Strukturelle und funktionale Beschreibung dynamisch rekonfigurierbarer Hardware in SystemC

Armin Felke

Leitung: Prof. Dr. Joachim K. Anlauf
Betreuer: Dipl.-Inform. Andreas Raabe

Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik II
Arbeitsgruppe Technische Informatik

07.11.2007

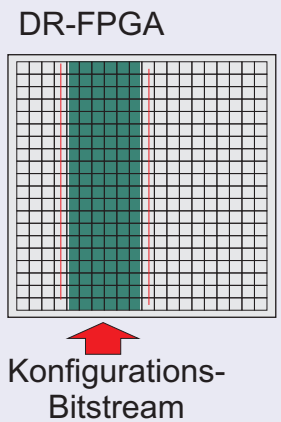
Abschlussvortrag zur Diplomarbeit

Übersicht

Dynamische Rekonfiguration

Dynamische Rekonfiguration (DR)

- Neuprogrammierung bestimmter Chip-Bereiche im laufenden Betrieb
- zunehmende Verwendung in der Hardware-Entwicklung (Unterstützung durch FPGAs)



Simulation von DR

- HW-Beschreibungssprachen meist statisch / Tool-Unterstützung schlecht
- Für Entwicklung komplexer Systeme existiert noch kein lückenloser Design-Flow

Simulation von DR

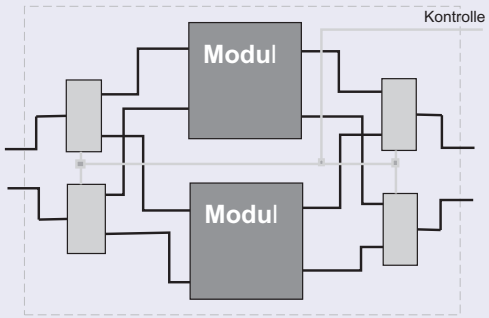
DR in SystemC

- Simulation: Austausch von Modulen und Channeln während der Laufzeit
- SystemC [2] bietet nativ keine Möglichkeit, dynamisch rekonfigurierbare Hardware (DRHW) zu simulieren. [1]
- Strikte Trennung: Konstruktions- und Simulationsphase
- Simulationsphase:
  - Modulhierarchie nicht mehr veränderbar
  - Port-Bindungen können nicht mehr eingegangen/gelöst werden

Rekonfiguration in SystemC

Klassische Multiplexer-"Lösung"

- unflexibel / viel "Handarbeit"
- nicht immer anwendbar (z.B. für TLM)
- Speziallösung (gemessen am Sprachumfang von SystemC)



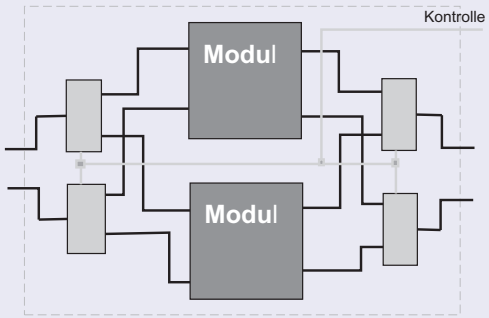
Allgemeine und komfortable Lösung wünschenswert

- anwendbar für beliebige Channel
- anwendbar auf sämtlichen Abstraktionsebenen (→ Refinement)
- Rekonfiguration von Kommunikation

Rekonfiguration in SystemC

Klassische Multiplexer-"Lösung"

- unflexibel / viel "Handarbeit"
- nicht immer anwendbar (z.B. für TLM)
- Speziallösung (gemessen am Sprachumfang von SystemC)



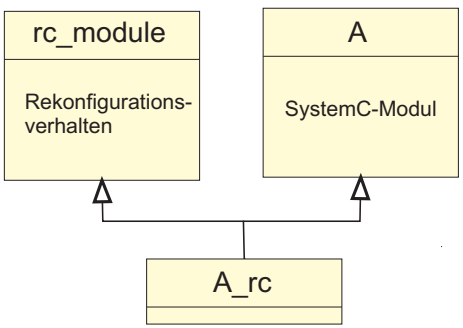
Allgemeine und komfortable Lösung wünschenswert

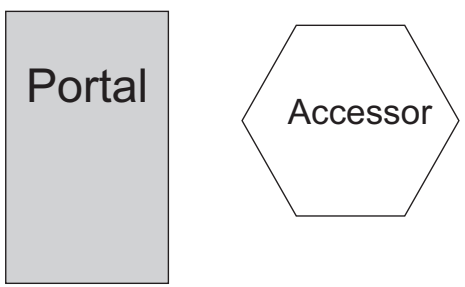
- anwendbar für beliebige Channel
- anwendbar auf sämtlichen Abstraktionsebenen ($\rightarrow$ Refinement)
- Rekonfiguration von Kommunikation

ReChannel

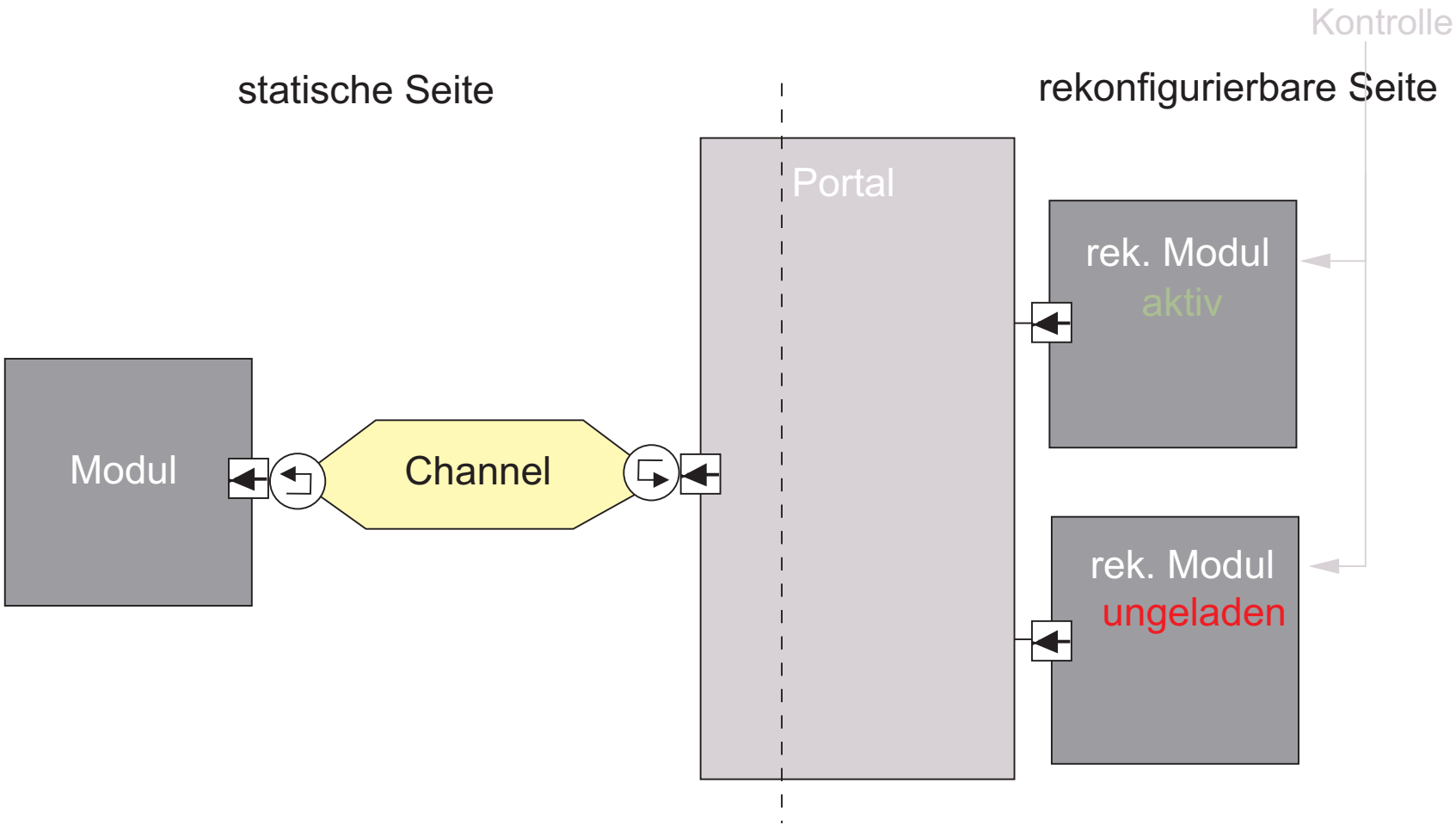
ReChannel-Bibliothek [4]

- statische C++-Library
- Spracherweiterung von SystemC zur allgemeinen Simulation von DRHW
- spezielle Anforderungen an ReChannel:
  - keine Änderungen am SystemC-Kernel
  - bewährter SystemC-Design-Flow (inkl. Refinement)
  - beliebige Module rekonfigurierbar (IP-Reuse)
  - anpassbar auf benutzerdefinierte Channel



