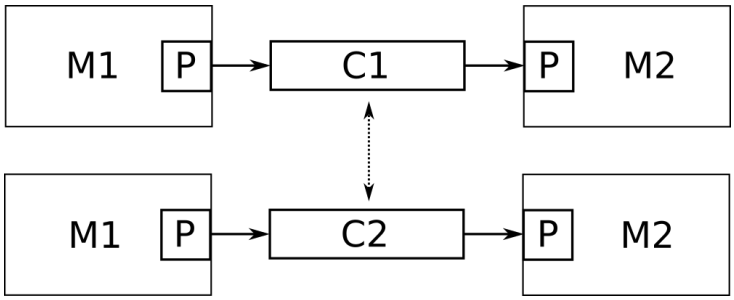
1.8 SystemC приступи

SystemC интерфејс је абстрактна класа која наслеђује класу `sc_interface` и пружа само чисте виртуелне декларације метода које се потом имплементирају у каналима и референцирају у пристпупима. SystemC интерфејси не садрже имплементацију и податке.

SystemC канал је класа која имплементира један или више SystemC интерфејса и наслеђује класу `sc_channel` или `sc_prim_channel`. Канал мора да имеплементира све методе интерфејса које наслеђује.

SystemC приступ је шаблон класа у чији шаблон се умеће класа која наслеђује SystemC интерфејс. SystemC приступ имплементиран је шаблон класом `sc_port`.

Идеја приступа је да се раздвоје правила комуникације од имплементације комуникације (слика 1.24). Модули који комуницирају преко приступа имају јасно дефинисана правила како се комуникација одвија. Уколико се имплементација комуникације промени, при чему се и даље поштују правија постављена приступима, модули могу остати непромењени.



Слика 1.24: Идеја приступа

Приступ се декларише на следећи начин:

```
sc_core::sc_port<interface> pname;
```

Пример улазног приступа који користи сигнални примитивни канал био би:

```
sc_core::sc_port< sc_signal_in_if<int> > sig_in;
```

Овај присту могао би да користи само методу `read` сигналног канала, док би метода `write` била недоступна. На сличан начин може се декларисати и излазни примитивни канал:

```
sc_core::sc_port< sc_signal_in_out<int> > sig_out;
```

Приликом моделовања хардверских система веома често се срећу приступи ка сигналним каналима. Стога су у SystemC језик додате посебне класе које скраћују декларацију сигналних приступа: `sc_in`, `sc_out`. Претходна два приступа могла би се декларисати и са овим класама.

```
sc_core::sc_in < int > sig_in;
sc_core::sc_out< int > sig_out;
```

Приступ се може разумети као показивач на канал који имплементира интерфејс наведен приликом декларације приступа. Ипак, приступ је и нешто више од самог показивача. За класу `sc_port` додате су методе које значајно олакшавају хијерархијско моделовање.

Модули садрже приступе ка каналима. Два модула се повезују тако што се прво инстанцирају модули, као и канал којим треба да се повежу. Потом се користе методе приступа да се модул повеже на конкретну инстанцу канала. За повезивање користи се метода приступа `bind` или преклопљени C++ оператор `operator ()`. Уколико се за повезивање користе методе приступа, тада је то именовано повезивање. Да би се постигло повезивање приступа `pt`, који је садржан у модулу `mod`, на канал са именом `ch` потребно је позвати неку од наредне две методе:

```
mod.pt.bind(ch);
mod.pt(ch);
```

Постоји и другачији начин повезивања који се назива позиционо повезивање. Повезивање канала на приступ модула могуће је и помоћу оператора `operator ()` класе `sc_module`. У случају позиционог повезивања, битан је редослед којим су приступи набројани унутар модула. Тада се приступ на одговарајућој позицији унутар модула повезује са одговарајућим каналом на истој тој позицији приликом позива оператора `operator ()`. На пример, нека модул `some_mod` садржи само два приступа `pr0` и `pr1`:

```
SC_MODULE(some_mod)
{
...
sc_in<int> pr0;
sc_in<int> pr1;
...
}
```

