1.9 SystemC модели за синтезу

Сви модели развијани у SystemC језику могу да се симулирају. Ипак, то није једина намена SystemC модела. Уколико су модели развијени на одговарајући начин, они могу да се аутоматски синтетизују, и на тај начин да се на основу модела добије резултат намењен имплементацији. Модели намењени синтези могу се развијати на два нивоа абстракције: RTL и HLS.

Модели намењени синтези често садрже процесе `SC_CTHREAD` врсте. Ова врста процеса је слична `SC_THERAD` процесима и често се користи у HLS синтези. Коришћењем ове врсте процеса добија се модел који је у потпуности синхрон. Додатно, ова врста процеса додаје неке могућности које олакшавају моделовање.

Процес `SC_CTHREAD` врсте региструју се на следећи начин:

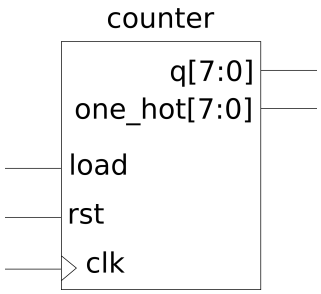
```
SC_CTHREAD(proc_name, clk.edge()) ;
```

Уместо `edge`, треба да стоји `pos` или `neg`. На овај начин добија се процес који је осетљив на сваку промену одговарајуће ивице синхроног сигнала. Додатно унутар ове врсте процеса, могу се користити додатне варијанте `wait` метода. Метода `wait(N)` омогућава да се извршавање процеса одложи `N` циклуса.

За моделовање комбинациних делова система, на располагању је `SC_METHOD` процес. Да би се добила комбинациона мрежа, потребно је да овај процес буде осетљив на све улазне сигнале мреже која се моделује.

1.9.1 Синтеза приступа и сигнала

Модул `counter` представља једноставан бројач (слика 1.28). Када је улаз `en` један, излаз `q`, повећава се за 1. Излаз `one_hot` предсавља "one hot" код нижа три бита излаза `q`.



Слика 1.28: Једноставан бројач

Бројач је моделован као компонента `counter` (листинг 1.24). Улази и излази компоненте реализовани су помоћу приступа. Компонента садржи и два прцеса, од којих један имплементира секвенцијални део бројача, а други илуструје како се имплементира комбинациони део дигиталног кола.

Листинг 1.24: Заглавље модула `counter`

```
#ifndef _COUNTER_HPP_
#define _COUNTER_HPP_

#include <systemc>

SC_MODULE(counter)
{
public:
        sc_core::sc_in<bool> clk;
        sc_core::sc_in<bool> rst;
        sc_core::sc_in<bool> en;
        sc_core::sc_out< sc_dt::sc_uint<8> > q;
        sc_core::sc_out< sc_dt::sc_lv<8> > one_hot;

        counter(sc_core::sc_module_name);
protected:
        sc_core::sc_signal< sc_dt::sc_uint<8> > val;
        void count();
        void one_hot_comb();
};

#endif
```

Процес `count` имплементира главну логику бројача (листинг 1.25). Пре бесконачне петље, унутар овог процеса, налази се део намењен иницијализацији. HLS алати интерпретирају ово као вредности које ће систем добити након ресетовања. Унутар бесконачне петље налати се главна логика. Методом `wait` означен је део модела који треба да се изврши када год се активира синхронизациони сигнал.

Листинг 1.25: Имплементација модула `counter`

```
#include "counter.hpp"

using namespace sc_core;
using namespace sc_dt;

SC_HAS_PROCESS(counter);

counter::counter(sc_module_name name) : sc_module(name)
{
        //SC_METHOD(count);
        //sensitive << clk.pos();
        SC_CTHREAD(count, clk.pos());
        reset_signal_is(rst,true);
        SC_METHOD(one_hot_comb);
        sensitive << val;
}

void counter::count()
{
        sc_uint<8> cnt;
        cnt = 0;
        wait();

        while(true)
        {
                if (en.read()) cnt++;
                q->write(cnt);
                val.write(cnt);
                wait();
        }
}
```

```
void counter::one_hot_comb()
{
        sc_lv<8> tmp(0);
        sc_uint<8> ind(val.read());
        tmp[ ind & 0x7 ] = 1;
        one_hot->write(tmp);
}
```

Процес `one_hot_comb` имплементира пребацивање бинарног кода у "one hot". HLS алат интерпретираће овај процес као комбинациону мрежу. Овај процес активираће се на сваку промену сигнала `val`. Као и код HDL језика, процес који се активира на све промене сигнала који имају утицај на сигнале које тај процес мења, биће интерпретиран од алата за синтезу као комбинациона мрежа.

Алат Vivado HLS може да направи Intellectual Property (IP) блок од овог SystemC модела (листинг 1.26).

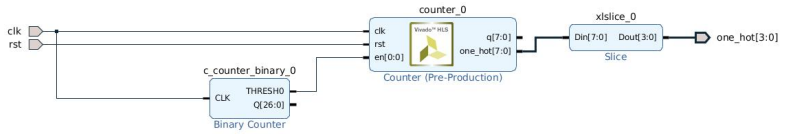
Листинг 1.26: Скрипта за имплементацију

```
set root ./
open_project counter_ip
set_top counter
add_files [glob "$root/counter.cpp"]
add_files -tb [glob "$root/main_counter.cpp"] -cflags -I$root/

# Solution settings
open_solution "solution1"
# Zedboard
# set_part {xc7z020clg484-1} -tool vivado
# Zybo
set_part {xc7z010clg400-1} -tool vivado
create_clock -period 10 -name default
#source "./hls_ip/solution1/directives.tcl"

# Command for tools
csim_design -compiler gcc
csynth_design
# cosim_design
export_design -format ip_catalog
```

Помоћу Vivado алата и IP Integratora може се моделовати једноставан систем који користи овај једноставан IP (слика 1.29). У случају да се користи Zybo плоча за имплементацију, на располагању стоје 4 диоде. На те диоде могу се послати вредности са неког од излаза, на пример са `one_hot`. Уколико је потребно да се промене дешавају сваке секунде, у систем је потербно убацити блок који ће да успори бројач и да шаље вредност 1 на улаз `en` сваку секунду.



Слика 1.29: Једноставан систем

IP блокове из репозиторијума фирме Xilinx, треба подесити тако да се промене у систему дешавају сваке секунде. Претпоставка је да

је улазна фреквенција синхронизационог сигнала 100 Mhz. Додатно, излазу `one_hot` потребно је смањити ширину са 8 бита на 4. Улаз `rst` у систем може се повезати на неки од прекидача на развојној плочи. Уз додатну Xilinx Design Constrains (XDC) датотеку (листинг 1.27), овај систем може се имплементирати и добијеном датотеком са екстензијом `bit` може се програмирати развојна плоча.

Листинг 1.27: Ограничења система

```
set_property PACKAGE_PIN L16 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -add -name clk -period 10.00 [get_ports clk];

set_property -dict { PACKAGE_PIN M14 IOSTANDARD LVCMOS33 } [get_ports { one_hot[0] }];
set_property -dict { PACKAGE_PIN M15 IOSTANDARD LVCMOS33 } [get_ports { one_hot[1] }];
set_property -dict { PACKAGE_PIN G14 IOSTANDARD LVCMOS33 } [get_ports { one_hot[2] }];
set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 } [get_ports { one_hot[3] }];

set_property -dict { PACKAGE_PIN G15 IOSTANDARD LVCMOS33 } [get_ports { rst }];
```

1.9.2 Синтеза других примитивних канала

Компонента `simfifo` илуструје како се може користи FIFO канал за моделе намењене синтези (листинг 1.28). Модул садржи улазни (`fin`) и излазни (`fout`) интерфејс ка FIFO каналу а додатно садржи и једну инстанцу канала (`fifo_int`). Два `SC_CTHREAD` процеса користе унутрашњи канал за комуникацију.

