Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
"Санкт-Петербургский политехнический университет Петра Великого"

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Лабораторная работа
по дисциплине «Высокоуровневое моделирование средствами SystemC»

Знакомство с описанием синхронных устройств на RTL уровне на языке SystemC

Выполнил
студент гр. 13541/2 Хуторной Я. В.

Преподаватель Мамутова О. В.

«__»_____2017г.

Санкт-Петербург
2017

Цели и задачи

1. Скопировать в локальную папку проект с примером регистра.
2. Выполнить компиляцию проекта. Запустить созданное приложение, наблюдая результаты моделирования устройства в консоли. Проверить правильность работы устройства, открыв сгенерированный *vcd* файл в *GTKWave*.
3. Разработать собственные устройства: счетчик и сдвигающий регистр, - с дополнительными функциями по индивидуальному заданию.
4. Создать тесты с самопроверкой для всех основных и дополнительных функций разработанных устройств.

Основные входы/выходы счетчика:

- *clk*
- *areset*
- *sreset_n*
- *dout*

Основные входы/выходы сдвигающего регистра:

- *clk*
- *areset*
- *sreset*
- *cin*
- *cout*
- *dout*

Ход работы

Был разработан сдвигающий регистр согласно индивидуальному заданию (наличие параллельной загрузи данных). Ниже представлен исходный код модуля регистра.

Листинг 1. Исходный код register.h

```
#include "systemc.h"

#ifndef DESIGN_H
#define    DESIGN_H

SC_MODULE(eightbit_register)
{
    //-----------------Ports Here--------------------------
    sc_in_clk clock;
    sc_in<bool> reset;
    sc_in<bool> areset;
    sc_in<bool> load;
    sc_in<bool> register_in;
```

```
    sc_out<bool> register_out;
    sc_out<sc_uint<8> > register_data;
    sc_in<sc_uint<8> > load_data;

    //------------Local Variables Here---------------------
    sc_uint<8> myregister;

    //------------Code Starts Here-------------------------

    void register_store();

    //------------Constructor Here-------------------------

    SC_CTOR(eightbit_register) :

           clock("clock"),
           reset("reset"),
           areset("areset"),
           load("load"),
           load_data("load_data"),
           register_data("register_data"),
           register_in("register_in"),
           register_out("register_out")

    {
        cout << "Executing new" << endl;
        SC_CTHREAD(register_store, clock.pos());
        async_reset_signal_is(areset, true);
        reset_signal_is(reset, true);
    }
};

#endif
```

Модуль в SystemC – это базовый элемент, включающий в себя процессы и другие модули.

В начале описания модуля содержится секция инициализации портов. Порты необходимы для взаимодействия модулей между собой. Каждый модуль может включать любое количество входных, выходных и смешанных *inout* портов.

Далее описана секция локальных переменных модуля. Эти переменные могут участвовать во всех взаимодействиях внутри создаваемого модуля.

Затем функция *register_store* объявляется в теле модуля и регистрируется как процесс внутри конструктора *SC_CTHREAD(register_store, clock.pos());*. Необходимо объявлять процесс как функцию void, не содержащую аргументов. Также задается чувствительность к фронту тактового сигнала и к сигналам синхронного

reset_signal_is(reset, true); и асинхронного *async_reset_signal_is(areset, true);* сброса. Тело процесса представлено ниже.

Листинг 2. Исходный код register.cpp

```
#include "register.h"

void eightbit_register::register_store()
{
    myregister = 0;
    register_data.write(myregister);
    wait();

    while (true)
    {
        if(load.read())
        {
            myregister = load_data.read();
            cout << "@" << sc_time_stamp() << " :: Have load " <<
            myregister << endl;
            register_data.write(myregister);
            wait();
        }
        else
        {
            register_out.write(myregister.bit(0));
            myregister = (register_in, myregister.range(7, 1));
            cout << "@" << sc_time_stamp() << " :: Have stored " <<
            myregister << endl;
            register_data.write(myregister);
            wait();
        }
    }
}
```

