1.6 SystemC процеси

SystemC има више врста процеса. Сви процеси се извршавају конкурентно. Процеси се могу налазити у три стања. Могу бити у стању извршавања, могу бити у стању чекања на неки догађај или могу бити у скупу RP. Врсте процеса су:

- SC_THREAD
- SC_METHOD
- SC_CTHREAD

У овом поглављу објаснићемо само `SC_THREAD` и `SC_METHOD` процесе. Да би објаснили `SC_CTHREAD` процес потребно је увести још неке концепте, па ће ова врста процеса бити објашњена у наредним поглављима.

1.6.1 SC_THREAD

Процеси `SC_THREAD` врсте стартују се само једном на почтку симулације. Када се процес стартовао, он има пуну контролу над симулацијом све до тренутка када се контрола врати кернелу симулатора.

Постоје два начина на која `SC_THREAD` враћа контролу симулатору. Први начин је командом `return`. Ова команда зауставља процес до краја симулације. Други начеин је позивом `wait` функције. Ова функција враћа контролу кернелу симулатора и пребацује процес у стање чекања.

Једном када се позове `return`, није више могуће поново покренути `SC_THREAD` процес. Стога, процес овог типа најчешће садрже бесконачну петљу која садржи барем један позив `wait` функције.

Променљиве које се декларишу у `SC_THREAD` прцесу су трајне. Због тога ова врста процеса често користи променљиве декларисане унутар самог процеса.

Ова врста процеса се покреће аутоматски на почеку симулације, осим ако се процес на означи методом `dont_initialize` приликом регистрације.

Алата за синтезу Register Transfer Level (RTL) модела, обично, не могу да синтетишу ову врсту процеса. Модели који користе `SC_THREAD` процесе могу да се синтетишу једино употребом High Level Synthesis (HLS) алата и то наравно коришћењем само одговарајућих конструкција. Више о овоме биће речи у неком од наредних поглавља.

1.6.2 SC_METHOD

Процеси SC_METHOD врсте не заустављају се у току свог извршавања. Позиви функције wait у оквиру SC_METHOD процеса нису дозвољени и проузрокују грешку током извршавања симулације. Ова врста процеса извршава се од почетка па до краја процеса или до позива наредбе return.

SC_METHOD процеси декларишу и иницијализују промењљиве сваки пут када се процес позове. Зато, ови процеси често користе локалне промењљиве које садржи компонента.

SC_METHOD процеси се декларишу, односно региструју, у оквиру конструктора на следећи начин:

```
SC_METHOD(process_name);
```

Кернел симулатора може позвати ову врсту процеса више пута у току симулације у зависности од њихове статичке или динамичке листе осетљивости. Статичка листа осетљивости конструише се одмах након декларисања SC_METHOD процеса у конструктору помоћу члана sensitive класе sc_module. Конструкција се може постићи на следећи начин:

```
SC_METHOD(process_name);
sensitive << event1 [<< event2 ...];
```

Код процеса са статичком листом осетљивости, сваки пут када се деси одговарајући догађај, процеси се пребацују у скуп RP. Када се одаберу за извршавање, извршавају се у целости и пребацују се у стање чекања на догађаје из статичке листе осетљивости.

Динамичка листа осетљивости се конструише методом next_trigger у току извравања самог процеса. Прототипови овог метода су:

```
next_trigger(time);
next_trigger(event);
next_trigger(time, event);
next_trigger();
```

Методом next_trigger са временским параметром обезбеђује се аутомаско покретање процеса након што истекне временски период. Када употребимо догађај као параметар, тада ће се SC_METHOD процесу променити листа осетљивости на догађаје које наведемо у аргумент листи. Ако се употребе и временски параметар time и догађај event,

тада ће процес чекати на догађај `event` највише временски период `time`. Уколико догађај не пошаље обавештење у задатом времену, процес ће се аутоматски покренути. Уколико употребимо ову методу без параметара, тада се за листу осетљивости враћа статичка листа осетљивости.

Ова врста процеса покреће се аутоматски на почеку симулације, осим ако се процес на маркира методом `dont_initialize` приликом регистрације.

Алати који могу да синтетишу SystemC моделе, `SC_METHOD` процесе, са статичком листом осетљивости посматрају као комбинационе мреже. Ова врста процеса користи се стога, приликом моделовања на RTL нивоу абстракције за моделе комбинационих мрежа. За RTL моделе са регистрима, алати обично очекују употребу `SC_CTHREAD` процеса.

1.6.3 Пример

Компонента `Proc` садржи више процеса различитог типа (листинг 1.9).

