

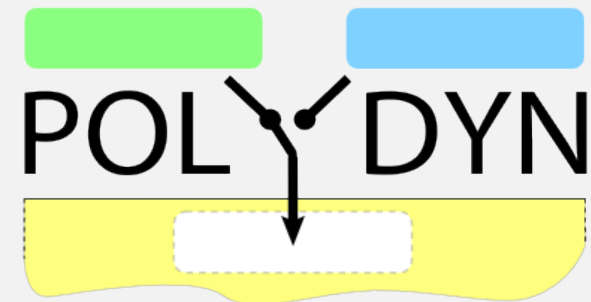
# Automatische Synthese rekonfigurierbarer Systeme

Zwischenkolloquium der dritten Förderperiode 2008

SPPRR 1148

27. und 28.05.2008

Andreas Schallenberg  
Universität Oldenburg



# Gliederung

- **Motivation**
- Gewählter Ansatz
- Schlaglichter auf Neuerungen und Erweiterungen
- Zusammenfassung

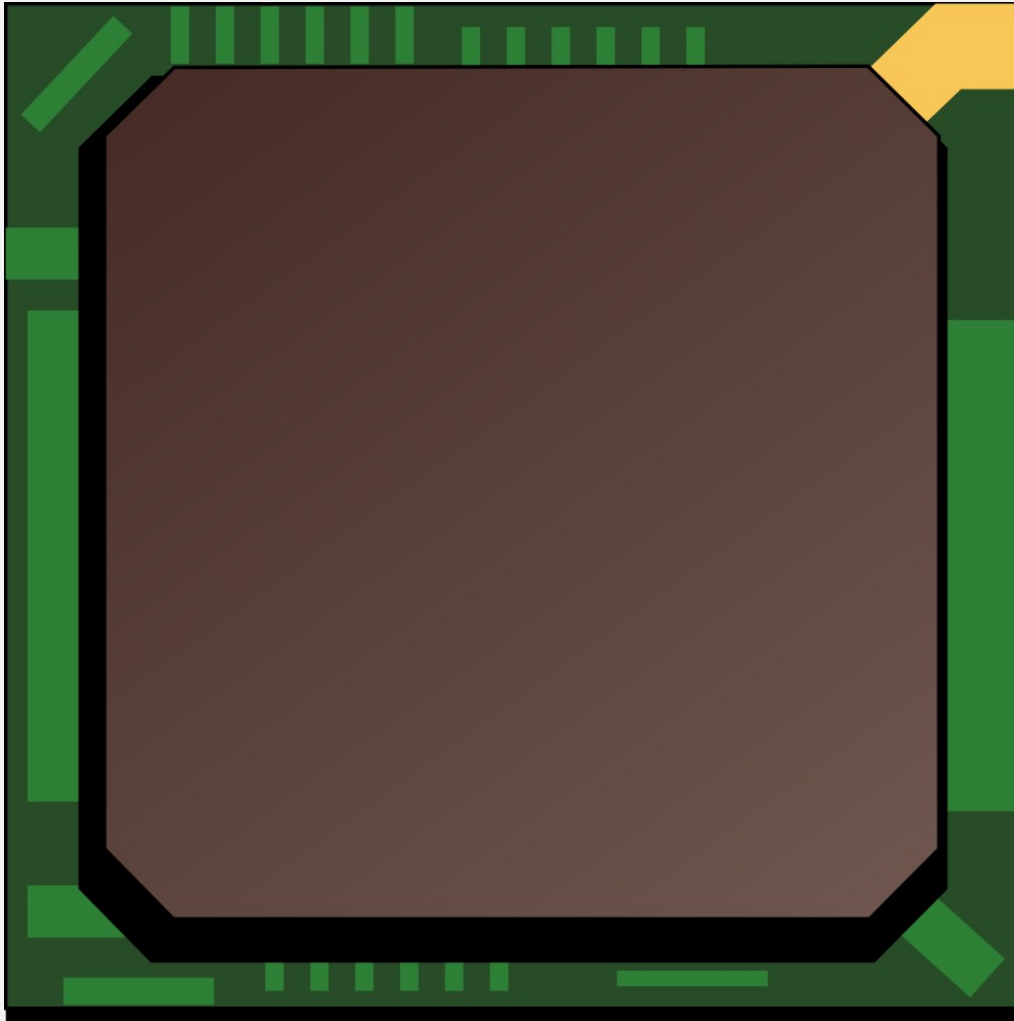
# Partielle Konfiguration (DPR) auf heutigen Standard-FPGAs

