Санкт-Петербургский государственный политехнический университет
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе

Дисциплина: Высокоуровневое моделирование средствами SystemC

Тема: Параметризированное FIFO на языке SystemC.

Выполнил студент гр. 13541/2 _____ В.Е. Бушин
(подпись)
Руководитель _____ О.В. Мамутова
(подпись)

“_” _____2017 г.

Санкт-Петербург
2017

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. Было разработано FIFO, работающее с 8-разрядными словами. В конструкторе задаётся размер FIFO. Реализация данного FIFO находится в файлах fifo.h и fifo.cpp.
   Содержимое файла fifo.h:

```
#include "systemc.h"

#ifndef FIFO_H
#define      FIFO_H
SC_MODULE(fifo){
       sc_in_clk clk;
       sc_in<bool> sreset_n;
       sc_in<sc_uint<8> > data_in;
       sc_in<bool> push;
       sc_in<bool> pop;
       sc_out<sc_uint<8> > data_out;
       sc_out<bool> empty;
       sc_out<bool> full;
```

```
        sc_signal<bool> is_read;
        sc_signal<bool> is_write;

        int max;
        unsigned int *data;
        int num_elements, first;

        void processing();
        void read();
        void write();
        void flags();

        SC_HAS_PROCESS(fifo);

        fifo(sc_module_name name, int _size) :
        clk("clk"),
                sreset_n("reset"),
                data_in("data_in"),
                push("push"),
                pop("pop"),
                data_out("data_out"),
                empty("empty"),
                full("full"),
                max(_size)
        {
                data = new unsigned int [max];
                cout << "Executing new" << endl;
                SC_CTHREAD(processing, clk.pos());
                reset_signal_is(sreset_n,true);
                SC_METHOD(flags);
                sensitive << is_read << is_write << clk;
        }

};
#endif /*FIFO_H*/
```

Содержимое файла fifo.cpp:

```
#include "fifo.h"

void fifo::processing()
{
        is_read = false;
        is_write = false;
        data_out = 0;
        first = 0;
        num_elements = 0;
        for(int i = 0; i<=max; i++)
                data[i] = 0;
        wait();
        while(true){
                if(num_elements == max)
                        read();
                else if(num_elements == 0)
                        write();
                else
                {
                        read();
                        write();
                }
                wait();
        }
}
```

```
void fifo::read()
{
	if(pop)
	{
		is_read = !is_read;
		data_out = data[first];
		--num_elements;
		first = (first + 1) % max;
	}
}

void fifo::write()
{
	if(push)
	{
		is_write = !is_write;
		data[(first + num_elements) % max] = data_in.read();
		++num_elements;
	}
}

void fifo::flags()
{
	if(num_elements == max)
		full = true;
	else if(num_elements == 0)
		empty = true;
	else
	{
		full = false;
		empty = false;
	}
}
```

Для данной реализации FIFO был разработан следующий тест:

```
int sc_main(int argc, char* argv[]) {
	sc_clock clk("clock", 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;
	sc_signal<bool> almost_empty;
	sc_signal<bool> almost_full;

	fifo test("test", 6);
	test.clk(clk);
	test.sreset_n(sreset_n);
	test.data_in(data_in);
	test.push(push);
	test.pop(pop);
	test.data_out(data_out);
	test.empty(empty);
	test.full(full);

	// Open VCD file
	sc_trace_file *wf = sc_create_vcd_trace_file("fifo_waveform");
	// Dump the desired signals
	sc_trace(wf, clk, "clock");
	sc_trace(wf, sreset_n, "reset");
	sc_trace(wf, data_in, "in");
```

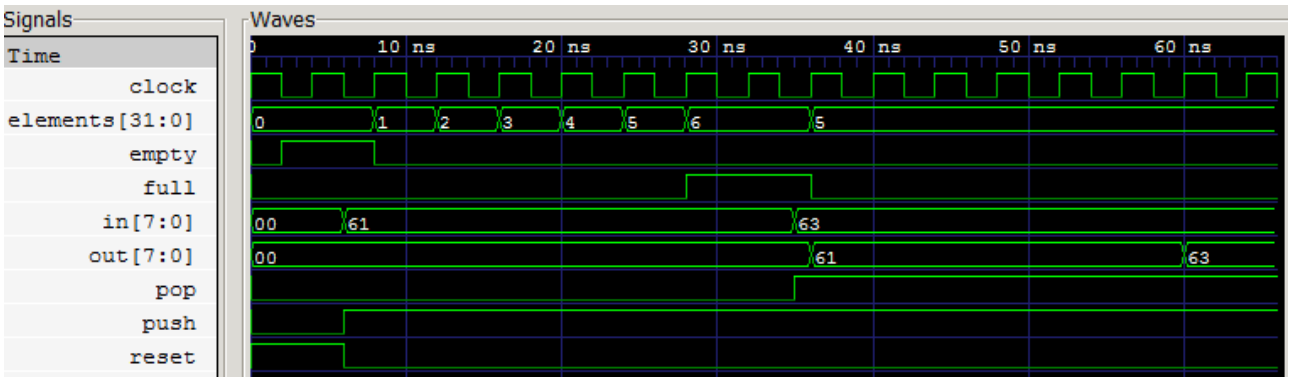
```
        sc_trace(wf, push, "push");
        sc_trace(wf, pop, "pop");
        sc_trace(wf, data_out, "out");
        sc_trace(wf, empty, "empty");
        sc_trace(wf, full, "full");
        sc_trace(wf, test.num_elements, "elements");
        sc_trace(wf, test.is_write, "is_wr");

        sreset_n = 1; // Assert the reset
        cout << "@" << sc_time_stamp() << " Asserting reset\n" << endl;
        sc_start(6, SC_NS);

        sreset_n = 0; // De-assert the reset
        cout << "@" << sc_time_stamp() << " De-Asserting reset\n" << endl;
        data_in = 'a';
        push = true;
        sc_start(29, SC_NS);

        data_in = 'c';
        pop = true;
        sc_start(33, SC_NS);
        return 0;
}
```

