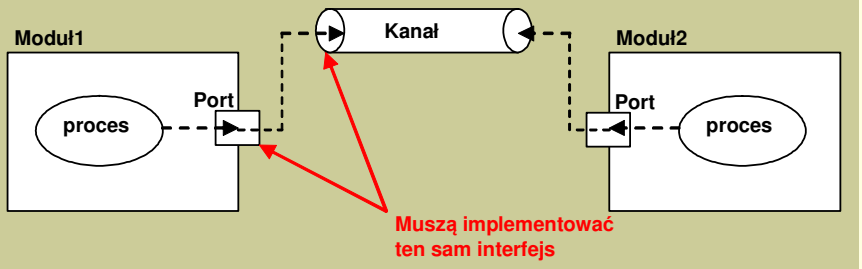
Systemy wbudowane

Wykład 7: Modelowanie systemów wbudowanych w SystemC, cz.II – komunikacja między procesami

Synchronizacja: zdarzenia

- ***sc_event*** e1;
- *e1.* ***notify()****, e1.****notify****(czas)* - generacja zdarzenia
- *e1.****cancel****()* - zaniechanie zdarzenia
- *sygnał.****value_changed_event****()* – zdarzenie związane ze zmiana sygnału
- *sygnał.****posedge_event****(), sygnał.****negedge_event****()* – zdarzenia związane z wybranymi zboczami zmian sygnałów (tylko jednobitowe)

Komunikacja między modułami



Transaction Level Modeling

Typy kanałów komunikacyjnych

- Kanały proste:
  - Klasa bazowa: sc_prim_channel
- Kanały hierarchiczne:
  - Klasa bazowa: sc_module

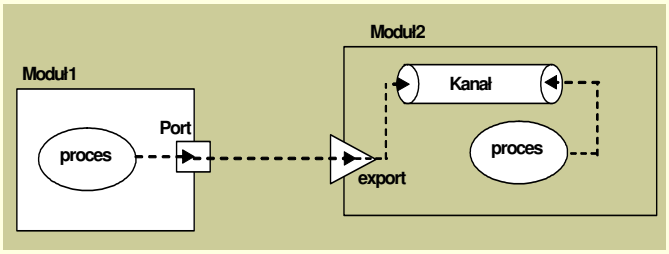
Każdy kanał może implementować wiele interfejsów!

Porty

- Klasa bazowa:
  - sc_port<interfejs, N=1, POL=SC_ONE_OR_MORE_BOUND>
- Przyłączanie portu do kanału:
  - moduł.port(kanał)
  - (*moduł)(kanał1, kanał2,…)

Port: sc_export

- Eksportowanie implementacji interfejsu/kanału



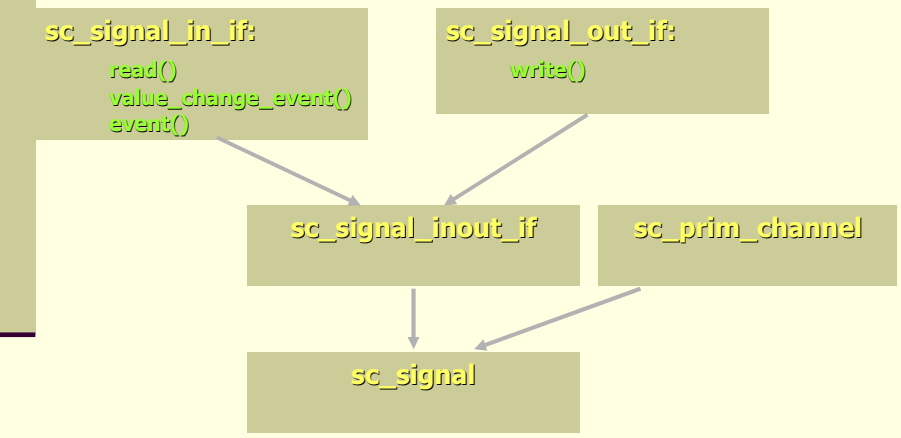
Standardowe kanały komunikacyjne

- Sygnały: sc_signal, sc_buffer, sc_clock, sc_signal_resolved, sc_signal_rv
  - interfejsy: sc_signal_inout_if
- Fifo: sc_fifo
  - Intefejsy: sc_fifo_in_if, sc_fifo_out_if
- Mutex: sc_mutex
  - Interfejsy: sc_mutex_if
- Semafor: sc_semaphore
  - Interfejsy: sc_semaphore_if
- Kolejka zdarzeń: sc_event_queue
  - Interfejsy: sc_event_queue_if