Процесс SC_CTHREAD в SystemC идентичен программному потоку. Ядро SystemC позволяет многим потокам выполняться параллельно. Процесс SC_CTHREAD запускается в процессе моделирования только один раз, однако его выполнение можно приостановить методом *wait()*. При завершении процесса, запустить его повторно нельзя, поэтому, данный тип процесса, как правило, содержит бесконечный цикл, имеющий по крайней мере один вызов функции *wait()*. В SC_CTHREAD нет отдельного списка чувствительности. Процесс SC_CTHREAD будет активирован каждый раз по перепаду тактового сигнала (перепад ждет функция *wait()*). В нашем случае процесс будет выполняться по каждому положительному фронту тактового сигнала.

В начале функции (до первого *wait()*) содержится секция, которая будет выполнятся в самом начале работы процесса, а также при срабатывании

сигналов синхронного и асинхронного сброса. Далее, в бесконечном цикле, описано тело процесса.

В SystemC, также как и в C/C++, есть строго определенная точка входа в программу. В случае SystemC – *sc_main()*. Функция *sc_main()* описана в файле *testbench.cpp*. Главная задача данной функции – это объявление всех объектов проекта (объект *eightbit_register*), соединение этих объектов посредством сигналов, а также запуск фазы моделирования.

Листинг 3. Исходный код testbench.cpp

```
#include "systemc.h"
#include "register.h"

int sc_main(int argc, char* argv[])
{
    sc_clock clock("clock", 4, SC_NS);
    sc_signal<bool> reset;
    sc_signal<bool> areset;
    sc_signal<bool> load;
    sc_signal<bool> register_in;
    sc_signal<bool> register_out;
    sc_signal<sc_uint<8> > register_data;
    sc_signal<sc_uint<8> > load_data;

    int i = 0;
    // Connect the DUT
    eightbit_register test_register("test_register");
    test_register.clock(clock);
    test_register.reset(reset);
    test_register.areset(areset);
    test_register.load(load);
    test_register.register_in(register_in);
    test_register.register_out(register_out);
    test_register.register_data(register_data);
    test_register.load_data(load_data);

    // Open VCD file
    sc_trace_file *wf = sc_create_vcd_trace_file("register_waveform");
    // Dump the desired signals
    sc_trace(wf, clock, "clock");
    sc_trace(wf, reset, "reset");
    sc_trace(wf, areset, "areset");
    sc_trace(wf, register_in, "din");
    sc_trace(wf, register_out, "dout");
    sc_trace(wf, register_data, "register_data");
    sc_trace(wf, load, "load");
    sc_trace(wf, load_data, "load_data");



    //Test code
```

```
    reset = 0;
    areset = 1;
    cout << "@" << sc_time_stamp() << " Asserting async reset\n" <<
endl;

    sc_start(6, SC_NS);
    areset = 0;
    cout << "@" << sc_time_stamp() << " De-Asserting async reset\n" <<
endl;

    cout << "@" << sc_time_stamp() << " Register_in = 1 " << endl;
    register_in = 1;
    sc_start(4, SC_NS);
    assert(register_data.read() == 128);
    cout << endl;

    cout << "@" << sc_time_stamp() << " Register_in = 1 " << endl;
    register_in = 1;
    sc_start(4, SC_NS);
    assert(register_data.read() == 192);
    cout << endl;

    cout << "@" << sc_time_stamp() << " Register_in = 0 " << endl;
    register_in = 0;
    sc_start(4, SC_NS);
    assert(register_data.read() == 96);
    cout << endl;

    cout << "@" << sc_time_stamp() << " Register_in = 0 " << endl;
    register_in = 0;
    sc_start(4, SC_NS);
    assert(register_data.read() == 48);
    cout << endl;

    reset = 1;
    cout << "@" << sc_time_stamp() << " Asserting sync reset\n" <<
endl;
    sc_start(4, SC_NS);
    assert(register_data.read() == 0);

    reset = 0;
    load_data = 25;
    load = 1;
    cout << "@" << sc_time_stamp() << " Asserting load\n" << endl;
    sc_start(4, SC_NS);
    assert(register_data.read() == load_data);

    load = 0;
    sc_start(12, SC_NS);

    areset = 1;
```

```
    cout << "@" << sc_time_stamp() << " Asserting async reset\n" <<
endl;
    sc_start(5, SC_NS);
    areset = 0;
    cout << "@" << sc_time_stamp() << " De-Asserting async reset\n" <<
endl;


    cout << "@" << sc_time_stamp() << " Terminating simulation\n" <<
endl;
    sc_close_vcd_trace_file(wf);
    return 0; // Terminate simulation

}
```