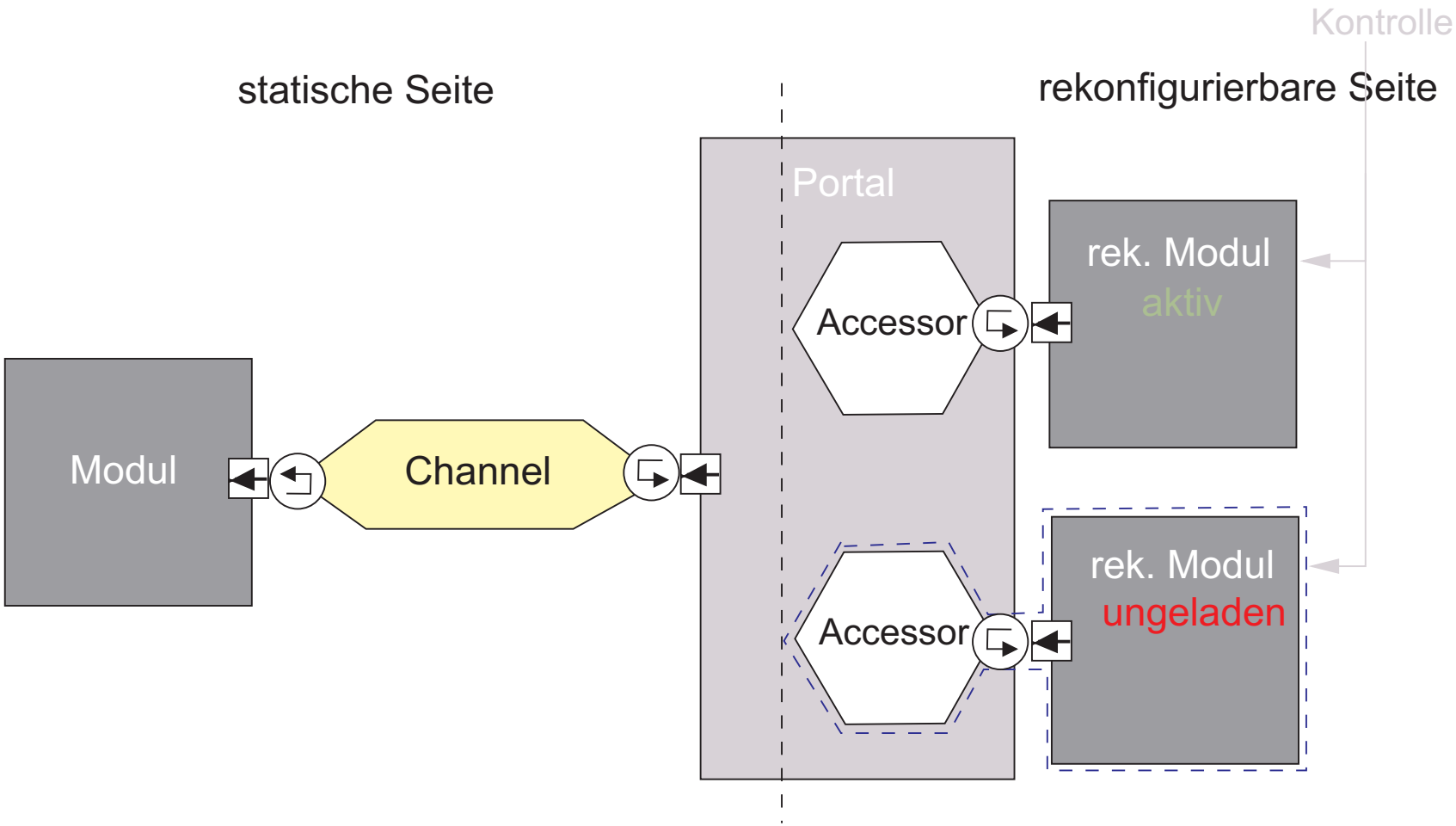


Prinzipielle Funktionsweise



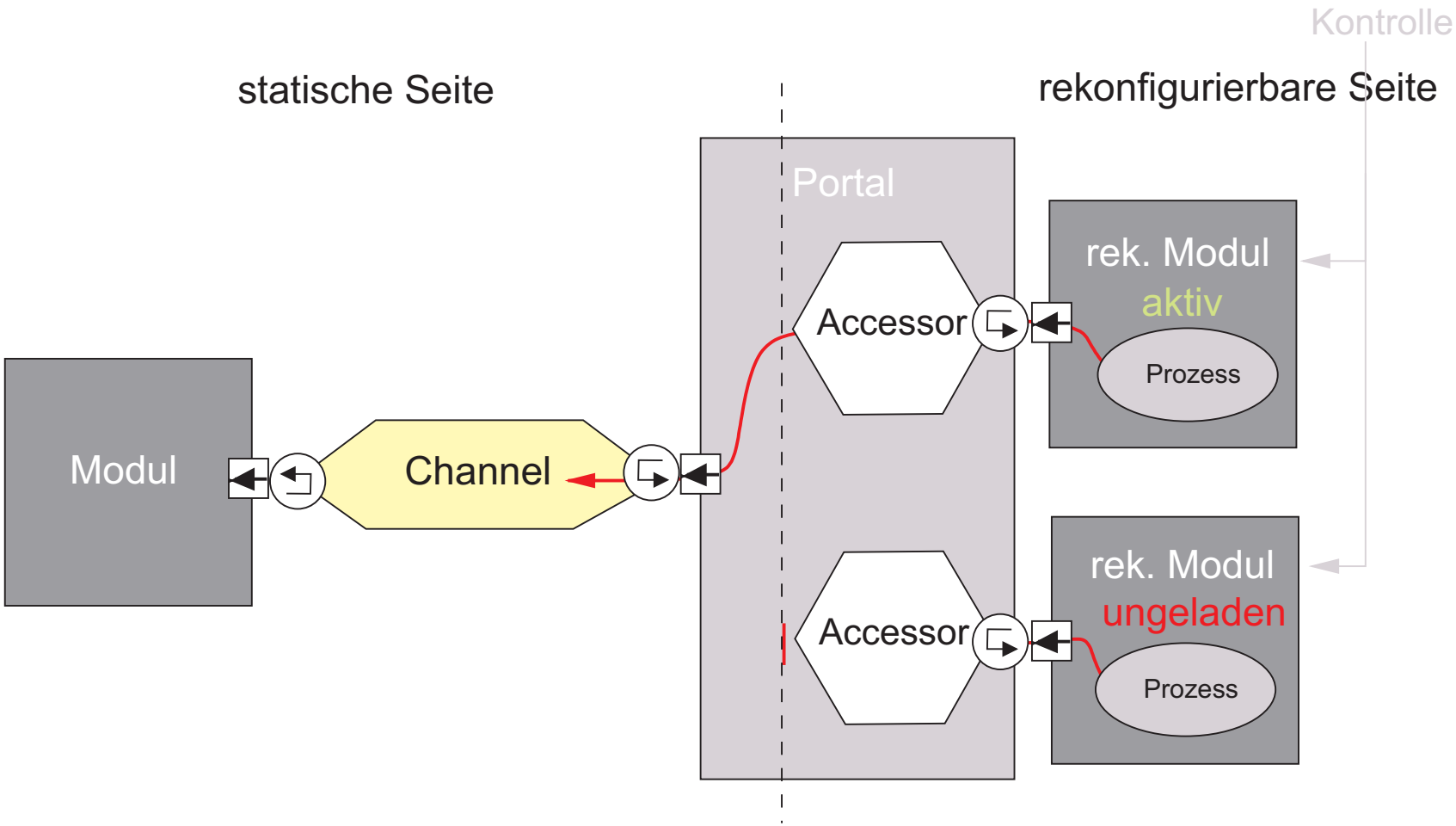
Portal (Außenansicht)

Prinzipielle Funktionsweise



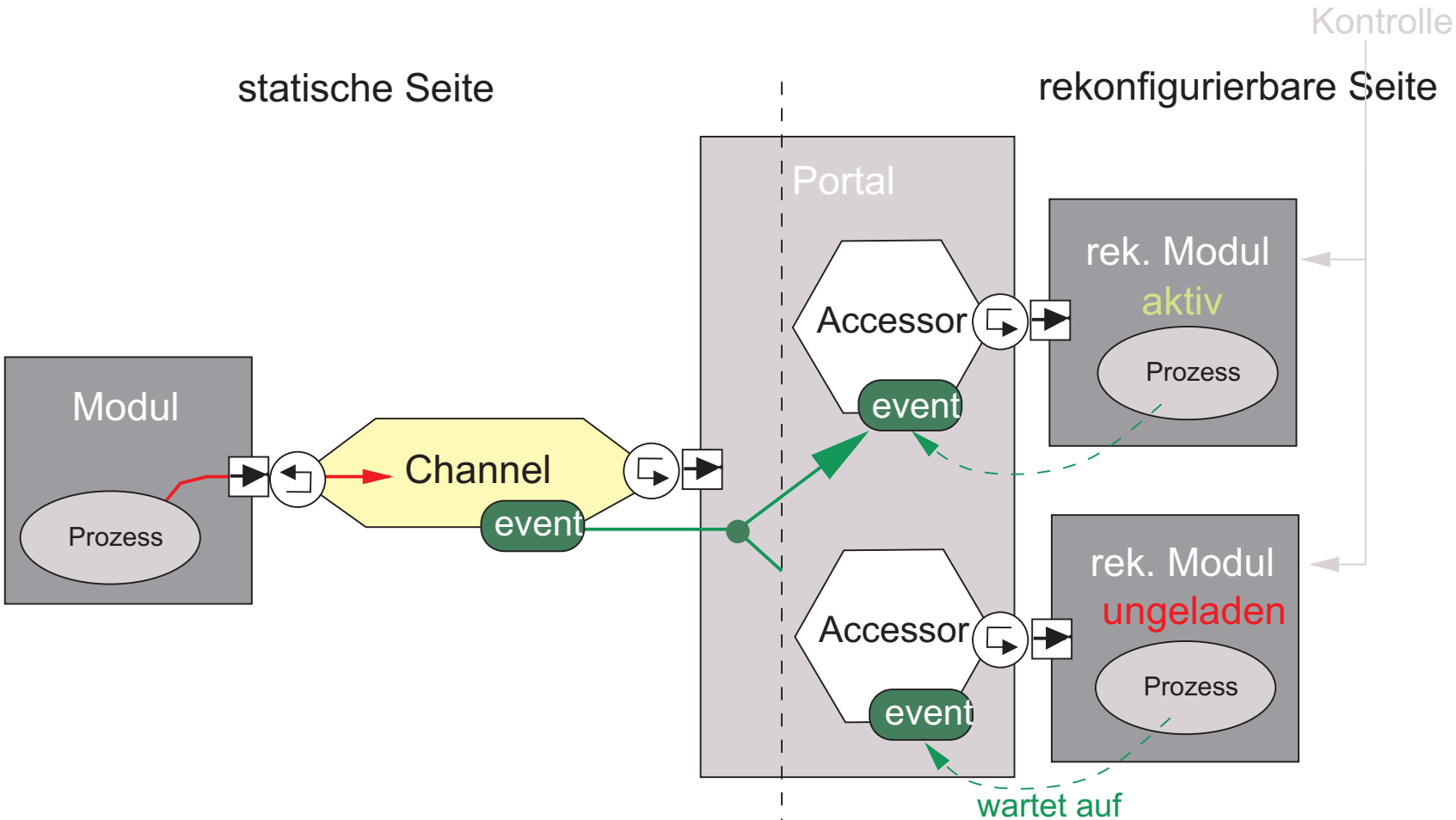
Portal (Innenansicht)

Prinzipielle Funktionsweise



Interface-Method-Call(IMC)-Forwarding

Prinzipielle Funktionsweise



Event-Forwarding

Aufgabenstellung

Aufgabenstellung der Arbeit

- Simulation von DR auf funktionaler sowie transaktionaler Abstraktionsebene in SystemC

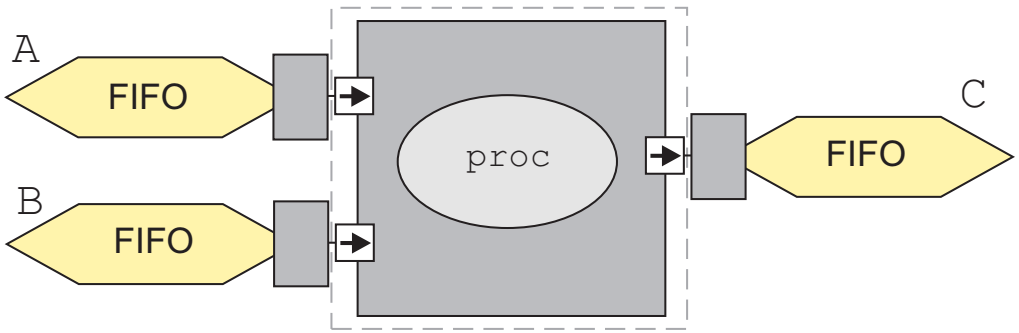
  $\rightarrow$ ReChannel erweitern um:

  - Sprachkonstrukte für grundlegende Synchronisationsfunktionalität
  - Rekonfiguration von Exports

- Refactoring an einigen bestehenden Klassen

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:




```
while(true) {
    a = portA.read();
    b = portB.read();
    c = berechnen(a, b);
    portC.write(c);
}
```

ohne Synchronisation

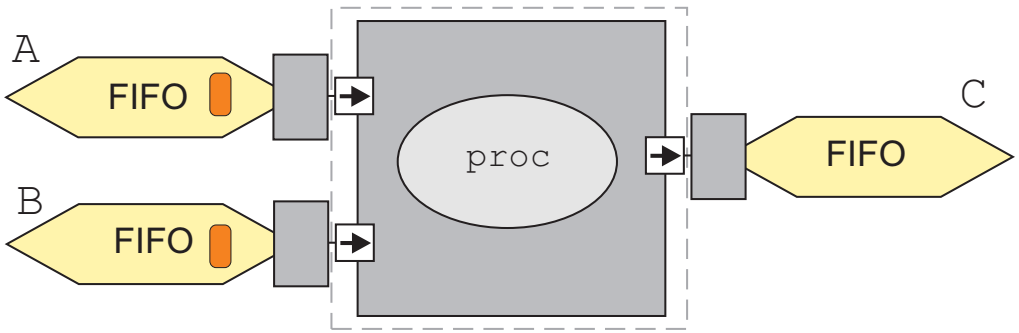
- Deaktivierung zu beliebigem Zeitpunkt

Dateninkonsistenzen

- Datenfluss-Beschreibungen: Reihenfolge u. Zuordenbarkeit von Daten entscheidend