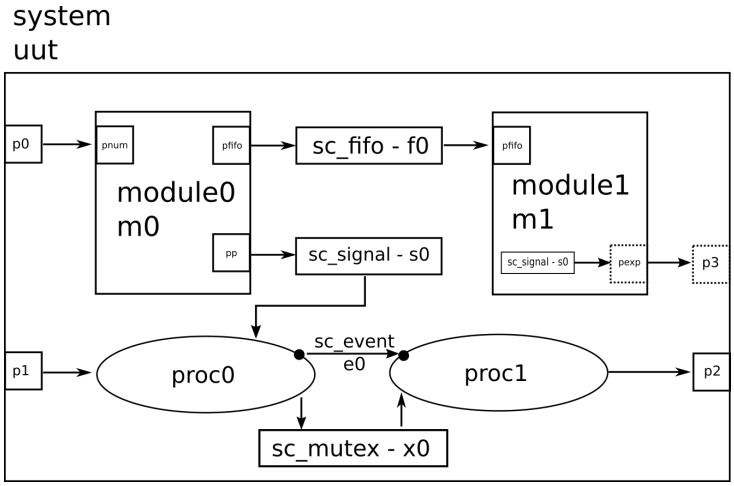
Тада се приступи инстанце овог модула `mod` могу повезати са одговарајућим каналима `ch0` и `ch1`, коришћењем преклопљеног оператора `operator ()` на следећи начин:

```
mod(ch0, ch1);
```

Као и у случају Hardware Description Languages (HDL) језика препоручљиво је користити именовано повезивање.

1.8.1 Примена приступа

Компонента `system` осмишљена је да илуструје разне начине комуникације који постоје у SystemC језику (слика 1.25). Компонента садржи модуле, процесе и већи број примитивних канала. Сви делови система повезани су на различит начин.



Слика 1.25: Различити начини комуникације

Компонента `module0` садржи три приступа: један улазни `pnum` и два излазна `pfifo` и `pp` (листинг 1.15). Инстанца ове компоненте у систему, `m0`, прима целе бројеве преко улазног приступа `p0` система на свој приступ `pnum` (слика 1.25). Приликом повезивања система, у овом случају повезиваће се два приступа.

Листинг 1.15: Заглавље модула `module0`

```
#ifndef _MODULE0_HPP_
#define _MODULE0_HPP_

#include <systemc>

namespace comm
{
```

```
using namespace sc_core;

SC_MODULE(module0)
{
public:
        SC_HAS_PROCESS(module0);

        module0(sc_module_name);

        sc_in<int> pnum;
        sc_port< sc_fifo_out_if<int> > pfifo;
        sc_out<int> pp;
protected:
        void process();
};

}
#endif
```

Преко `pfifo`, инстанца `m0` комуницира са инстацом `m1`. За имплементацију комуникације користи се примитивни канал `sc_fifo`, конкретно инстаца `f0`. У овом случају приказана је комуникација између два модула.

Ова компонента унутар система, повезана је и са процесом `proc0` који је садржан у компоненти `system`. Веза је постигнута помоћу пристпа `pp` а за имплементацију користи се примитивни канал `sc_signal`, његова инстанца `s0`. Комуникација се одвија између компоненте и процеса.

Имплементација модула је једноставна (листинг 1.16). Компонента `module0` унутар свог процеса који је типа `SC_METHOD` и осетљив је на улазни приступ `pnum`, чита пристигли податак и потом га повећава за 1. Затим га прослеђује на излазне приступе. Процес може бити осетљив на приступ и у том случају процес се активира када год догађај канала пошаље поруку. То се дешава увек када дође до промене вредности унутар канала.

Листинг 1.16: Имплементација модула `module0`

```
#include "module0.hpp"

namespace comm
{
        using namespace sc_core;
        using namespace std;

        module0::module0(sc_module_name n) : sc_module(n)
        {
                cout << name() << " constructed.\n";
                SC_METHOD(process);
                sensitive << pnum;
        }

        void module0::process()
        {
                int v = pnum->read();
                cout << name() << " read: " << v << endl;
                v++;
                pfifo->write(v);
                pp->write(v);
        }
}
```

Компонента `module1` садржи само један улазни приступ `pfifo` (листинг 1.17). Преко овог приступа, инстанца ове компоненте у систему, `m1`, повезана је са инстанцом `m0` компоненте `module0`.

