_seen = bits_seen.read() + 1;
    }
}

```

## Creating a Module With Multiple SC\_METHOD Processes

---

```
/*
 * SC_METHOD: seq_logic()
 *   All registers have asynchronous resets
 */
void count_zeros_seq::seq_logic() {
    if (reset) {
        zeros_s = 0;
        bits_seen = 0;
        seenZero = false;
        seenTrailing = false;
        is_legal_s = true;
        data_ready_s = false;
    } else {
        zeros_s = new_zeros;
        bits_seen = new_bits_seen;
        seenZero = new_seenZero;
        seenTrailing = new_seenTrailing;
        is_legal_s = new_is_legal;
        data_ready_s = new_data_ready;
    }
}
```

## Creating a Module With Multiple SC\_METHOD Processes

---

```
/*
 * SC_METHOD: assign_outputs()
 * Zero time assignments of signals to their associated outputs
 */
void count_zeros_seq::assign_outputs() {
    zeros = zeros_s;
    is_legal = is_legal_s;
    data_ready = data_ready_s;
}

/*
 * method: set_defaults()
 * sets the default values of the new_* signals for the comb_logic
 * process.
 */
void count_zeros_seq::set_defaults() {
    new_is_legal = is_legal_s;
    new_seenZero = seenZero;
    new_seenTrailing = seenTrailing;
    new_zeros = zeros_s;
    new_bits_seen = bits_seen;
    new_data_ready = data_ready_s;
}
```

---



# Creating a Hierarchical RTL Module

---

- To create a hierarchical module
  - Create data members in the top-level module that are pointers to the instantiated modules.
  - Allocate the instantiated modules inside the constructor of the top-level module, giving each instance a unique name.
  - Bind the ports of the instantiated modules to the ports or signals of the top-level module. Use either binding by position or binding by name coding style

# Creating a Hierarchical RTL Module

---

```
/***fir_top.h***/
#include <systemc.h>
#include "fir_fsm.h"
#include "fir_data.h"

SC_MODULE(fir_top) {

    sc_in_clk          CLK;
    sc_in<bool>        RESET;
    sc_in<bool>        IN_VALID;
    sc_in<int>         SAMPLE;

    sc_out<bool>       OUTPUT_DATA_READY;
    sc_out<int>        RESULT;

    sc_signal<unsigned> state_out; //Communication between
                                   //two peer modules
}
```

---

# Creating a Hierarchical RTL Module

```
// Create data members - pointers to instantiated
// modules
fir_fsm *fir_fsm1;
fir_data *fir_data1;
SC_CTOR(fir_top) {
    // Create new instance of fir_fsm module
    fir_fsm1 = new fir_fsm("FirFSM");

    // Binding by name
    fir_fsm1->clock(CLK);
    fir_fsm1->reset(RESET);
    fir_fsm1->in_valid(IN_VALID);
    fir_fsm1->state_out(state_out);

    // Binding by position alternative
    //fir_fsm1 (CLK, RESET, IN_VALID, state_out);

    // Create new instance
    // of fir_data module and bind by name
    fir_data1 = new fir_data("FirData");
    fir_data1->reset(RESET);
    fir_data1->state_out(state_out);
    fir_data1->sample(SAMPLE);
    fir_data1->result(RESULT);
    fir_data1->output_data_ready(OUTPUT_DATA_READY);
    fir_data1->clk(CLK);
    ...
}
};
```

# Creating a Hierarchical RTL Module

---

```
/*fir_fsm.h*/
SC_MODULE(fir_fsm) {

    sc_in<bool>      clock;
    sc_in<bool>      reset;
    sc_in<bool>      in_valid;
    sc_out<unsigned> state_out;
    ...

