technische universität
dortmund

fakultät für informatik
informatik 12

Synthese Eingebetteter Systeme

Sommersemester 2011

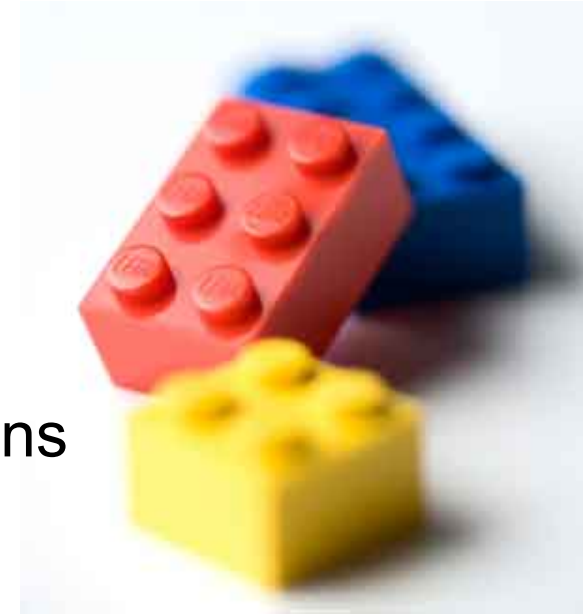
4 – SystemC-Kommunikation

Michael Engel
Informatik 12
TU Dortmund

2011/04/15

SystemC-Komponenten

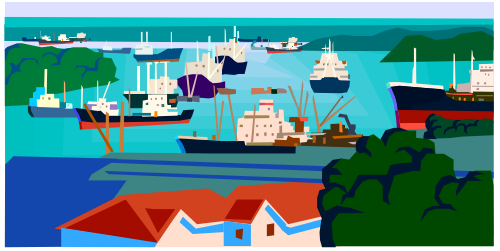
- Zeitmodell
- Datentypen
- Instanziierung
- Stolpersteine
- Module, Hierarchie und Struktur
- Nebenläufigkeit
- Ereignisse, Sensitivität und Notifications
- Kommunikation
- Ports, Interfaces und Kanäle



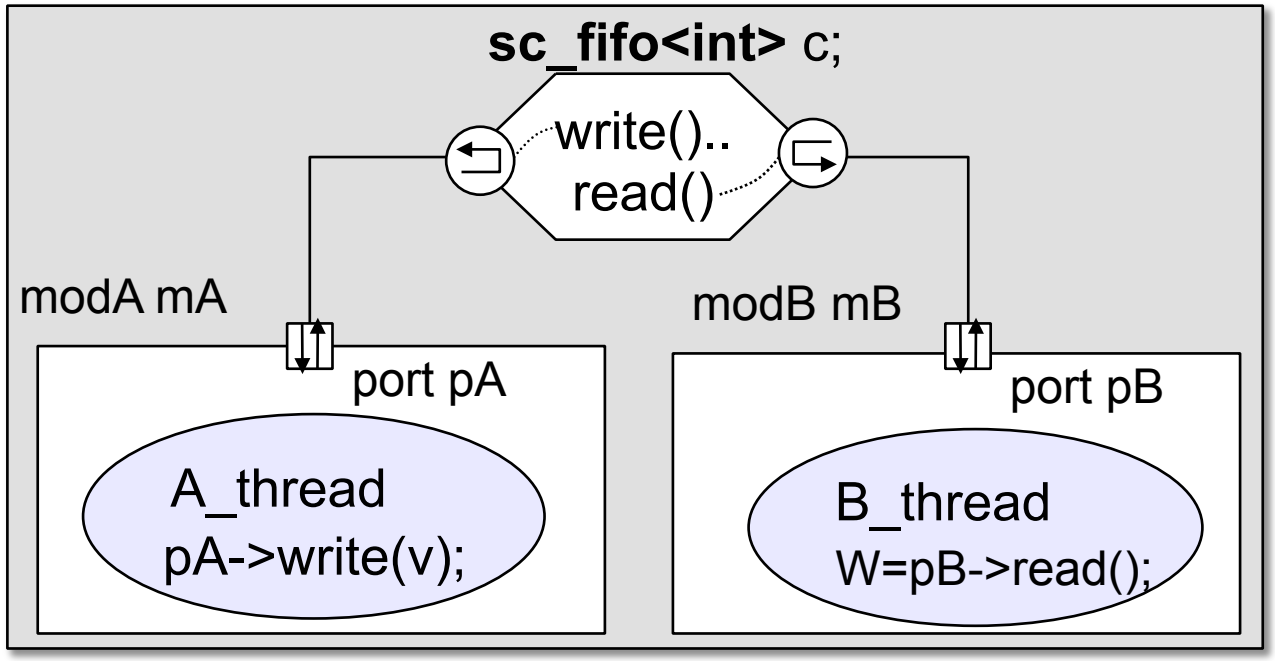
Ports

- Wir können bereits Hierarchien definieren
  - SC_MODULEs, Instanziierungen
- Wie können lokale Module kommunizieren?
  - Implizite Schnittstellen?
    - Globale Variablen vermeiden!

  - **Ports** sind wohldefinierte Schnittstellen, die über Modulgrenzen hinweg interagieren
  - Ports werden über **interfaces** mit **channels** verbunden.
    - Diese verkapseln die Ports
- Ports sind Pointer auf Kanäle



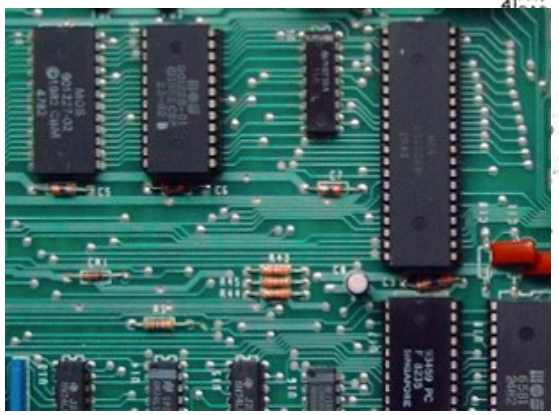
Beispiel

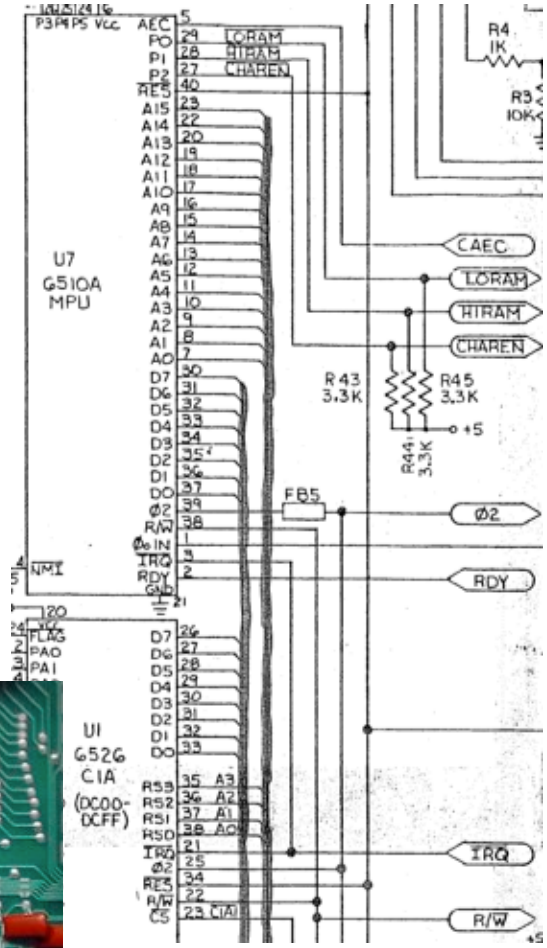


- "A_thread in Modul modA kommuniziert einen Wert, der in der lokalen Variablen v steht, durch Aufruf der **write**-Methode des Kanals c im übergeordneten Modul."
- A_thread muss dafür nur die **write**-Methode kennen
- Trennung von Belangen durch ☞*Interfaces.*

Analogie zu Hardware-Systemen

- Chips haben interne, verborgene Funktionalität
- Schittstelle nach außen:(elektrische) *Signale*
- Verbindung von Signalen von Chips über *Leiterbahnen*
  - Leiterbahnen auf höherer Ebene (Platine) -> separates Layout
  - Entwurf des Chips unabhängig von Platinenentwurf
- Austausch von Chips durch *schnittstellen-kompatible* Chips mit anderer Funktionalität möglich
  - z.B. Prozessoren mit und ohne FPU: Intel 80486DX vs. SX