Листинг 1.17: Заглавље модула `module1`

```
#ifndef _MODULE1_HPP_
#define _MODULE1_HPP_

#include <systemc>

namespace comm
{
        using namespace sc_core;

        SC_MODULE(module1)
        {
        public:
                SC_HAS_PROCESS(module1);

                module1(sc_core::sc_module_name);

                sc_fifo_in<int> pfifo;
                sc_export<sc_signal_out_if<int> > pexp;
                //sc_port< sc_fifo_in_if<int> > pfifo;
        protected:
                void process();
                sc_signal<int> s0;
        };

}
#endif
```

Компонента имплементира један процес (листинг 1.18). Овај процес је `SC_THREAD` типа и сваку наносекунду проверава да ли су пристигле неки подаци на улазни приступ и у случају да јесу исписује их.

Листинг 1.18: Имплементација модула `module1`

```
#include "module1.hpp"

namespace comm
{
        using namespace sc_core;
        using namespace std;

        module1::module1(sc_module_name n) : sc_module(n)
        {
                //pexp(s0);
                pexp.bind(s0);
                cout << name() << " constructed.\n";
                SC_THREAD(process);
        }

        void module1::process()
        {
                while(1)
                {
                        if (pfifo->num_available())
                        {
                                int v = pfifo->read();
                                cout << name() << " read: "
                                        << v << endl;
                                s0.write(v);
                        }
                        wait(1, SC_NS);
                }
        }

}
```

Компонента `system` је најинтересантнија (листинг 1.19). Компонента садржи два улазна приступа `p0` и `p1` и један излазни `p2`. Потом, ова компонента имплементира процесе `proc0` и `proc1`. Додатно, ова компонента садржи инстанце компоненти `module0` и `module1` као и примитивних канала.

Листинг 1.19: Заглавље модула `system`

```
#ifndef _SYS_HPP_
#define _SYS_HPP_

#include <systemc>
#include "module0.hpp"
#include "module1.hpp"

namespace comm
{
using namespace sc_core;

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

        system(sc_module_name);

        sc_port<sc_signal_in_if<int> > p0;
        sc_in<int> p1;
        sc_out<int> p2;
        sc_export< sc_signal_out_if<int> > p3;
protected:
        void proc0();
        void proc1();

        sc_fifo<int> f0;
        sc_signal<int> s0;
        sc_event e0;
        module0 m0;
        module1 m1;
        sc_mutex x0;
};


}
#endif
```

Подкомпоненте у SystemC језику инстанцирају се унутар конструктора (листинг 1.20). Конструктори свих инстанци позивају се у иницијлизационој листи чланова. Имена свих компоненти прослеђују се као параматар. Конструктори примитивних канала могу да се наведу али не морају. На пример, за мутекс је позван конструктор и прослеђно му је име. За остале канале није позван конструктор. То значи да ће се позвати подразумевани конструктор примитивног канала, који ће дати произвољно, јединствено име каналу.

Листинг 1.20: Имплементација модула `system`

```
#include "system.hpp"

namespace comm
{
        using namespace sc_core;
        using namespace std;

        system::system(sc_module_name n) :
                sc_module(n),
                f0("f0"), // Member initializer list
                m0("m0"),
                m1("m1"),
                x0("x0")
        {
                cout << name() << " constructed.\n";
                m0.pnum(p0);
                m0.pfifo.bind(f0);
                m0.pp(s0);
                // Position connecting is possible
                // but not recommended.
                //m0(p0, f0);
                m1.pfifo(f0);
                p3(m1.pexp);

                SC_THREAD(proc0);
```

```
                sensitive << p1 << s0;
                SC_THREAD(proc1);
        }

        void system::proc0()
        {
                sc_process_handle h = sc_get_current_process_handle();
                cout << h.name() << " of type " << h.proc_kind() << " activated @ " << sc_time_stamp() << endl;
                x0.lock();
                wait(1, SC_NS);
                x0.unlock();
                cout << "Proc0 deactivated " << sc_time_stamp() << endl;
                e0.notify();
        }

        void system::proc1()
        {
                int v = 200;
                while(1)
                {
                        wait(e0);
                        cout << "Event e0 happened " << sc_time_stamp() << endl;
                        x0.lock();
                        wait(2, SC_NS);
                        x0.unlock();
                        p2.write(v++);
                }
        }
}
```

