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

**Лабораторная работа №3**

**Знакомство с описанием параметризованных устройств на языке SystemC**

**Программа работы:**

1. Создать потактовое описание FIFO, работающего с 8-разрядными словами. Глубина FIFO должна задаваться в конструкторе.

Обязательные входы:

- clk

- sreset\_n

- data\_in

- push

- pop

Обязательные выходы:

- data\_out

- empty

- full

2. Создать на основе класса FIFO шаблон класса FIFOParam, в котором параметром выступает тип данных.

3. Унаследовать от созданного шаблона класса FIFOParam, шаблон класса FIFOParamExtended с дополнительными выходами:

- almost\_empty

- almost\_full

Для всех трёх описаний необходимо написать соответствующие тесты.

**Выполнение работы.**

1. Создать потактовое описание FIFO, работающего с 8-разрядными словами. Глубина FIFO должна задаваться в конструкторе.

Листинг 1.1. Файл FIFO.h

|  |
| --- |
| #include "systemc.h"  #ifndef DESIGN\_H  #define DESIGN\_H  class FIFO : public sc\_module {  public:  //input  sc\_in\_clk clk; // Clock input of the design  sc\_in<bool> sreset\_n; // active high, synchronous Reset input  sc\_in<sc\_uint<8>> data\_in; // 8 bit vector input  sc\_in<bool> push;  sc\_in<bool> pop;  //output  sc\_out<sc\_uint<8>> data\_out; // 8 bit vector output  sc\_out<bool> empty;  sc\_out<bool> full;  //------------Local Variables Here---------------------  int \*data;  int size;  sc\_signal<sc\_uint<32>> pop\_pointer;  sc\_signal<sc\_uint<32>> push\_pointer;  sc\_signal<bool> push\_event;  sc\_signal<bool> pop\_event;  sc\_signal<bool> flag\_can\_push;  sc\_signal<bool> flag\_can\_pop;  sc\_signal<int> count\_of\_elements;  void write\_FIFO();  void read\_FIFO();  void elem\_counter();  void compare\_pointer();  // Constructor  SC\_HAS\_PROCESS(FIFO);  FIFO(sc\_module\_name name, int max) :  clk("clk"),  sreset\_n("sreset\_n"),  data\_in("data\_in"),  data\_out("data\_out") {  data=new int [max];  size=max;  //flag\_push\_on\_pop=false;  push\_pointer=0;  pop\_pointer=0;  //compare\_pointer  SC\_METHOD(compare\_pointer);  sensitive << push\_pointer << pop\_pointer;  //elements count  SC\_CTHREAD(elem\_counter, clk.pos());  reset\_signal\_is(sreset\_n, false);  //write\_FIFO  SC\_CTHREAD(write\_FIFO, clk.pos());  reset\_signal\_is(sreset\_n, false);  //read\_FIFO  SC\_CTHREAD(read\_FIFO, clk.pos());  reset\_signal\_is(sreset\_n, false);  } // End of Constructor  }; // End of Module  #endif /\* DESIGN\_H \*/ |