Variabilität in Entwürfen: Virtuelle Methoden und Polymorphismus in C++

Schnittstelle: Abstrakte Klasse

```
class My_Interface {
    virtual T1 My_methA(...)=0;
    virtual T2 My_methC(…)=0;
};
```

Abstrakte Klasse:
– Rein virtuelle Methoden
– Keine Daten

Implementierung 1

```
class My_Derived1:
 public My_Interface {
    T1 My_methA(...) { … }
    T2 My_methC(…) { … }
 private:
    T5 My_data1;
};
```

Implementierung 2

```
class My_Derived2:
 public My_Interface {
    T1 My_methA(...) { … }
    T2 My_methC(…) { … }
 private:
    T3 My_private_method(…);
    T3 My_data2;
};
```

Variabilität in Entwürfen: Virtuelle Methoden und Polymorphismus in C++

Schnittstelle: Abstrakte Klasse

```
class My_Interface {
    virtual T1 My_methA(...)=0;
    virtual T2 My_methC(…)=0;
};
```

Abstrakte Klasse:
– Rein virtuelle Methoden
– Keine Daten

Polymorphismus: Aufruf von My_methA bewirkt
- Aufruf von My_Derived1::My_methA, wenn Objekt vom Typ My_Derived1 ist
- Oder Aufruf von My_Derived2::My_methA, wenn Objekt vom Typ My_Derived2 ist

Implementierung 1

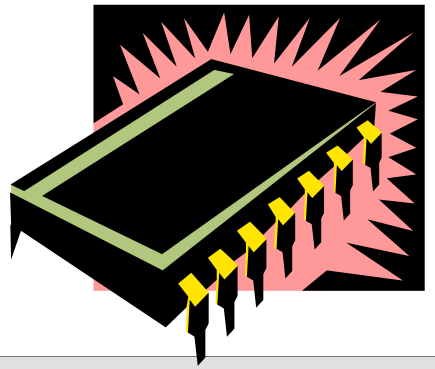
```
class My_D
 public My_Interface {
    T1 My_methA(...) { … }
    T2 My_methC(…) { … }
 private:
    T5 My_data1;
};
```

Implementierun

```
 public My_Interface {
    T1 My_methA(...) { … }
    T2 My_methC(…) { … }
 private:
    T3 My_private_method(…);
    T3 My_data2;
};
```

Variabilität in Entwürfen: Beispiel

```
struct multiport_memory_arch: public my_interface {
      virtual void write(unsigned addr, int data) {
        mem[addr] = data;
      } // Ende von write
      virtual int read(unsigned addr) ) {
        return mem[addr];
      } // Ende von read
      private:  int mem[1024];
};
```



Zwei Modelle für Speicher – *identische Schnittstellen*

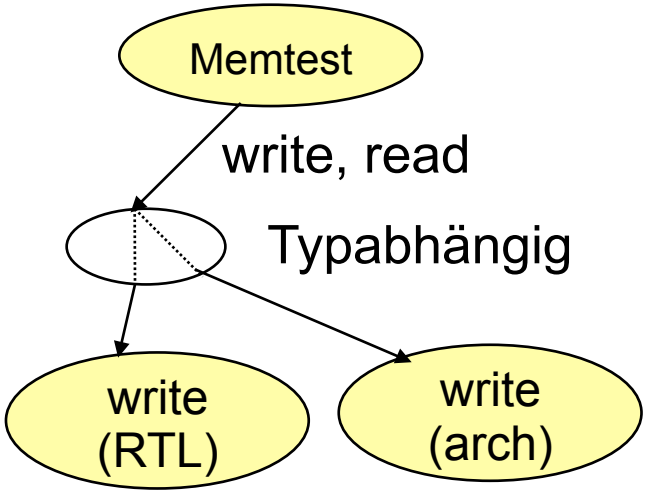
```
struct multiport_memory_RTL: public my_interface {
  virtual void write(unsigned addr, int data) {
            // Komplexe Implementierung von write
   } // Ende von write
  virtual int read(unsigned addr) ) {
            // Komplexe Implementierung von read
  } // Ende von read
  private:   // Komplexe Details des Speichers
};
```

Beispiel: Anwendung bei Speichertests

```
void memtest(my_interface mem) {
        // Komplexer Speichertest
   }

multiport_memory_arch fast;
multiport_memory_RTL slow;
memtest(fast);
memtest(slow);
```

Aufrufgraph:



Die gleiche Speichertestmethode verwendet Modelle des Speichers mit unterschiedlicher Präzision

Definition von Schnittstellen und Kanälen

- **SystemC interface:** Abstrakte Klasse, erbt von **sc_interface**
  - Stellt nur rein virtuelle Methodendeklarationen zur Verfügung, die von SystemC *channels* und *ports* referenziert werden
  - Enthält keine Methodenimplementierungen oder Daten
- **SystemC channel:** Klasse, die ≥ 1interface-Klasse implementiert
  - Erbt von **sc_channel** oder **sc_prim_channel**
  - Channel implementiert *alle* Methoden ererbter interface-Klassen

Durch die Verwendung von interfaces zur Verbindung an channels können Module *unabhängig* von Implementierungsdetails des Kommunikationskanals implementiert werden!

Einfache Port-Deklarationen in SystemC

- **Definition:** Ein **SystemC port** ist eine Klasse, die von einem SystemC **interface** erbt und diese als template nutzt
- Ports ermöglichen den Zugriff von channels über Modulgrenzen hinweg

Syntax:

sc_port<*interface*> *portname*;

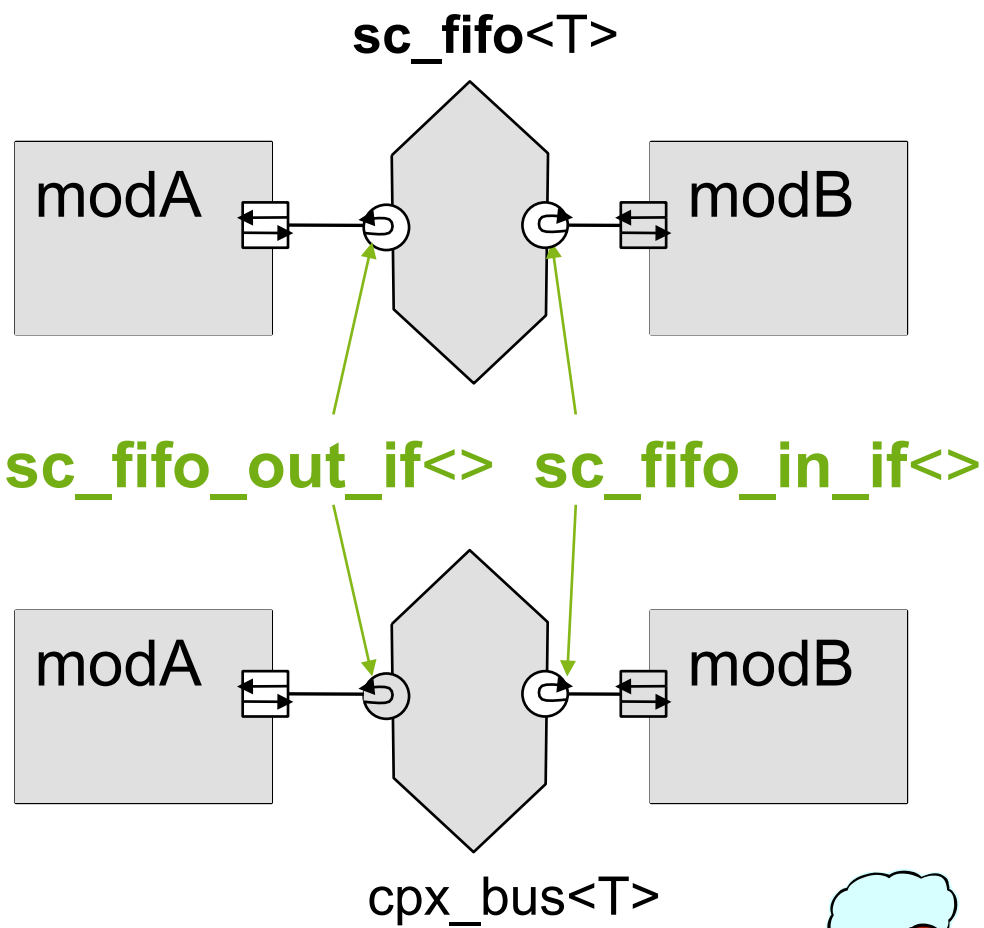
// verwendet in Klassendefinition des Moduls

Beispiel:

```
SC_MODULE(stereo_amp) {
 sc_port<sc_fifo_in_if<int> > soundin_p;
 sc_port<sc_fifo_out_if<int> > soundin_p;
 …};
```

Leerzeichen nicht vergessen!