Geeignete FPGAs sind am Markt und im Einsatz,  
dennoch wird mit ihnen selten partiell konfiguriert.

Gründe:

- Vorsicht: Wenige Applikationen verlangen nach DPR, Entwickler bleiben im Zweifel bei Bekanntem
- Time-to-market: Hoher Entwicklungsaufwand
- Kosten: DPR muss Kosten senken, nicht erhöhen

# Typische Entwicklungsaufgaben



- Konfigurationssteuerung
- Arbitrierung des Konfigurationsports
- Buchhaltung über Konfigurationen
- Auswahl der rekonfig. Fläche („Placement“)
- Konfliktfreie Kommunikation

# Entwurfsentscheidungen

- Wenig Entscheidungen vom Designer zwingend fordern
- Wenn möglich, viele änderbare Voreinstellungen
- Möglichst viel Umsetzung an Werkzeuge abgeben

Nachteil:

Einschränkung des Entwurfsraumes

Vorteil:

Lösungen in kurzer Zeit erreichbar

# Gliederung

- Motivation
- **Gewählter Ansatz**
- Schlaglichter auf Erweiterungen und Neuerungen
- Zusammenfassung

# Gewählter Ansatz

High-Level Modellierung:

- Abstrakt: Objekt-Orientiert, methodenbasiert
- SystemC domänenspezifische Bibliothek
- „Bedarf ausdrücken, nicht Verwaltungsschritte“
- Simulation
- Zeitliche Analyse, Visualisierung
- Automatische Synthese

# Annahmen über Entwurfsraum

- Rekonfigurierbare Flächen:
  - Feste Anordnung (Ort und Größe)
  - Dienstanbieter: Reaktives Verhalten
- Selbstrekonfiguration, kein Prozessor erforderlich
- Keine unnötigen Konfigurationen
- Dynamische Anforderungen: Online-Scheduling

# Kontext-Objekte

Kontexte aus Sicht der Applikationslogik:

- Unbeschränkte Lebenszeit (fester Satz!)
- Verwendbar wie alle anderen Variablen
- Polymorph (Laufzeittyp änderbar)

