

Exercise 4: SystemC and Virtual Prototyping

Exercise on Custom `sc_interface`

Lukas Steiner

WS 2023/2024

The source code to start this exercise is available here:
<https://github.com/TUK-SCVP/SCVP.Exercise4>

Task 1

Custom Interfaces, Ports and Channels

Figure 1 shows an example for a very simple *Petri Net* (PN). The goal of this exercise is to implement the semantics of PN in SystemC by using custom interfaces, templated ports and channels. We will reach this goal going step by step through the exercise. *Transitions* will be implemented as `SC_MODULES` and *places* will be implemented as custom channels that are connected to ports of the transitions.

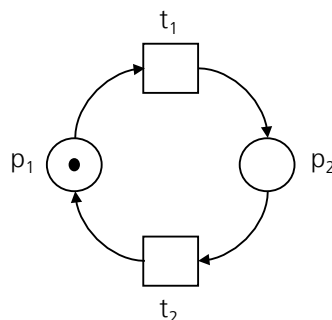


Fig. 1: Simple Petri Net

First, an interface `placeInterface` has to be implemented in `place.h`, which describes the access methods of a place channel by using pure virtual functions. The abstract class `placeInterface` inherits from `sc_interface` and should have the following pure virtual functions:

```
virtual void addTokens(unsigned int n) = 0;
virtual void removeTokens(unsigned int n) = 0;
virtual unsigned int testTokens() = 0;
```

Note: a virtual function is specified *pure* by using the *pure specifier* "`= 0`".

Second, a `place` channel should be created inside `place.h`, which inherits from `placeInterface`. The channel should have a member variable `unsigned int tokens`, which specifies the current number of tokens in the place. The initial number of tokens should be passed as a parameter to the constructor and then set in the constructor of the channel `place`.

Next the virtual functions have to be implemented:

- The method `addTokens(unsigned int n)` should add `n` tokens to the member variable `tokens`.
- The method `removeTokens(unsigned int n)` should subtract `n` tokens from the member variable `tokens`.
- The method `testTokens()` should return the value of the member variable `tokens`.

Third, an `SC_MODULE` called `transition` should be defined in `transition.h`. As usual, the constructor of the module must be defined, but there is no need to register any member function with the SystemC kernel, i.e., no need to call `SC_METHOD()` or `SC_THREAD()` for this module.

The module has two `sc_ports` from template type `placeInterface` called `in` and `out`. Furthermore, the `transition` module should have a member function `fire()`¹. This function should check if the transition is enabled by using the method `testTokens` of the `in` port, i.e., if there exists at least one token in the place. In this case it should print

```
std::cout << this->name() << ": Fired" << std::endl;
```

remove one token from the `in` port and add a token to the `out` port. If the number of tokens in the place is zero it should print

```
std::cout << this->name() << ": NOT Fired" << std::endl;
```

¹For the beginning of this exercise, we assume that all weights of the arcs are one and that the Petri Net has no forks or joins, i.e., a place is connected to exactly one transition's output and one transition's input, e.g., Figure 1.

In order to test our initial Petri Net implementation build an `SC_MODULE` called `toplevel` in `main.cpp` that reflects the PN shown in Figure 1. As a test bench the module should use the following function as an `SC_THREAD`.

```
void process()
{
    while (true)
    {
        wait(10, SC_NS);
        t1.fire();
        wait(10, SC_NS);
        t1.fire();
        wait(10, SC_NS);
        t2.fire();
        sc_stop();
    }
}
```

Create one instance of `toplevel` with the name `t` in the `sc_main` function. After compiling and running your code you should see the following output:

```
t.t1: Fired
t.t1: NOT Fired
t.t2: Fired
```

After firing `t1` the transaction cannot be fired again because a token is missing in the input place.

Multiports

In order to have more than one port for in and out we template the transition module in the following way in order to support multiple input ports:

```
template<unsigned int N = 1, unsigned int M = 1>
SC_MODULE(transition)
{
public:
    sc_port<placeInterface, N, SC_ALL_BOUND> in;
```

place.h

```
1  #ifndef PLACE_H
2  #define PLACE_H
3
4  #include <systemc.h>
5  using namespace std;
6
7
8  class placeInterface : public sc_interface {    //^Father class
9
10     public:
11     //----Task 1 and 2
12     virtual void addTokens(unsigned int n) = 0;
13     virtual void removeTokens(unsigned int n) = 0;
14     virtual unsigned int testTokens() = 0;
15
16     //-----Task 3
17     /*
18     virtual void addTokens() = 0;
19     virtual void removeTokens() = 0;
20     virtual bool testTokens() = 0;
21     */
22 };
23
24 //-----Task 1 and 2
25 class placeChannel: public placeInterface    //^Regular child class, not SystemC module
26 {
27     public:
28     unsigned int tokens;
29
30     placeChannel(unsigned int n=0): tokens(n) {}    //^Class constructor
31
32     void addTokens(unsigned int n)    //^ VVVVV Virtual functions
33     {
34         tokens += n;
35     }
36
37     void removeTokens(unsigned int n)
38     {
39         tokens -= n;
40     }
41
42     unsigned int testTokens()
43     {
44         return tokens;
45     }
46 };
47
48
49 #endif // PLACE_H
50
```