Möglichkeiten von interfaces



Im oberen Entwurf sind Module über eine FIFO verbunden, im unteren über einen komplexen Bus.

Wenn die interfaces, hier **sc_fifo_out_if**<> und **sc_fifo_in_if**<>, gleich sind, ist bei Änderung des Kanaltyps keine Änderung an modA und modB erforderlich.

Ermöglicht Wiederverwendung von *intellectual property* (IP).

Ports

- Ports haben einen **Modus** (Richtung) und einen **Typ**
  - **Modus:** in, out, inout
  - **Typ**: C++-Typ, SystemC-Typ, benutzerdefinierter Typ
- Beispiele:
  - **sc_in**<*type*> *in_port_name*; // deklariert Eingang
  - **sc_out**<*type*> *out_port_name*; // deklariert Ausgang
  - **sc_inout**<*type*> *inout_port_name*; // bidirektionaler Port
  - **sc_out<int>** result [32]; // Vektorport/Portarray

Wo werden Ports deklariert?

Beispiel:

```
SC_MODULE(adder) {
    sc_in<int> a;                  // schlecht: Verwendung von int!
    sc_in<sc_uint<32> > b;         // besser: SystemC-Typ!
    sc_out<bool> c;
    // Prozesse usw.
    SC_CTOR(adder) {
        // Konstruktor-Body
        // Prozessdeklarationen, Sensitivitäten usw.
        c.initialize(0);
    }
};
```

Ports

- Spezielle Ports können von der Klasse **sc_port** abgeleitet werden
- SystemC stellt Signal-Ports zur Verfügung:

```
template<class T>
  class sc_in : public sc_port<sc_signal_in_if<T> > …;

template<class T>
  class sc_inout : public sc_port<sc_signal_inout_if<T> > ..
```

Zugriff auf Ports aus Prozessen

- **sc_port** überlädt den **–>** Operator für Portzugriffe

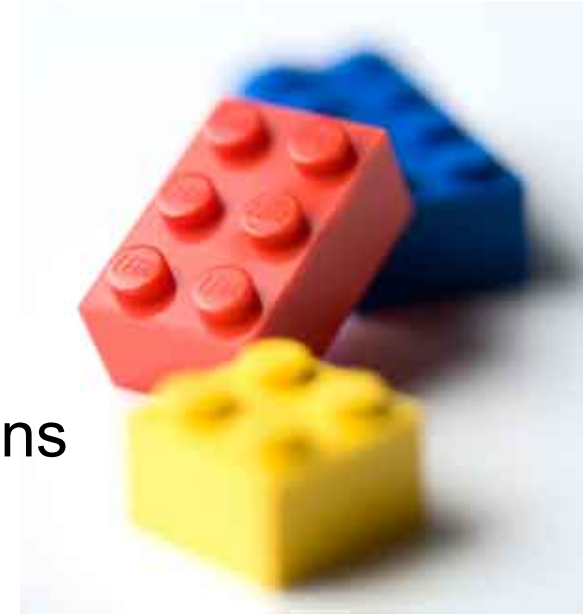
Beispiel:

```
void Video_Mixer::Mixer_thread() {
  …
  switch (control–>read()) {
    case MOVIE: K.write(0.0f);    break;
    case MENU:  K.write(1.0f);    break;
    case FADE:  K.write(0.5f);    break;
    default: status–>write(Error); break;
   }
…
}
```

Jede interface-Methode kann über den Port aufgerufen werden.
read() und **write**() sind Methodenaufrufe des interface.

SystemC-Komponenten

- Zeitmodell
- Datentypen
- Instanziierung
- Stolpersteine
- Module, Hierarchie und Struktur
- Nebenläufigkeit
- Ereignisse, Sensitivität und Notifications
- Kommunikation
- Ports, Interfaces und Kanäle



Interfaces

- Menge von Operationen
- Geben nur *Signatur* einer Operation an:
  - Name, Parameter, Rückgabewert
- Alle SystemC-Interfaces von **sc_interface** abgeleitet
- Enthält virtuelle Funktion **register_port**()
  - Verbinder Port mit Channel durch das Interface
  - Parameter: Port-Objekt und Typname des Interfaces, das der Port implementiert

sc_mutex_if

sc_mutex stellt ebenfalls ein Interface zur Verfügung, das mit Ports verwendet werden kann

```
struct sc_mutex_if: virtual public sc_interface {
  virtual int lock()    = 0;
  virtual int trylock() = 0;
  virtual int unlock() = 0;
};
```

Keine Methode für Ereignis-Sensitivität

sc_semaphore_if

sc_semaphore stellt ebenfalls ein Interface zur Verfügung, das mit Ports verwendet werden kann

```
struct sc_semaphore_if: virtual public sc_interface {
  virtual int wait()     = 0;
  virtual int trywait() = 0;
  virtual int post()     = 0;
  virtual int get_value() const = 0;
};
```

Auch hier: Keine Methode für Ereignis-Sensitivität

sc_fifo interfaces (1)

Es gibt zwei Interfaces für **sc_fifo**:

1. **sc_fifo_out_if<>**
2. **sc_fifo_in_if<>**

Zusammen stellen sie alle Funktionen für **sc_fifo** zur Verfügung

sc_fifo interfaces (2): sc_fifo_out_if<>

```
template <class T>
struct sc_fifo_out_if: virtual public sc_interface {
  virtual void write(const T&) =0;
  virtual bool nb_write(const T&) =0;
  virtual int num_free() const =0;
  virtual const sc_event& data_read_event() const =0;
  …
};
```

sc_fifo interfaces (3): sc_fifo_in_if<>

```
template <class T>
struct sc_fifo_in_if: virtual public sc_interface {
  virtual void read( T& ) = 0;
  virtual T read() = 0;
  virtual bool nb_read( T& ) =0;
  virtual int num_available() const =0;
  virtual const sc_event& data_written_event() const =0;
  …
};
```

sc_signal interfaces

Interfaces für **sc_signal**:

1. **sc_signal_out_if<>**
2. **sc_signal_in_if<>**
3. **sc_signal_inout_if<>**

Zusammen stellen sie alle Methoden für **sc_signal** bereit

- **Beispiel:**

```
template <class T>
struct sc_signal_inout_if: virtual public
sc_signal_in_if<T>
   { virtual void write(const T&) =0;   … };
#define sc_signal_out_if sc_signal_inout_if
```

  - **sc_signal_inout_if** besitzt alle Methoden für die anderen beiden Interfaces

Ports und statische Sensitivität

- Oft sollen Prozesse beschrieben werden, die auf Änderungen an Ports reagieren
- Ports sind eigentlich Pointer
  - Können bis Auswertungsende undefiniert sein
  - IEEE 1666 legt dies fest
  - Zum Zeitpunkt der Analyse der statischen Sensitivität undefiniert

☞ Verbindung zu Ereignissen muss in nachfolgendem Schritt erfolgen. Events werden “später” lokalisiert

☞ **Event-Finder** sind Methoden, um in statischen Sensivitätslisten die Sensitivität auf Ports festzulegen

☞ Einige spezielle Ports besitzen Methoden, die Event-Finder beinhalten

  ☞ Erspart eigene Implementierung!