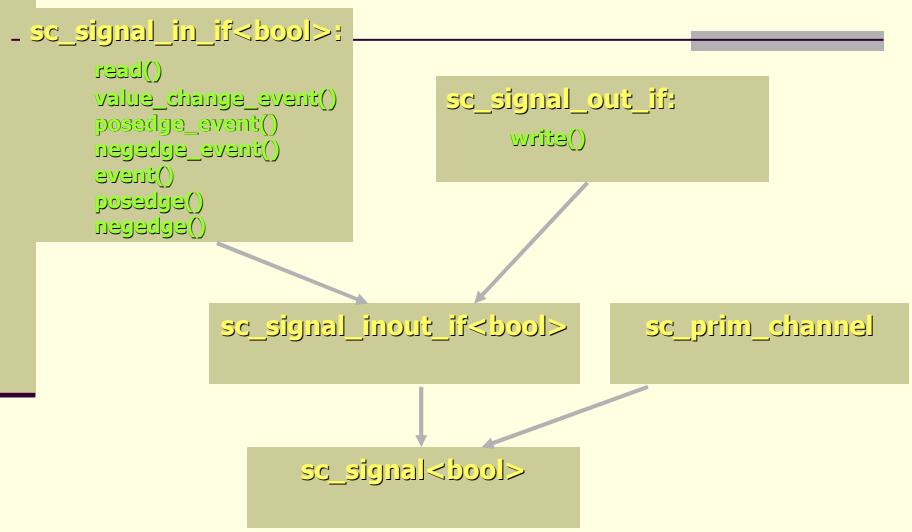
Sygnały

- jednoelementowy bufor
- komunikacja asynchroniczna
- komunikacja bez blokowania

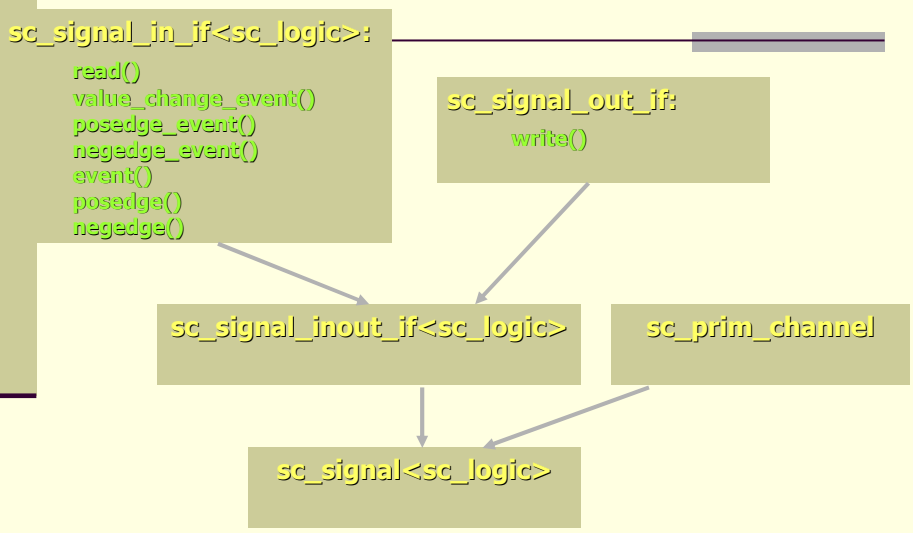
sc_signal



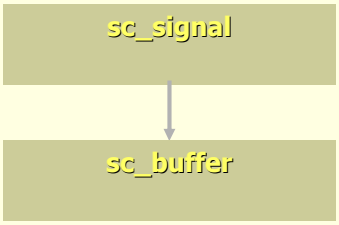
sc_signal<bool>



sc_signal<sc_logic>



sc_buffer



Każdy zapis powoduje generację zdarzenia!

sc_clock

sc_signal<bool>

↓

sc_clock:
- period()
- duty_cycle()
- start_time()
- posedge_first()

Sygnał tylko do odczytu!

sc_signal_resolved

sc_signal<sc_logic>

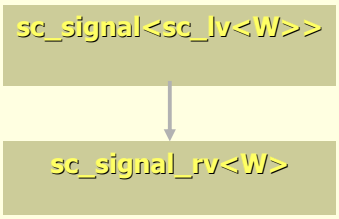
↓

sc_signal_resolved

Można łączyć kilka wyjść!