В начале создается объект класса *sc_clock* - тактовая частота с периодом 4 нс, а также описываются сигналы для моделирования.

Затем создается экземпляр объекта *eightbit_register*, к портам которого соединяются созданные сигналы выше, а также происходит создание и инициализация *vcd*-файла.

Далее описывается процедура моделирования. Имитатор SystemC имеет 3 главные фазы работы: разработка, выполнение и постобработка. Выполнение всех операторов до конструкции *sc_start()* есть фаза разработки. На этом этапе происходит инициализация структур данных и подготовка к следующей фазе выполнения. Фаза выполнения передает управление ядру моделирования SystemC, которое управляет работой всех процессов и создает иллюзию параллельности их выполнения. Постобработка связана с удалением всех созданных структур данных, освобождением памяти и завершением этапа моделирования. Метод *sc_start()* запускает фазу моделирования, которая состоит из инициализации и выполнения. Метод может принимать параметр, который является ограничением максимального времени моделирования.

Разработка и моделирование проводилось в среде Visual Studio 2012. Результат запуска программы представлен на рис. 2. Более наглядное представление результатов моделирования изображено на рис. 1. На рис. 1. представлена программа *GTKWave*, в которой запущен файл *register_waveform.vcd*, сгенерированный в процессе моделирования.