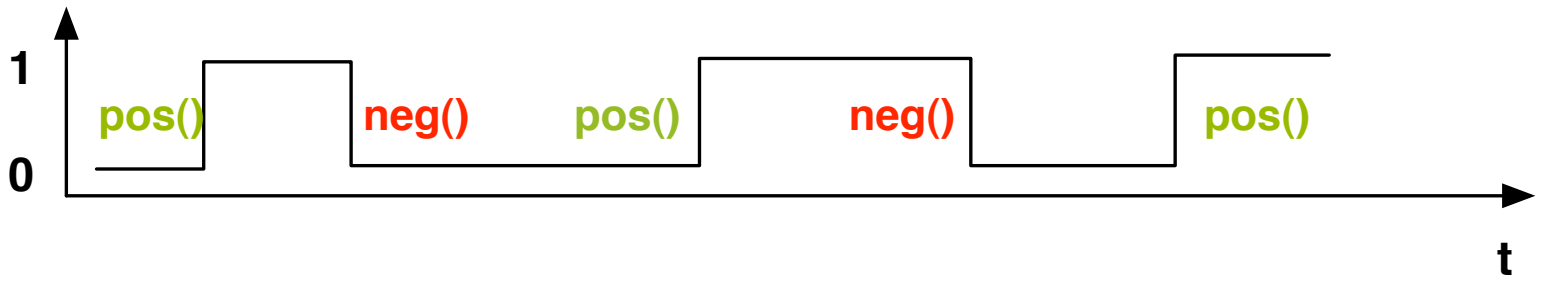
sc_in<T>

- **sc_in:** Spezialisierung von **sc_port<sc_signal_in_if**<T> >
- Methoden beinhalten Event-Finder:
  - **sc_in**<T> name_sig_ip;
  - **sensitive** << name_sig_ip.**value_changed**();
  - // Für T= **bool** und T= **sc_logic**:
  - **sc_in<bool>** name_bool_sig_ip;
  - **sc_in<sc_logic>** name_log_sig_ip;
  - **sensitive** << name_log_ip_sig.**pos**();
  - **sensitive** << name_bool_ip_sig.**neg**();

“.”-Schreibweise nicht empfohlen, aber in Deklaration notwendig
“->” -Schreibweise wird bevorzugt (…soweit die Theorie)

Sensitivitätsarten

- Hardware-Äquivalent: **Flankentriggerung**
  - Positive und negative Signalübergänge
- // Für T= **bool** und T= **sc_logic**:
- **sc_in<bool>** name_bool_sig_ip;
- **sc_in<sc_logic>** name_log_sig_ip;
- **sensitive** << name_log_ip_sig.**pos**();
- **sensitive** << name_bool_ip_sig.**neg**();

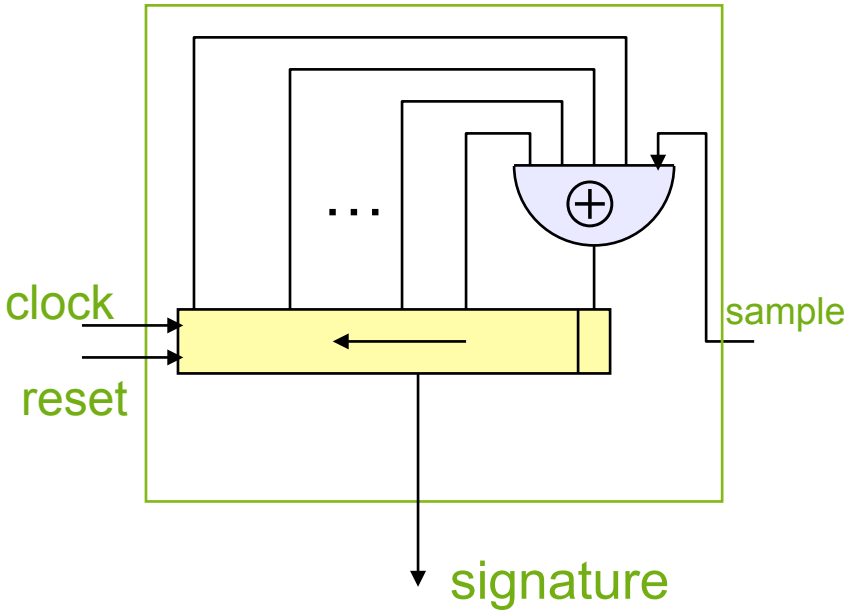


sc_out<T>

- Für **sc_out** existieren entsprechend Methoden **value_changed()** und **initialize()**

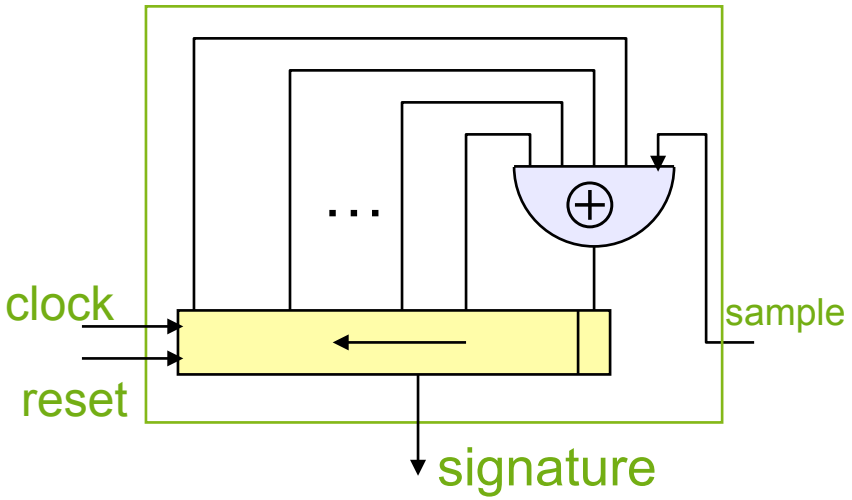
Beispiel: Header

```
// Datei LFSR_ex.h
#ifndef LFSR_EX_H
#define LFSR_EX_H
SC_MODULE (LFSR_ex) // Ports
  sc_in<bool> sample;
  sc_out<sc_int<32> > signature;
  sc_in<bool> clock;
  sc_in<bool> reset;
  SC_CTOR(LFSR_ex){
    SC_METHOD(LFSR_ex_method);
    sensitive << clock.pos() << reset;
    signature.initialize(0);
   }
  void LFSR_ex_method();
  sc_int<32> LFSR_reg;  };
#endif
```