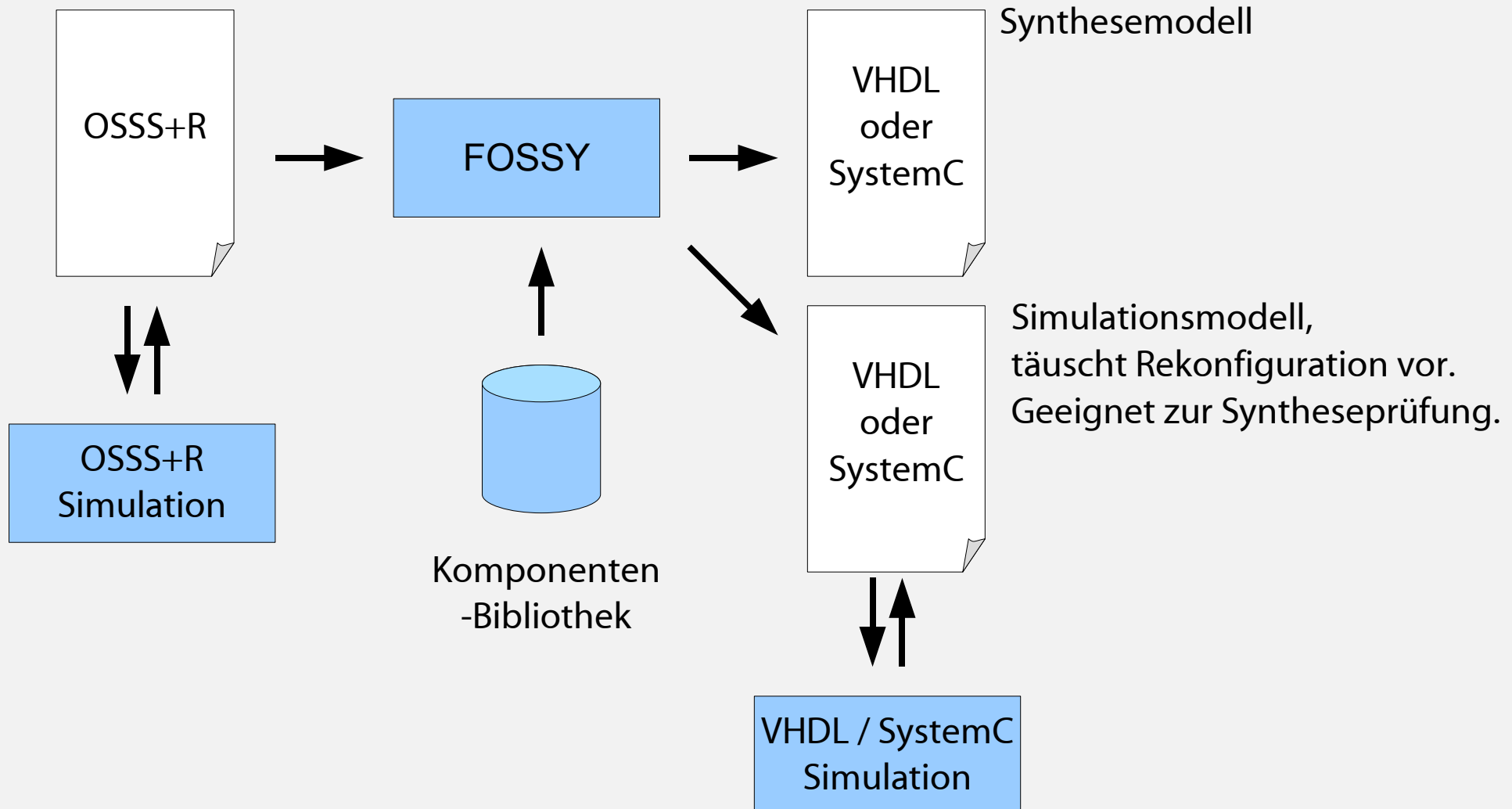
Kontexte aus Sicht der Implementation:

- Tupel aus Laufzeittyp und Wertebelegung
- Teilen sich eine FPGA-Fläche (ggf. Verdrängung)
- Bei Verdrängung ist Persistenz zu erhalten