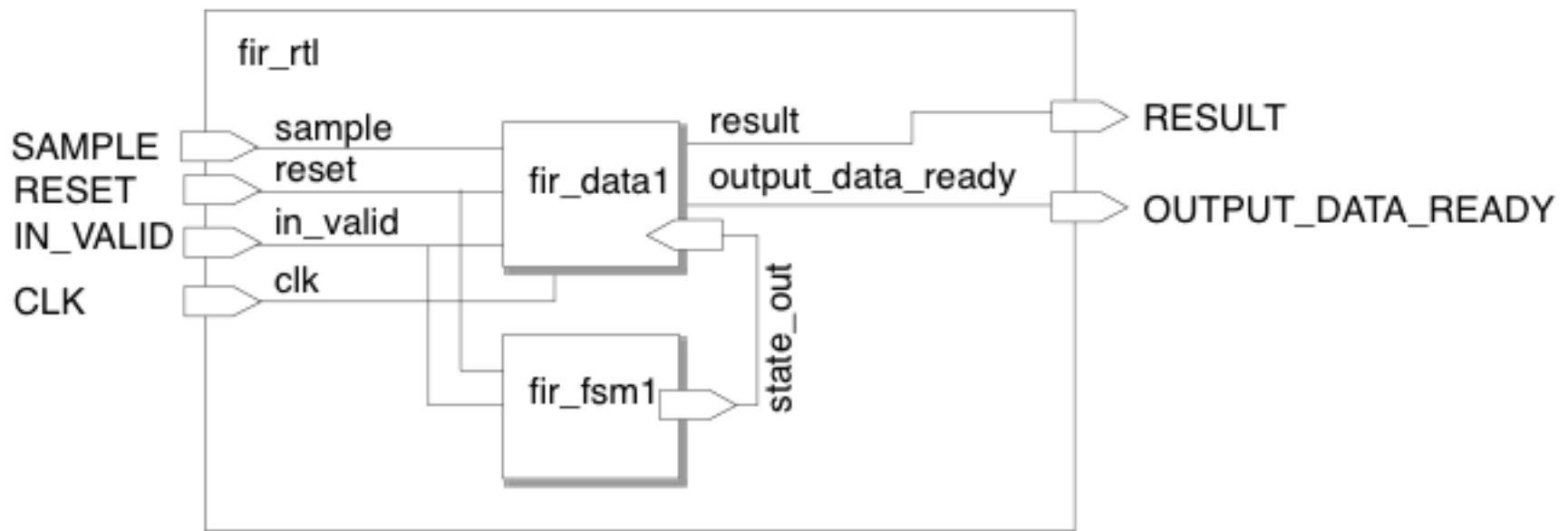
/*fir_data.h*/
SC_MODULE(fir_data) {

    sc_in<bool>      clk;
    sc_in<bool>      reset;
    sc_in<unsigned>  state_out;
    sc_in<int>       sample;
    sc_out<int>      result;
    sc_out<bool>     output_data_ready;
    ...
```

---

# Hierarchical RTL Module Example

---



# Hierarchical RTL Module Example

---

```

/****fir_rtl.h file****/
#include <systemc.h>
#include "fir_fsm.h"
#include "fir_data.h"

SC_MODULE(fir_rtl) {

    sc_in<bool>      clk;
    sc_in<bool>      reset;
    sc_in<bool>      in_valid;
    sc_in<int>       sample;
    sc_out<bool>     output_data_ready;
    sc_out<int>      result;

    sc_signal<unsigned> state_out;

    fir_fsm *fir_fsm1;
    fir_data *fir_data1;

    SC_CTOR(fir_rtl) {

        fir_fsm1 = new fir_fsm("FirFSM");
        fir_fsm1->clock(clk);
        fir_fsm1->reset(reset);

        fir_fsm1->in_valid(in_valid);
        fir_fsm1->state_out(state_out);

        fir_data1 = new fir_data("FirData");
        fir_data1->state_out(state_out);
        fir_data1->sample(sample);
        fir_data1->clock(clk);
        fir_data1->result(result);
        fir_data1->output_data_ready(output_data_ready);
    }
};

```

# Hierarchical RTL Module Example

---