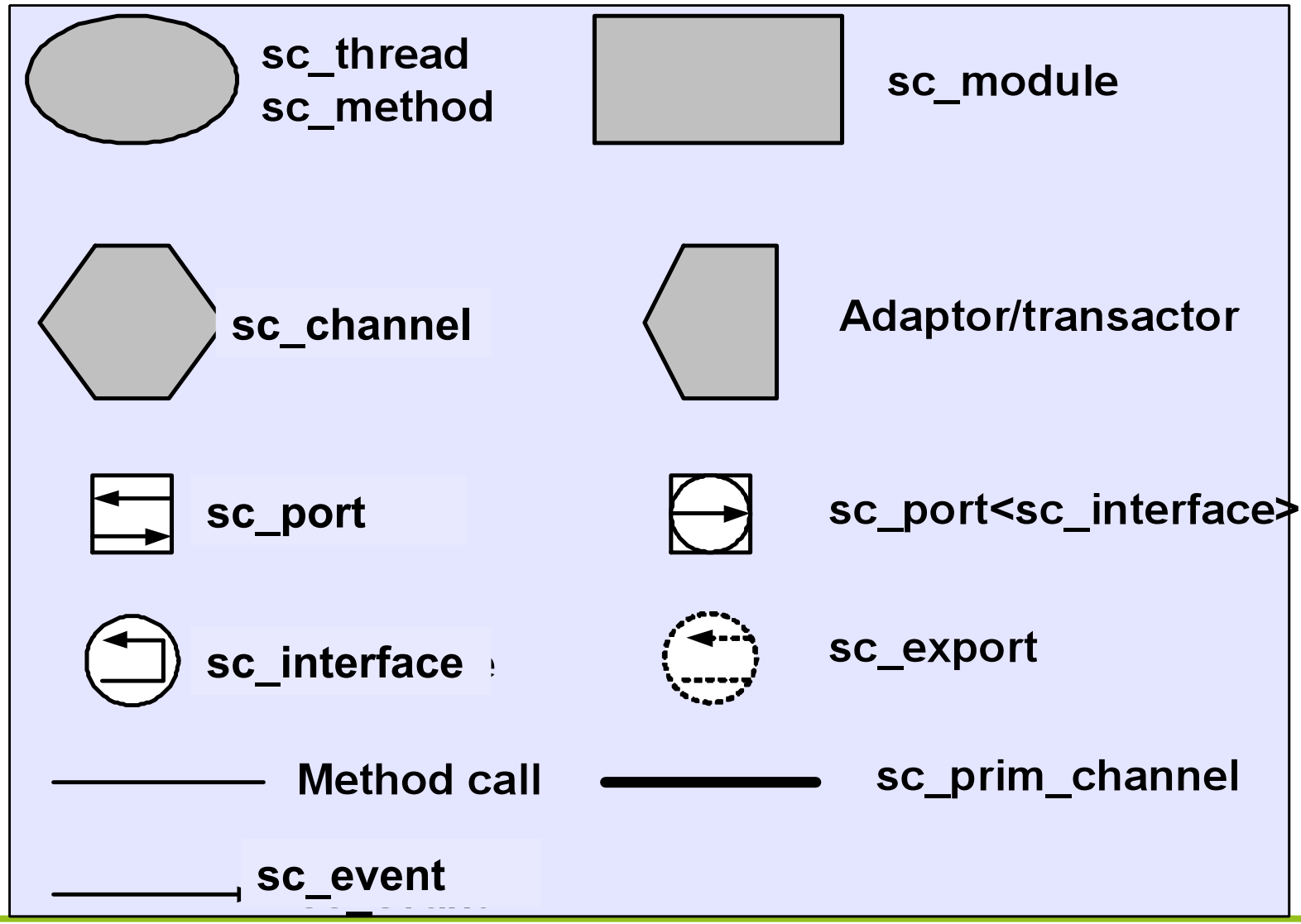


Beispiel: Implementierung

```
// Datei LFSR_ex.cpp
#include <systemc.h>
#include "LFSR_ex.h"
void LFSR_ex::LFSR_ex_method() {
  if (reset->read() == true) {
     LFSR_reg = 0;
      signature->write(LFSR_reg);
  } else {
    bool lsb = LFSR_reg[31] ^ LFSR_reg[25] ^ LFSR_reg[22]
    ^ LFSR_reg[21] ^ LFSR_reg[15] ^ LFSR_reg[11] ^ LFSR_reg[10]
    ^ LFSR_reg[ 9] ^ LFSR_reg[ 7] ^ LFSR_reg[ 6] ^ LFSR_reg[ 4]
    ^ LFSR_reg[ 3] ^ LFSR_reg[ 1] ^ LFSR_reg[ 0] ^ sample->read();
  LFSR_reg.range(31,1) = LFSR_reg.range(30,0); LFSR_reg[0] = lsb;
  signature->write(LFSR_reg);
 }
}
```



Standard für grafische Darstellung



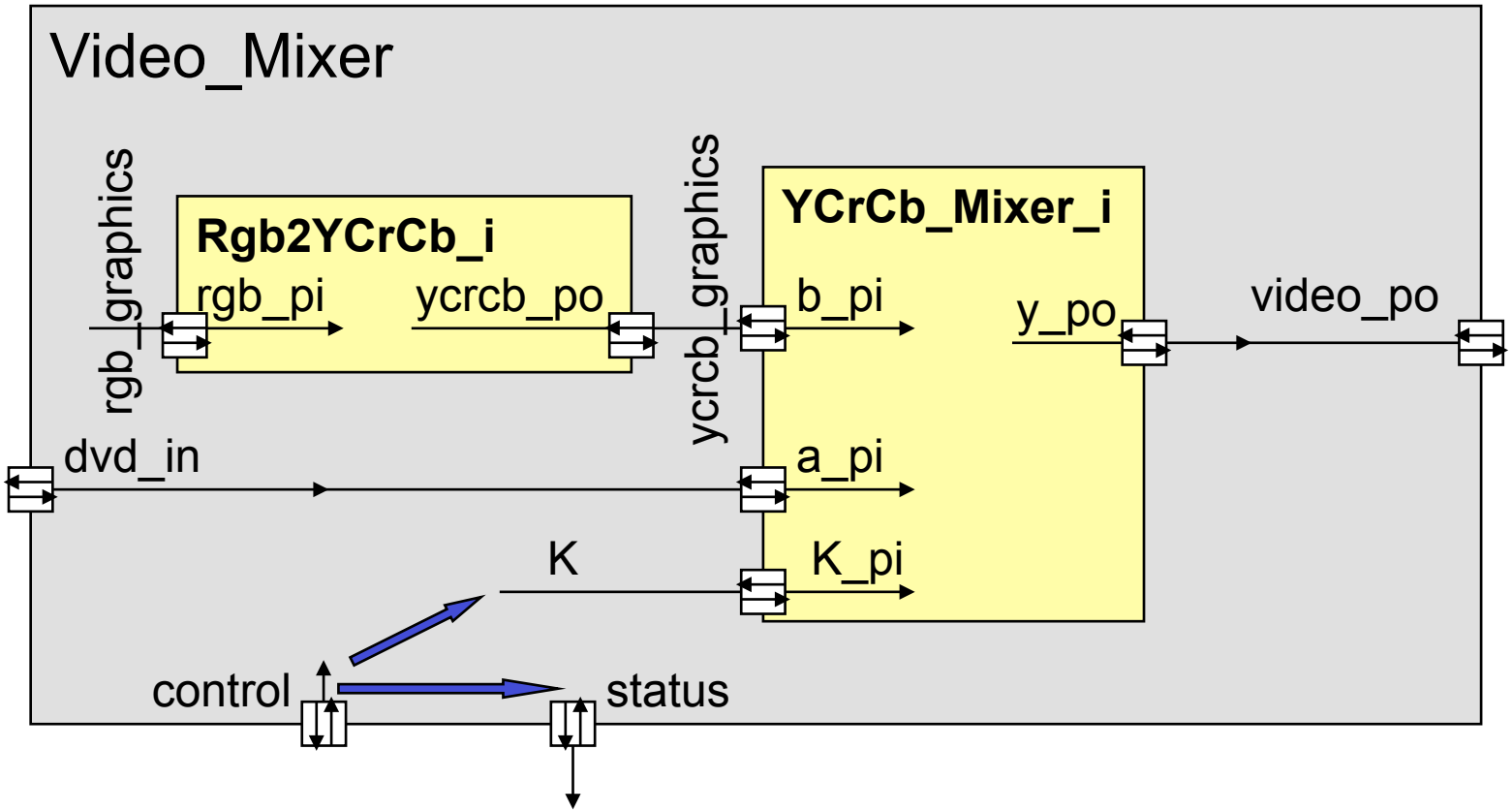
Verbindungsmechanismen von Ports

- Module werden mit Channels verbunden, nachdem alle Module und Kanäle instanziiert wurden

Syntax:

mod_instance(*channel_instance*, …); // Reihenfolge (riskant)
mod_inst.portname(*channel_instance*); // Explizit (sicherer)

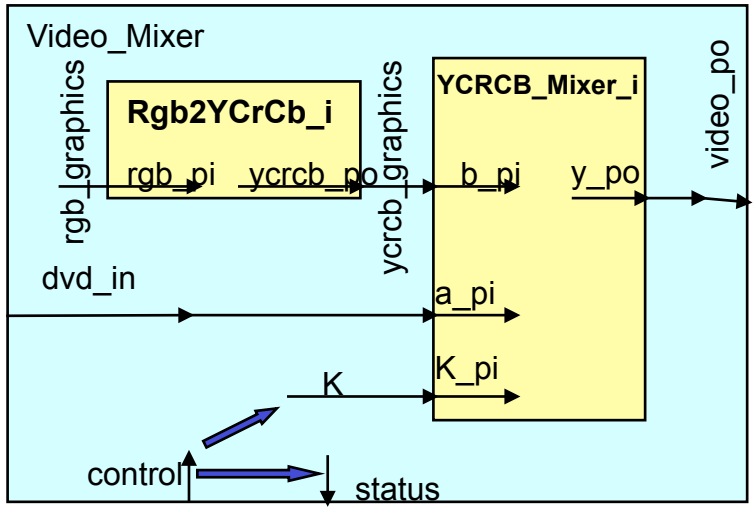
Beispiel: Video-Mixer



Darstellung so (un)vollständig wie der Quelltext

Beispiel: Video-Mixer (2)

```
// Datei: Rgb2YCrCb.h
SC_MODULE(Rgb2YCrCb) {
    sc_port<sc_fifo_in_if<RGB_frame> > rgb_pi;
    sc_port<sc_fifo_out_if<YCRCB_frame> > ycrcb_po;
};
```

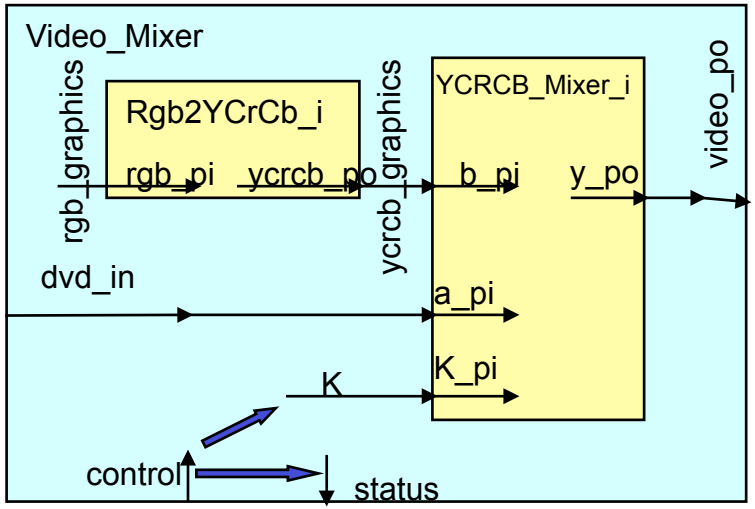



```
// Datei: YCRCB_Mixer.h
SC_MODULE(YCRCB) {
 sc_port<sc_fifo_in_if<float> > K_pi;
 sc_port<sc_fifo_in_if<YCRCB_frame> > a_pi, b_pi;
 sc_port<sc_fifo_out_if<YCRCB_frame> > y_po;
};
```