Листинг 1.2. Файл FIFO.cpp

|  |
| --- |
| #include "FIFO.h"  void FIFO::write\_FIFO() {  push\_pointer=0;  push\_event=false;  empty=1;  full=0;  wait();  while (true) {  if ((push==1) && (flag\_can\_push==true)) {  data[push\_pointer.read()]=data\_in.read();  cout<<sc\_time\_stamp()<<" Write data = "<<data\_in.read()<<"; push\_pointer change from "<<push\_pointer;  push\_pointer=push\_pointer.read()+1;  int push\_pointer\_temp=push\_pointer.read()+1;  push\_event=true;  if(push\_pointer\_temp==size){ push\_pointer=0; push\_pointer\_temp=0;}  cout<< " to "<< push\_pointer\_temp <<endl;  }  else  {  push\_event=false;  }  wait();  }  }  void FIFO::read\_FIFO() {  pop\_pointer=0;  pop\_event==false;  data\_out=0;  wait();  while (true) {  if ((pop==1) && (flag\_can\_pop==true)){  data\_out=data[pop\_pointer.read()];  cout<<sc\_time\_stamp()<<" Read data = "<<data[pop\_pointer.read()]<<"; pop\_pointer change from "<<pop\_pointer;  pop\_pointer=pop\_pointer.read()+1;  pop\_event=true;  int pop\_pointer\_temp=pop\_pointer.read()+1;  if (pop\_pointer\_temp==size) {pop\_pointer=0; pop\_pointer\_temp=0;}  cout<< " to "<< pop\_pointer\_temp<<endl;  }  else  {  pop\_event=false;  }  wait();  }  }  void FIFO::elem\_counter() {  count\_of\_elements=0;  wait();  while (true) {  if((push\_event==true) && (pop\_event==false))  {  count\_of\_elements=count\_of\_elements+1;  }  else  if((push\_event==false) && (pop\_event==true))  {  count\_of\_elements=count\_of\_elements-1;    }  wait();  }  }  void FIFO::compare\_pointer(){  // Формируем флаг can\_push  if((push\_pointer.read()==pop\_pointer.read()) && ((count\_of\_elements==size) || ((count\_of\_elements==size-1) && (push\_event==true))) && (pop\_event!=true))  {  flag\_can\_push=false;  }  else{  flag\_can\_push=true;  }  // формируем флаг can\_pop  if((push\_pointer.read()==pop\_pointer.read()) && ((count\_of\_elements==0) || ((count\_of\_elements==1) && (pop\_event==true))) && (push\_event!=true))  {  flag\_can\_pop=false;  }  else{  flag\_can\_pop=true;  }  // Формируем флаг empty  if(((count\_of\_elements==0) || ((count\_of\_elements==1) && (pop\_event==true))) && (push\_event!=true))  {  empty=1;  }  else  {  empty=0;  }  // Формируем флаг full  if(((count\_of\_elements==size) || ((count\_of\_elements==size-1) && (push\_event==true))) && (pop\_event!=true))  {  full=1;  }  else  {  full=0;  }  } |

Листинг 1.3. Файл testbench.cpp

|  |
| --- |
| //-----------------------------------------------------  // Testbench for the FIFO ---------------->  //-----------------------------------------------------  #include "systemc.h"  #include "FIFO.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\_core::sc\_report\_handler::set\_actions( "/IEEE\_Std\_1666/deprecated", sc\_core::SC\_DO\_NOTHING );  sc\_clock clk("clk", 4, SC\_NS);  sc\_signal<bool> sreset\_n;  sc\_signal<sc\_uint<8> > data\_in;  sc\_signal<bool> push;  sc\_signal<bool> pop;  sc\_signal<sc\_uint<8> > data\_out;  sc\_signal<bool> empty;  sc\_signal<bool> full;  // Connect the DUT  FIFO test\_FIFO("test\_FIFO", 2);  test\_FIFO.clk(clk);  test\_FIFO.sreset\_n(sreset\_n);  test\_FIFO.data\_in(data\_in);  test\_FIFO.push(push);  test\_FIFO.pop(pop);  test\_FIFO.data\_out(data\_out);  test\_FIFO.empty(empty);  test\_FIFO.full(full);  // Open VCD file  sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("FIFO\_waveform");  // Dump the desired signals  sc\_trace(wf, clk, "clk");  sc\_trace(wf, sreset\_n, "sreset\_n");  sc\_trace(wf, data\_in, "data\_in");  sc\_trace(wf, push, "push");  sc\_trace(wf, pop, "pop");  sc\_trace(wf, data\_out, "data\_out");  sc\_trace(wf, empty, "empty");  sc\_trace(wf, full, "full");  sreset\_n = 0; // Assert the reset  push=0;  pop=0;  cout << "@" << sc\_time\_stamp() << " Asserting reset\n" << endl;  sc\_start(6, SC\_NS);  //6ns  assert(empty.read() == 1);  assert(full.read() == 0);  sreset\_n = 1; // De-assert the reset  cout << "@" << sc\_time\_stamp() << " De-Asserting reset\n" << endl;  data\_in=3;  push=1;  sc\_start(4, SC\_NS);  //10ns  assert(empty.read() == 0);  push=0;  data\_in=0;  pop=1;  sc\_start(4, SC\_NS);  //14ns  assert(empty.read() == 1);  assert(data\_out.read() == 3);  data\_in=4;  pop=0;  push=1;  sc\_start(4, SC\_NS);  //18ns  assert(empty.read() == 0);  pop=1;  data\_in=5;  sc\_start(4, SC\_NS);  //22ns  assert(data\_out.read() == 4);    pop=0;  data\_in=6;  sc\_start(4, SC\_NS);  //26ns  assert(full.read() == 1);  data\_in=8;  sc\_start(4, SC\_NS);  //30ns  assert(full.read() == 1);  pop=1;  push=0;  sc\_start(4, SC\_NS);  //34ns  assert(full.read() == 0);  assert(data\_out.read() == 5);  sc\_start(4, SC\_NS);  //38ns  assert(full.read() == 0);  assert(data\_out.read() == 6);  assert(empty.read() == 1);  sc\_start(8, SC\_NS);  //42ns  sreset\_n = 0;  sc\_start(17, SC\_NS);  cout << "@" << sc\_time\_stamp() << " Asserting reset\n" << endl;  cout << "@" << sc\_time\_stamp() << " Terminating simulation\n" << endl;  sc\_close\_vcd\_trace\_file(wf);  return 0; // Terminate simulation  } |

Для проверки корректности работы FIFO был создан тест.