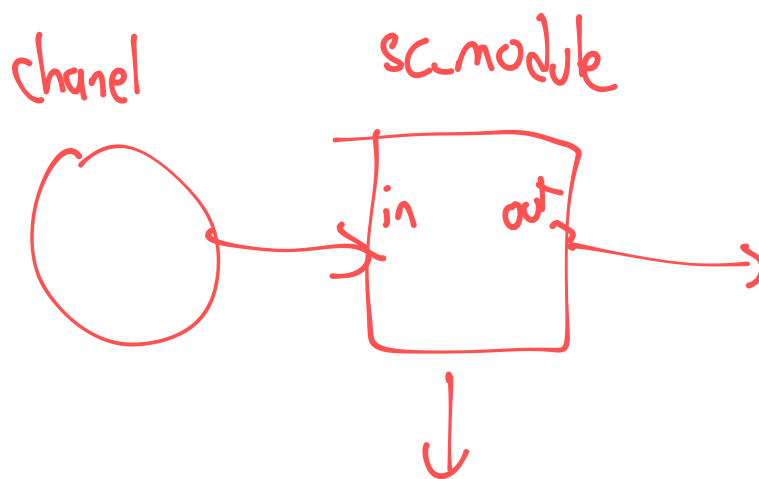
transition.h

```

1  #ifndef TRANSITION_H
2  #define TRANSITION_H
3
4  #include <systemc.h>
5
6  #include "place.h"
7
8  using namespace std;
9
10 //----task 1 and 2
11
12 SC_MODULE(transition){
13     sc_port< placeInterface > in;
14     sc_port< placeInterface > out;
15
16     SC_CTOR(transition): in("in"), out("out"){
17
18     void fire()
19     {
20         if (in->testTokens() >= 1)    //^Using arrow -> because these are virtual fuctions
21         {
22             std::cout << this->name() << ": Fired" << std::endl;
23             in->removeTokens(1);
24             out->addTokens(1);
25
26         }else
27         {
28             std::cout << this->name() << ": NOT Fired" << std::endl;
29         }
30     }
31 };
32
33
34
35
36
37 #endif // TRANSITION_H
38

```

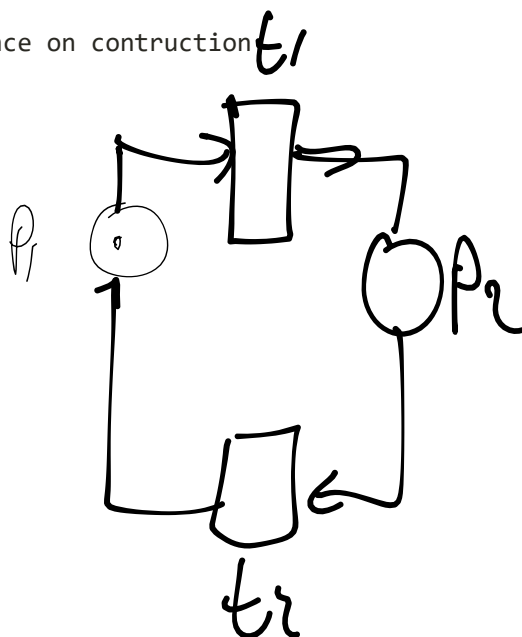
→ interface → no channel



has access to the
functions of the
connected channels

main.cpp

```
1  #include <systemc.h>
2
3  #include "place.h"
4  #include "transition.h"
5  // #include "subnet.h"
6
7  using namespace std;
8
9  //----- Task 1
10 SC_MODULE(toplayer)
11 {
12     placeChannel p1; // ^Channels are places
13     placeChannel p2;
14     transition t1, t2; // ^Transitions are modules
15
16
17     SC_CTOR(toplayer): t1("t1"), t2("t2"), p1(1), p2(0) // ^Here I define my initial state
18     {
19         t1.in.bind(p1);
20         t1.out.bind(p2);
21         t2.in.bind(p2);
22         t2.out.bind(p1);
23         SC_THREAD(process); // ^Execute once on construction
24     }
25
26     void process()
27     {
28         while (true)
29         {
30             wait(10, SC_NS);
31             t1.fire();
32             wait(10, SC_NS);
33             t1.fire();
34             wait(10, SC_NS);
35             t2.fire();
36             sc_stop();
37         }
38     }
39 };
40
41
42
43 int sc_main(int, char**)
44 {
45     toplayer top("t");
46
47     sc_start();
48     return 0;
49 }
50
```



```

    sc_port<placeInterface, M, SC_ALL_BOUND> out;

    ...
}

```

The template parameter N denotes the number of input ports and M denotes the number of output ports, respectively.