Beispiel: Video-Mixer (3)

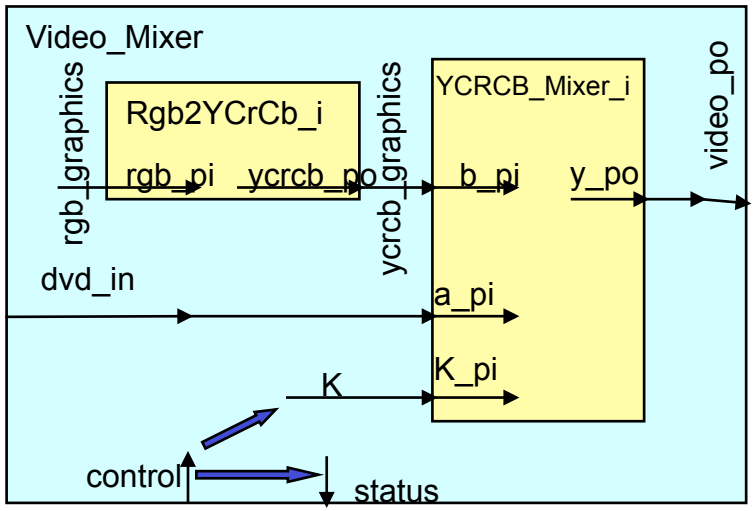
```
// Datei: Video_Mixer.h
SC_MODULE (VIDEO_MIXER) {
// Ports
sc_port<sc_fifo_in_if<YCRCB_frame> > dvd_in;
sc_port<sc_fifo_out_if<YCRCB_frame> > video_po;
sc_port<sc_fifo_in_if<MIXER_ctrl> > control;
sc_port<sc_fifo_out_if<MIXER_state> > status;
// lokale Kanäle
sc_fifo<float> K;
sc_fifo<RGB_frame> rgb_graphics;
sc_fifo<YCRCB_frame> ycrcb_graphics;
// Konstruktor
SC_HAS_PROCESS(VIDEO_MIXER);
VIDEO_MIXER(sc_module_name nm);*
void Mixer_thread();
};
```

*__sc_module_name__ ist der Typ dersModul-namens (intern als Strings dargestellt), muss erster Parameter sein, wenn nicht SC_CTOR verwendet wird.



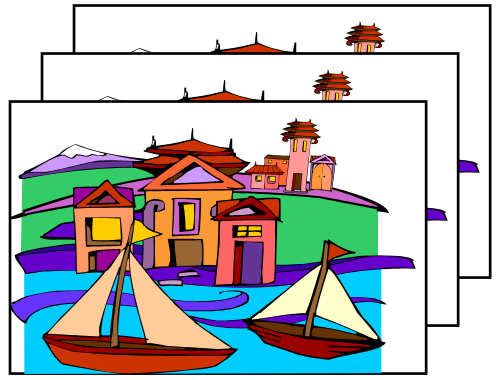
Beispiel: Video-Mixer (4)

```
VIDEO_Mixer::VIDEO_Mixer(sc_module_name nm)
:sc_module(nm)
{
  // Instanziieren
  Rgb2YCrCb Rgb2YCrCb_i("Rgb2YCrCb_i");
  YCRCB_Mixer YCRCB_Mixer_i(" YCRCB_Mixer_i ");
  // Verbindungen
  Rgb2YCrCb_i.rgb_pi(rgb_graphics);
  Rgb2YCrCb_i.ycrcb_po(ycrcb_graphics);
  YCRCB_Mixer_i.K_pi(K);
  YCRCB_Mixer_i.a_pi(dvd_in);
  YCRCB_Mixer_i.b_pi(ycrcb_graphics);
  YCRCB_Mixer_i.y_po(video_po);
};
```

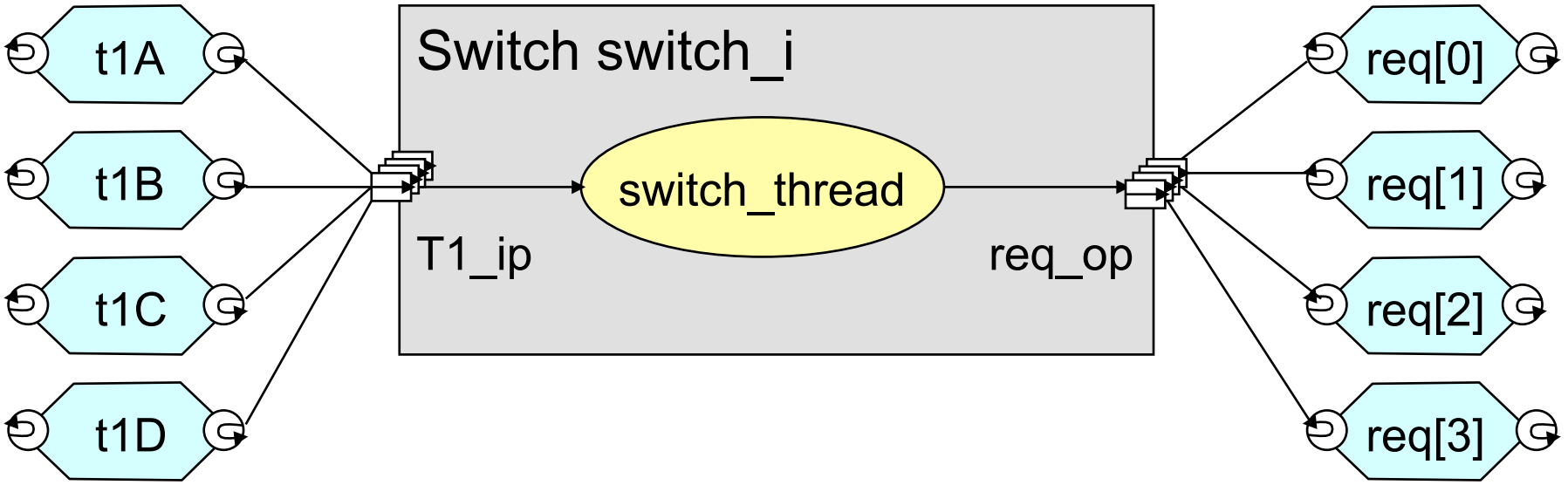


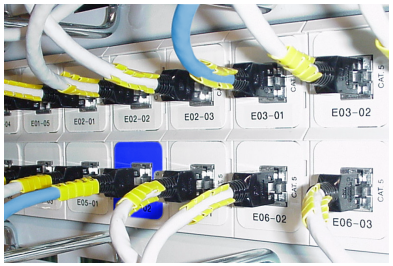
Port-Arrays

- Beschreibung von Arrays von Ports
  - z.B. zur Beschreibung von Multiplexer-Eingängen nützlich
- **Syntax:**
  - **sc_port** <*interface*[,N]> *portname*;
  - N=0..max, Standard=1
  - Wenn N=0, dann können beliebig viele Interfaces mit dem Port verbunden werden



Beispiel





```
// Datei: Switch.h
SC_MODULE(Switch) {
    sc_port<sc_fifo_in_if<int>, 4> T1_ip;
    sc_port<sc_fifo_out_if<int>, 0> req_op;
  …}
```