**Полученные результаты моделирования:**

Сообщения из консоли:

|  |
| --- |
| SystemC 2.3.1-Accellera --- Feb 18 2017 01:17:28  Copyright (c) 1996-2014 by all Contributors,  ALL RIGHTS RESERVED  @0 s Asserting reset  Info: (I702) default timescale unit used for tracing: 1 ps (FIFO\_waveform.vcd)  @6 ns De-Asserting reset  8 ns Write data = 3; push\_pointer change from 0 to 1  12 ns Read data = 3; pop\_pointer change from 0 to 1  16 ns Write data = 4; push\_pointer change from 1 to 0  20 ns Read data = 4; pop\_pointer change from 1 to 0  20 ns Write data = 5; push\_pointer change from 0 to 1  24 ns Write data = 6; push\_pointer change from 1 to 0  32 ns Read data = 5; pop\_pointer change from 0 to 1  36 ns Read data = 6; pop\_pointer change from 1 to 0  @63 ns Asserting reset  @63 ns Terminating simulation  Для продолжения нажмите любую клавишу . . . |

Результаты моделирования:

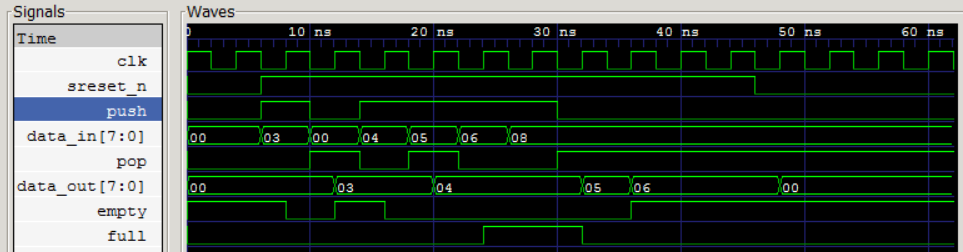


Рис 1.1. Результаты моделирования конечного автомата

Результаты моделирования соответствуют ожидаемым результатам, устройство работает корректно.

1. Создать на основе класса FIFO шаблон класса FIFOParam, в котором параметром выступает тип данных