	'0'	'1'	'Z'	'X'
'0'	'0'	'X'	'0'	'X'
'1'	'X'	'1'	'1'	'X'
'Z'	'0'	'1'	'Z'	'X'
'X'	'X'	'X'	'X'	'X'

sc_signal_rv



Standardowe porty implementujące interfejsy sygnałów

- sc_in: sc_port<sc_signal_in_if,1>
- sc_in<bool>: sc_port<sc_signal_in_if<bool>,1>
- sc_in<sc_logic>: sc_port<sc_signal_in_if<sc_logic>,1>
- sc_inout: sc_port<sc_signal_inout_if,1>
- sc_inout<bool>: sc_port<sc_signal_inout_if<bool>,1>
- sc_inout<sc_logic>: sc_port<sc_signal_inout_if<sc_logic>,1>
- sc_out: sc_port<sc_signal_out_if,1>
- sc_in_resolved: sc_in<sc_logic>
- sc_inout_resolved: sc_inout<sc_logic>
- sc_out_resolved: sc_out<sc_logic>
- sc_in_rv<W>: sc_in<sc_lv<W>>
- sc_inout_rv<W>: sc_inout<sc_lv<W>>
- sc_out_rv<W>: sc_out<sc_lv<W>>

Przykład komunikacji wykorzystującej sygnały

M1:	M2:	Main:
sc_out<bool> o1;	sc_in<bool> i1;	sc_signal<bool> s;
….	…	M1->o1(s);
o1.write(true);	ss=i1.read();	M2->i1(s);

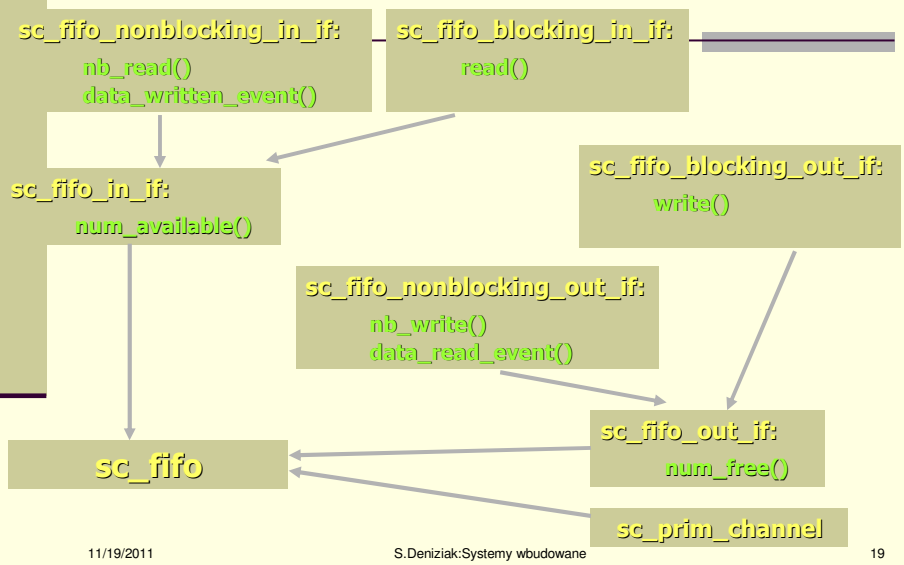
Lub:

```
o1=true;      ss=i1;
```


FIFO

- wieloelementowy bufor
- komunikacja asynchroniczna
- komunikacja bez blokowania lub z blokowaniem