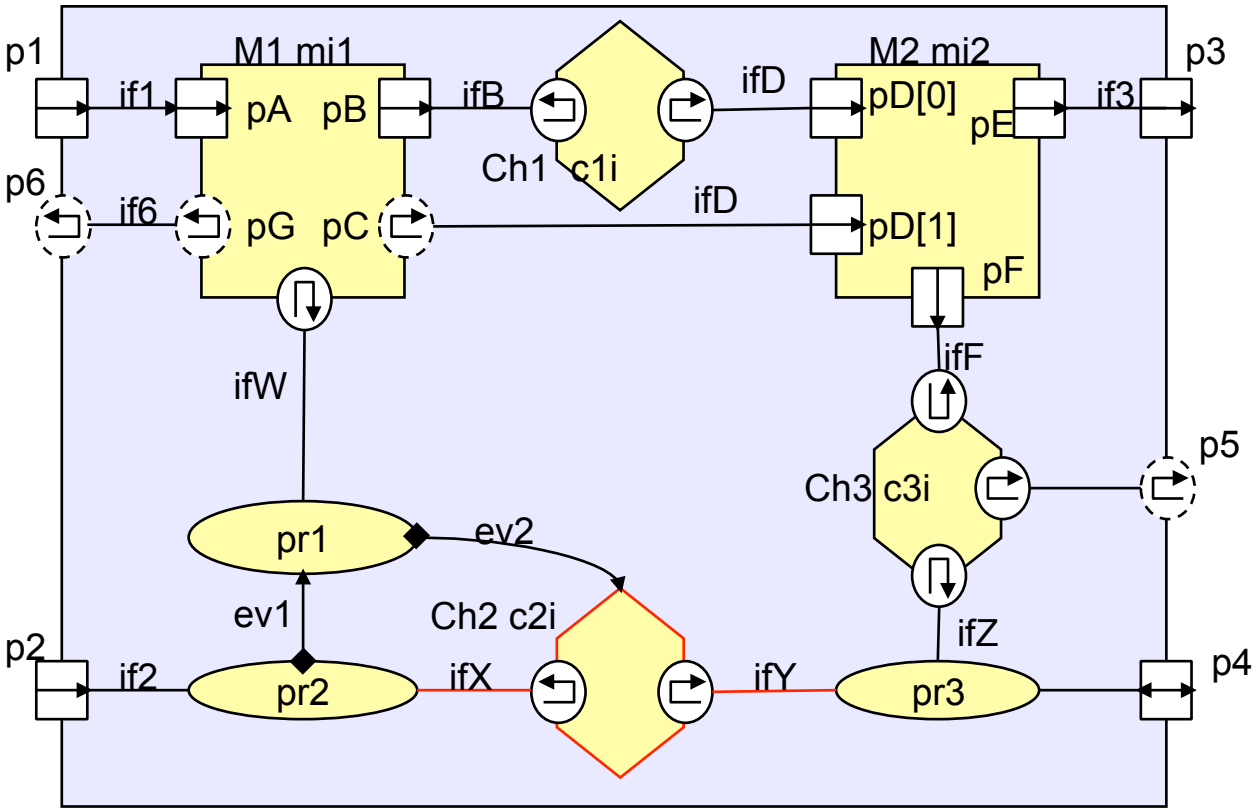
Array-Verbindungen

```
// Datei: board.h
#include "Switch.h”
SC_MODULE(Board) {
    const unsigned int REQS;
    Switch switch_i;
    sc_fifo<int> t1A, t1B, t1C, t1D;
    sc_signal<bool> req[REQS];
    SC_CTOR(Board): switch_i("switch_i");
      {  // Verbinde 4 Channels mit dem Switch
      switch_i.T1_ip(t1A); switch_i.T1_ip(t1B);
      switch_i.T1_ip(t1C); switch_i.T1_ip(t1D);
      for (unsigned int i=0; i<4; i++) {
         switch_i.req_op(req[i]);
} } };
```

Code des Prozesses

```
// Datei: Switch.cpp
void Switch::switch_thread() {
    for (unsigned int i=0; i<req.size(); i++) {
          req[i]->write(true);
    }
    // Starte nach Aktivierung des ersten Ports
    wait(T1_ip[0]->data_written_event()      // liefert Referenz auf Event,
          | T1_ip[1]->data_written_event()    // der beim Schreiben von
          | T1_ip[2]->data_written_event()    // Daten auftritt
          | T1_ip[3]->data_written_event() );  // (Methode von sc_fifo).
     for (;;) {
       for (unsigned i=0; i!=T1_ip.size(); i++) {
         // Jeden Port bearbeiten…
         int value = T1_ip[i]->read();…
    }}}
```

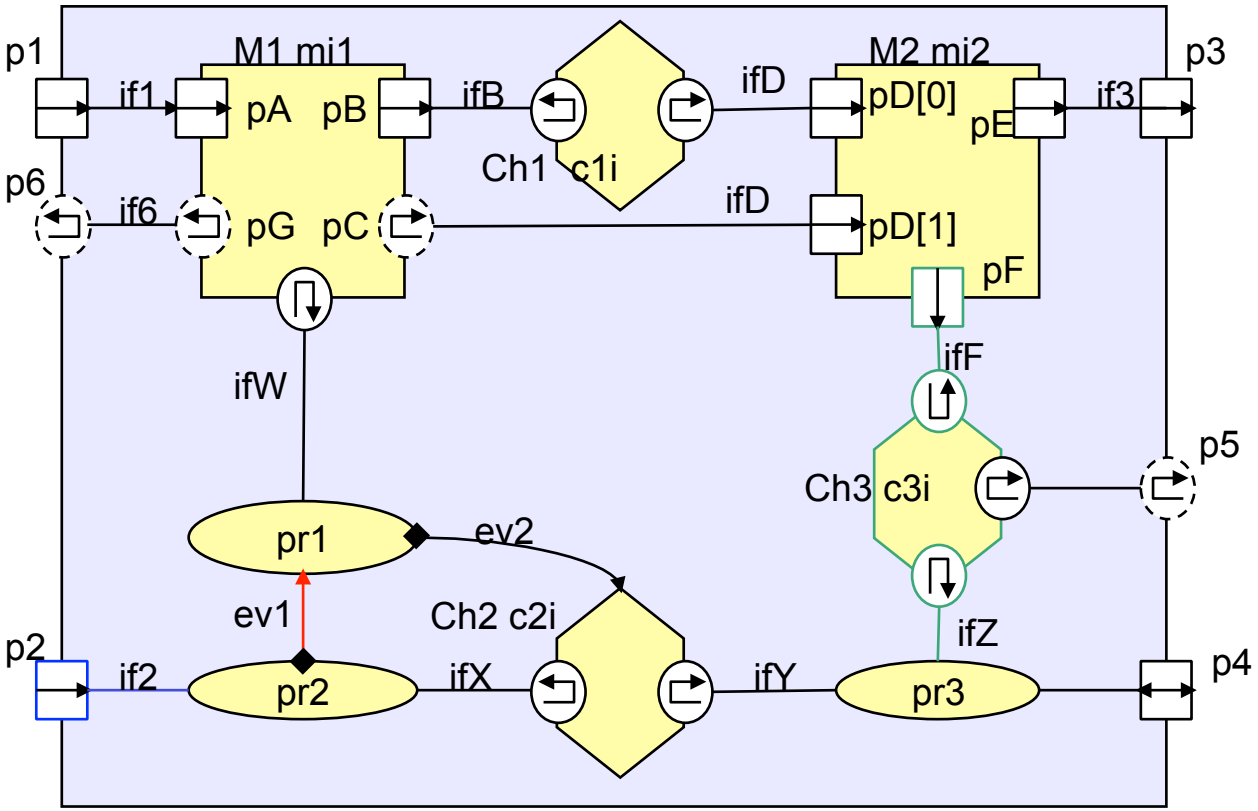
Verbindungsarten (1)



- Channel c1i implements interfaces ifB, ifD,
- channel c2i implements interfaces ifX, ifY,
- channel c3i implements interfaces ifF, if5, ifZ.