Листинг 2.1. Файл FIFOParam.h

|  |
| --- |
| #include "systemc.h"  #ifndef DESIGN\_H  #define DESIGN\_H  template <class T>  class FIFOParam : public sc\_module {  public:  //input  sc\_in\_clk clk; // Clock input of the design  sc\_in<bool> sreset\_n; // active high, synchronous Reset input  sc\_in<T> data\_in; // vector input with format T  sc\_in<bool> push;  sc\_in<bool> pop;  //output  sc\_out<T> data\_out; // vector output with format T  sc\_out<bool> empty;  sc\_out<bool> full;  //------------Local Variables Here---------------------  T \*data;  int size;  sc\_signal<sc\_uint<32>> pop\_pointer;  sc\_signal<sc\_uint<32>> push\_pointer;  sc\_signal<bool> push\_event;  sc\_signal<bool> pop\_event;  sc\_signal<bool> flag\_can\_push;  sc\_signal<bool> flag\_can\_pop;  sc\_signal<int> count\_of\_elements;  void write\_FIFO();  void read\_FIFO();  void elem\_counter();  void compare\_pointer();  // Constructor  SC\_HAS\_PROCESS(FIFOParam);  FIFOParam(sc\_module\_name name, int max) :  clk("clk"),  sreset\_n("sreset\_n"),  data\_in("data\_in"),  data\_out("data\_out") {  data=new T [max];  size=max;  //flag\_push\_on\_pop=false;  push\_pointer=0;  pop\_pointer=0;  //compare\_pointer  SC\_METHOD(compare\_pointer);  sensitive << push\_pointer << pop\_pointer;  //elements count  SC\_CTHREAD(elem\_counter, clk.pos());  reset\_signal\_is(sreset\_n, false);  //write\_FIFO  SC\_CTHREAD(write\_FIFO, clk.pos());  reset\_signal\_is(sreset\_n, false);  //read\_FIFO  SC\_CTHREAD(read\_FIFO, clk.pos());  reset\_signal\_is(sreset\_n, false);  } // End of Constructor  }; // End of Module  template <class T>  void FIFOParam<T>::write\_FIFO() {  push\_pointer=0;  push\_event=false;  empty=1;  full=0;  wait();  while (true) {  if ((push==1) && (flag\_can\_push==true)) {  data[push\_pointer.read()]=data\_in.read();  cout<<sc\_time\_stamp()<<" Write data = "<<data\_in.read()<<"; push\_pointer change from "<<push\_pointer;  push\_pointer=push\_pointer.read()+1;  int push\_pointer\_temp=push\_pointer.read()+1;  push\_event=true;  if(push\_pointer\_temp==size){ push\_pointer=0; push\_pointer\_temp=0;}  cout<< " to "<< push\_pointer\_temp <<endl;  }  else  {  push\_event=false;  }  wait();  }  }  template <class T>  void FIFOParam<T>::read\_FIFO() {  pop\_pointer=0;  pop\_event==false;  data\_out=0;  wait();  while (true) {  if ((pop==1) && (flag\_can\_pop==true)){  data\_out=data[pop\_pointer.read()];  cout<<sc\_time\_stamp()<<" Read data = "<<data[pop\_pointer.read()]<<"; pop\_pointer change from "<<pop\_pointer;  pop\_pointer=pop\_pointer.read()+1;  pop\_event=true;  int pop\_pointer\_temp=pop\_pointer.read()+1;  if (pop\_pointer\_temp==size) {pop\_pointer=0; pop\_pointer\_temp=0;}  cout<< " to "<< pop\_pointer\_temp<<endl;  }  else  {  pop\_event=false;  }  wait();  }  }  template <class T>  void FIFOParam<T>::elem\_counter() {  count\_of\_elements=0;  wait();  while (true) {  if((push\_event==true) && (pop\_event==false))  {  count\_of\_elements=count\_of\_elements+1;  }  else  if((push\_event==false) && (pop\_event==true))  {  count\_of\_elements=count\_of\_elements-1;    }  wait();  }  }  template <class T>  void FIFOParam<T>::compare\_pointer(){  // Формируем флаг can\_push  if((push\_pointer.read()==pop\_pointer.read()) && ((count\_of\_elements==size) || ((count\_of\_elements==size-1) && (push\_event==true))) && (pop\_event!=true))  {  flag\_can\_push=false;  }  else{  flag\_can\_push=true;  }  // формируем флаг can\_pop  if((push\_pointer.read()==pop\_pointer.read()) && ((count\_of\_elements==0) || ((count\_of\_elements==1) && (pop\_event==true))) && (push\_event!=true))  {  flag\_can\_pop=false;  }  else{  flag\_can\_pop=true;  }  // Формируем флаг empty  if(((count\_of\_elements==0) || ((count\_of\_elements==1) && (pop\_event==true))) && (push\_event!=true))  {  empty=1;  }  else  {  empty=0;  }  // Формируем флаг full  if(((count\_of\_elements==size) || ((count\_of\_elements==size-1) && (push\_event==true))) && (pop\_event!=true))  {  full=1;  }  else  {  full=0;  }  }  #endif /\* DESIGN\_H \*/ |