Повезивање инстанци компоненти налази се увек унутар тела конструктора. Методе приступа употребљене су за повезивање компоненти. Употребљена је `bind` метода, као и преклопљени оператор `operator ()`. Додатно, регистровани су и процеси компоненте.

Процеси `proc0` и `proc1` комуницирају помоћу мутекс примитивног канала `x0` и додатно се синхронизују преко порука догађаја `e0`. Тиме је илустрована комуникација између два процеса, помоћу примитивних канала и догађаја. Процес `proc0` исписује поруку о свом имену и врсти. Потом закључава мутекс, чека неко време, откључава мутекс и на крају догађај `e0` шаље поруку. Овај процес се потом завршава. Процес `proc1` чека поруку догађаја. Када порука стигне, закључава мутекс, чека неко време, откључава мутекс и потом шаље податке на излазни приступ `p2`. Процес може коришћењем приступа слати поруке.

Компонента `tbcomm` инстанцира компоненту `system`, шаље јој стимулус и представља једноставан тестбенч за компоненту `system` (листинг 1.21). Унутар компоненте декларисани су примитивни канали који ће бити повезани на улазне и излазне приступе `system` компоненте.

Листинг 1.21: Заглавље модула `tbcomm`

```
#ifndef _TBCOMM_HPP_
#define _TBCOMM_HPP_

#include <systemc>
#include "system.hpp"

namespace comm
{

SC_MODULE(tbcomm)
{
public:
        SC_HAS_PROCESS(tbcomm);
```

```
        tbcomm(sc_module_name);

protected:
        system uut;
        sc_signal<int> num0;
        sc_signal<int> num1;
        sc_signal<int> res;

        void driver();
        void monitor();
};

}
#endif
```

У иницијлизационој листи чланова инстанцира се `system` компонента, док се у конструктору повезују се примитивни канали на прситупе. Процес `driver` уписује неке вредности на улазне приступе инстанциране `system` компоненте.

Листинг 1.22: Имплементација модула `tbcomm`

```
#include "tbcomm.hpp"

namespace comm
{
        using namespace sc_core;
        using namespace std;

        tbcomm::tbcomm(sc_module_name n) : sc_module(n), uut("UUT")
        {
                cout << name() << " constructed.\n";
                uut.p0(num0);
                uut.p1(num1);
                uut.p2(res);
                SC_THREAD(driver);
                SC_METHOD(monitor);
                sensitive << uut.p3;
        }

        void tbcomm::driver()
        {
                int v = 0xA;
                int t = 100;
                while(1)
                {
                        num0.write(v++);
                        wait(2, SC_NS);
                        num1.write(t--);
                        wait(1, SC_NS);
                }
        }

        void tbcomm::monitor()
        {
                cout << "From export: " << uut.p3->read() << endl;
        }
}
```

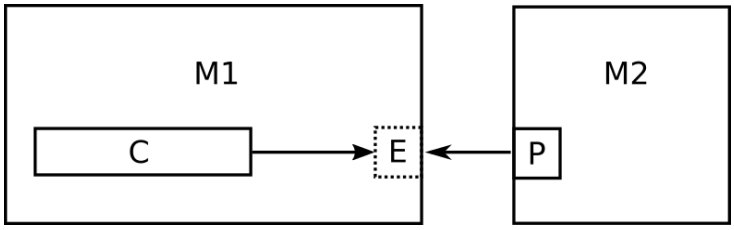
Главни програм инстацира компоненту `tbcomm` и покреће симулацију (листинг 1.23).