Открыв сгенерированный vcd файл в GTKWave, мы увидим корректные результаты работы устройства:



2. На основе созданного раннее класса был реализован шаблонный класс FIFOParam, в котором параметром выступает тип данных хранящихся в FIFO. Реализация данного класса находится в файле fifoparam.h:

```
#include <systemc.h>

#ifndef FIFOPARAM_H
#define       FIFOPARAM_H

template <typename T>
SC_MODULE(fifoparam){
public:
        sc_in_clk clk;
        sc_in<bool> sreset_n;
        sc_in<T> data_in;
        sc_in<bool> push;
        sc_in<bool> pop;
        sc_out<T> data_out;
        sc_out<bool> empty;
        sc_out<bool> full;
```

```
	sc_signal<bool> is_write;
	sc_signal<bool> is_read;

	int max;
	T *data;
	int num_elements, first;

	void processing();
	void read();
	void write();
	void flags();

	SC_HAS_PROCESS(fifoparam);

	//SC_CTOR(fifoparam):
	fifoparam(sc_module_name _name, int _size):
	clk("clk"),
		sreset_n("reset"),
		data_in("data_in"),
		push("push"),
		pop("pop"),
		data_out("data_out"),
		empty("empty"),
		full("full"),
		max(_size)
	{
		data = new T[max];
		cout << "Executing new" << endl;
		SC_CTHREAD(processing, clk.pos());
		reset_signal_is(sreset_n,true);
		SC_METHOD(flags);
		sensitive << is_read << is_write << clk;
	}

};

template <typename T>
void fifoparam<T>::processing()
{
	is_read = false;
	is_write = false;
	data_out = 0;
	first = 0;
	num_elements = 0;
	for(int i = 0; i<=max; i++)
		data[i] = 0;
	wait();
	while(true){
		if(num_elements == max)
			read();
		else if(num_elements == 0)
			write();
		else{
			read();
			write();
		}
		wait();
	}
}

template <typename T>
void fifoparam<T>::read()
{
	if(pop)
```

```
	{
		is_read = !is_read;
		data_out = data[first];
		--num_elements;
		first = (first + 1) % max;
	}
}

template <typename T>
void fifoparam<T>::write()
{
	if(push)
	{
		is_write = !is_write;
		data[(first + num_elements) % max] = data_in.read();
		++num_elements;
	}
}

template <typename T>
void fifoparam<T>::flags()
{
	if(num_elements == max)
		full = true;
	else if(num_elements == 0)
		empty = true;
	else
	{
		full = false;
		empty = false;
	}
}
#endif /*FIFO_H*/
```

Для данной реализации FIFO был разработан следующий тест:

```
int sc_main(int argc, char* argv[]) {
	sc_clock clk("clock", 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;

	fifoparam<char> test("test", 7);
	test.clk(clk);
	test.sreset_n(sreset_n);
	test.data_in(data_in);
	test.push(push);
	test.pop(pop);
	test.data_out(data_out);
	test.empty(empty);
	test.full(full);

	// Open VCD file
	sc_trace_file *wf = sc_create_vcd_trace_file("fifo_waveform");
	// Dump the desired signals
	sc_trace(wf, clk, "clock");
	sc_trace(wf, sreset_n, "reset");
	sc_trace(wf, data_in, "in");
```

