Санкт-Петербургский государственный политехнический университет

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторным работам**

**Дисциплина**: Высокоуровневое моделирование средствами SystemC

Выполнил студент гр. 13541/2 Муравьев Ф.Э.

(подпись)

Руководитель Мамутова О.В.

(подпись)

“ ” 2017 г.

Санкт-Петербург

2017

**Вводная работа**

**Программа работы:**

1. Используя исходный код программы из примера /usr/share/doc/systemc/examples/sysc/simple\_fifo, создать программу, моделирующую работу очереди.

<описание архитектуры проекта>

1. Выполнить компиляцию проекта, убедиться в правильности подключения библиотеки SystemC.
2. Запустить созданное приложение, наблюдая результаты моделирования устройства в консоли.
3. Модифицировать созданное устройство, изменив тип данных, с которыми работает очередь, на int. Соответствующим образом модифицировать тест.

Пример simple fifo с изменениями для вывода в формате int:

Листинг 1.1. Файл main.cpp

|  |
| --- |
| #include <systemc.h>  class write\_if : virtual public sc\_interface  {  public:  virtual void write(int) = 0;  virtual void reset() = 0;  };  class read\_if : virtual public sc\_interface  {  public:  virtual void read(int &) = 0;  virtual int num\_available() = 0;  };  class fifo : public sc\_channel, public write\_if, public read\_if  {  public:  fifo(sc\_module\_name name) : sc\_channel(name), num\_elements(0), first(0) {}  void write(int c) {  if (num\_elements == max)  wait(read\_event);  data[(first + num\_elements) % max] = c;  ++ num\_elements;  write\_event.notify();  }  void read(int &c){  if (num\_elements == 0)  wait(write\_event);  c = data[first];  -- num\_elements;  first = (first + 1) % max;  read\_event.notify();  }  void reset() { num\_elements = first = 0; }  int num\_available() { return num\_elements;}  private:  enum e { max = 10 };  char data[max];  int num\_elements, first;  sc\_event write\_event, read\_event;  };  class producer : public sc\_module  {  public:  sc\_port<write\_if> out;  SC\_HAS\_PROCESS(producer);  producer(sc\_module\_name name) : sc\_module(name)  {  SC\_THREAD(main);  }  void main()  {  const char \*str =  "Visit www.accellera.org and see what SystemC can do for you today!\n";  while (\*str)  out->write(\*str++);  }  };  class consumer : public sc\_module  {  public:  sc\_port<read\_if> in;  SC\_HAS\_PROCESS(consumer);  consumer(sc\_module\_name name) : sc\_module(name)  {  SC\_THREAD(main);  }  void main()  {  int c;  cout << endl << endl;  while (true) {  in->read(c);  cout << c << flush;  cout<<"\_";  if (in->num\_available() == 1)  cout << "<1>" << flush;  if (in->num\_available() == 9)  cout << "<9>" << flush;  }  }  };  class top : public sc\_module  {  public:  fifo \*fifo\_inst;  producer \*prod\_inst;  consumer \*cons\_inst;  top(sc\_module\_name name) : sc\_module(name)  {  fifo\_inst = new fifo("Fifo1");  prod\_inst = new producer("Producer1");  prod\_inst->out(\*fifo\_inst);  cons\_inst = new consumer("Consumer1");  cons\_inst->in(\*fifo\_inst);  }  };  int sc\_main (int, char \*[]) {  top top1("Top1");  sc\_start();  return 0;  } |

Результаты моделирования:

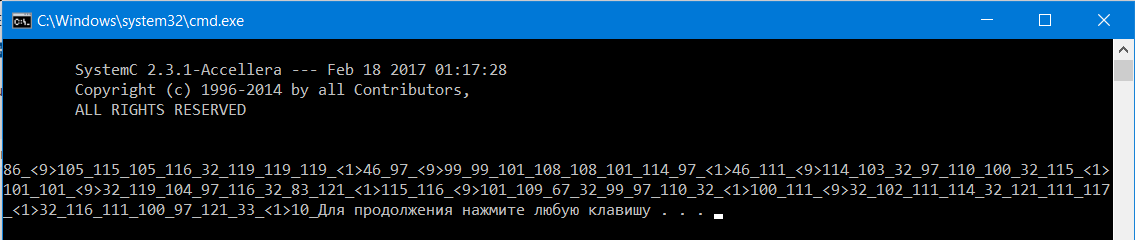


Рис 1.1. Результаты моделирования simple fifo

**Лабораторная работа №1**

**Программа работы:**

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

**Вариант индивидуального задания №2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | counter | | | shift\_register | | |
| ena | load | up/down | ena | load | left/right |
| 2 | + |  |  |  |  | + |

**Выполнение работы.**

1. Скопировали проект с примером регистра
2. Скомпилировали проект. Запустили созданное приложение.
3. Проверили полученные результаты моделирования устройства.
4. Разработать собственные устройства: счетчик и сдвигающий регистр, с дополнительными функциями по индивидуальному заданию.

**Счетчик:**

Счетчик с разрешением работы (ena)

Листинг 2.1. Файл Counter.h

|  |
| --- |
| #include "systemc.h"  #ifndef DESIGN\_H  #define DESIGN\_H  SC\_MODULE(cnt) {  sc\_in\_clk clock; // Clock input of the design  sc\_in<bool> reset; // active high, synchronous Reset input  sc\_in<bool> sync\_reset;  sc\_in<bool> ena;  sc\_out<sc\_uint<8> > cnt\_register\_out; // 8 bit vector output  sc\_uint<8> myregister;  void cnt\_count();  SC\_CTOR(cnt) :  clock("clock"),  reset("reset"),  cnt\_register\_out("register\_out") {  cout << "Executing new" << endl;  SC\_CTHREAD(cnt\_count, clock.pos());  reset\_signal\_is(sync\_reset, true);  async\_reset\_signal\_is(reset, true);  } // End of Constructor  }; // End of Module  #endif |

Листинг 2.2. Файл Counter.cpp

|  |
| --- |
| #include "counter.h"  void cnt::cnt\_count() {  // We check if reset is active  myregister = 0;  cnt\_register\_out.write(myregister);  wait();  while (true) {  if (ena.read()) {  myregister += 1;  cout << "@" << sc\_time\_stamp() << " :: Have count " << myregister << endl;  cnt\_register\_out.write(myregister);  }  wait();  }  } |

Листинг 2.3. Файл testbench.cpp

|  |
| --- |
| #include "systemc.h"  #include "counter.h"  #define soft\_assert(signal, expected) \  if (signal.read() != expected) { \  cerr << "@" << sc\_time\_stamp() << " Check failed. Expected: " << expected << ". Actual: " << signal.read() << ".\n" << endl; \  }  int sc\_main(int argc, char\* argv[]) {  sc\_clock clock("clock", 4, SC\_NS);  sc\_signal<bool> reset;  sc\_signal<bool> sync\_reset;  sc\_signal<bool> ena;  sc\_signal<sc\_uint<8> > cnt\_register\_out;  int i = 0;  // Connect the DUT  cnt test\_cnt("test\_cnt");  test\_cnt.clock(clock);  test\_cnt.reset(reset);  test\_cnt.sync\_reset(sync\_reset);  test\_cnt.ena(ena);  test\_cnt.cnt\_register\_out(cnt\_register\_out);  // Open VCD file  sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("cnt\_waveform");  // Dump the desired signals  sc\_trace(wf, clock, "clock");  sc\_trace(wf, reset, "reset");  sc\_trace(wf, sync\_reset, "sync\_reset");  sc\_trace(wf, ena, "ena");  sc\_trace(wf, cnt\_register\_out, "cnt\_register\_out");  reset = 0;  sync\_reset = 0;  ena = 0;  sc\_start(4, SC\_NS);  assert(cnt\_register\_out.read() == 0);  sc\_start(3, SC\_NS);    ena = 1;  sc\_start(3, SC\_NS);  assert(cnt\_register\_out.read() == 1);  reset = 1;  sc\_start(9, SC\_NS);  assert(cnt\_register\_out.read() == 0);    reset = 0;  sc\_start(2, SC\_NS);    sync\_reset = 1;  sc\_start(10, SC\_NS);  sync\_reset = 0;  assert(cnt\_register\_out.read() == 0);  sc\_start(1, SC\_NS);  sc\_start(27, SC\_NS);  assert(cnt\_register\_out.read() == 7);  ena = 0;  sc\_start(26, SC\_NS);  assert(cnt\_register\_out.read() == 7);  reset = 1; // Assert the reset  cout << "@" << sc\_time\_stamp() << " Asserting reset\n" << endl;  sc\_start(2, SC\_NS);  assert(cnt\_register\_out.read() == 0);  sc\_start(20, SC\_NS);  cout << "@" << sc\_time\_stamp() << " Terminating simulation\n" << endl;  sc\_close\_vcd\_trace\_file(wf);  return 0; // Terminate simulation  } |

Полученные результаты моделирования:

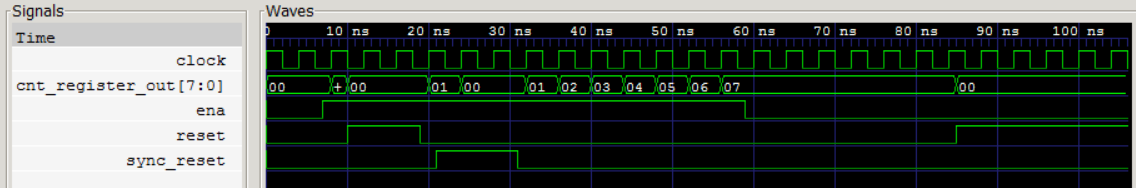


Рис 2.1. Результаты моделирования счетчика с разрешением работы

Результаты моделирования соответствуют ожидаемым результатам, устройство работает корректно.

**Сдвигающий регистр:**

Сдвигающий регистр с направлением сдвига

Листинг 2.4. Файл sdvig\_register.h

|  |
| --- |
| #include "systemc.h"  #ifndef DESIGN\_H  #define DESIGN\_H  SC\_MODULE(sd\_register) {  sc\_in\_clk clock; // Clock input of the design  sc\_in<bool> reset; // active high, synchronous Reset input  sc\_in<bool> sync\_reset;  sc\_in<bool> left\_right;  sc\_in<bool> d;  sc\_out<sc\_uint<8> > register\_out; // 8 bit vector output  sc\_uint<8> myregister;  void reg\_sd();  SC\_CTOR(sd\_register) :  clock("clock"),  reset("reset"),  register\_out("register\_out") {  cout << "Executing new" << endl;  SC\_CTHREAD(reg\_sd, clock.pos());  reset\_signal\_is(sync\_reset, true);  async\_reset\_signal\_is(reset, true);  } // End of Constructor  }; // End of Module  #endif |

Листинг 2.5. Файл sdvig\_register.cpp

|  |
| --- |
| #include "sdvig\_register.h"  void sd\_register::reg\_sd() {  // We check if reset is active  myregister = 0;  register\_out.write(myregister);  wait();  while (true) {  if (left\_right.read()) {  myregister = myregister>>1;  if(d.read()){  myregister = myregister|128;  }  else  {  myregister = myregister&127;  }  //myregister = myregister&127;  cout << "@" << sc\_time\_stamp() << " :: Have out " << myregister << endl;  register\_out.write(myregister);  }  else  {  myregister = myregister<<1;  if(d.read()){  myregister = myregister|1;  }  else  {  myregister = myregister&254;  }  cout << "@" << sc\_time\_stamp() << " :: Have out " << myregister << endl;  register\_out.write(myregister);  }  wait();  }  } |

Листинг 2.6. Файл testbench.cpp

|  |
| --- |
| //-----------------------------------------------------  // Testbench for the register ---------------->  //-----------------------------------------------------  #include "systemc.h"  #include "sdvig\_register.h"  #define soft\_assert(signal, expected) \  if (signal.read() != expected) { \  cerr << "@" << sc\_time\_stamp() << " Check failed. Expected: " << expected << ". Actual: " << signal.read() << ".\n" << endl; \  }  int sc\_main(int argc, char\* argv[]) {  sc\_clock clock("clock", 4, SC\_NS);  sc\_signal<bool> reset;  sc\_signal<bool> sync\_reset;  sc\_signal<bool> left\_right;  sc\_signal<bool> d;  sc\_signal<sc\_uint<8> > register\_out;  int i = 0;  // Connect the DUT  sd\_register test\_cnt("test\_cnt");  test\_cnt.clock(clock);  test\_cnt.reset(reset);  test\_cnt.sync\_reset(sync\_reset);  test\_cnt.left\_right(left\_right);  test\_cnt.d(d);  test\_cnt.register\_out(register\_out);  // Open VCD file  sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("reg\_waveform");  // Dump the desired signals  sc\_trace(wf, clock, "clock");  sc\_trace(wf, reset, "reset");  sc\_trace(wf, sync\_reset, "sync\_reset");  sc\_trace(wf, left\_right, "left\_right");  sc\_trace(wf, d, "d");  sc\_trace(wf, register\_out, "register\_out");  reset = 0;  sync\_reset = 0;  left\_right = 0;  d=1;  sc\_start(5, SC\_NS);  assert(register\_out.read() == 1);  sc\_start(2, SC\_NS);  reset = 1;  sc\_start(10, SC\_NS);  reset = 0;  assert(register\_out.read() == 0);  sc\_start(6, SC\_NS);    sync\_reset = 1;  sc\_start(7, SC\_NS);  sync\_reset = 0;  sc\_start(7, SC\_NS);  d=0;  assert(register\_out.read() == 3);  sc\_start(13, SC\_NS);  assert(register\_out.read() == 24);  left\_right = 1;  sc\_start(23, SC\_NS);  d=1;  sc\_start(25, SC\_NS);    reset = 1; // Assert the reset  cout << "@" << sc\_time\_stamp() << " Asserting reset\n" << endl;  sc\_start(2, SC\_NS);  sc\_start(20, SC\_NS);  cout << "@" << sc\_time\_stamp() << " Terminating simulation\n" << endl;  sc\_close\_vcd\_trace\_file(wf);  return 0; // Terminate simulation  } |

Полученные результаты моделирования:

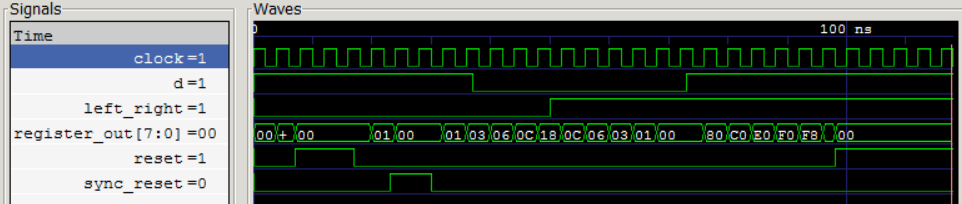


Рис 2.2. Результаты моделирования счетчика с разрешением работы

Результаты моделирования соответствуют ожидаемым результатам, устройство работает корректно.