```
/** fir_fsm.h file */

SC_MODULE(fir_fsm) {

    sc_in<bool>      clock;
    sc_in<bool>      reset;
    sc_in<bool>      in_valid;
    sc_out<unsigned> state_out;

    // defining the states of the ste machine
    enum {reset_s, first_s, second_s, third_s, output_s,
wait_s} state;

    SC_CTOR(fir_fsm)
    {
        SC_METHOD(entry);
        sensitive_pos(clock);
    };
    void entry();
};
```

---

# Hierarchical RTL Module Example

```
/***fir_fsm.cpp file***/
#include <systemc.h>
#include "fir_fsm.h"
void fir_fsm::entry() {
    sc_uint<3> state_tmp;

    // reset behavior
    if(reset.read()==true) {
        state = reset_s;
    }
    // main state machine
    switch(state) {
    case reset_s:
        state = wait_s;
        state_tmp = 0;
        state_out.write(state_tmp);
        break;
    case first_s:
        state = second_s;
        state_tmp = 1;
        state_out.write(state_tmp);
        break;
```

```
    case second_s:
        state = third_s;
        state_tmp = 2;
        state_out.write(state_tmp);
        break;
    case third_s:
        state = output_s;
        state_tmp = 3;
        state_out.write(state_tmp);
        break;
    case output_s:
        state = wait_s;
        state_tmp = 4;
        state_out.write(state_tmp);
        break;

    default:
        if(in_valid.read()==true) {
            state = first_s;
        };
        state_tmp = 5;
        state_out.write(state_tmp);
        break;
```

```
    }
}
```



# Hierarchical RTL Module Example

```

/****fir_data.h file****/

SC_MODULE(fir_data) {

    sc_in<unsigned>  state_out;
    sc_in<int>      sample;
    sc_out<int>     result;
    sc_out<bool>    output_data_ready;
    sc_in<bool>     clock;

    sc_int<19> acc;
    sc_int<8>  shift[16];
    sc_int<9>  coefs[16];

    SC_CTOR(fir_data)
    {
        SC_METHOD(entry);
        sensitive_pos(clock);
    };
    void entry();
};

/****fir_data.cpp file****/

#include <systemc.h>
#include "fir_data.h"

void fir_data::entry()
{
    #include "fir_const_rtl.h"
    sc_int<8> sample_tmp;

    sc_uint<3> state = state_out.read();

    switch (state) {
    case 0:
        sample_tmp = 0;
        acc = 0;
        for (int i=0; i<=15; i++) {
            shift[i] = 0;}
        result.write(0);
        output_data_ready.write(false);
        break;
    case 1 :
        sample_tmp = sample.read();
        acc = sample_tmp*coefs[0];
        acc += shift[14]* coefs[15];
        acc += shift[13]*coefs[14];
        acc += shift[12]*coefs[13];
        acc += shift[11]*coefs[12];
        output_data_ready.write(false);
        break;
    }
}
```

# Hierarchical RTL Module Example

---

```
case 2 :
    acc += shift[10]*coefs[11];
    acc += shift[9]*coefs[10];
    acc += shift[8]*coefs[9];
    acc += shift[7]*coefs[8];
    output_data_ready.write(false);
    break;
case 3 :
    acc += shift[6]*coefs[7];
    acc += shift[5]*coefs[6];
    acc += shift[4]*coefs[5];
    acc += shift[3]*coefs[4];
    output_data_ready.write(false);
    break;
case 4 :
    acc += shift[2]*coefs[3];
    acc += shift[1]*coefs[2];
    acc += shift[0]*coefs[1];
    for(int i=14; i>=0; i--) {
        shift[i+1] = shift[i];
    };
    shift[0] = sample.read();
    result.write(acc);

    output_data_ready.write(true);
    break;
case 5 :
    // This state waits for valid input
    output_data_ready.write(false);
    break;
default :
    output_data_ready.write(false);
    result.write(0);
}
}
```

---





# Final issues

---

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