# Gliederung

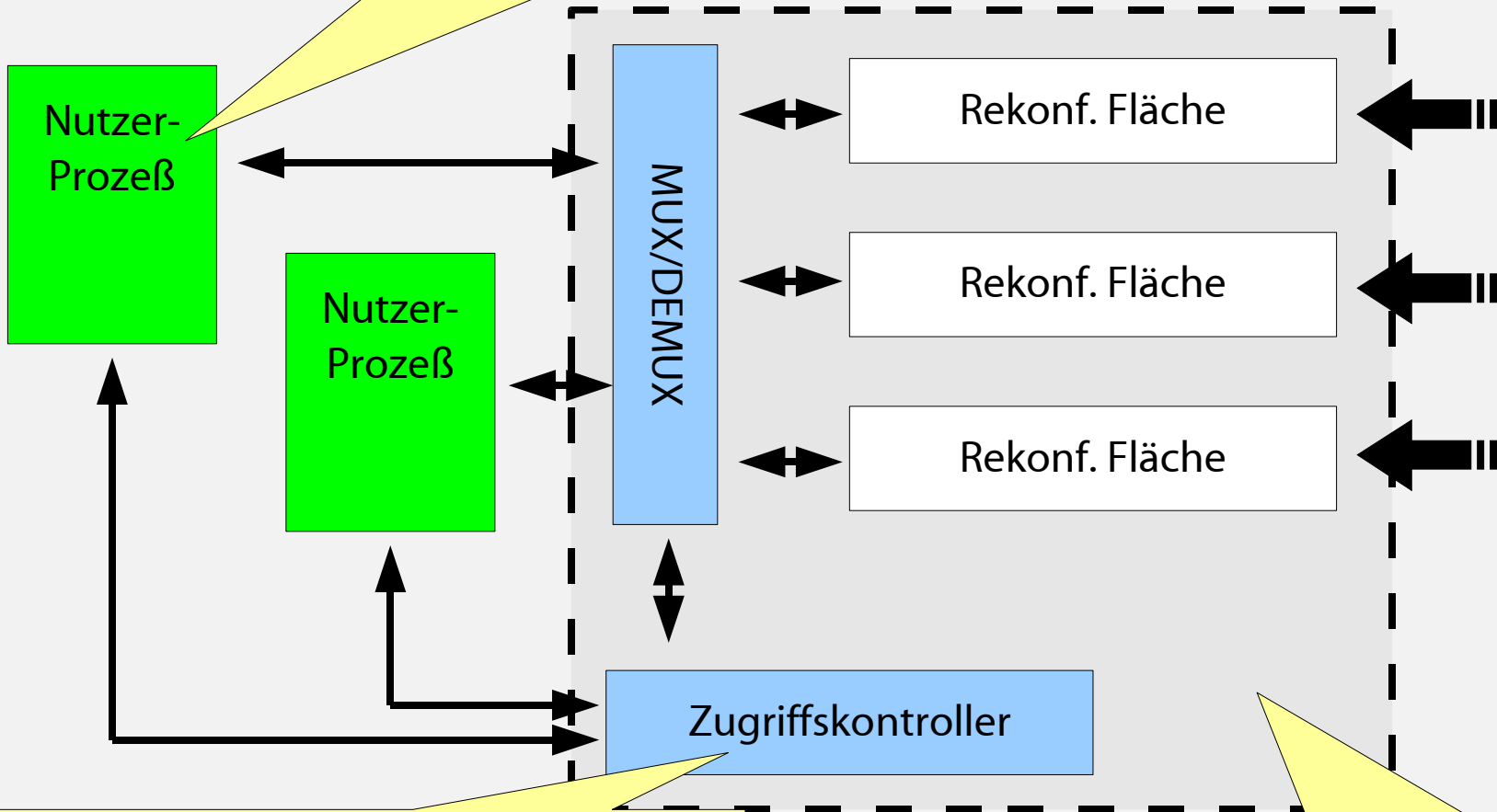
- Motivation
- Gewählter Ansatz
- **Schlaglichter auf Erweiterungen und Neuerungen**
  - Platzierung und Scheduling
  - Prefetching
  - Post-Synthese Simulation
  - Burst-Zugriffssequenzen
  - Visualisierung
- Zusammenfassung