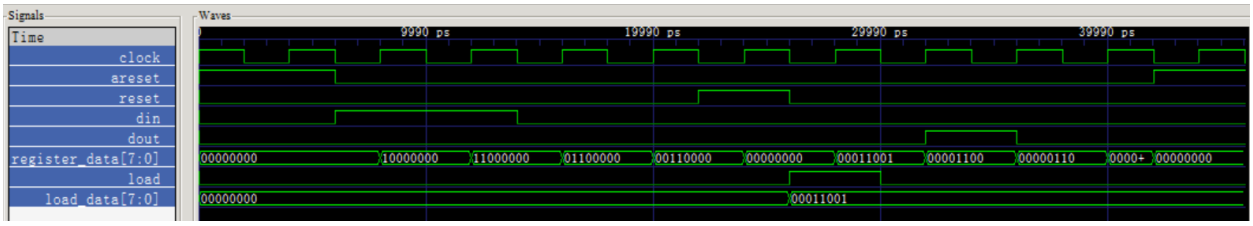

Рис. 1. Результат моделирования в программе *GTKWave*

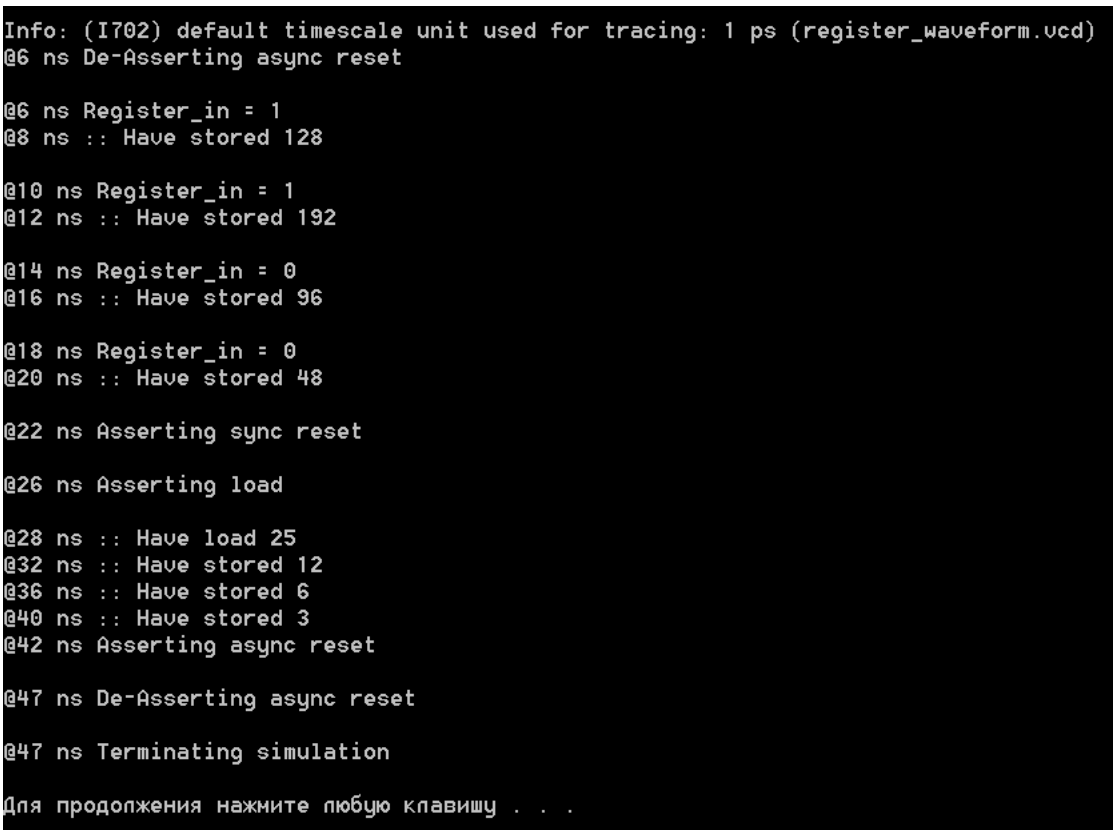

Рис. 2. Результат выполнения программы

Таким образом, результаты моделирования подтверждают корректность работы созданного устройства.

Далее был разработан счетчик согласно индивидуальному заданию (двунаправленный счет). Ниже представлен исходный код модуля.

Листинг 4. Исходный код counter.h

```
#include "systemc.h"

#ifndef DESIGN_H
#define    DESIGN_H

SC_MODULE(eightbit_counter)
{
    //--------------------Ports Here-----------------------
    sc_in_clk clock;
    sc_in<bool> areset;
    sc_in<bool> reset;
    sc_in<bool> up_down;
    sc_out<sc_uint<8> > counter_data;

    //------------Local Variables Here---------------------
    sc_uint<8> mycounter;

    //------------Code Starts Here-------------------------
    void counter_action();
```

```
    //------------Constructor Here-------------------------
    SC_CTOR(eightbit_counter) :

            clock("clock"),
            reset("reset"),
            areset("areset"),
            up_down("up_down")

    {
        cout << "Executing new" << endl;
        SC_CTHREAD(counter_action, clock.pos());
        async_reset_signal_is(areset, true);
        reset_signal_is(reset, true);
    }
};

#endif
```

Тело процесса представлено в листинге 5.

Листинг 5. Исходный код counter.cpp

```
#include "counter.h"

void eightbit_counter::counter_action()
{

    mycounter = 0;
    cout << "@" << sc_time_stamp() << " :: Counter sreset " <<
mycounter << endl;
    counter_data.write(mycounter);
    wait();

    while (true)
    {
        if(up_down.read())
        {
            mycounter++;
            cout << "@" << sc_time_stamp() << " :: Counter add " <<
mycounter << endl;
            counter_data.write(mycounter);
            wait();
        }
        else
        {
            mycounter--;
            cout << "@" << sc_time_stamp() << " :: Counter sub " <<
mycounter << endl;
            counter_data.write(mycounter);
            wait();
```

```
        }
    }
}
```

Исходный код главной функции *sc_main()* представлен ниже.