The function `fire()` must be modified to the previous example. It should check if there is one token in each input port using the `in[i]->testTokens()` method call (where $0 \leq i < N$).

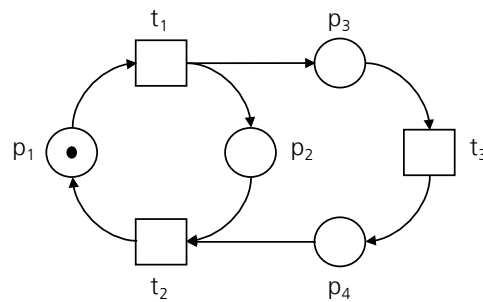


Fig. 2: Simple Petri Net

If there are enough tokens it should remove one token from each input port and add one token to each output port. The printing when firing should be done as in the previous example. In order to test our implementation, implement the PN shown in Figure 2 and use the following function as a test bench (SC_THREAD).

```

void process()
{
    while (true)
    {
        wait(10, SC_NS);
        t1.fire();
        wait(10, SC_NS);
        t2.fire();
        wait(10, SC_NS);
        t3.fire();
        wait(10, SC_NS);
        t2.fire();
    }
}

```

```
        wait(10, SC_NS);
        s1.ACT.fire();
        wait(10, SC_NS);
        s1.RD.fire();
        wait(10, SC_NS);
        s1.WR.fire();
        wait(10, SC_NS);
        s1.PRE.fire();
        wait(10, SC_NS);
        s1.ACT.fire();
        wait(10, SC_NS);
        s2.ACT.fire();
        wait(10, SC_NS);
        s2.ACT.fire();
        wait(10, SC_NS);
        s1.PRE.fire();
        wait(10, SC_NS);
        s2.PRE.fire();
        wait(10, SC_NS);
        sc_stop();
    }
}
```

As an output you should see the following:

```
t.s1.ACT: Fired
t.s1.ACT: NOT Fired
t.s1.RD: Fired
t.s1.WR: Fired
t.s1.PRE: Fired
t.s1.ACT: Fired
t.s2.ACT: Fired
t.s2.ACT: NOT Fired
t.s1.PRE: Fired
t.s2.PRE: Fired
```


place.h

```
1  #ifndef PLACE_H
2  #define PLACE_H
3
4  #include <systemc.h>
5  using namespace std;
6
7
8  class placeInterface : public sc_interface {    //^Father class
9
10     public:
11     //----Task 1 and 2
12     virtual void addTokens(unsigned int n) = 0;
13     virtual void removeTokens(unsigned int n) = 0;
14     virtual unsigned int testTokens() = 0;
15
16     //-----Task 3
17     /*
18     virtual void addTokens() = 0;
19     virtual void removeTokens() = 0;
20     virtual bool testTokens() = 0;
21     */
22 };
23
24 //-----Task 1 and 2
25 class placeChannel: public placeInterface    //^Regular child class, not SystemC module
26 {
27     public:
28     unsigned int tokens;
29
30     placeChannel(unsigned int n=0): tokens(n) {}    //^Class constructor
31
32     void addTokens(unsigned int n)    //^ VVVVV Virtual functions
33     {
34         tokens += n;
35     }
36
37     void removeTokens(unsigned int n)
38     {
39         tokens -= n;
40     }
41
42     unsigned int testTokens()
43     {
44         return tokens;
45     }
46 };
47
48
49 #endif // PLACE_H
50
```

transition.h

```
1  #ifndef TRANSITION_H
2  #define TRANSITION_H
3
4  #include <systemc.h>
5
6  #include "place.h"
7
8  using namespace std;
9
10 //----task 1 and 2
11 template < unsigned int N = 1, unsigned int M = 1 >
12 SC_MODULE(transition){
13
14     sc_port<placeInterface, N, SC_ALL_BOUND> in; //^PlaceInterfaces is created as a vector of N
15     sc_port<placeInterface, M, SC_ALL_BOUND> out; elements as inputs
16
17     SC_CTOR(transition): in("in"), out("out"){
18
19     void fire()
20     {
21         //Check tokens
22         int counterToken = 0;
23         for (int i = 0; i < N; i++)
24         {
25             if (in[i]->testTokens() >= 1)
26             {
27                 counterToken++; //^Add how many of my previous places have at least one token
28             }
29         }
30
31         if (counterToken == N) //^If all previous places have a token, then...
32         {
33             //if all previous places have tokens
34             cout << this->name() << ": Fired" << endl;
35
36             for (int i = 0; i < N; i++){
37                 in[i]->removeTokens(1); //----Task 1 and 2
38             }
39             for (int i = 0; i < M; i++){
40                 out[i]->addTokens(1); //-----Task 1 and 2
41             }
42
43         }else{
44             cout << this->name() << ": NOT Fired" << endl;
45         }
46     }
47
48 };
49
50 };
51
52
53
54 #endif // TRANSITION_H
55
```