# Entwurfsfluss



ement

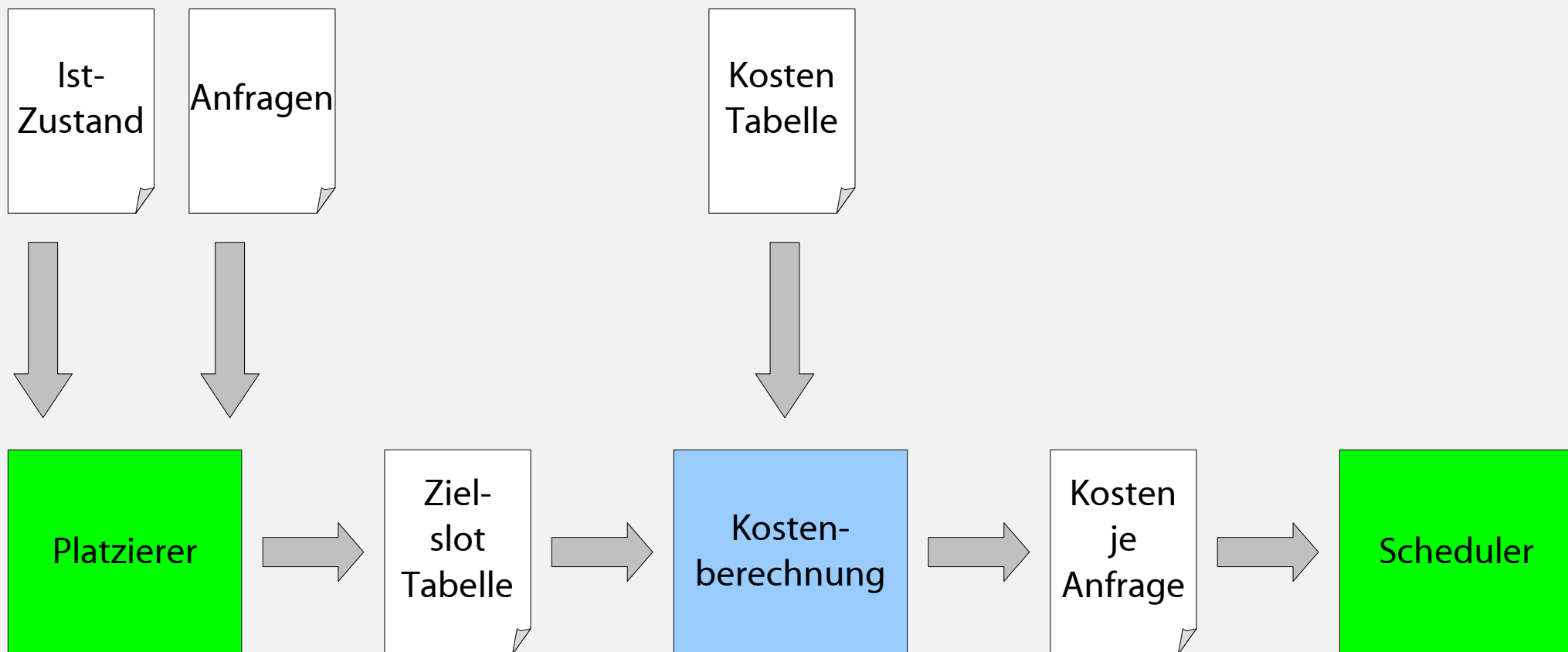
Platzierungsentscheidung ist für die Nutzerprozesse transparent



```
...
ro.setPlacer<myPlacerAlg>( );
...
```