Листинг 2.2. Файл testbench.cpp

|  |
| --- |
| //-----------------------------------------------------  // Testbench for the FIFO ---------------->  //-----------------------------------------------------  #include "systemc.h"  #include "FIFOParam.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\_core::sc\_report\_handler::set\_actions( "/IEEE\_Std\_1666/deprecated", sc\_core::SC\_DO\_NOTHING );  sc\_clock clk("clk", 4, SC\_NS);  sc\_signal<bool> sreset\_n;  sc\_signal<char> data\_in;  sc\_signal<bool> push;  sc\_signal<bool> pop;  sc\_signal<char> data\_out;  sc\_signal<bool> empty;  sc\_signal<bool> full;  // Connect the DUT  FIFOParam<char> test\_FIFO("test\_FIFO", 2);  test\_FIFO.clk(clk);  test\_FIFO.sreset\_n(sreset\_n);  test\_FIFO.data\_in(data\_in);  test\_FIFO.push(push);  test\_FIFO.pop(pop);  test\_FIFO.data\_out(data\_out);  test\_FIFO.empty(empty);  test\_FIFO.full(full);  // Open VCD file  sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("FIFO\_waveform");  // Dump the desired signals  sc\_trace(wf, clk, "clk");  sc\_trace(wf, sreset\_n, "sreset\_n");  sc\_trace(wf, data\_in, "data\_in");  sc\_trace(wf, push, "push");  sc\_trace(wf, pop, "pop");  sc\_trace(wf, data\_out, "data\_out");  sc\_trace(wf, empty, "empty");  sc\_trace(wf, full, "full");  sreset\_n = 0; // Assert the reset  push=0;  pop=0;  cout << "@" << sc\_time\_stamp() << " Asserting reset\n" << endl;  sc\_start(6, SC\_NS);  //6ns  assert(empty.read() == 1);  assert(full.read() == 0);  sreset\_n = 1; // De-assert the reset  cout << "@" << sc\_time\_stamp() << " De-Asserting reset\n" << endl;  data\_in='a';  push=1;  sc\_start(4, SC\_NS);  //10ns  assert(empty.read() == 0);  push=0;  data\_in=0;  pop=1;  sc\_start(4, SC\_NS);  //14ns  assert(empty.read() == 1);  assert(data\_out.read() == 'a');  data\_in='2';  pop=0;  push=1;  sc\_start(4, SC\_NS);  //18ns  assert(empty.read() == 0);  pop=1;  data\_in='f';  sc\_start(4, SC\_NS);  //22ns  assert(data\_out.read() == '2');    pop=0;  data\_in='a';  sc\_start(4, SC\_NS);  //26ns  assert(full.read() == 1);  pop=1;  push=0;  sc\_start(4, SC\_NS);  //30ns  assert(full.read() == 0);  assert(data\_out.read() == 'f');  sc\_start(4, SC\_NS);  //34ns  assert(full.read() == 0);  assert(data\_out.read() == 'a');  assert(empty.read() == 1);  sc\_start(8, SC\_NS);  //38ns  sreset\_n = 0;  sc\_start(17, SC\_NS);  cout << "@" << sc\_time\_stamp() << " Asserting reset\n" << endl;  cout << "@" << sc\_time\_stamp() << " Terminating simulation\n" << endl;  sc\_close\_vcd\_trace\_file(wf);  return 0; // Terminate simulation  } |

Для проверки корректности работы FIFO был создан тест.

**Полученные результаты моделирования:**

Сообщения из консоли:

|  |
| --- |
| SystemC 2.3.1-Accellera --- Feb 18 2017 01:17:28  Copyright (c) 1996-2014 by all Contributors,  ALL RIGHTS RESERVED  @0 s Asserting reset  Info: (I702) default timescale unit used for tracing: 1 ps (FIFO\_waveform.vcd)  @6 ns De-Asserting reset  8 ns Write data = a; push\_pointer change from 0 to 1  12 ns Read data = a; pop\_pointer change from 0 to 1  16 ns Write data = 2; push\_pointer change from 1 to 0  20 ns Read data = 2; pop\_pointer change from 1 to 0  20 ns Write data = f; push\_pointer change from 0 to 1  24 ns Write data = a; push\_pointer change from 1 to 0  28 ns Read data = f; pop\_pointer change from 0 to 1  32 ns Read data = a; pop\_pointer change from 1 to 0  @59 ns Asserting reset  @59 ns Terminating simulation  Для продолжения нажмите любую клавишу . . . |

Результаты моделирования:

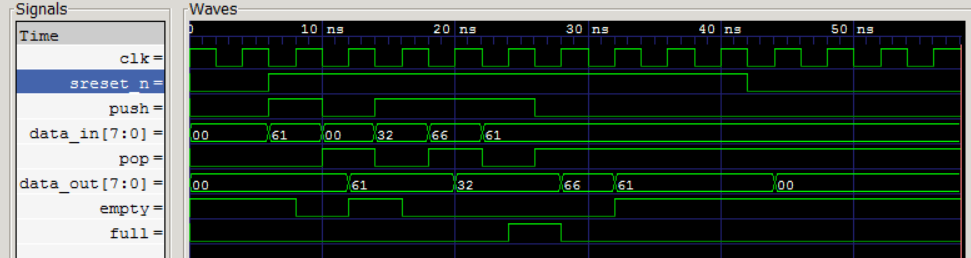


Рис 2.1. Результаты моделирования конечного автомата

Результаты моделирования соответствуют ожидаемым результатам, устройство работает корректно.