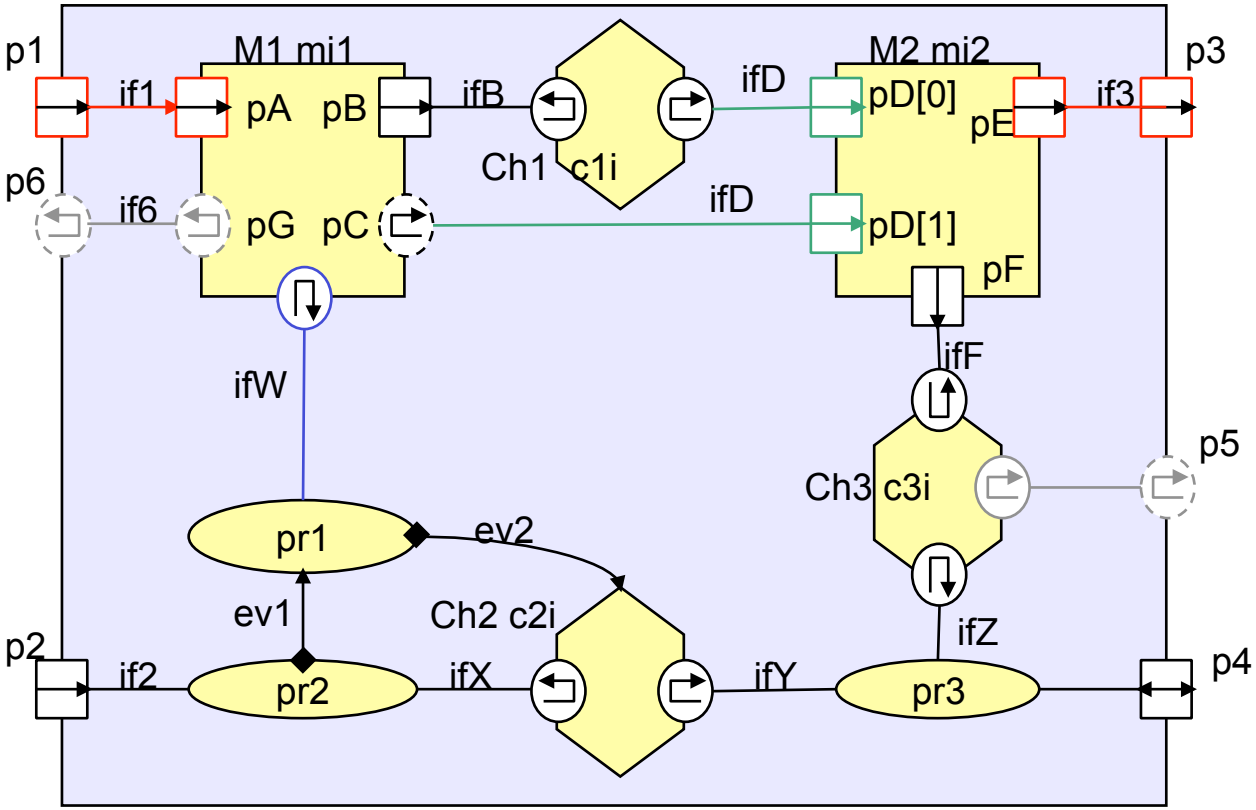
- pD is an array port
- mi1 implements an interface ifW (☞ custom channels, not explained in course)
- processes may communicate with other processes at the same level using channels (see pr2 & pr3)
- pC is of type **sc_export** (SystemC 2.1: channel moved into module, port can be used like a channel)

Verbindungsarten (2)



- Processes may synchronize with other processes at the same level using event (see pr1 & pr2)
- Processes may communicate upwards in the hierarchy via ports using interfaces (see pr2 & p2).
- Processes may communicate with sub-modules via channels connected to sub-module ports (see pr3 & mi2).

Verbindungsarten (3)



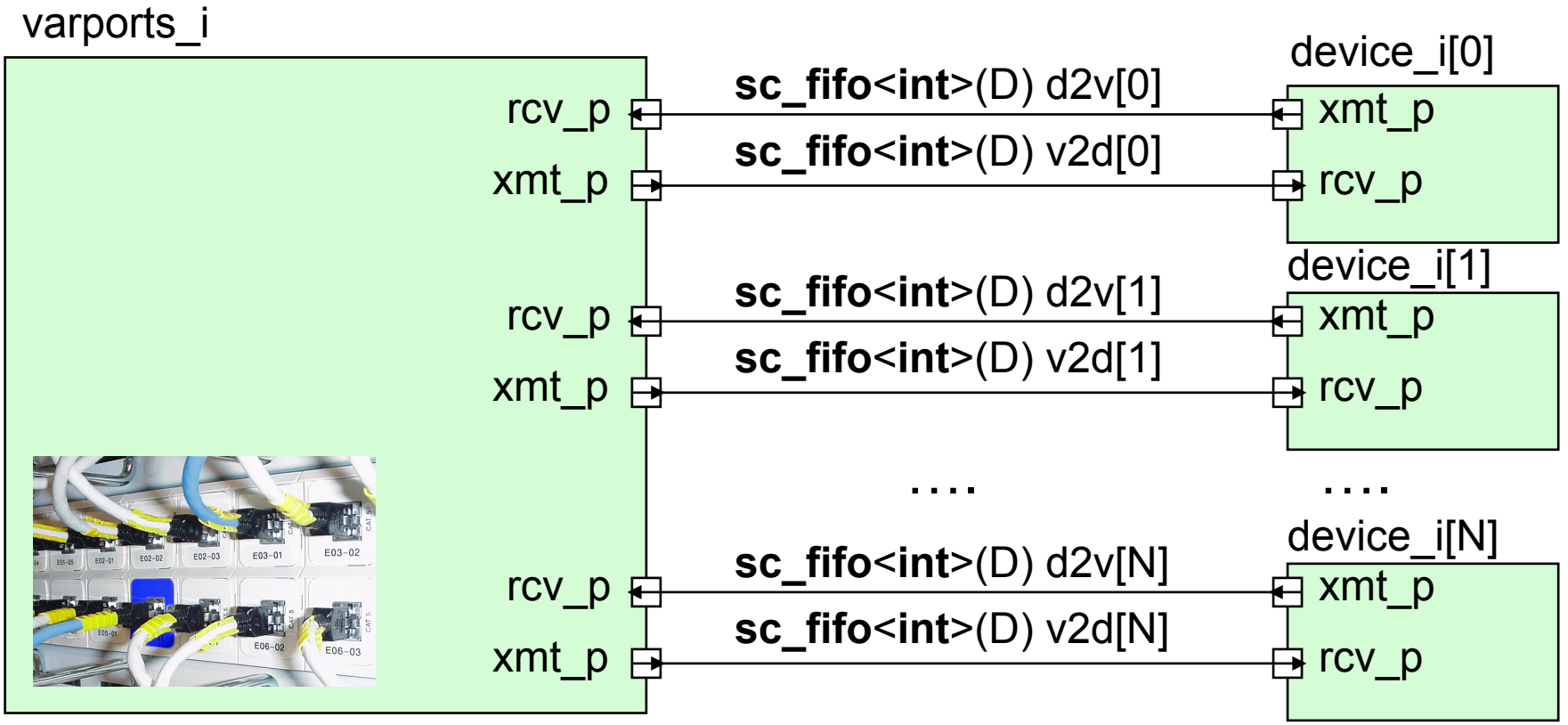
- Processes may communicate with sub-modules by letting the process access the port interface (see pr1 & mi1).
- **sc_ports** may connect directly to ports of sub-modules (see p1 & mi1)
- port arrays can share the same interface (see pD).
- **sc_exports** may connect to **sc_exports** of submodules and channels.

Verbindungsarten (4)

Von	Nach	Methode
Port	Untermodul	Direkt verbinden über **sc_port**
Prozess	Port	Direkter Zugriff durch Prozess
Untermodul	Untermodul	Lokale Channel-Verbindung
Prozess	Untermodul	Lokale Channel-Verbindung oder über **sc_export** or über Interfaces eines Untermoduls (hierarchischer Channel)
Prozess	Prozess	Events oder lokaler channel
Port	Lokaler Channel	Direkt verbinden über **sc_export**

Programmierbare Hierarchie

- Verwendung von Kontrollstrukturen wie **for** oder **if then else** usw. bei der Erzeugung von Strukturen, z.B.:



Konfigu-rierbarer Code

```
#include <systemc.h>
#include "varports.h"
#include "device.h"
int sc_main(int argc, char* argv[]) {
    // N von Befehlszeile..
    int N = (argc > 1) ? atoi(argv[1]) : 1;
    varports* varports_i;
    device* device_i[N];
    sc_fifo<int>* v2d[N];
    sc_fifo<int>* d2v[N];
    varports_i = new varports(…initialisiere Parameter…);
    for (unsigned i=0; i<N; i++) {
      sc_string nm;
      nm = sc_string::to_string("device_name_i[%d]",i);
      device_i[i] = new device(nm.c_str());
      nm = sc_string::to_string("v2d[%d]",i);
      v2d[i] = new sc_fifo<int>(nm.c_str(),fifo_depth);
      nm = sc_string::to_string("d2v[%d]",i);
      d2v[i] = new sc_fifo<int>(nm.c_str(),fifo_depth);
      device_i[i]->rcv_p(*v2d[i]);
      device_i[i]->xmt_p(*d2v[i]);
      varports_i->rcv_p(*d2v[i]);
      varports_i->xmt_p(*v2d[i]); }
```

Zusammenfassung

- Ports
- Interfaces
- Ports und statische Sensivität
- **Pos** und **neg**-Methoden
- Beispiel: LFSR
- Verbindung von Ports
- Beispiel: Video-Mixer
- Port-Arrays
- Verbindungsmöglichkeiten
- Programmierbare Hierarchie