It's not needed

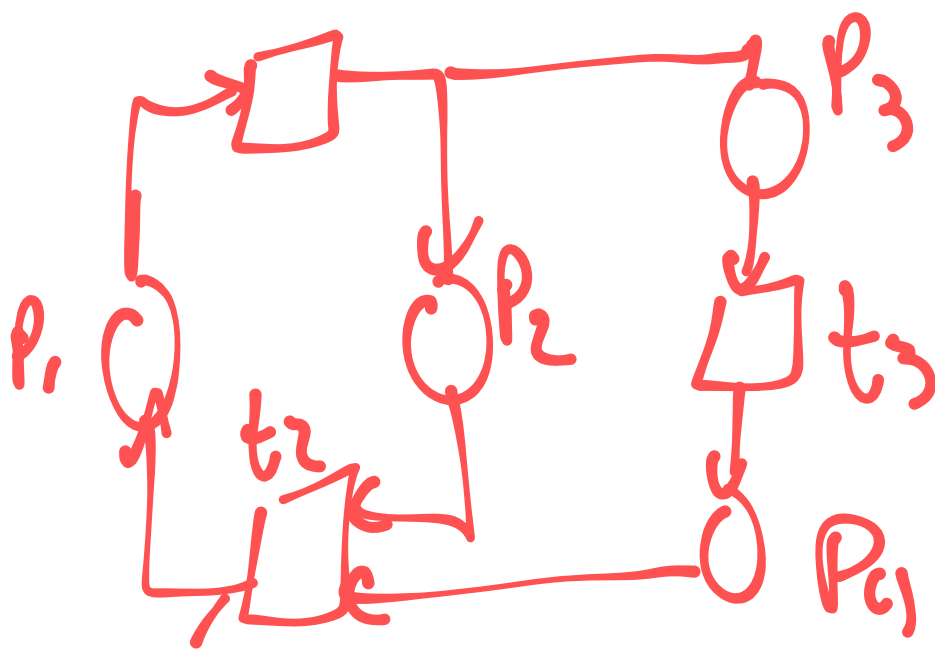
★ ⇒ In this example we use

[N]

Dynamic port creation

if not ⇒ use arrays - [N]

t1



```

1  #include <systemc.h>
2
3  #include "place.h"
4  #include "transition.h"
5  //#include "subnet.h"
6
7  using namespace std;
8
9
10
11 //----- Task 2
12
13 SC_MODULE(toplayer)
14 {
15     transition<1,2> t1;
16     transition<2,1> t2;
17     transition<1,1> t3;
18
19     placeChannel p1;
20     placeChannel p2;
21     placeChannel p3;
22     placeChannel p4;
23
24     SC_CTOR(toplayer): t1("t1"), t2("t2"), t3("t3"), p1(1), p2(0), p3(0), p4(0)
25     {
26         //^Check figure 2
27         t1.in.bind(p1);
28         t1.out.bind(p3);
29         t1.out.bind(p2);
30
31         t2.in.bind(p2);
32         t2.in.bind(p4);
33         t2.out.bind(p1);
34
35         t3.in.bind(p3);
36         t3.out.bind(p4);
37
38         SC_THREAD(process);
39     }
40
41     void process()
42     {
43         while (true)
44         {
45             wait(10, SC_NS);
46             t1.fire();
47             cout << "p1: " << p1.testTokens() << " p2: " << p2.testTokens() << " p3: " <<
p3.testTokens() << " p4: " << p4.testTokens() << endl;
48             wait(10, SC_NS);
49             t2.fire();
50             cout << "p1: " << p1.testTokens() << " p2: " << p2.testTokens() << " p3: " <<
p3.testTokens() << " p4: " << p4.testTokens() << endl;
51             wait(10, SC_NS);
52             t3.fire();
53             cout << "p1: " << p1.testTokens() << " p2: " << p2.testTokens() << " p3: " <<
p3.testTokens() << " p4: " << p4.testTokens() << endl;
54             wait(10, SC_NS);
55             t2.fire();

```

See SL, 3 - 38/39

$t_1.out[0].bind(p_1)$
 $t_2.out[1].bind(p_2)$

Not needed since we are using Dynamic port otherwise needed

```
56         cout << "p1. " << p1.testTokens() << " p2: " << p2.testTokens() << " p3: " <<
p3.testTokens() << " p4: " << p4.testTokens() << endl;
57         sc_stop();
58     }
59 }
60 };
61
62 int sc_main(int, char**)
63 {
64     toplayer top("t");
65
66     sc_start();
67     return 0;
68 }
69
```

```
        sc_stop();
    }
}
```

Note that the order of binding determines to which port number of the port array the channels are bound.

After running the application you should see the following output:

```
t.t1: Fired
t.t2: NOT Fired
t.t3: Fired
t.t2: Fired
```

Note that t2 cannot fire until t3 is fired.

Templated Channels

As next step we want to add weights to the channels at the input arc of a place and at the output arc of a place. First, we change the methods of the interface class as follows:

```
virtual void addTokens() = 0;
virtual void removeTokens() = 0;
virtual bool testTokens() = 0;
```

As for the toplevel module we use templates for the place channel

```
template<unsigned int Win = 1, unsigned int Wout = 1>
class place : public placeInterface
...
```

where Win denotes the input weight of a place and Wout the output weight of a place. The methods of the place channel have to be changed according to the new placeInterface:

- The method addTokens() should now add Win tokens to the member variable tokens.

place.h

```
1  #ifndef PLACE_H
2  #define PLACE_H
3
4  #include <systemc.h>
5  using namespace std;
6
7
8  class placeInterface : public sc_interface {    //^Father class
9
10     public:
11     //----Task 1 and 2
12     /*
13     virtual void addTokens(unsigned int n) = 0;
14     virtual void removeTokens(unsigned int n) = 0;
15     virtual unsigned int testTokens() = 0;
16     */
17
18     //-----Task 3
19     virtual void addTokens() = 0;
20     virtual void removeTokens() = 0;
21     virtual bool testTokens() = 0;
22
23 };
24
25 //-----Task 3
26 template<unsigned int Win = 1, unsigned int Wout = 1>
27 class placeChannel : public placeInterface {
28
29     public:
30     unsigned int tokens;
31
32     placeChannel(unsigned int n=0): tokens(n) {}
33
34     void addTokens(){
35         tokens += Win;
36     }
37     void removeTokens()
38     {
39         tokens -= Wout;
40     }
41
42     bool testTokens()
43     {
44         return (tokens >= Wout); //^Different from task 1 and 2, return a BOOL of weather I have more
of same number of tokens as in Wout
45     }
46 };
47
48
49 #endif // PLACE_H
50
```

transition.h

```
1  #ifndef TRANSITION_H
2  #define TRANSITION_H
3
4  #include <systemc.h>
5
6  #include "place.h"
7
8  using namespace std;
9
10 //-----Task 3
11 template<unsigned int N=1, unsigned int M=1>
12 SC_MODULE(transition){
13     public:
14         sc_port<placeInterface, N, SC_ALL_BOUND> in;
15         sc_port<placeInterface, M, SC_ALL_BOUND> out;
16
17         SC_CTOR(transition): in("in"), out("out"){
18
19         void fire()
20         {
21             //check Places
22             int counterToken = 0;
23             for (int i = 0; i < N; i++)
24             {
25                 if (in[i]->testTokens())
26                 {
27                     counterToken++;
28                 }
29             }
30
31             //^if all previous can fire
32             if (counterToken == N)
33             {
34                 cout << this->name() << ": Fired" << endl;
35
36                 for (int i = 0; i < N; i++){
37                     in[i]->removeTokens();
38                 }
39                 for (int i = 0; i < M; i++){
40                     out[i]->addTokens();
41                 }
42
43             }else{
44                 cout << this->name() << ": NOT Fired" << endl;
45             }
46         }
47     };
48
49
50
51 #endif // TRANSITION_H
52
```

```

1  #include <systemc.h>
2
3  #include "place.h"
4  #include "transition.h"
5  // #include "subnet.h"
6
7  using namespace std;
8
9
10
11 //----- Task 2
12
13 SC_MODULE(toplayer)
14 {
15     transition<1,2> t1;  //^ <NumIn, NumOut>
16     transition<2,1> t2;
17     transition<1,1> t3;
18
19     placeChannel<1,1> p1;  //^ <Win, Wout>
20     placeChannel<1,1> p2;
21     placeChannel<1,1> p3;
22     placeChannel<1,1> p4;
23
24     SC_CTOR(toplayer): t1("t1"), t2("t2"), t3("t3"), p1(1), p2(0), p3(0), p4(0)
25     {
26         //^Check figure 2
27         t1.in.bind(p1);
28         t1.out.bind(p3);
29         t1.out.bind(p2);
30
31         t2.in.bind(p2);
32         t2.in.bind(p4);
33         t2.out.bind(p1);
34
35         t3.in.bind(p3);
36         t3.out.bind(p4);
37
38         SC_THREAD(process);
39     }
40
41     void process()
42     {
43         while (true)
44         {
45             wait(10, SC_NS);
46             t1.fire();
47             cout << "p1: " << p1.testTokens() << " p2: " << p2.testTokens() << " p3: " <<
p3.testTokens() << " p4: " << p4.testTokens() << endl;
48             wait(10, SC_NS);
49             t2.fire();
50             cout << "p1: " << p1.testTokens() << " p2: " << p2.testTokens() << " p3: " <<
p3.testTokens() << " p4: " << p4.testTokens() << endl;
51             wait(10, SC_NS);
52             t3.fire();
53             cout << "p1: " << p1.testTokens() << " p2: " << p2.testTokens() << " p3: " <<
p3.testTokens() << " p4: " << p4.testTokens() << endl;
54             wait(10, SC_NS);
55             t2.fire();

```



```
56         cout << "p1. " << p1.testTokens() << " p2: " << p2.testTokens() << " p3: " <<
p3.testTokens() << " p4: " << p4.testTokens() << endl;
57         sc_stop();
58     }
59 }
60 };
61
62
63 int sc_main(int, char**)
64 {
65     toplayer top("t");
66
67     sc_start();
68     return 0;
69 }
70
```

- The method `removeTokens()` should now subtract `Wout` tokens from the member variable `tokens`.
- The method `testTokens()` should test if the number of tokens is greater than or equal to `Wout`.

The `transition` and `topLevel` modules should be adapted accordingly. By using weights of 1 the new application output should be identical to the previous one.

Modelling a Single Memory Bank

With the current code base we want to model the single memory bank example from the lecture, as shown in Figure 3. Create a `topLevel` module that reflects the memory bank; make sure to use the same names for transitions and places as shown in the picture.

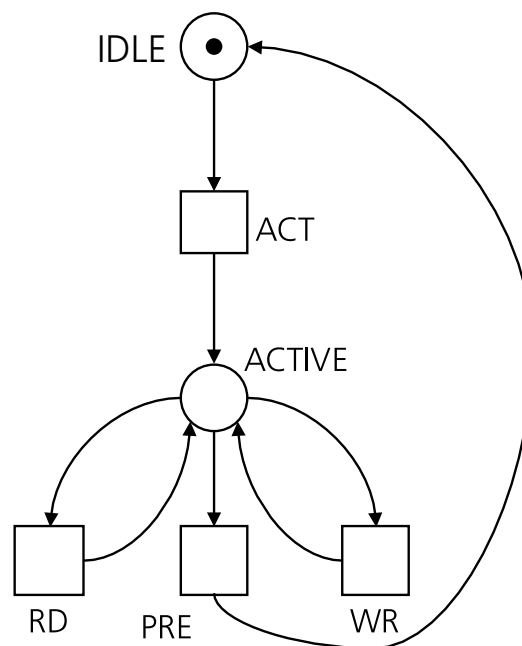


Fig. 3: Single Memory Bank Example

Test your code with the following process:

```

void process()
{
    while (true)

```

main.cpp

```
1  #include <systemc.h>
2
3  #include "place.h"
4  #include "transition.h"
5  // #include "subnet.h"
6
7  using namespace std;
8
9  //-----Task4
10 SC_MODULE(toplayer)
11 {
12     transition<1,1> ACT, RD, PRE, WR;
13     placeChannel<1,1> IDLE, ACTIVE;
14
15     SC_CTOR(toplayer): ACT("ACT"), RD("RD"), PRE("PRE"), WR("WR"), IDLE(1), ACTIVE(0)
16     {
17         ACT.in.bind(IDLE);
18         ACT.out.bind(ACTIVE);
19         RD.in.bind(ACTIVE);
20         RD.out.bind(ACTIVE);
21         PRE.in.bind(ACTIVE);
22         PRE.out.bind(IDLE);
23         WR.in.bind(ACTIVE);
24         WR.out.bind(ACTIVE);
25         SC_THREAD(process);
26     }
27
28     void process()
29     {
30         while (true)
31         {
32             wait(10, SC_NS);
33             ACT.fire();
34             wait(10, SC_NS);
35             ACT.fire();
36             wait(10, SC_NS);
37             RD.fire();
38             wait(10, SC_NS);
39             WR.fire();
40             wait(10, SC_NS);
41             PRE.fire();
42             wait(10, SC_NS);
43             ACT.fire();
44             sc_stop();
45         }
46     }
47 };
48
49
50 int sc_main(int, char**)
51 {
52     toplayer top("t");
53
54     sc_start();
55     return 0;
56 }
57
```

```
{
    wait(10, SC_NS);
    ACT.fire();
    wait(10, SC_NS);
    ACT.fire();
    wait(10, SC_NS);
    RD.fire();
    wait(10, SC_NS);
    WR.fire();
    wait(10, SC_NS);
    PRE.fire();
    wait(10, SC_NS);
    ACT.fire();
    sc_stop();
}
```

The application output should be the following:

```
t.ACT: Fired
t.ACT: NOT Fired
t.RD: Fired
t.WR: Fired
t.PRE: Fired
t.ACT: Fired
```

Implementation of Inhibitor Arcs

In order to implement inhibitor arcs add an additional template parameter `L` for the number of inhibitor inputs,

```
template<unsigned int N=1, unsigned int M=1, unsigned int L=0>
```

and an additional `sc_port` to the transition module:

```
sc_port<placeInterface, L, SC_ZERO_OR_MORE_BOUND> inhibitors;
```

In addition to the firing check for enough tokens in the input ports it is necessary to check if there are no tokens in the connected channels of all inhibitor ports by using the `testTokens()` method. In other words, the firing is only performed if there are enough tokens in all input places and no tokens in places that would inhibit it. Adjust the `fire()` method accordingly. Why did we use the `SC_ZERO_OR_MORE_BOUND` flag?

Building Hierarchical PNs

To finish the exercise create an `SC_MODULE` called `subnet` in `subnet.h` that implements the PNs in the green boxes of Figure 4².

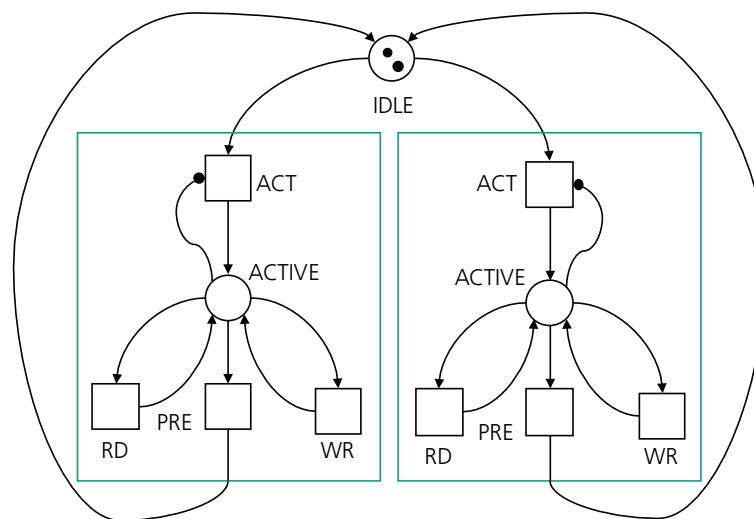


Fig. 4: Two Memory Bank Example with Subnets

Afterwards, implement the `toplevel` module in such a way that it includes two instances of `subnet`, called `s1` and `s2`. Take care that all ports are bound correctly. You can test your implementation with the following process:

```
void process()
{
    while (true)
    {
        wait(10, SC_NS);
        s1.ACT.fire();
    }
}
```

²Hint: you can also bind input ports to input ports and output ports to output ports.

place.h

```
1  #ifndef PLACE_H
2  #define PLACE_H
3
4  #include <systemc.h>
5  using namespace std;
6
7
8  class placeInterface : public sc_interface {    //^Father class
9
10     public:
11     //----Task 1 and 2
12     /*
13     virtual void addTokens(unsigned int n) = 0;
14     virtual void removeTokens(unsigned int n) = 0;
15     virtual unsigned int testTokens() = 0;
16     */
17
18     //-----Task 3
19     virtual void addTokens() = 0;
20     virtual void removeTokens() = 0;
21     virtual bool testTokens() = 0;
22     virtual unsigned int getTokens() = 0; //-- Task 4 and 5
23
24 };
25
26 //-----Task 3
27 template<unsigned int Win = 1, unsigned int Wout = 1>
28 class placeChannel : public placeInterface {
29
30     public:
31     unsigned int tokens;
32
33     placeChannel(unsigned int n=0): tokens(n) {}
34
35     void addTokens(){
36         tokens += Win;
37     }
38     void removeTokens()
39     {
40         tokens -= Wout;
41     }
42
43     bool testTokens()
44     {
45         return (tokens >= Wout); //^Different from task 1 and 2, return a BOOL of weather I have more
of same number of tokens as in Wout
46     }
47
48     unsigned int getTokens()
49     {
50         return tokens;
51     }
52 };
53
54
55 #endif // PLACE_H
56
```

transition.h

```

1  #ifndef TRANSITION_H
2  #define TRANSITION_H
3
4  #include <systemc.h>
5
6  #include "place.h"
7
8  using namespace std;
9
10
11
12
13 //-----Task 3
14 template<unsigned int N=1, unsigned int M=1, unsigned int L=0>
15 SC_MODULE(transition){
16     public:
17         sc_port<placeInterface, N, SC_ALL_BOUND> in;  //^SC_ALL_BOUND all inputs and output have to be
connected
18         sc_port<placeInterface, M, SC_ALL_BOUND> out;
19         sc_port<placeInterface, L, SC_ZERO_OR_MORE_BOUND> inhibitors;    //^SC_ZERO_OR... means my Inhibitor
doesn't have to be connected to something, see task 4 IDLE place
20
21         SC_CTOR(transition): in("in"), out("out"), inhibitors("inhibitors"){
22
23         void fire()
24         {
25             //check Places
26             int counterToken = 0;
27             int counterInh = 0;
28             for (int i = 0; i < N; i++)
29             {
30                 if (in[i]->testTokens())
31                 {
32                     counterToken++;
33                 }
34             }
35
36
37             for (int i = 0; i < L; i++)
38             {
39                 if (inhibitors[i]->getTokens() > 0)
40                 {
41                     counterInh++;
42                 }
43             }
44
45             //if all previous places have tokens
46             //if (counterToken == N)  //-----Task before 5
47             if ((counterToken == N) && (counterInh == 0))  //----- Task 5
48             {
49                 cout << this->name() << ": Fired" << endl;
50
51                 for (int i = 0; i < N; i++){
52                     //in[i]->removeTokens(1);    //-----Task 1 and 2
53                     in[i]->removeTokens();
54                     //cout << "test" << endl;
55                 }
56                 for (int i = 0; i < M; i++){

```

```
57         //out[i]->addTokens(1); //-----Task 1 and 2
58         out[i]->addTokens();
59     }
60
61     }else{
62         cout << this->name() << ": NOT Fired" << endl;
63     }
64 }
65 };
66
67
68
69
70 #endif // TRANSITION_H
71
```


subnet.h

```
1  #ifndef SUBNET_H
2  #define SUBNET_H
3
4  #include <systemc.h>
5
6  #include "place.h"
7  #include "transition.h"
8
9
10
11 SC_MODULE(subnet)
12 {
13     public:
14     placeChannel<1, 1> ACTIVE;
15
16     transition<1,1,1> ACT;
17     transition<1,1> RD, PRE, WR;
18
19
20     SC_CTOR(subnet): ACT("ACT"), RD("RD"), PRE("PRE"), WR("WR"), ACTIVE(0)
21     {
22         ACT.inhibitors.bind(ACTIVE);
23         ACT.out.bind(ACTIVE);
24         RD.in.bind(ACTIVE);
25         RD.out.bind(ACTIVE);
26         PRE.in.bind(ACTIVE);
27         WR.in.bind(ACTIVE);
28         WR.out.bind(ACTIVE);
29     }
30 };
31
32 #endif // SUBNET_H
33
```

main.cpp

```
1  #include <systemc.h>
2
3  #include "place.h"
4  #include "transition.h"
5  #include "subnet.h"
6
7  using namespace std;
8
9
10
11 //-----Task 5
12 SC_MODULE(toplayer)
13 {
14     subnet s1, s2;
15     placeChannel<1,1> IDLE;
16
17     SC_CTOR(toplayer): s1("s1"), s2("s2"), IDLE(2)
18     {
19         s1.ACT.in.bind(IDLE);
20         s1.PRE.out.bind(IDLE);
21         s2.ACT.in.bind(IDLE);
22         s2.PRE.out.bind(IDLE);
23         SC_THREAD(process);
24     }
25
26     void process()
27     {
28         while(true)
29         {
30             wait(10, SC_NS);
31             s1.ACT.fire();
32             wait(10, SC_NS);
33             s1.ACT.fire();
34             wait(10, SC_NS);
35             s1.RD.fire();
36             wait(10, SC_NS);
37             s1.WR.fire();
38             wait(10, SC_NS);
39             s1.PRE.fire();
40             wait(10, SC_NS);
41             s1.ACT.fire();
42             wait(10, SC_NS);
43             s2.ACT.fire();
44             wait(10, SC_NS);
45             s2.ACT.fire();
46             wait(10, SC_NS);
47             s1.PRE.fire();
48             wait(10, SC_NS);
49             s2.PRE.fire();
50             wait(10, SC_NS);
51             sc_stop();
52         }
53     }
54 };
55
56 int sc_main(int, char**)
57 {
```

```
58     toplayer top("t");
59
60     sc_start();
61     return 0;
62 }
63
```