Листинг 1.9: Илустрација различитих процеса

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

using namespace sc_core;

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

        Proc(sc_module_name);

protected:
        void print(const char *str);

        sc_event clk, a, b;
        void clk_thread();
        void ab_thread();
        void static_method();
        void dynamic_method();

        int ab_int;
};

Proc::Proc(sc_module_name name) : sc_module(name)
{
        SC_THREAD(clk_thread);
        SC_THREAD(ab_thread);
        SC_METHOD(static_method);
        sensitive << a << b;
        SC_METHOD(dynamic_method);
        sensitive << a << b;
        ab_int = 0;
}

void Proc::print(const char *str)
{
        std::cout <<
                "@ " <<
                sc_time_stamp() << " " <<
                str <<
                std::endl;
}

void Proc::clk_thread()
```

```
{
        while(1)
        {
                wait(5, SC_NS);
                clk.notify();
        }
}

void Proc::ab_thread()
{
        while(1)
        {
                wait(5, SC_NS);
                a.notify();
                wait(5, SC_NS);
                b.notify();
        }
}

void Proc::static_method()
{
        print("Static method");
}

void Proc::dynamic_method()
{
        sc_time d(3, SC_NS);
        switch(ab_int)
        {
        case 0:
                print("Dynamic method AB");
                next_trigger(a);
                break;
        case 1:
                print("Dynamic method A");
                next_trigger(b);
                break;
        case 2:
                print("Dynamic method B");
                next_trigger(d);
                break;
        case 3:
                print("Dynamic method - timeout");
                next_trigger(d, a | b);
                break;
        case 4:
                print("Dynamic method - timeout or AB");
                next_trigger();
                break;
        default:
                assert(false);
        }
        ab_int = (ab_int + 1) % 5;
}

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

        sc_start(100, SC_NS);

    return 0;
}
```

Процеси `clk_thread` и `ab_thread` су `SC_THREAD` процеси. Процес `clk_thread` шаље обавештења од догађаја `clk` сваких 5 ns. Процес `ab_thread` шаље обавештења од догађаја `a` и `b`.

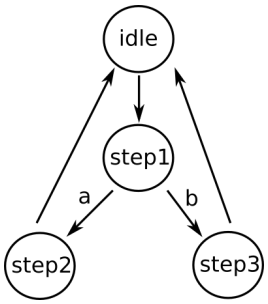
Процеси `static_method` и `dynamic_method` су `SC_METHOD` процеси. Оба процеса садрже догађаје `a` и `b` у својим статичким листама осетљивости.

Процес `static_method` веома је једноставан. Сваки пут када се пошаље обавештење од догађаја `a` и `b` овај процес одштампа поруку.

Процес `dynamic_method` је компликованији. Овај процес мења своју листу осетљивости током извршавања коришћењем методе `next_trigger`. Постоје 4 различита коришћења ове методе унутар овог процеса. Прво, позивима `next_trigger(a)` и `next_trigger(b)` мења се листа осетљивости на обавештења од догађаја `a` and `b`. Друго, позив `next_trigger(d)` аутоматски активира овај процес након временског интервала `d`. Треће, позив `next_trigger(d, a | b)` аутоматски активира процес након временског интервала `d` или док се не деси обавештење од догађаја `a` или `b`. На крају, позив `next_trigger()` враћа оригиналну статичку листу осетљивости. Када год се процес активира штампа се одговарајућа порука.

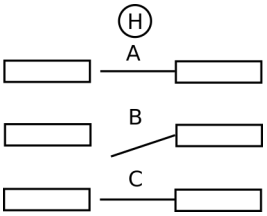
1.6.4 Вежбе

1. Моделовати аутомат задат дијаграмом (слика 1.17) коришћењем `SC_METHOD` процеса и већег броја догађаја. Стање аутомата треба да се памти променом осетљивости процеса. Прелази аутомата се дешавају када се пошаље порука одговарајућег догађаја. Улазе аутомата моделовати догађајима.



Слика 1.17: Дијаграм аутомата

2. Процесом `SC_METHOD` врсте моделовати ситуацију човека који покушава да прође кроз троје врата при следећим правилима (слика 1.18). На почетку човек - `H`, чека испред врата `A`. Када прође кроз врата `A`, треба да прође кроз врата `B`, па потом `C`. Уколико испред било којих врата човек чека дуже од 5 секунди, мора да се врати испред врата `A`. Врата се отварају насумично у интервалу од 3 до 7 секунди. Симулацијом одредити проценат успешности проласка човека кроз сво троје врата.



Слика 1.18: Човек и врата