1. Унаследовать от созданного шаблона класса FIFOParam, шаблон класса FIFOParamExtended с дополнительными выходами:

- almost\_empty

- almost\_full

Листинг 3.1. Файл FIFOParam.h

|  |
| --- |
| #include "systemc.h"  #ifndef DESIGN\_H  #define DESIGN\_H  template <class T>  class FIFOParam : public sc\_module {  public:  //input  sc\_in\_clk clk; // Clock input of the design  sc\_in<bool> sreset\_n; // active high, synchronous Reset input  sc\_in<T> data\_in; // vector input with format T  sc\_in<bool> push;  sc\_in<bool> pop;  //output  sc\_out<T> data\_out; // vector output with format T  sc\_out<bool> empty;  sc\_out<bool> full;  //------------Local Variables Here---------------------  T \*data;  int size;  sc\_signal<sc\_uint<32>> pop\_pointer;  sc\_signal<sc\_uint<32>> push\_pointer;  sc\_signal<bool> push\_event;  sc\_signal<bool> pop\_event;  sc\_signal<bool> flag\_can\_push;  sc\_signal<bool> flag\_can\_pop;  sc\_signal<int> count\_of\_elements;  void write\_FIFO();  void read\_FIFO();  void elem\_counter();  void compare\_pointer();  // Constructor  SC\_HAS\_PROCESS(FIFOParam);  FIFOParam(sc\_module\_name name, int max) :  clk("clk"),  sreset\_n("sreset\_n"),  data\_in("data\_in"),  data\_out("data\_out") {  data=new T [max];  size=max;  //flag\_push\_on\_pop=false;  push\_pointer=0;  pop\_pointer=0;  //compare\_pointer  SC\_METHOD(compare\_pointer);  sensitive << push\_pointer << pop\_pointer;  //elements count  SC\_CTHREAD(elem\_counter, clk.pos());  reset\_signal\_is(sreset\_n, false);  //write\_FIFO  SC\_CTHREAD(write\_FIFO, clk.pos());  reset\_signal\_is(sreset\_n, false);  //read\_FIFO  SC\_CTHREAD(read\_FIFO, clk.pos());  reset\_signal\_is(sreset\_n, false);  } // End of Constructor  }; // End of Module  template <class T>  void FIFOParam<T>::write\_FIFO() {  push\_pointer=0;  push\_event=false;  empty=1;  full=0;  wait();  while (true) {  if ((push==1) && (flag\_can\_push==true)) {  data[push\_pointer.read()]=data\_in.read();  cout<<sc\_time\_stamp()<<" Write data = "<<data\_in.read()<<"; push\_pointer change from "<<push\_pointer;  push\_pointer=push\_pointer.read()+1;  int push\_pointer\_temp=push\_pointer.read()+1;  push\_event=true;  if(push\_pointer\_temp==size){ push\_pointer=0; push\_pointer\_temp=0;}  cout<< " to "<< push\_pointer\_temp <<endl;  }  else  {  push\_event=false;  }  wait();  }  }  template <class T>  void FIFOParam<T>::read\_FIFO() {  pop\_pointer=0;  pop\_event==false;  data\_out=0;  wait();  while (true) {  if ((pop==1) && (flag\_can\_pop==true)){  data\_out=data[pop\_pointer.read()];  cout<<sc\_time\_stamp()<<" Read data = "<<data[pop\_pointer.read()]<<"; pop\_pointer change from "<<pop\_pointer;  pop\_pointer=pop\_pointer.read()+1;  pop\_event=true;  int pop\_pointer\_temp=pop\_pointer.read()+1;  if (pop\_pointer\_temp==size) {pop\_pointer=0; pop\_pointer\_temp=0;}  cout<< " to "<< pop\_pointer\_temp<<endl;  }  else  {  pop\_event=false;  }  wait();  }  }  template <class T>  void FIFOParam<T>::elem\_counter() {  count\_of\_elements=0;  wait();  while (true) {  if((push\_event==true) && (pop\_event==false))  {  count\_of\_elements=count\_of\_elements+1;  }  else  if((push\_event==false) && (pop\_event==true))  {  count\_of\_elements=count\_of\_elements-1;    }  wait();  }  }  template <class T>  void FIFOParam<T>::compare\_pointer(){  // Формируем флаг can\_push  if((push\_pointer.read()==pop\_pointer.read()) && ((count\_of\_elements==size) || ((count\_of\_elements==size-1) && (push\_event==true))) && (pop\_event!=true))  {  flag\_can\_push=false;  }  else{  flag\_can\_push=true;  }  // формируем флаг can\_pop  if((push\_pointer.read()==pop\_pointer.read()) && ((count\_of\_elements==0) || ((count\_of\_elements==1) && (pop\_event==true))) && (push\_event!=true))  {  flag\_can\_pop=false;  }  else{  flag\_can\_pop=true;  }  // Формируем флаг empty  if(((count\_of\_elements==0) || ((count\_of\_elements==1) && (pop\_event==true))) && (push\_event!=true))  {  empty=1;  }  else  {  empty=0;  }  // Формируем флаг full  if(((count\_of\_elements==size) || ((count\_of\_elements==size-1) && (push\_event==true))) && (pop\_event!=true))  {  full=1;  }  else  {  full=0;  }  }  #endif /\* DESIGN\_H \*/ |

Листинг 3.2. Файл FIFOParamExtended.h

|  |
| --- |
| #include "FIFOParam.h"  template <class T>  class FIFOParamExtended : public FIFOParam<T>{  public:  sc\_out<bool> almost\_empty;  sc\_out<bool> almost\_full;  int almost\_level;  void almost\_out();  SC\_HAS\_PROCESS(FIFOParamExtended);  FIFOParamExtended(sc\_module\_name name, int max, int almost=1) : FIFOParam(name, max)  {  almost\_level=almost;  SC\_METHOD(almost\_out);  sensitive << FIFOParam<T>::push\_pointer << FIFOParam<T>::pop\_pointer;  };  };  template <class T>  void FIFOParamExtended<T>::almost\_out() {  if ((FIFOParam<T>::count\_of\_elements<almost\_level) ||  ((FIFOParam<T>::count\_of\_elements==almost\_level) &&  ((FIFOParam<T>::push\_event==false) || (FIFOParam<T>::pop\_event==true && FIFOParam<T>::push\_event==true))) ||  ((FIFOParam<T>::count\_of\_elements==almost\_level+1) && (FIFOParam<T>::pop\_event==true && FIFOParam<T>::push\_event==false))  ) almost\_empty=1;  else almost\_empty=0;    if ((FIFOParam<T>::count\_of\_elements>FIFOParam<T>::size-almost\_level) ||  ((FIFOParam<T>::count\_of\_elements==FIFOParam<T>::size-almost\_level) && ((FIFOParam<T>::pop\_event==false) || (FIFOParam<T>::pop\_event==true && FIFOParam<T>::push\_event==true))) ||  ((FIFOParam<T>::count\_of\_elements==FIFOParam<T>::size-almost\_level-1) && (FIFOParam<T>::push\_event==true && FIFOParam<T>::pop\_event==false))) almost\_full=1;  else almost\_full=0;  } |