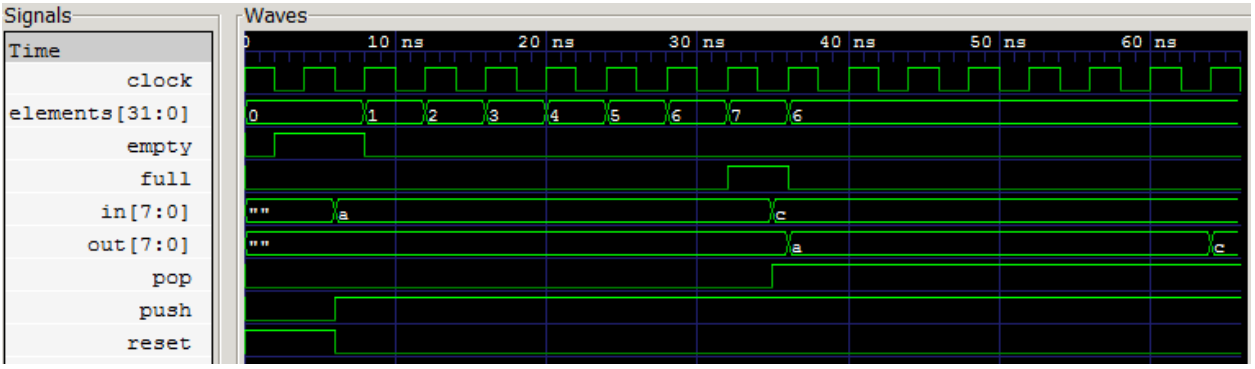
```
        sc_trace(wf, push, "push");
        sc_trace(wf, pop, "pop");
        sc_trace(wf, data_out, "out");
        sc_trace(wf, empty, "empty");
        sc_trace(wf, full, "full");
        sc_trace(wf, test.num_elements, "elements");
        sc_trace(wf, test.is_write, "is_wr");

        sreset_n = 1; // Assert the reset
        cout << "@" << sc_time_stamp() << " Asserting reset\n" << endl;
        sc_start(6, SC_NS);

        sreset_n = 0; // De-assert the reset
        cout << "@" << sc_time_stamp() << " De-Asserting reset\n" << endl;
        data_in = 'a';
        push = true;
        sc_start(29, SC_NS);

        data_in = 'c';
        pop = true;
        sc_start(33, SC_NS);
        return 0;
}
```

Открыв сгенерированный vcd файл в GTKWave, мы увидим корректные результаты работы устройства:



3. Далее от созданного шаблонного класса FIFOParam был унаследован класс FIFOParamExtended с 2 дополнительными выходами:
   - almost_empty;
   - almost_full.

   Реализация класса FIFOParamExtended находится в файле fifoparamextend.h:

```
#include <systemc.h>
#include "fifoparam.h"

#ifndef FIFOPARAMEXTEND_H
#define       FIFOPARAMEXTEND_H

template <typename T>
struct fifoparamextend : public fifoparam<T> {
public:
        sc_out<bool> almost_empty;
        sc_out<bool> almost_full;
```

```
        unsigned int almost;

        void almostfe();

        SC_HAS_PROCESS(fifoparamextend);

        fifoparamextend(sc_module_name name, int _size, unsigned int _almost):
                fifoparam<T>(name, _size),
                almost(_almost)
        {
                cout << "Executing new" << endl;
                SC_METHOD(almostfe);
                sc_module::sensitive << fifoparam<T>::is_write << is_read << clk;
        }
};

template <typename T>
void fifoparamextend<T>::almostfe()
{
        if(fifoparam<T>::num_elements <= almost)
                almost_empty = true;
        else almost_empty = false;
        if(fifoparam<T>::num_elements >= fifoparam<T>::max - almost)
                almost_full = true;
        else almost_full = false;
}
#endif /* FIFOPARAMEXTEND_H */
```

Для данной реализации FIFO был разработан следующий тест:

```
int sc_main(int argc, char* argv[]) {
        sc_clock clk("clock", 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;

        fifoparamextend<char> test("test", 7, 2);
        test.clk(clk);
        test.sreset_n(sreset_n);
        test.data_in(data_in);
        test.push(push);
        test.pop(pop);
        test.data_out(data_out);
        test.empty(empty);
        test.full(full);
        test.almost_empty(almost_empty);
        test.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, "clock");
        sc_trace(wf, sreset_n, "reset");
        sc_trace(wf, data_in, "in");
        sc_trace(wf, push, "push");
        sc_trace(wf, pop, "pop");
        sc_trace(wf, data_out, "out");
        sc_trace(wf, empty, "empty");
```

```
        sc_trace(wf, full, "full");
        sc_trace(wf, test.num_elements, "elements");
        sc_trace(wf, test.is_write, "is_wr");
        sc_trace(wf, almost_empty, "alm_empty");
        sc_trace(wf, almost_full, "alm_full");

        sreset_n = 1; // Assert the reset
        cout << "@" << sc_time_stamp() << " Asserting reset\n" << endl;
        sc_start(6, SC_NS);

        sreset_n = 0; // De-assert the reset
        cout << "@" << sc_time_stamp() << " De-Asserting reset\n" << endl;
        data_in = 'a';
        push = true;
        sc_start(29, SC_NS);

        data_in = 'c';
        pop = true;
        sc_start(33, SC_NS);
        return 0;
}
```

Открыв сгенерированный vcd файл в GTKWave, мы увидим корректные результаты работы устройства: