1.4 Симулационо време

Приликом системског моделовања, време има суштинску улогу. Да би се омогућило системског моделовање било је потребно додати концепт времена у C++ језик. У наредним поглављима биће описано како се то постиже. За сада, биће описана класа која служи за манипулацију временом у оквиру језика SystemC, `sc_time`.

1.4.1 Манипулација временом

Класа `sc_time` има већи број конструктора од којих су два:

```
sc_core::sc_time var_name(double, sc_core::sc_time_unit)
sc_core::sc_time var_name;
```

Конструктор са два параметра очекује величину времена, као први параметар, изражену у јединицама које се прослеђују као други параметар. Јединице времена могу бити следеће:

- SC_SEC - секунде
- SC_MS - милисекунде
- SC_US - микросекунде
- SC_NS - наносекунде
- SC_PS - пикосекунде
- SC_FS - фемптосекунде

Конструктор без параметара прави време од 0 секунди.

Многи оператори су преклопљени за класу `sc_time`. Временске тренутке је могуће сабирати, одузимати, поредити... (листинг 1.12). Додатно, пријатељске функције за слање на стандардни излаз су имплементиране.

Листинг 1.12: Манипулација временом

```
#include <systemc>
#include <iostream>

int sc_main(int argc, char* argv[])
{
        sc_core::sc_time t1;
        sc_core::sc_time t2(10000, sc_core::SC_PS);
```

```
        std::cout << "t1 = " << t1 << std::endl;
        std::cout << "t2 = " << t2 << std::endl;

        sc_core::sc_time period(5, sc_core::SC_NS);
        sc_core::sc_time delay(4000, sc_core::SC_PS);
        sc_core::sc_time hold(1.2, sc_core::SC_NS);

        sc_core::sc_time skew = period - delay;
        if (skew < hold)
                std::cout <<
                        "Time violation: skew = " <<
                        skew << std::endl;

        std::cout
                << "Constant SC_ZERO_TIME = "
                << sc_core::SC_ZERO_TIME <<  std::endl;

        sc_start(60, sc_core::SC_NS);

        std::cout
                << "Current sim time = "
                << sc_core::sc_time_stamp()
                << std::endl;

    return 0;
}
```

У SystemC језику постоји предефинисана константа `SC_ZERO_TIME` која је једнака времену од 0 секунди, `sc_time(0, SC_SEC)`.

Функција која стартује SystemC симулацију је `sc_start` (листинг 1.12). Као опциони параметар ова функција може да прими максимално време за које ће симулација бити покренута. На пример, позив `sc_start(60, sc_core::SC_NS)`, би покренуо SystemC симулацију за највише 60 ns. Може се десити и да се симулација заврши пре овог наведеног времена.

Понекад је потребно приказати тренутно симулационо време. Да би се ово постигло, може се користити функција `sc_time_stamp`. Ова функција нема улазне параметре, а као излаз враћа тренутно време симулације.

1.4.2 Вежбе

1. Написати SystemC симулацију која ће у случајном тренутку времена, у интервалу од 10 ns до 100 ns у корацима од 1 ns, исписати поруку. Порука може бити произвољна и треба да садржи у себи тренутно симулационо време.

2. Написати SystemC симулацију која ће садржати два случајна времеснак интервала, од 1 ns до 10 ns у корацима од 1 ns. Симулација треба да траје временски интервал једнак збиру два случајна интервала. На крају, потребно је исписати поруку која приказује два случајна временска интервала, као и тренутно симулационо време.

1.5 Модули

Комплексни електронски системи се састоје од великог броја компнената. Те компоненте могу бити софтверске, хардверске или механичке. У SystemC језику компоненте система називају се модули. Дефинисање модула постиже се одговарајућим техникама C++ језика.

1.5.1 Дефинисање модула

Базна класа свих SystemC компоненти је `sc_module`. Ова класа дефинисана је у простору имена `sc_core`. Да би се користиле компоненете потребно је наследити ову класи. Макро `SC_MODULE` јавно наслеђује ову класу. Једна од могућих имеплементација овог макроа могла би да буде:

```
#define SC_MODULE(user_module_name) \
    struct user_module_name : ::sc_core::sc_module
```

Модул се може дефинисати експлицитним наслеђивањем класе `sc_module` или коришћењем макроа `SC_MODULE`. Кодни фрагмент који илуструје два начина дефинисања модула је дат.

```
class module_name : public sc_core::sc_module
{

};

SC_MODULE(module_name)
{

};
```

Базна класа има велики број јавних метода. Ова класа садржи и име компоненте које је класе `sc_module_name`. Приликом конструкције, класа које наслеђују `sc_module`, параметар имена је једини обавезан параметар. Ово име је јединствено за сваку компоненту и може се видети на таласним облицима у симулаторима који имају могућност приказивања таласних облика.

1.5.2 Конструкција модула

Језик SystemC омогућава конкурентно извршавање метода у језику C++. Да би се метода прогласила конкурентном потребно је:

- Регистровати класу да у њој постоје конкурентне методеа
- Означити, односно регистровати, методу да је конкурента

Постоје два макроа која региструју класу за конкурентне методе.

- SC_CTOR
- SC_HAS_PROCESS

Макро `SC_CTOR` региструје класу за коришћење конкурентних метода и дефинише/декларише конструктор са обавезним параметром имена `sc_module_name`. Једна могућа имплементација овог макроа би могла бити:

```
#define SC_CTOR(user_module_name) \
    typedef user_module_name SC_CURRENT_USER_MODULE; \
    user_module_name( ::sc_core::sc_module_name )
```

У овој имплементацији класа се региструје са линијом која почиње са кључном речју `typedef`. Након тога је написан почетак дефиниције/декларације конструктора. Овај макро може се користити уколико постоји конструктор са само једним параметром, именом. У супротном користи се макро `SC_HAS_PROCESS`.

Макро `SC_HAS_PROCESS` региструје класу за конкурентне наредбе и ништа додатно. Могућа имплементација овог макроа могла би да буде:

```
#define SC_HAS_PROCESS(user_module_name) \
    typedef user_module_name SC_CURRENT_USER_MODULE
```

Макрои `SC_CTOR` и `SC_HAS_PROCESS` су узајамно искључиви.

1.5.3 SystemC процеси

Конкурентне методе у SystemC језику се називају још и SystemC процеси. Прототип SystemC процеса је

```
void process_name(void);
```

SystemC процес не прима аргументе и нема повратну вредност. Метода се може прогласити SystemC процесом коришћењем следећих макроа:

- SC_THREAD
- SC_METHOD
- SC_CTHREAD

Метода се мора прогласити са SystemC процес унутар конструктора. Сваки од наведених макроа чини методу нешто другачијим SystemC процесом. Разлике између ових процеса биће објашњене у наредним поглављима. До тада ће бити коришћен само `SC_THREAD` процес. Методу проглашавамо процесом на следећи начин:

```
SC_THREAD(my_method);
```

1.5.4 SystemC wait функција

Често је потребно зауставити извршавање SystemC процеса неки временски период. На пример, на овај начин се може симулирати кашњење дигиталних блокова. Функција `wait` користи се за моделовање кашењења процеса. Када се употреби функција `wait` унутар `SC_THREAD` процеса, тада ова врста процеса блокира своје извршаваање за време наведено као параметар wait функције. На пример, да би се извршавање процеса блокирало 5 ns, може се позвати `wait(5, sc_core::SC_NS);`

1.5.5 Пример

У првом примеру дефинисан је модул `Module01`, са једним конструктором и једним процесом (листинг 1.13).

Листинг 1.13: Једноставан модул

```
#include <systemc>
#include <iostream>

SC_MODULE(Module01)
{
public:
        SC_CTOR(Module01)
        {
                SC_THREAD(main_method);
        }

        void main_method(void)
        {
                wait(5, sc_core::SC_NS);
                std::cout <<
                        sc_core::sc_time_stamp() <<
                        ": Message from \"" <<
```

```
                        name() <<
                        "\"␣module.\n";
        }
};

int sc_main(int argc, char* argv[])
{
        Module01 uut ("Inst_Module01");

        sc_start(10, sc_core::SC_NS);
        return 0;
}
```

У модулу `Module01` унутар конструктора дефинисаног макроом `SC_CTOR` регистрован је један процес `main_method`. Овај процес чека одређено време и потом исписује поруку која садржи и име са којим је модул инстанциран. За испис имена коришћена је метода `name` базне класе `sc_module`.