Ein rekonfigurierbares Objekt. Hier: Verwaltet mehrere rekonfigurierbare Flächen

# Zusammenspiel Scheduling & Platzierung



# Prefetching

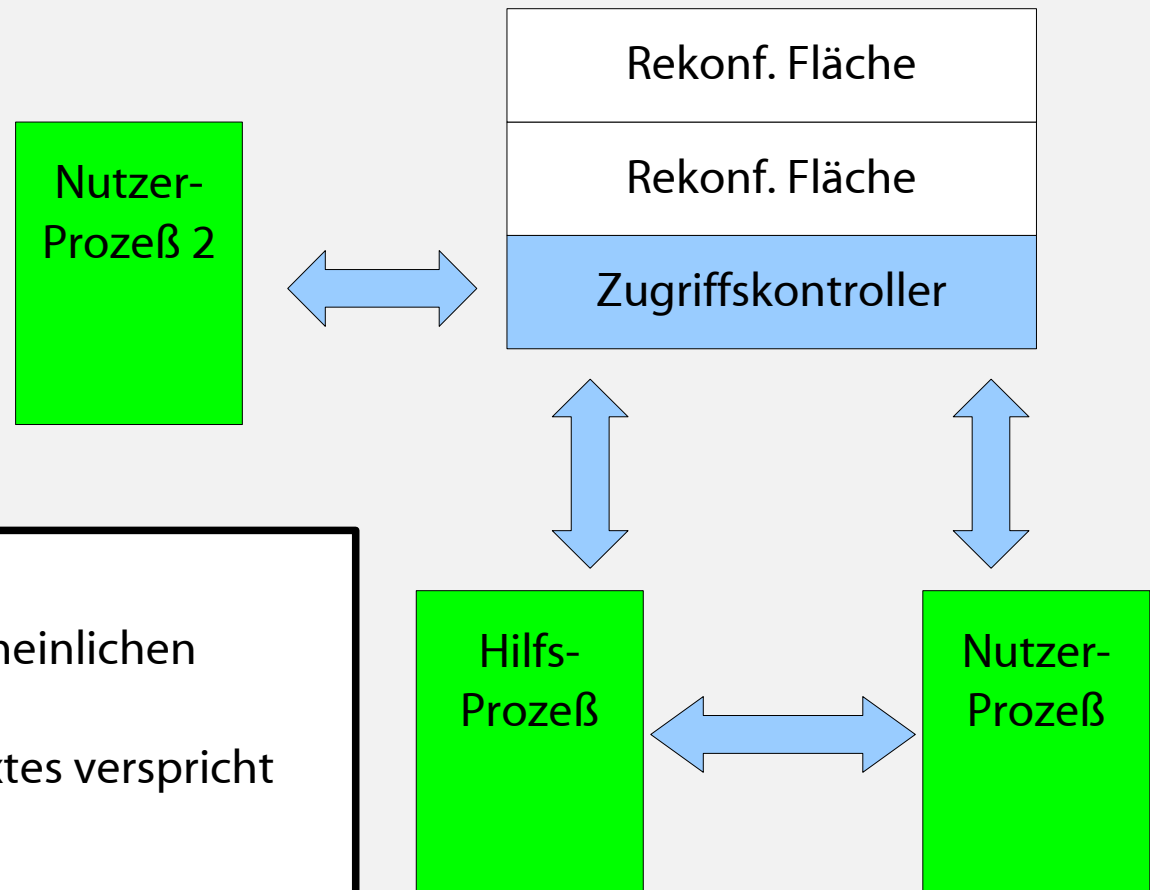
Problem:  
Konkurrierende Nutzung  
der rekonfigurierbaren  
Flächen kann zu **Thrashing**  
führen!

Gegeben:

- Ein Prozess kennt seine wahrscheinlichen zukünftigen Bedarf
- Vorladen des benötigten Kontextes verspricht Zeiteinsparungen

Frage:

Wie lade ich Kontexte vor ihrer Nutzung?



# Lösung 1: Kontextinterne Sperre

```
class A : public B
{
public:
bool prefetched;
...
}
```

```
OSSS_NEW_RELEASE_GUARD( !prefetched );
```

```
void prefetch(){ prefetched = true; }
```

```
...
```

```
void do_something(...){ prefetched = false; ... }
```

## Semantik:

„Objekte vom Typ A dürfen nur dann aus der rekonfigurierbaren Fläche entfernt werden, wenn die Bedingung wahr ist.“

Einsatzzwecke:

- „Schutz“ transienter Attribute
- Vermeiden von Thrashingeffekten

# Lösung 2:

## Externe Sperre durch Hilfsprozeß

```
...  
// Auf Prefetch-Anfrage warten  
...  
  
OSSS_KEEP_ENABLED( my_object )  
{  
    my_object->enable();  
    ...  
    // Auf Nutzungsbestätigung warten  
    while (false == signal_used.read()){ wait(); }  
    ...  
}  
...
```

Hinweis:

Beide Lösungen verhindern das Verdrängen des Kontextes, erlauben aber weiterhin die konkurrierende Nutzung durch mehrere Prozesse !