Листинг 3.3. Файл testbench.cpp

|  |
| --- |
| //-----------------------------------------------------  // Testbench for the FIFO ---------------->  //-----------------------------------------------------  #include "systemc.h"  #include "FIFOParamExtended.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\_core::sc\_report\_handler::set\_actions( "/IEEE\_Std\_1666/deprecated", sc\_core::SC\_DO\_NOTHING );  sc\_clock clk("clk", 4, SC\_NS);  sc\_signal<bool> sreset\_n;  sc\_signal<char> data\_in;  sc\_signal<bool> push;  sc\_signal<bool> pop;  sc\_signal<char> data\_out;  sc\_signal<bool> empty;  sc\_signal<bool> full;  sc\_signal<bool> almost\_empty;  sc\_signal<bool> almost\_full;  // Connect the DUT  FIFOParamExtended<char> test\_FIFO("test\_FIFO", 3, 1);  test\_FIFO.clk(clk);  test\_FIFO.sreset\_n(sreset\_n);  test\_FIFO.data\_in(data\_in);  test\_FIFO.push(push);  test\_FIFO.pop(pop);  test\_FIFO.data\_out(data\_out);  test\_FIFO.empty(empty);  test\_FIFO.full(full);  test\_FIFO.almost\_empty(almost\_empty);  test\_FIFO.almost\_full(almost\_full);  // Open VCD file  sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("FIFO\_waveform");  // Dump the desired signals  sc\_trace(wf, clk, "clk");  sc\_trace(wf, sreset\_n, "sreset\_n");  sc\_trace(wf, data\_in, "data\_in");  sc\_trace(wf, push, "push");  sc\_trace(wf, pop, "pop");  sc\_trace(wf, data\_out, "data\_out");  sc\_trace(wf, empty, "empty");  sc\_trace(wf, full, "full");  sc\_trace(wf, almost\_empty, "almost\_empty");  sc\_trace(wf, almost\_full, "almost\_full");  sreset\_n = 0; // Assert the reset  push=0;  pop=0;  cout << "@" << sc\_time\_stamp() << " Asserting reset\n" << endl;  sc\_start(6, SC\_NS);  //6ns  assert(empty.read() == 1);  assert(full.read() == 0);  sreset\_n = 1; // De-assert the reset  cout << "@" << sc\_time\_stamp() << " De-Asserting reset\n" << endl;  data\_in='a';  push=1;  sc\_start(4, SC\_NS);  //10ns  assert(empty.read() == 0);  assert(almost\_empty.read() == 1);  push=0;  data\_in=0;  pop=1;  sc\_start(4, SC\_NS);  //14ns  assert(empty.read() == 1);  assert(almost\_empty.read() == 1);  assert(data\_out.read() == 'a');  data\_in='2';  pop=0;  push=1;  sc\_start(4, SC\_NS);  //18ns  assert(empty.read() == 0);  assert(almost\_empty.read() == 1);  pop=1;  data\_in='f';  sc\_start(4, SC\_NS);  //22ns  assert(data\_out.read() == '2');  assert(almost\_empty.read() == 1);  pop=0;  data\_in='a';  sc\_start(4, SC\_NS);  //26ns  assert(almost\_full.read() == 1);  data\_in='z';  sc\_start(4, SC\_NS);  //30ns  assert(full.read() == 1);  assert(almost\_full.read() == 1);  pop=1;  push=0;  sc\_start(4, SC\_NS);  //34ns  assert(data\_out.read() == 'f');  assert(almost\_full.read() == 1);  sc\_start(4, SC\_NS);  //38ns  assert(data\_out.read() == 'a');  assert(almost\_empty.read() == 1);  sc\_start(4, SC\_NS);  //42ns  assert(data\_out.read() == 'z');  assert(empty.read() == 1);  assert(almost\_empty.read() == 1);  sc\_start(4, SC\_NS);  //46ns  sreset\_n = 0;  sc\_start(17, SC\_NS);  cout << "@" << sc\_time\_stamp() << " Asserting reset\n" << endl;  cout << "@" << sc\_time\_stamp() << " Terminating simulation\n" << endl;  sc\_close\_vcd\_trace\_file(wf);  return 0; // Terminate simulation  } |

Для проверки корректности работы FIFO был создан тест.

**Полученные результаты моделирования:**

Сообщения из консоли:

|  |
| --- |
| SystemC 2.3.1-Accellera --- Feb 18 2017 01:17:28  Copyright (c) 1996-2014 by all Contributors,  ALL RIGHTS RESERVED  @0 s Asserting reset  Info: (I702) default timescale unit used for tracing: 1 ps (FIFO\_waveform.vcd)  @6 ns De-Asserting reset  8 ns Write data = a; push\_pointer change from 0 to 1  12 ns Read data = a; pop\_pointer change from 0 to 1  16 ns Write data = 2; push\_pointer change from 1 to 2  20 ns Read data = 2; pop\_pointer change from 1 to 2  20 ns Write data = f; push\_pointer change from 2 to 0  24 ns Write data = a; push\_pointer change from 0 to 1  28 ns Write data = z; push\_pointer change from 1 to 2  32 ns Read data = f; pop\_pointer change from 2 to 0  36 ns Read data = a; pop\_pointer change from 0 to 1  40 ns Read data = z; pop\_pointer change from 1 to 2  @63 ns Asserting reset  @63 ns Terminating simulation  Для продолжения нажмите любую клавишу . . . |

Результаты моделирования:

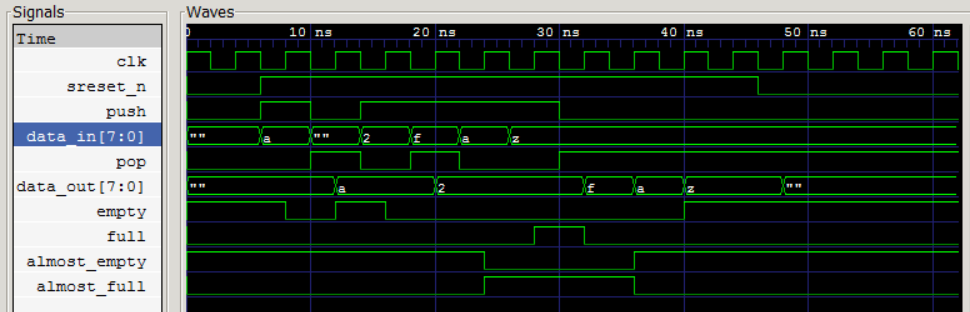


Рис 3.1. Результаты моделирования конечного автомата

Результаты моделирования соответствуют ожидаемым результатам, устройство работает корректно.