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:




```
 while(true) {
>    a = portA.read();
     b = portB.read();
     c = berechnen(a, b);
     portC.write(c);
}
```

ohne Synchronisation

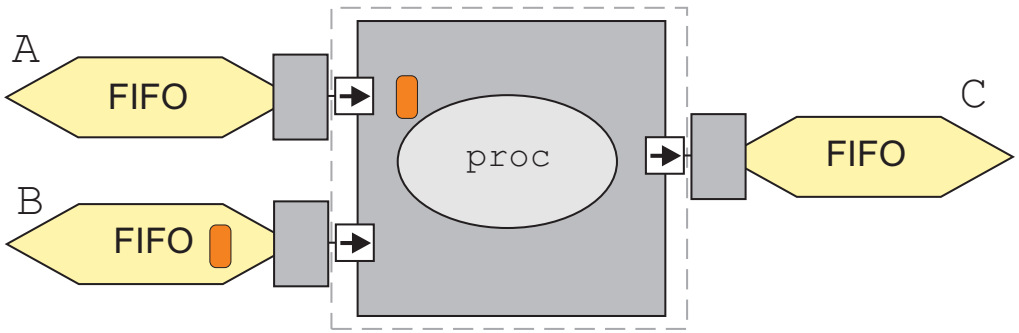
- Deaktivierung zu beliebigem Zeitpunkt

Dateninkonsistenzen

- Datenfluss-Beschreibungen: Reihenfolge u. Zuordenbarkeit von Daten entscheidend

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:




```
 while(true) {
     a = portA.read();
>    b = portB.read();
     c = berechnen(a, b);
     portC.write(c);
}
```

ohne Synchronisation

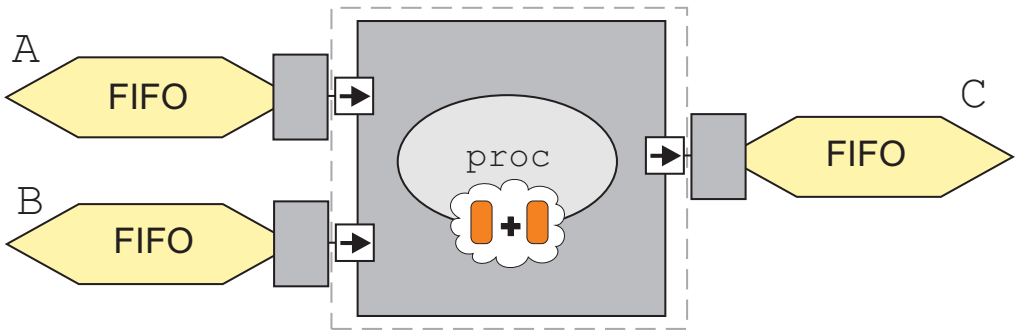
- Deaktivierung zu beliebigem Zeitpunkt

Dateninkonsistenzen

- Datenfluss-Beschreibungen: Reihenfolge u. Zuordenbarkeit von Daten entscheidend

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:




```
 while(true) {
     a = portA.read();
     b = portB.read();
>    c = berechnen(a, b);
     portC.write(c);
}
```

ohne Synchronisation

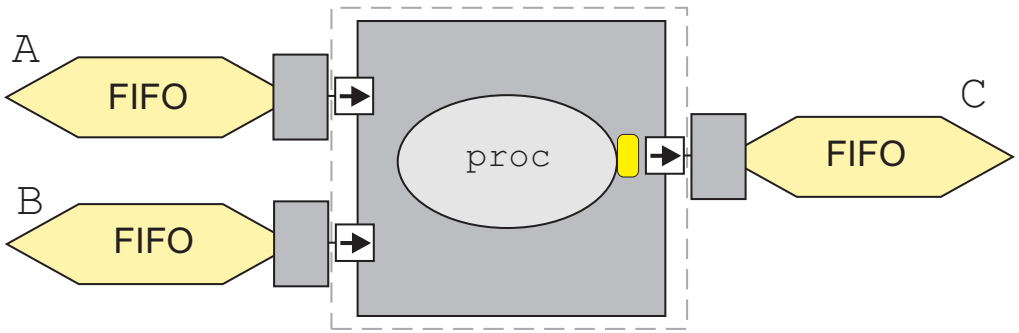
- Deaktivierung zu beliebigem Zeitpunkt

Dateninkonsistenzen

- Datenfluss-Beschreibungen: Reihenfolge u. Zuordenbarkeit von Daten entscheidend

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:




```
 while(true) {
     a = portA.read();
     b = portB.read();
     c = berechnen(a, b);
>    portC.write(c);
}
```

ohne Synchronisation

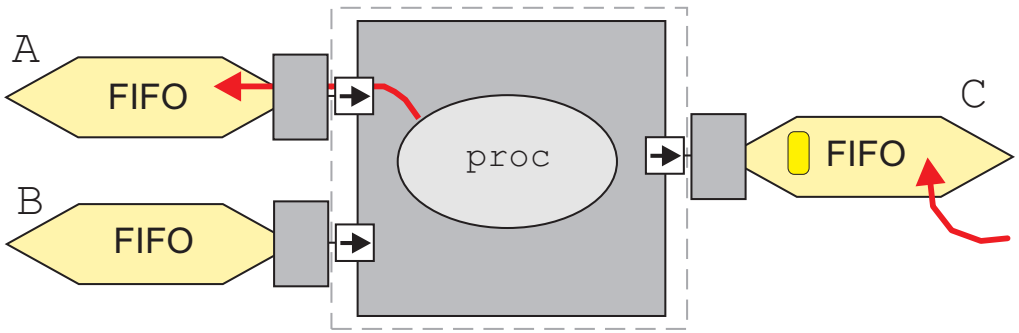
- Deaktivierung zu beliebigem Zeitpunkt

Dateninkonsistenzen

- Datenfluss-Beschreibungen: Reihenfolge u. Zuordenbarkeit von Daten entscheidend

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:



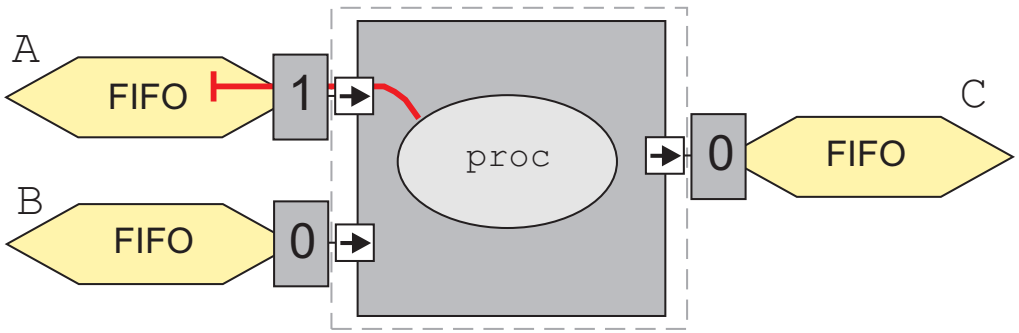

```
 while(true) {
>    a = portA.read();
     b = portB.read();
     c = berechnen(a, b);
     portC.write(c);
}
```

Timing

- Ziel: Deaktivierung zu bestimmtem Zeitpunkt
- Aber: Wenn Ausgabe gelesen werden kann, ist es bereits zu spät
- `proc` beginnt sofort wieder mit dem Lesen weiterer Daten

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:



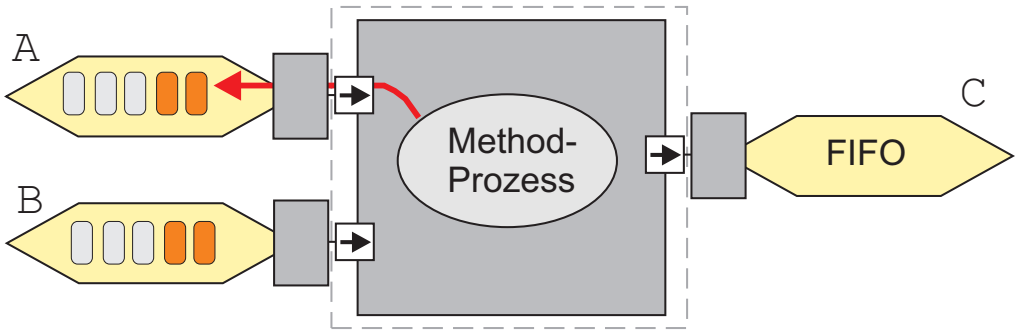

```
 while(true) {
>    a = portA.read();
     b = portB.read();
     c = berechnen(a, b);
     portC.write(c);
}
```

Deadlock

- FIFO leer: Prozess wartet auf neue Daten
- Modul soll entladen werden, da Arbeit getan
- Deaktivierung nicht möglich während externer Zugriff läuft
- $\rightarrow$ Deadlock in der Rekonfigurationskontrolle

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:



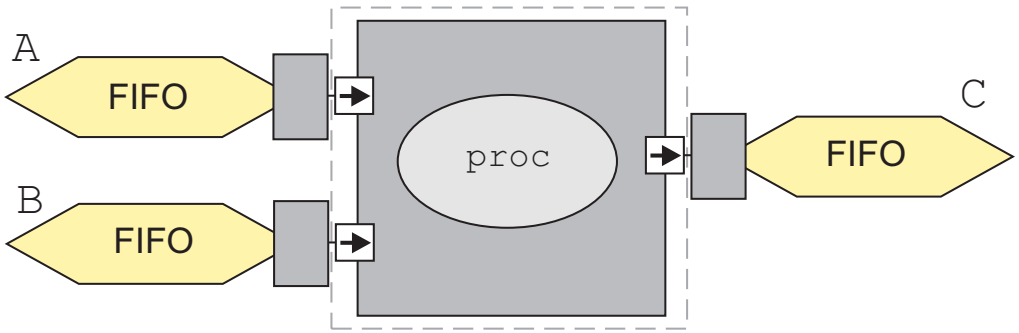
`nb_read()` in einer Schleife

Nichtblockierende Zugriffe

- Grundsätzlich problematisch: Method-Prozesse mit Zugriff auf abstrakte Channel
- Accessor kann nichtblockierende Zugriffe nicht abblocken
- Prozess kann nicht daran gehindert werden, alle Daten auszulesen

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:

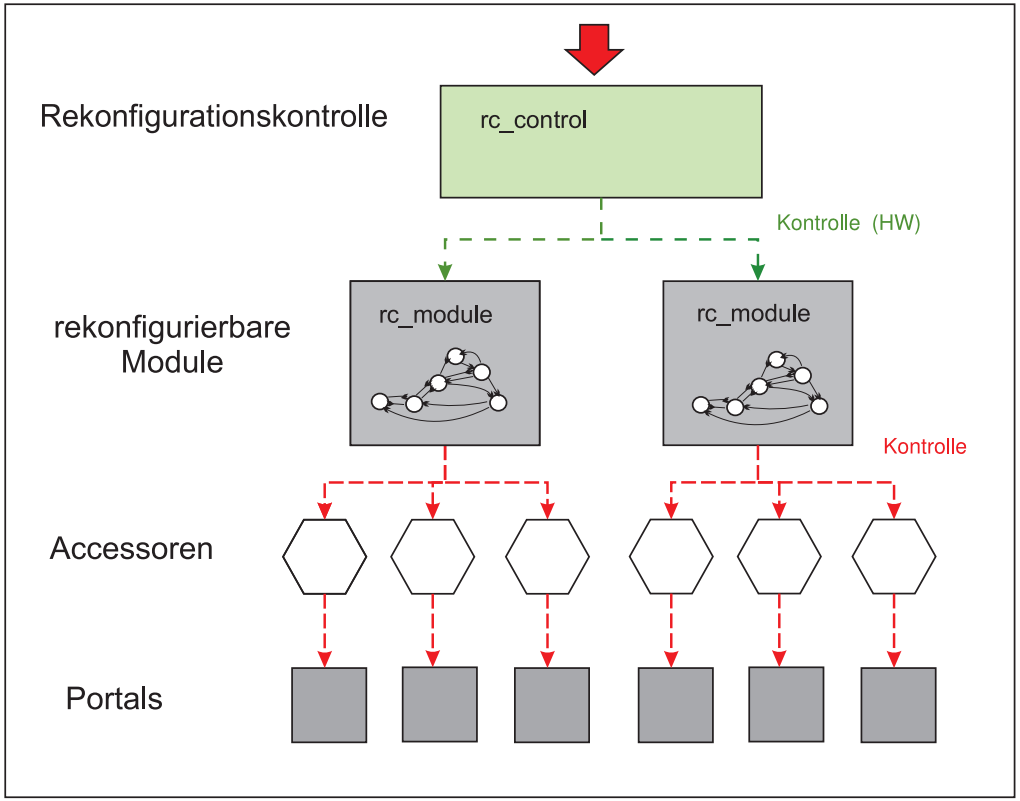


Was muss möglich sein?

- den inneren Zustand eines Moduls anhand externer Zugriffe bestimmen
- Nichtblockierende Zugriffe abblocken (z. B. leere FIFO vortäuschen)
- Transaktionen für Eingabe-/Ausgabe-Zugriffe definieren

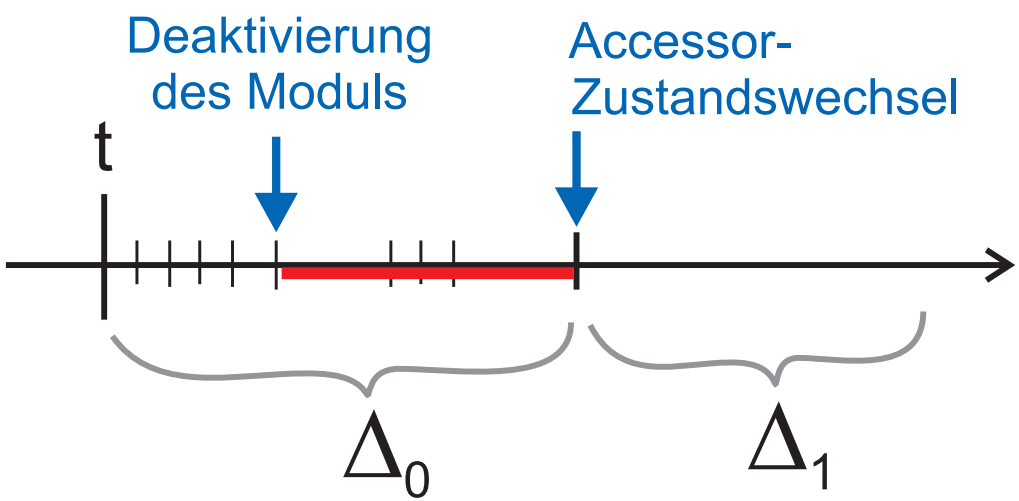
Kontrollmechanismus

- Kommunikation zwischen Komponenten unterschiedlich
- Instabil bei mehreren Kontrollprozessen
  → Inkonsistente Zustände
  → Deadlock
- Interne Verwaltung komplex
  z. B.:
  → `rc_module`: 7 Zustände
  → Accessor: 6 Zustände
- Ursache für ein Synchronisationsproblem



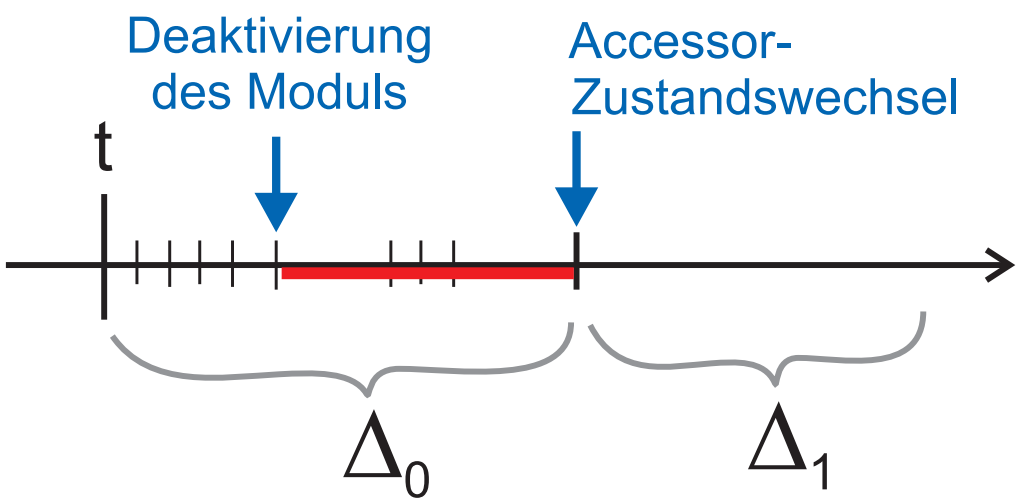
Kontrollmechanismus

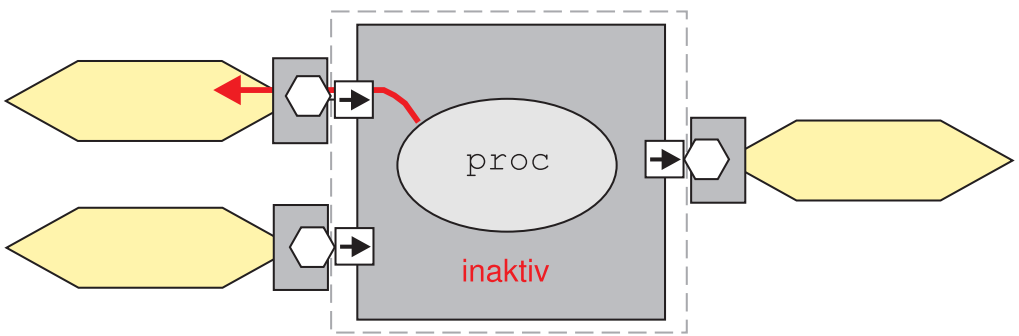
- Accessor-Zustand ist ein Signal
- neuer Zustand wird daher erst im nächsten Delta-Zyklus übernommen
- wenn Accessor sofort blockierte, dann würden Zugriffe vor und nach Deaktivierung ungleich behandelt
- Problem: blockierende Zugriffe kurz nach Deaktivierung werden nicht aufgehalten



Kontrollmechanismus

- Accessor-Zustand ist ein Signal
- neuer Zustand wird daher erst im nächsten Delta-Zyklus übernommen
- wenn Accessor sofort blockierte, dann würden Zugriffe vor und nach Deaktivierung ungleich behandelt
- Problem: blockierende Zugriffe kurz nach Deaktivierung werden nicht aufgehalten



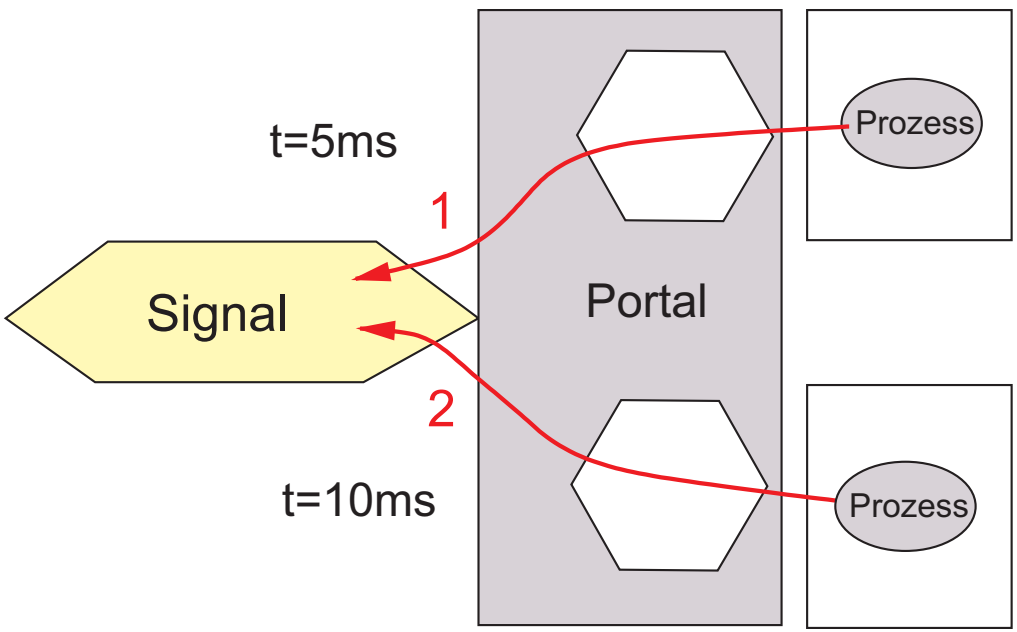


Kompatibilität zum neuen SystemC-Standard

Treiberkonflikte

- SystemC-Standard IEEE-1666: Treiberkonflikt bei Signalen führt unweigerlich zum Abbruch der Simulation

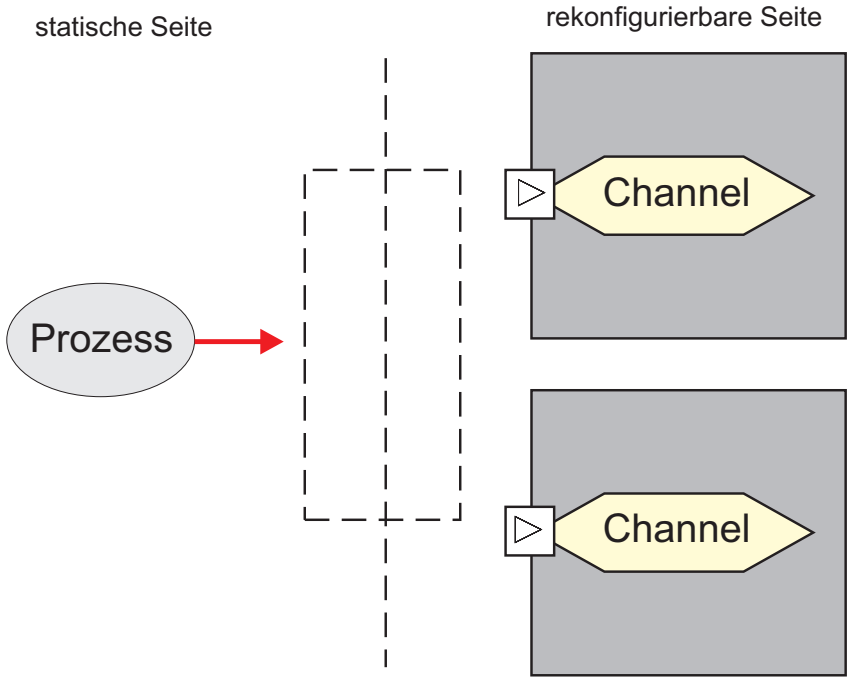
→ ReChannel ist betroffen, da es Zugriffe direkt "durchleitet"