У главном програму инстанцирана је једна компонента овог модула и покренута је симулација.

У другом примеру имплементиран је модул са два процеса. Један процес генерише случајне бројеве, а други процес их прима и приказује. Овај модул имплементиран је класом `Module02` (листинг 1.14).

Листинг 1.14: Заглавље модула

```
#ifndef MODULE02_H
#define MODULE02_H

#include <systemc>
#include <deque>

SC_MODULE(Module02)
{
public:
        SC_HAS_PROCESS(Module02);

        Module02(sc_core::sc_module_name name,
                 int limit = 10);

        void generate();
        void monitor();
protected:
        std::deque<int> ints;
        int limit;
};

#endif
```

Заглавље садржи заштитне директиве које спречавају вишеструко декларисање ове класе унутар других датотека. Када се имплементација класе раздвоји на две датотеке, као што је то урађено у овом случају, ова заштита је обавезна. Потом су укључена сва заглавља неопходна за декларацију ове класе.

Унутар класе `Module02` макроом `SC_HAS_PROCESS` означено је да ће ова класа имати процесе. Потом је декларисан један конструктор као и две методе. У заштићеном делу класе декларисане су променљиве које су коришћене приликом имплементација метода: `ints` и `limit`.

Имплементација класе `Module02` укључује заглавље са декларацијом модула (листинг 1.15). Потом је дефинисан конструктор, унутар кога су

регистрована два процеса `generate` и `monitor`. Пошто имплементације процеса користе случајне бројеве, `srand` функцијом иницијализован је генератор случајних вредности.

Листинг 1.15: Имплементација модула

```
#include "Module02.hpp"
#include <cstdlib>
#include <ctime>

using namespace sc_core;
using namespace std;

Module02::Module02(sc_core::sc_module_name name,
                                        int limit) :
        limit(limit)
{
        SC_THREAD(generate);
        SC_THREAD(monitor);
        srand(time(NULL));
}

void Module02::generate()
{
        while(true)
        {
                wait(1, SC_NS);
                int val = rand() % limit;
                ints.push_back(val);
        }
}

void Module02::monitor()
{
        for(;;)
        {
                wait(1500, SC_PS);
                while(ints.size() != 0)
                {
                        int num = ints.front();
                        ints.pop_front();
                        std::cout <<
                                "Number " << num <<
                                " at time " << sc_time_stamp() <<
                                ".\n";
                }
        }
}
```

Процес `generate` генерише случајне бројеве у опсегу од 0 до `limit`. Овај процес је имплементиран бесконачном петљом. Пре генерисања случајног броја, чека се неко време, а генерисани број се смешта у ред `ints`.

Процес `monitor` имплементиран је бесконачном петљом. Овај процес чека неко време, па провера да ли у реду `ints` постоје неке вредности. Уколико вредности постоје, процес их прикаже и уклања из реда.

У главном програму инстанцирана је компоненета `Module02` и покренута је симулација (листинг 1.16).

Листинг 1.16: Глани програм

```
#include <systemc>
#include "Module02.hpp"

int sc_main(int argc, char* argv[])
{
        Module02 uut("UUT");

        sc_start(10, sc_core::SC_NS);
```

```
        std::cout << "Simulation finished at " << sc_core::sc_time_stamp() << std::endl;
        return 0;
}
```

1.5.6 Вежбе

1. Моделовати модул који садржи два процеса. Први процес ствара кружну секвенцу првих 64 Фибоначијевих бројева у случајним временским интервалима. Нови број се ствара у интервалу од 10 ns до 100 ns. Други процес исписује број створених бројева сваке секунде. Симлуација треба да траје 20 секунди. За комуникацију између процеса користити ред или вектор.

2. Моделовати модул са пет процеса. Први процес представља лото 7 од 39 машину, који сваке секунде ствара комбинацију од 7 бројева. Следећа три процеса представљају играче лото игре. Играчи одлучују да ли ће у неком од извлачења учествовати у игри. Уколико учествују, праве своју комбинацију. Пети процес упоређује комбинације играча који су учествовали са комбинацијом машине и проглашава победнике. Победници су они играчи који имају највећи број погодака. Овај процес треба да испише симулационо време, победнике, као и њихове кобинације са назначеним погођеним бројевима.