Листинг 6. Исходный код testbench.cpp

```
#include "systemc.h"
#include "counter.h"

int sc_main(int argc, char* argv[])
{
    sc_clock clock("clock", 5, SC_NS);
    sc_signal<bool> reset;
    sc_signal<bool> areset;
    sc_signal<bool> up_down;
    sc_signal<sc_uint<8> > counter_data;

    int i = 0;

    // Connect the DUT
    eightbit_counter test_counter("test_counter");
    test_counter.clock(clock);
    test_counter.reset(reset);
    test_counter.areset(areset);
    test_counter.up_down(up_down);
    test_counter.counter_data(counter_data);

    // Open VCD file
    sc_trace_file *wf = sc_create_vcd_trace_file("counter_waveform");

    // Dump the desired signals
    sc_trace(wf, clock, "clock");
    sc_trace(wf, reset, "reset");
    sc_trace(wf, up_down, "up_down");
    sc_trace(wf, counter_data, "counter_data");

    //Test code
    up_down = 1;
    reset = 0;
    areset = 1;
    cout << "@" << sc_time_stamp() << " Asserting async reset\n" <<
endl;

    sc_start(5, SC_NS);

    areset = 0;
    up_down = 1;
    sc_start(5, SC_NS);
    assert(counter_data.read() == 1);
```

```
    cout << endl;

    sc_start(5, SC_NS);
    assert(counter_data.read() == 2);
    cout << endl;

    sc_start(5, SC_NS);
    assert(counter_data.read() == 3);
    cout << endl;

    up_down = 0;
    sc_start(5, SC_NS);
    assert(counter_data.read() == 2);
    cout << endl;

    up_down = 1;
    sc_start(4, SC_NS);
    assert(counter_data.read() == 3);
    cout << endl;

     reset = 1;
    sc_start(5, SC_NS);
    assert(counter_data.read() == 0);
    cout << endl;

    cout << "@" << sc_time_stamp() << " Terminating simulation\n" <<
endl;
    sc_close_vcd_trace_file(wf);
    return 0; // Terminate simulation
}
```

Результаты работы программы представлены на рис. 3 и рис. 4.

```
@0 s Asserting async reset

@0 s :: Counter sreset 0
@0 s :: Counter sreset 0

Info: (I702) default timescale unit used for tracing: 1 ps (counter_waveform.vcd)
@5 ns :: Counter add 1

@10 ns :: Counter add 2

@15 ns :: Counter add 3

@20 ns :: Counter sub 2

@25 ns :: Counter add 3

@30 ns :: Counter sreset 0

@34 ns Terminating simulation

Для продолжения нажмите любую клавишу . . .
```

Рис. 3. Результат работы программы

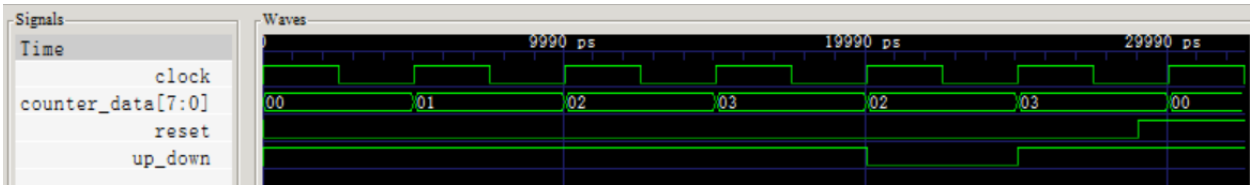

Рис. 4. Результат моделирования в *GTKWave*

Таким образом, результаты моделирования подтверждают корректность работы созданного устройства.