SystemC-Sprachumfang

Abdeckung des SystemC-Sprachumfangs

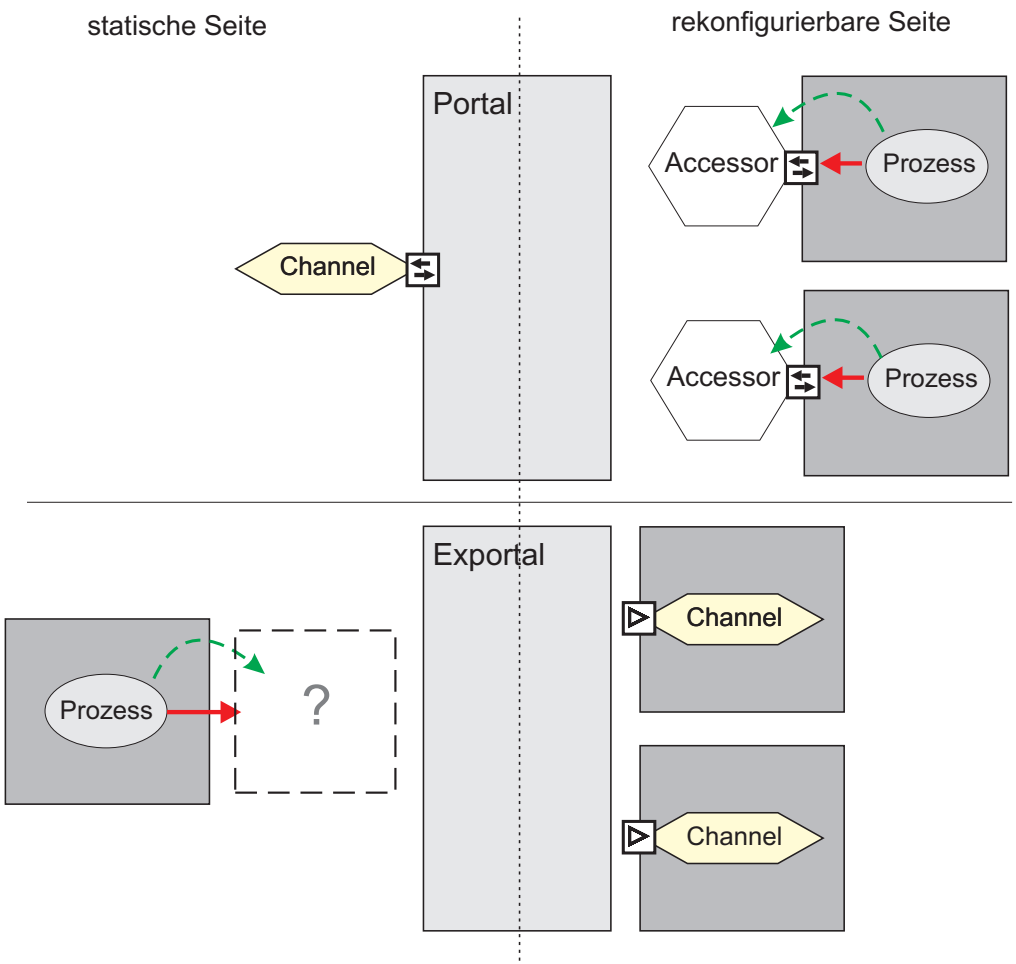
- Exports häufig verwendet in TLM-Beschreibungen (z. B. [3])
- Rekonfiguration von Exports bisher nicht möglich
- Rekonfiguration von Channeln ebenfalls nicht möglich



Rekonfiguration von Exports

Exportal

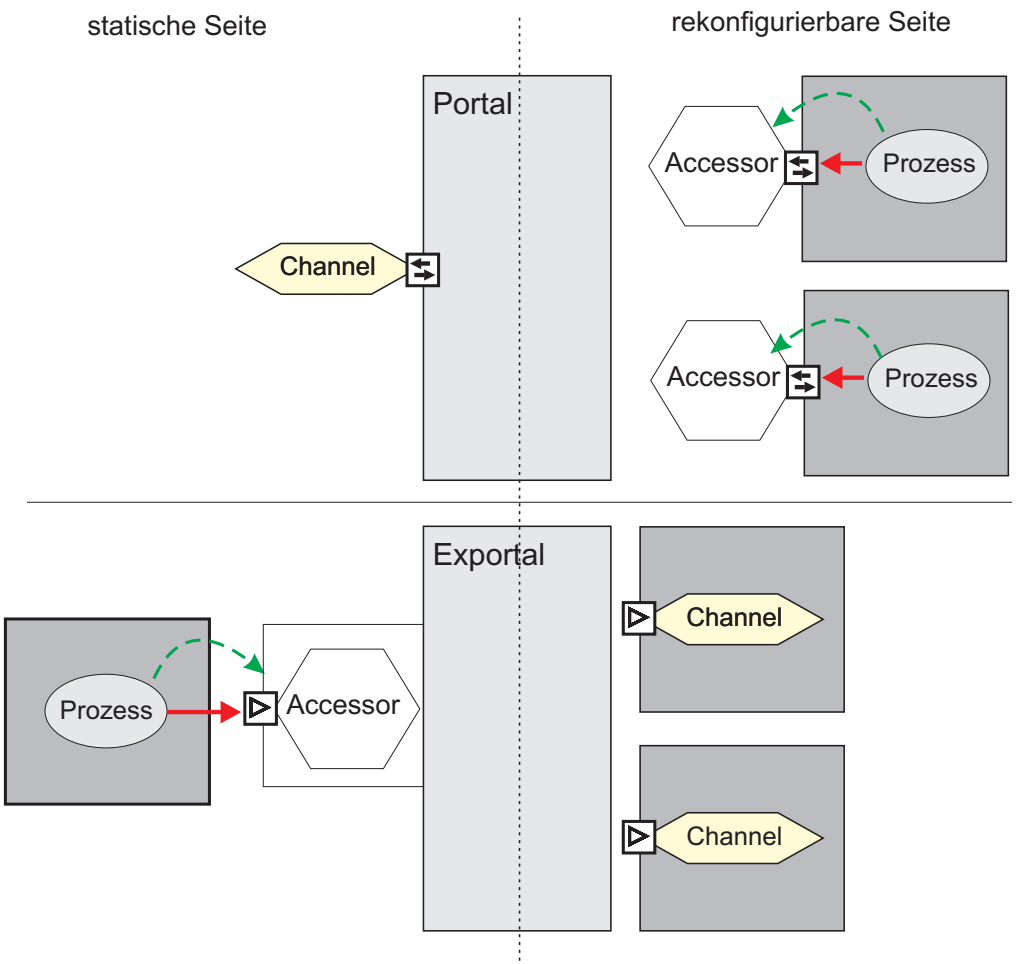
- Verallgemeinerung des Kontrollmechanismus notwendig
- Suche nach Gemeinsamkeiten von Portal und Exportal
- Worauf greift der Prozess der statischen Seite zu?
- Wünschenswert: Wiederverwendung des Accessors



Rekonfiguration von Exports

Exportal

- Verallgemeinerung des Kontrollmechanismus notwendig
- Suche nach Gemeinsamkeiten von Portal und Exportal
- Worauf greift der Prozess der statischen Seite zu?
- Wünschenswert: Wiederverwendung des Accessors



Re-Design von ReChannel

Leider: Erweiterbarkeit um neue Aspekte extrem gering

- Funktionsweise fest zugeschnitten auf ganz spezielle Basisklassen
- Kontroll-Logik über viele Klassen verteilt
- viele direkte Abhängigkeiten der Klassen/Objekte untereinander

Zusätzlich erschwerend:

- z.T. unnötig komplexe interne Verwaltung

→ sauberes Re-Design sinnvoller als Versuch einer Änderung

Re-Design von ReChannel

Leider: Erweiterbarkeit um neue Aspekte extrem gering

- Funktionsweise fest zugeschnitten auf ganz spezielle Basisklassen
- Kontroll-Logik über viele Klassen verteilt
- viele direkte Abhängigkeiten der Klassen/Objekte untereinander

Zusätzlich erschwerend:

- z.T. unnötig komplexe interne Verwaltung

$\longrightarrow$ sauberes Re-Design sinnvoller als Versuch einer Änderung

Konzept

Konzept

- Abstraktion des Algorithmus und der Datenstrukturen

  Kontrolle $\leftrightarrow$ rekonf.Objekt $\leftrightarrow$ Switch $\leftrightarrow$ Interface

- Entwurf einer Klassenhierarchie
  - Definition abstrakter Schnittstellen
  - minimale Abhängigkeiten

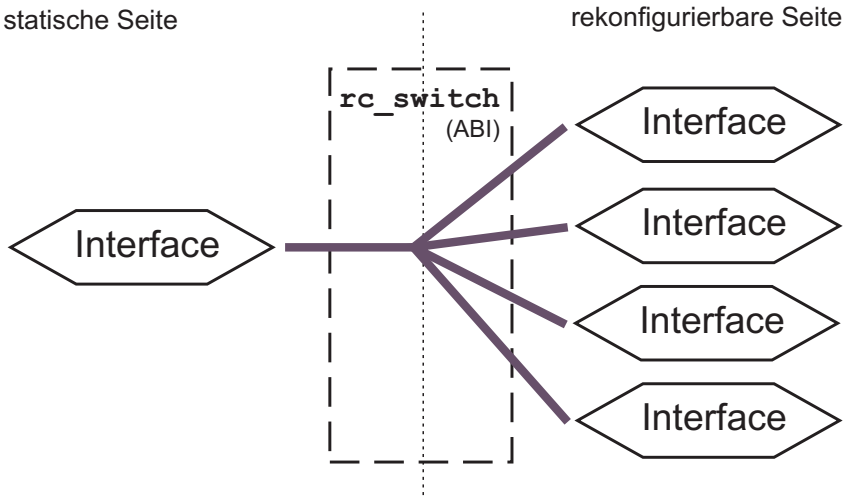
  $\rightarrow$ einfache Erweiterbarkeit um neue Switches (z.B. Exportal)
  $\rightarrow$ hohe Wiederverwendbarkeit von Code (z.B. Accessor)

- dabei bekannte u. neu entdeckte Probleme beheben
- Synchronisationsproblem lösen durch Interface-Filter
- Explizite Beschreibung von DRHW (Verhalten, Struktur)

Der Switch

`rc_switch` (ABI)

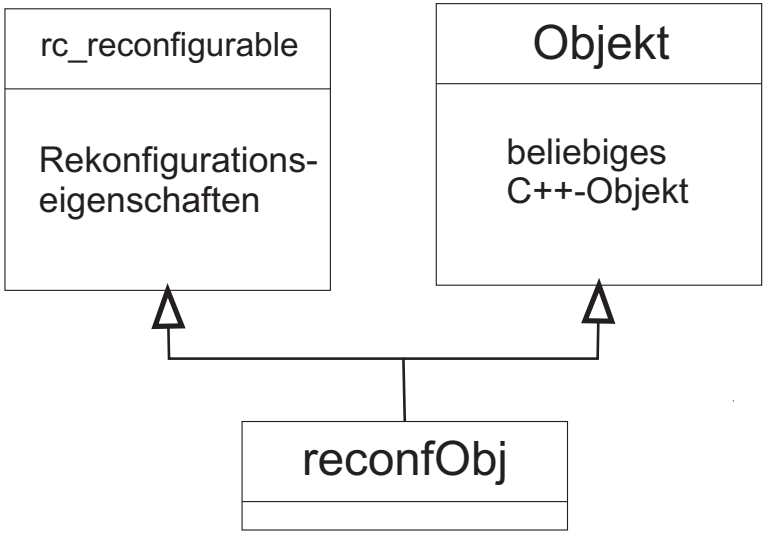
- Spezifikation eines Schalters für die Simulation von DR
- Trennung von Implementation u. Kontrollmechanismus
- 3 Zustände:
  OPEN, CLOSED, UNDEF
- Schnittstelle für Öffnung, Schließung sowie Registrierung, Bindung
- Implementation zuständig für: Kommunikationsspezifikation, Typüberprüfung, Bindung, etc.