Листинг 1.23: Главни програм

```
#include <systemc>

#include "tbcomm.hpp"

int sc_main(int argc, char* argv[])
{
        comm::tbcomm uut("TB");

        sc_start(50, sc_core::SC_NS);

    return 0;
}
```

1.8.2 Примена извозника

SystemC извозник је сличан SystemC приступу по начину декларисања али је другачији по начину повезивања. Идеја извозника је да се унутрашњи канал члан модула учини доступним за повезивање са осталим деловима система (слика 1.26). У случају приступа, повезивање се спроводи са каналом који је изван модула, док се у случају извозника повезивање изводи са каналом који је унутар модула. Модул (`M2`) се помоћу приступа (`P`), преко извозника (`E`) другог модула (`M1`) може повезати на канал (`C`) које је члан тог другог модула.



Слика 1.26: Идеја извозника

Извозник је имплементиран класом `sc_export` и декларише се на следећи начин:

```
sc_core::sc_export<interface> ename;
```

Слично као и у случају приступа, за повезивање користи се метода извозника `bind` или преклопљени C++ оператор `operator ()`.

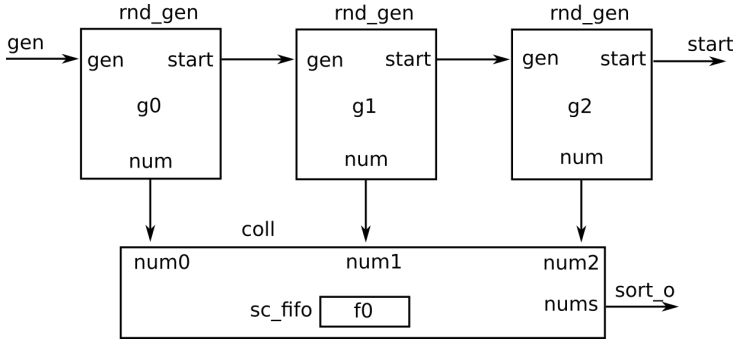
Компонента `system` садржи неколико извозника (слика 1.25). Компонента `m1` садржи сигнални канал `s0`, који је помоћу извозника `pexp` начињен доступним (листинг 1.17). Канал `s0` је унутар заштићеног дела модула, али је ипак доступан преко `pexp`. Ова два објекта повезана су помоћу методе `bind` извозника (листинг 1.18).

Компонента `system` затим показује једну додатну особину извозника, а то је могућност да се извозник избаци даље на следећи виши ниво хијерархије. Извозник `pexp` инстанце `m1` избачен је даље на извозник `p3` компоненте `system` (листинг 1.19). У овом случају коришћен је оператор `operator ()` (листинг 1.20).

Тестбенч система, `tbcomm`, на крају, користи извозник `p3` компоненте `system` да прима податке директно од канала `s0` компоненте `m1` (листинг 1.22). Ова компонента садржи и процес `monitor` који је осетљив на извозник. Када год стигне неки податак од извозника, исписује се порука која садржи вредност податка.

1.8.3 Вежбе

1. Моделовати систем који садржи три компоненте које генеришу случајне бројеве (слика 1.27). Када се подигне улаз `gen`, компонента у интервалу времена од 5 ns до 10 ns на свој излаз `num` шаље један случајан број од 0 до 99. Одмах након тога, активира свој излаз `start`. Све улазе и излазе моделовати приступима. Систем садржи и једну компоненту `coll` која прима бројеве на три улаза. Када прими 100 бројева слаже их сортиране у FIFO канал, који се преко извозника чини доступним на излазу компоненте `nums`.



Слика 1.27: Једноставан систем са приступима и извозницима

Моделовати и компоненту која представља једноставан тестбенч за овај систем. У току тестирања потребно је већи број пута прочитати све вредности са излаза целог система, `sort_o`.

2. Моделовати филтар задат релацијом 1.1, структурним, хијерархијским стилом. Сабираче, множаче и регистре моделовати појединачним компонентама. Све компоненте повезати у оквиру једног модела целокупног филтра. Вредности излаза добијене оваквим моделом упоредити са златним вредностима модела.