Листинг 1.28: Заглавље модула `simfifo`

```
#ifndef _SIMFIFO_H_
#define _SIMFIFO_H_

#include <systemc>

class simfifo :
        public sc_core::sc_module
{
public:
        sc_core::sc_in<bool> clk;
        sc_core::sc_in<bool> rst;
        sc_core::sc_in<bool> en;

        typedef sc_dt::sc_lv<4> data_t;

        sc_core::sc_fifo_in< data_t > fin;
        sc_core::sc_fifo_out< data_t > fout;
        sc_core::sc_out< data_t > dout;

        simfifo(sc_core::sc_module_name);
protected:
        sc_core::sc_fifo< data_t > fifo_int;
        void input_cthread();
        void output_cthread();
};

#endif
```

Процес `input_cthread` прима податке преко улазног FIFO интерфејса и прослеђује их у унутрашни FIFO канал (листинг 1.29). Процес `output_cthread` пролази кроз три фазе. У првој шаље неколико података на излазни FIFO интерфејс. У другој фази, прима податак

са унутрашњег канала и избацује тај податак на излазни интерфејс, као и на излазни FIFO интерфејс. У трећој фази рачуна просечну вредност бројева, које је добио са унутрашњег канала и ту вредност шаље на излазни интерфејс. Потом се фазе два и три понављају.

Листинг 1.29: Имплементација модула `simfifo`

```
#include "simfifo.hpp"
#include <string>

SC_HAS_PROCESS(simfifo);

using namespace sc_dt;
using namespace sc_core;
using namespace std;

simfifo::simfifo(sc_module_name name) :
        sc_module(name),
        fifo_int(4)
{
        SC_CTHREAD(input_cthread, clk.pos());
        reset_signal_is(rst,true);
        SC_CTHREAD(output_cthread, clk.pos());
        reset_signal_is(rst,true);
}

void simfifo::input_cthread()
{
        data_t val;
        while(true)
        {
                do wait(); while(en->read() != 1);

                if(fin->num_available() != 0)
                {
                        fin->nb_read(val);
                        // string info = "FIFO = " + val.to_string() +
                        //       " written @ " + sc_time_stamp().to_string();
                        // SC_REPORT_INFO(name(), info.c_str());
                        fifo_int.nb_write( val );
                }
        }
}

void simfifo::output_cthread()
{
        sc_uint<12> avg = 0;
        sc_uint<4> cnt = 0;
        data_t val;

        dout->write(15);

        while(true)
        {
                for (int i = 0; i < 4; ++i)
                {
                        fout->nb_write(4 + i);
                        wait();
                }

                while(true)
                {
                        do wait(); while(en->read() != 1);

                        if (fifo_int.num_available() != 0)
                        {
                                fifo_int.nb_read(val);
                                sc_uint<4> tmp = static_cast< data_t > (val);
                                avg = (avg * cnt) + tmp;
                                cnt++;
                                fout->nb_write( val );
                                dout->write( val );
                                // string info = "Val = " + val.to_string() +
                                //       " written @ " + sc_time_stamp().to_string();
                                // SC_REPORT_INFO(name(), info.c_str());

                                do wait(); while(en->read() != 1);
                                if (cnt == 0)
                                        avg = val;
```

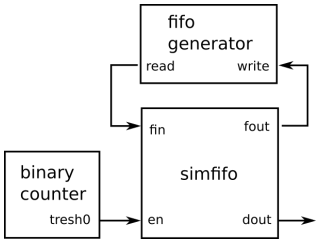
```
                                        else
                                                avg /= cnt;
                                        dout->write( avg );
                                        // string info2 = "Avg = " + avg.to_string() +
                                        //         " written @ " + sc_time_stamp().to_string();
                                        // SC_REPORT_INFO(name(), info2.c_str());
                                }
                        }
                }
}
```

Компонента `simfifo` може да се синтетише коришћењем Vivado HLS алата. Од унутрашњег канала биће направљен FIFO модул. Од улазних и излазних интерфејса настаће Xilinx стандардни интерфејси за комуникацију са FIFO модулима.

1.9.3 Вежбе

1. Имплементирати систем са једноставним бројачем (слика 1.29) коришћењем Vivado Integratora. Покренути имплементацију и спустити добијену `bit` датотеку на развојну плочу.

2. Имплементирати `simfifo` модул помоћу Vivado HLS алата. Потом, искористити добијени IP блок за интеграцију у систем (слика 1.30). Покренути имплементацију и спустити добијену `bit` датотеку на развојну плочу.



Слика 1.30: Систем са модулаом `simfifo`