Rekonfigurierbare Objekte

`rc_reconfigurable`

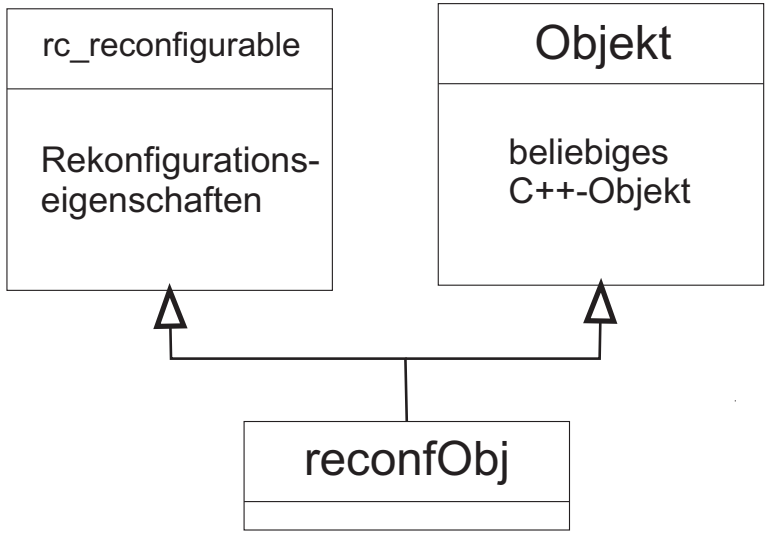
- repräsentiert Rekonfigurations-eigenschaften (wie vormals die Klasse `rc_module`)
- Abstraktion:
  beliebiges Objekt rekonfigurierbar
- 3 Zustände:
  UNLOADED, INACTIVE, ACTIVE
- Zustandsübergang wird mit Delta-Zyklen synchronisiert (Delta-Sync-Object)
- bildet zusammen mit `rc_control` den Kern von ReChannel

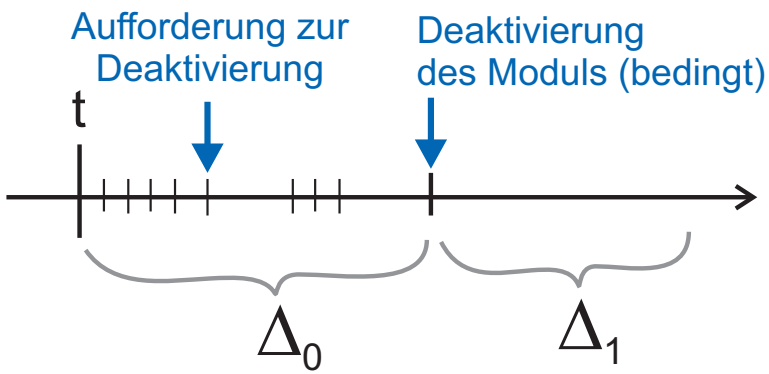


Rekonfigurierbare Objekte

`rc_reconfigurable`

- repräsentiert Rekonfigurations-eigenschaften (wie vormals die Klasse `rc_module`)
- Abstraktion:
  beliebiges Objekt rekonfigurierbar
- 3 Zustände:
  UNLOADED, INACTIVE, ACTIVE
- Zustandsübergang wird mit Delta-Zyklen synchronisiert (Delta-Sync-Object)
- bildet zusammen mit `rc_control` den Kern von ReChannel





ReChannel-v2 Kontrollhierarchie

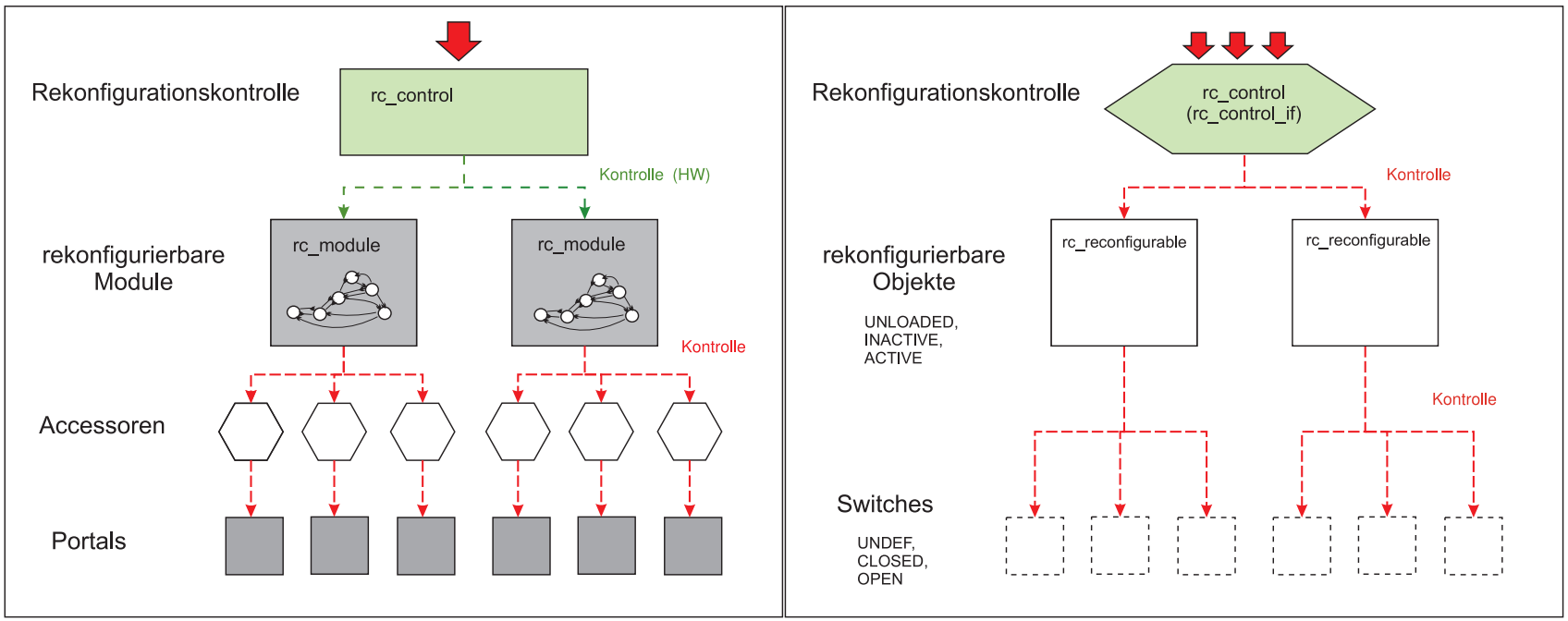
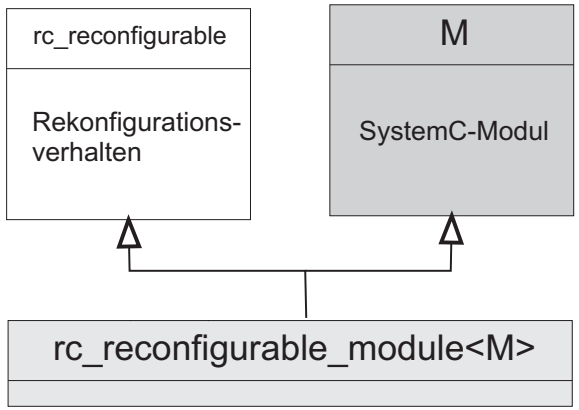


Abbildung: Vergleich mit Kontrollhierarchie von ReChannel-v1

Rekonfigurierbare Module und Switch-Implementationen

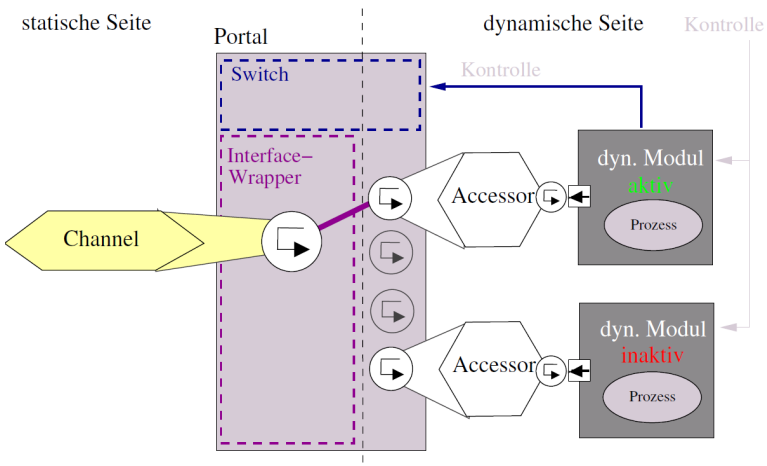
Rekonfigurierbare Module

- `rc_reconfigurable_module<M>`
- sofort verwendbar
- Makro für benutzerdefiniertes Modul: `RC_RECONFIGURABLE_MODULE()`



Portal und Exportal

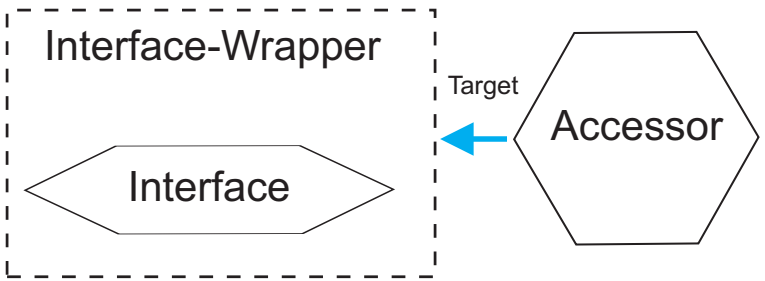
- Switch-Implementationen
- interner Aufbau nach Baukastenprinzip
- benutzerdefinierbar, z. B. mit Makros: `RC_PORTAL()`, `RC_EXPORTAL()`
- Callbacks für Schalterverhalten



Interface-Wrapper und Accessor

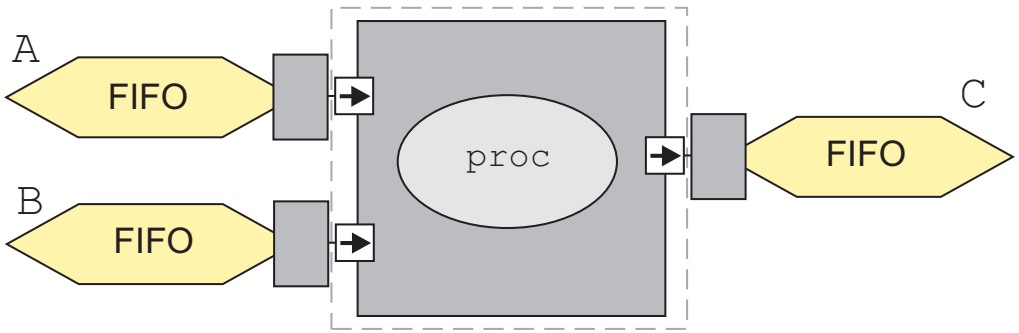
`rc_interface_wrapper<IF>`

- Ermöglicht Zugriff ohne Kenntnis der Interface-Methoden.
  - ⇒ kapselt Interface, Zugriffskontrolle, Event-Weiterleitung, Treiberprozesse, Fallback-Interface, etc. ...
- verschiedene Zugriffsarten (nichtblockierend, blockierend, nb-Treiber, Treiber)
- Lösung für eine Reihe von Problemen



Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:




```
while(true) {
    a = portA.read();
    b = portB.read();
    c = berechnen(a, b);
    portC.write(c);
}
```

ohne Synchronisation

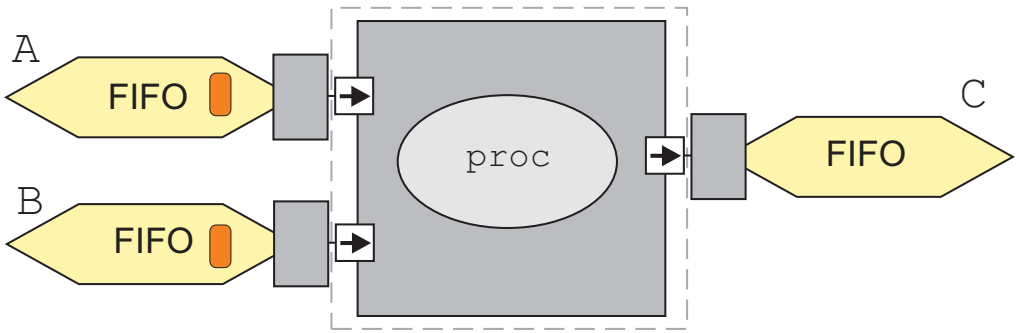
- Deaktivierung zu beliebigem Zeitpunkt

Dateninkonsistenzen

- Datenfluss-Beschreibungen: Reihenfolge u. Zuordenbarkeit von Daten entscheidend

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:




```
 while(true) {
>    a = portA.read();
     b = portB.read();
     c = berechnen(a, b);
     portC.write(c);
}
```

ohne Synchronisation

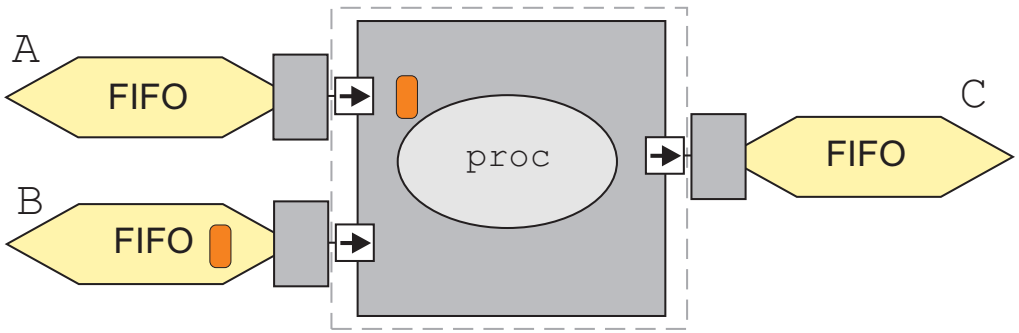
- Deaktivierung zu beliebigem Zeitpunkt

Dateninkonsistenzen

- Datenfluss-Beschreibungen: Reihenfolge u. Zuordenbarkeit von Daten entscheidend

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:




```
 while(true) {
     a = portA.read();
>    b = portB.read();
     c = berechnen(a, b);
     portC.write(c);
}
```

ohne Synchronisation

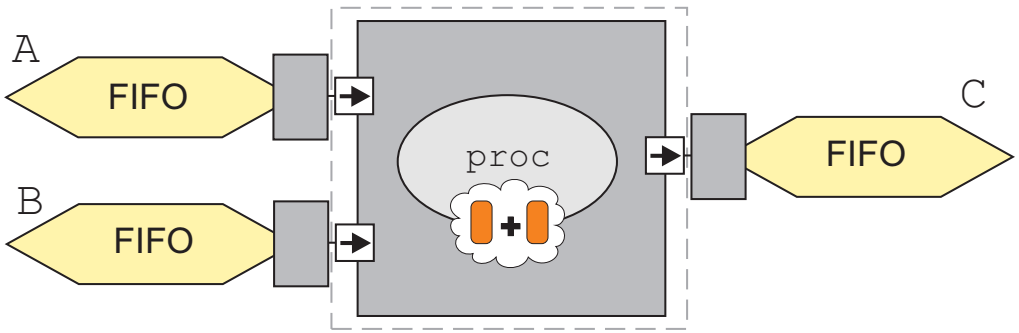
- Deaktivierung zu beliebigem Zeitpunkt

Dateninkonsistenzen

- Datenfluss-Beschreibungen: Reihenfolge u. Zuordenbarkeit von Daten entscheidend

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:




```
 while(true) {
     a = portA.read();
     b = portB.read();
>    c = berechnen(a, b);
     portC.write(c);
}
```

ohne Synchronisation

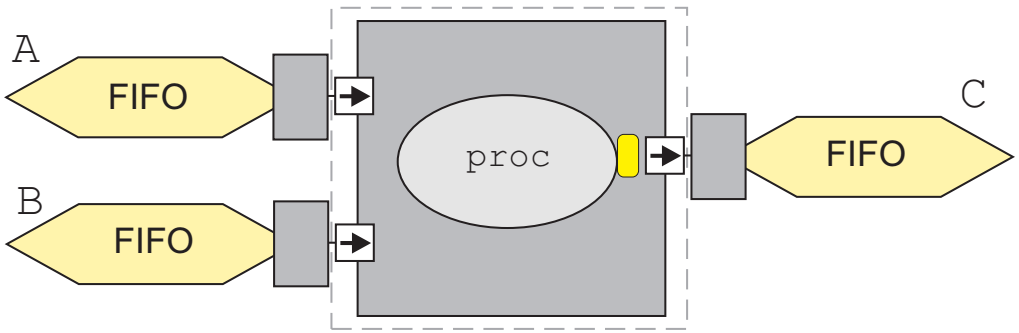
- Deaktivierung zu beliebigem Zeitpunkt

Dateninkonsistenzen

- Datenfluss-Beschreibungen: Reihenfolge u. Zuordenbarkeit von Daten entscheidend

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:




```
 while(true) {
     a = portA.read();
     b = portB.read();
     c = berechnen(a, b);
>    portC.write(c);
}
```

ohne Synchronisation

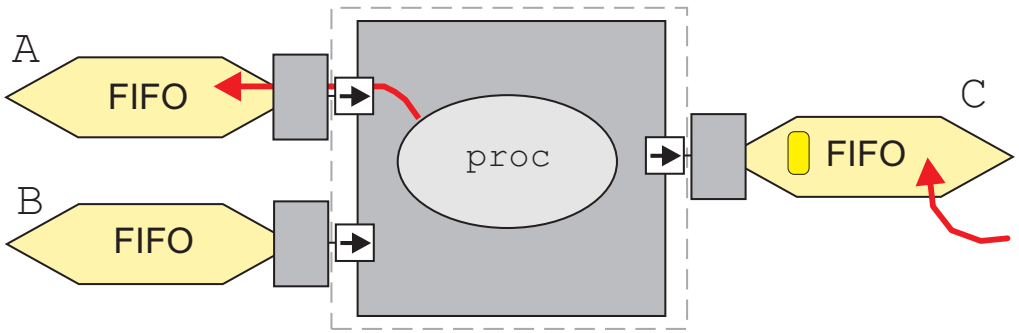
- Deaktivierung zu beliebigem Zeitpunkt

Dateninkonsistenzen

- Datenfluss-Beschreibungen: Reihenfolge u. Zuordenbarkeit von Daten entscheidend

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:



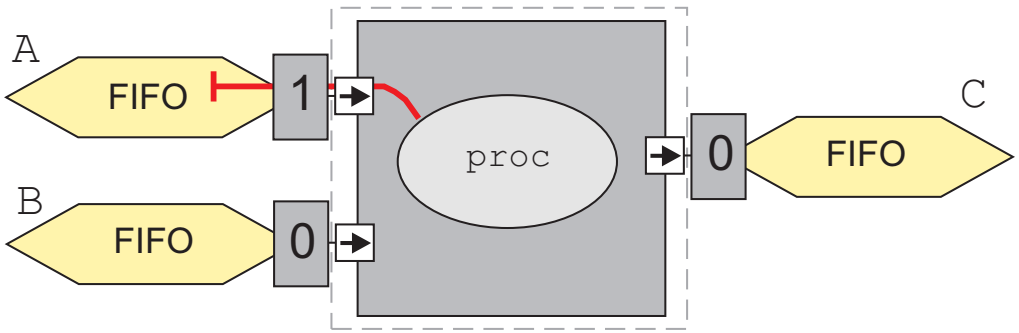

```
 while(true) {
>    a = portA.read();
     b = portB.read();
     c = berechnen(a, b);
     portC.write(c);
}
```

Timing

- Ziel: Deaktivierung zu bestimmtem Zeitpunkt
- Aber: Wenn Ausgabe gelesen werden kann, ist es bereits zu spät
- `proc` beginnt sofort wieder mit dem Lesen weiterer Daten

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:



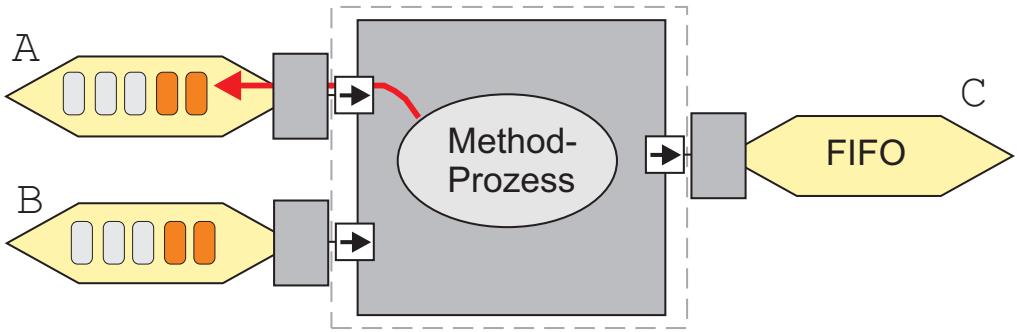

```
 while(true) {
>    a = portA.read();
     b = portB.read();
     c = berechnen(a, b);
     portC.write(c);
}
```

Deadlock

- FIFO leer: Prozess wartet auf neue Daten
- Modul soll entladen werden, da Arbeit getan
- Deaktivierung nicht möglich während externer Zugriff läuft
- $\rightarrow$ Deadlock in der Rekonfigurationskontrolle

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:



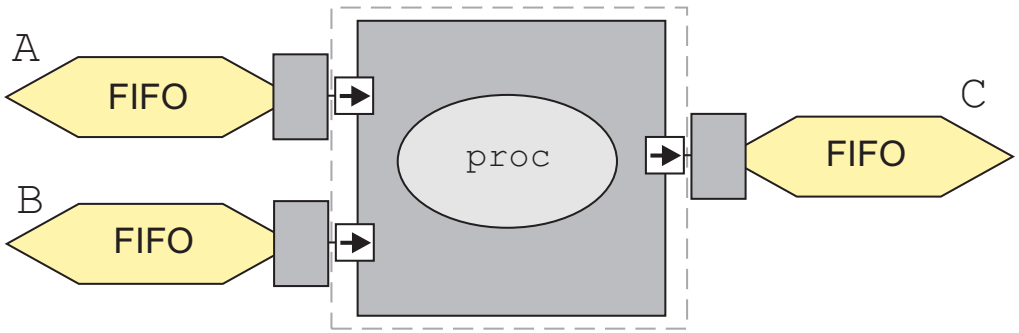
`nb_read()` in einer Schleife

Nichtblockierende Zugriffe

- Grundsätzlich problematisch: Method-Prozesse mit Zugriff auf abstrakte Channel
- Accessor kann nichtblockierende Zugriffe nicht abblocken
- Prozess kann nicht daran gehindert werden, alle Daten auszulesen

Probleme mit abstrakten Beschreibungen

Datenfluss-Beispiel:



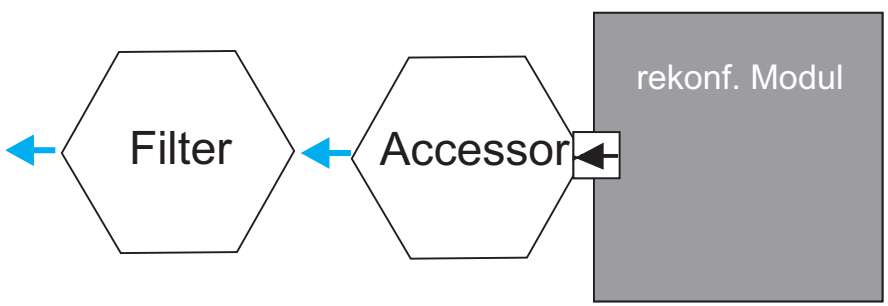
Was muss möglich sein?