sc_fifo



Standardowe porty implementujące interfejsy FIFO

- sc_fifo_in: sc_port<sc_fifo_in_if,0>
- sc_fifo_out: sc_port<sc_fifo_out_if,0>

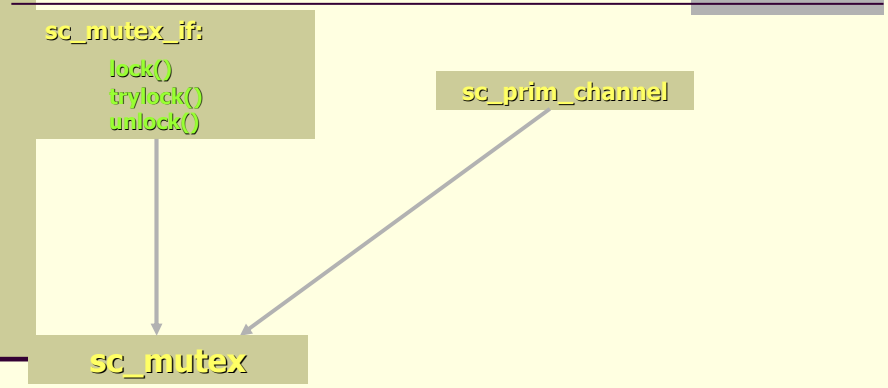
Przykład

```
SC_MODULE(writer){
 sc_port<sc_fifo_out_if<int> > out;
 void main_action(){
  int val = 0;
  while( true ) {
   wait( 10, SC_NS );
   for( int i = 0; i < 20; i ++ )
     out->write( val ++ );
  }
 }
 SC_CTOR(writer){
  SC_THREAD(main_action);
 }
};
```

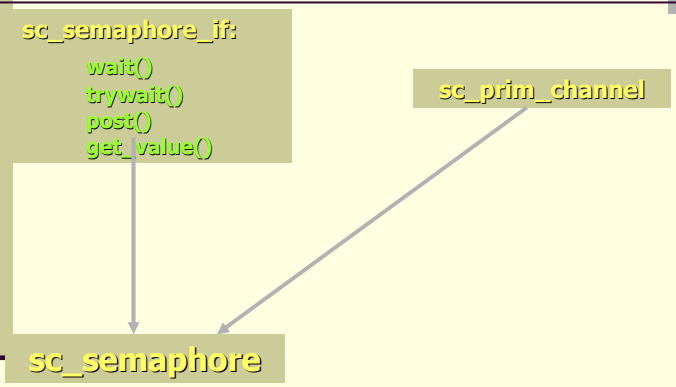
```
SC_MODULE(reader){
 sc_port<sc_fifo_in_if<int> > in;
 void main_action(){
  int val;
  while( true ) {
   wait( 10, SC_NS );
   for( int i = 0; i < 15; i ++ )
    in->read( val );
  }
 }
 SC_CTOR(reader){
  SC_THREAD( main_action );
 }
};
```

```
int sc_main(){
sc_fifo<int> fifo( 10 );
writer w("writer"); reader r("reader");
w.out( fifo ); r.in( fifo );
 ...
```

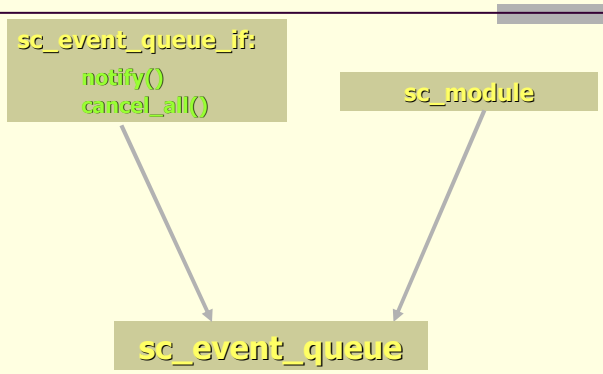

sc_mutex



sc_semaphore



sc_event_queue



Przykład

```
sc_event_queue EQ;
SC_CTOR(Mod)  {
        SC_THREAD(T);
        SC_METHOD(M);
        sensitive << EQ;
        dont_initialize();
}
void T() {
EQ.notify(2, SC_NS); // M aktywuje się w  chwili 2ns
EQ.notify(1, SC_NS); // M aktywuje się w chwili 1ns
EQ.notify(SC_ZERO_TIME); // M aktywuje się w chwili 0ns
EQ.notify(1, SC_NS); // M aktywuje się w chwili 1ns
}
```

Aktywacje w kolejnych cyklach symulacyjnych

Koniec