- den inneren Zustand eines Moduls anhand externer Zugriffe bestimmen
- Nichtblockierende Zugriffe abblocken (z. B. leere FIFO vortäuschen)
- Transaktionen für Eingabe-/Ausgabe-Zugriffe definieren

Synchronisation der Rekonfiguration

Interface-Filter

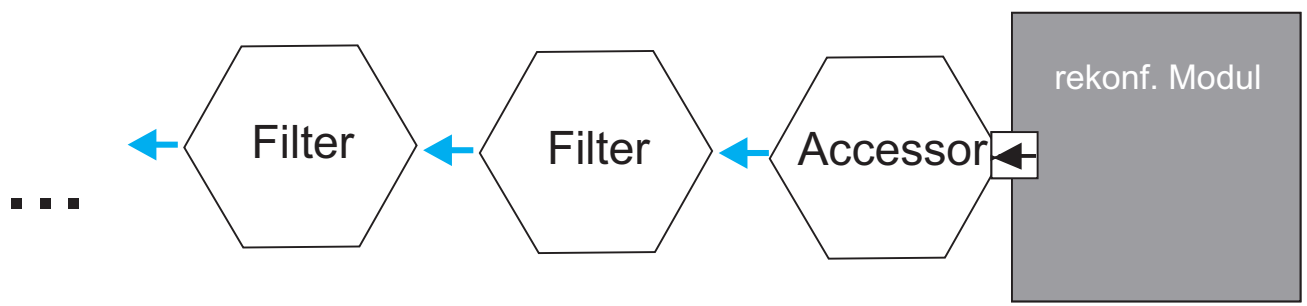
- "filtert" Kommunikation zwischen Accessor und Interface-Wrapper
- Filterung und Manipulation von Zugriffen und Events möglich
- kann daher auch nb-Zugriffe abblocken
- individuell jedem einzelnen Port/Export zuordenbar
- Accessor ist ein Filter



Synchronisation der Rekonfiguration

Interface-Filter

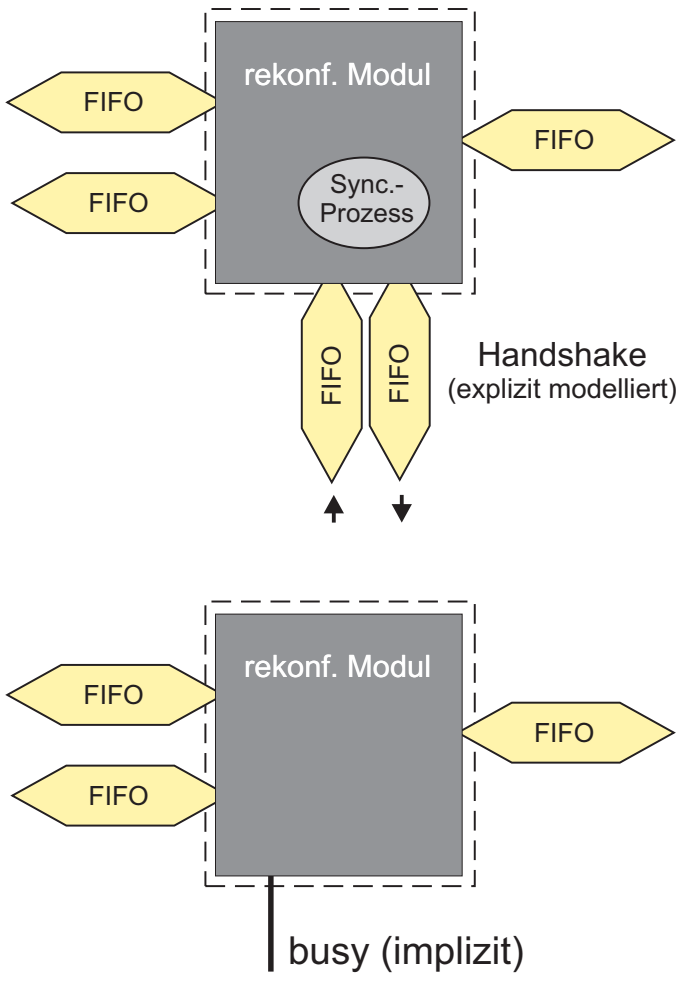
- "filtert" Kommunikation zwischen Accessor und Interface-Wrapper
- Filterung und Manipulation von Zugriffen und Events möglich
- kann daher auch nb-Zugriffe abblocken
- individuell jedem einzelnen Port/Export zuordenbar
- Accessor ist ein Filter



Synchronisation der Rekonfiguration

Synchronisation mit Filtern

- explizite Modellierung von Synchronisationsfunktionalität möglich
- nachträglich erweiterbar mittels Filter-Ketten (“Zwiebelschalenmodell”)
- benutzerdefinierter Filter durch Ableitung von Accessor
- vordefinierte Filter für SystemC-Channel (z. B. FIFO-Filter)



Explizite Modellierung von DRHW

Explizite Modellierung von DRHW mit ReChannel

- Implizite Reset-Funktionalität für Prozesse, Channel und Variablen
- rekonfigurierbare Entsprechung für jede SystemC-Komponente (Prefix: “rc_” und “RC_”)
- Synchronisation von DR mittels Transaktionsblöcken definierbar:

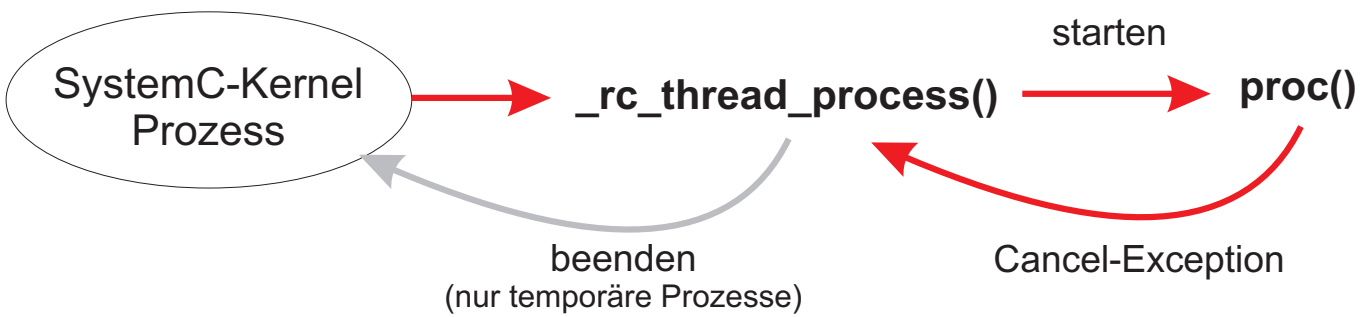
```
RC_TRANSACTION {
  ...
}
```

- Verwendung der Beschreibungen auch im nichtrekonfigurierbaren Kontext möglich

Explizite Modellierung von DRHW

Rücksetzbare Prozesse (Verhalten)

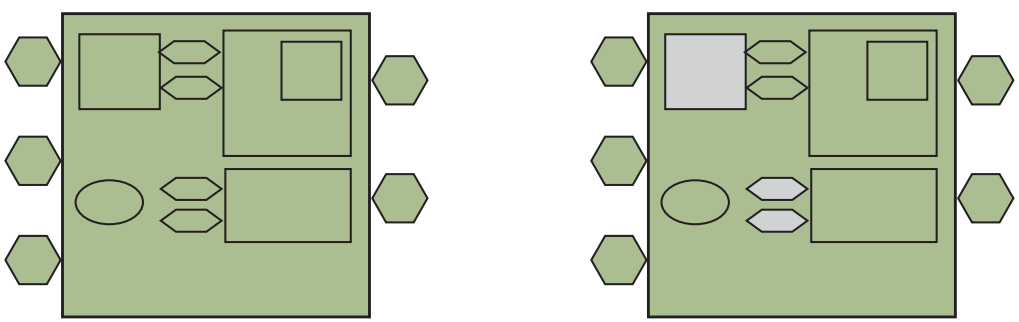
- `RC_HAS_PROCESS()`, `RC_METHOD()`, `RC_THREAD()`, `RC_CTHREAD()`
- `reset_signal_is()` für CThread- und Thread-Prozesse
- Ausführung der Prozesse gebunden an Rekonfigurationszustand
- Prozesse werden von anderem Einsprungpunkt aus gestartet
- Voraussetzung: preparierte wait()-/next_trigger()-Methoden
- Exception wird geworfen, wenn Reset-Bedingung erfüllt ist



Explizite Modellierung von DRHW

Rücksetzbare Komponenten (Struktur)

- Voraussetzung: Komponente implementiert `rc_resettable` und registriert sich beim rekonfigurierbaren Kontext
- `rc_on_reset()` wird vor Aktivierung und bei Deaktivierung des Moduls aufgerufen
- Vordefiniert: `rc_module`, `rc_signal`, `rc_fifo`, `rc_event`, etc.
- Rücksetzbare Variablen mittels `rc_var` definierbar
- Mischung von SystemC- und ReChannel-Komponenten möglich



Zusammenfassung

Zusammenfassung

- ReChannel-Bibliothek zur Simulation von DR in SystemCwurde vorgestellt
- ReChannel-Funktionalität und Problemstellungen wurden analysiert
- Re-Design von ReChannel wurde durchgeführt
  - Abstraktion des ReChannel-Algorithmus und Datenstrukturen
  - bekannte und neu entdeckte Probleme wurden beseitigt
  - Exportals wurden implementiert
  - Interface-Filter zur Lösung des Synchronisationsproblems
  - Spracherweiterung zur expliziten Modellierung von DRHW


IEEE Standards Association ("IEEE-SA") Standards Board.

IEEE Std 1666-2005 Open SystemC Language Reference Manual, 2005.

http://www.systemc.org.


Open SystemC Initiative (OSCI).

SystemC.

http://www.systemc.org.


Open SystemC Initiative (OSCI).

TLM Transaction Level Modeling Library, Release 2.0 Draft 1, 2006.

http://www.systemc.org.


Raabe, A., Hartmann, P. A., and Anlauf, J. K.

Rechannel: Describing and Simulating Reconfigurable Hardware in SystemC.

ACM Transactions on Design Automation of Electronic Systems (accepted).