# „Burst“-Zugriffssequenzen

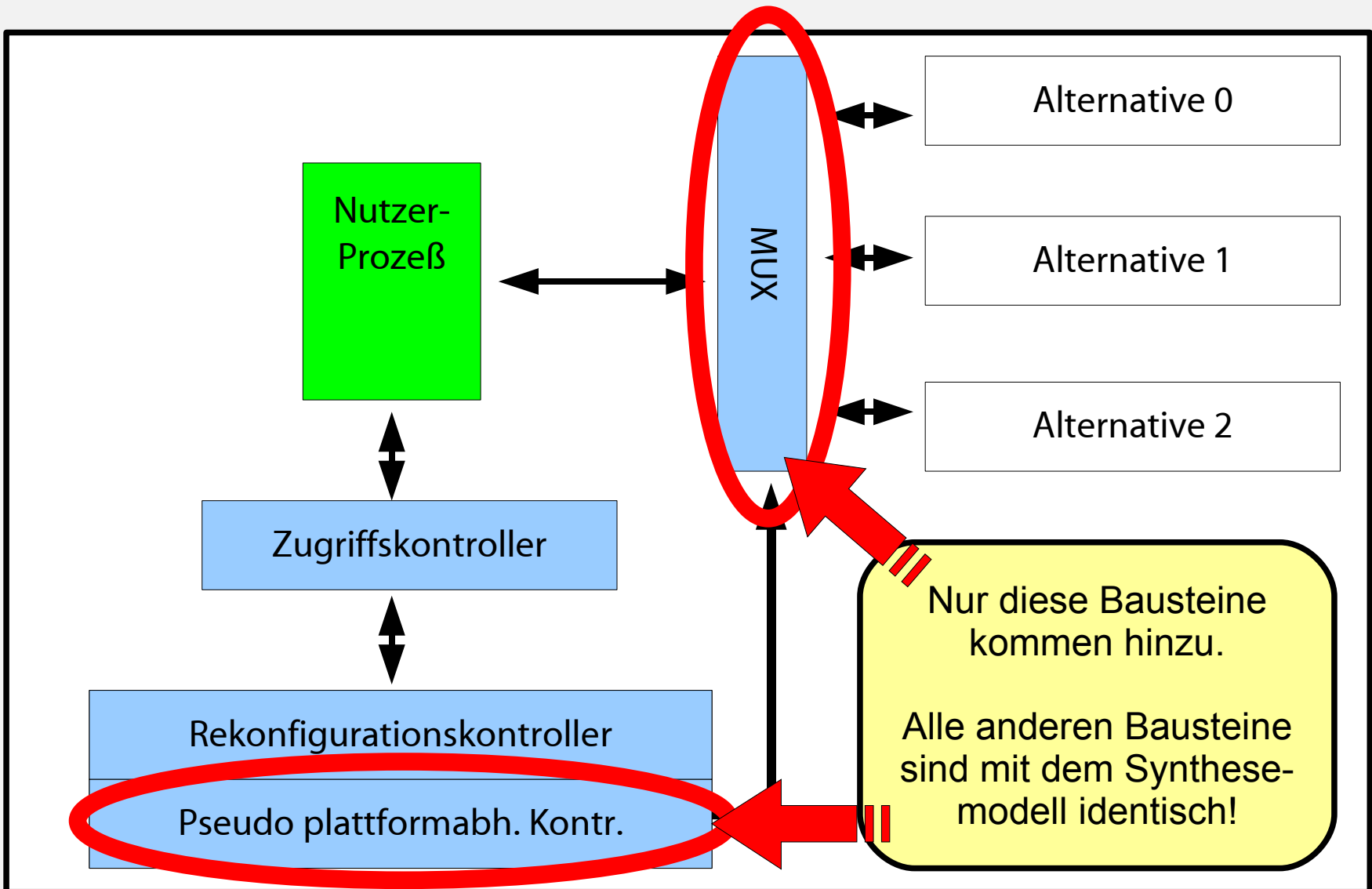
```
...  
OSSS_LAZY_PERMISSION_RETURN( my_ro )  
{  
    ...  
    my_ro->do_this();  
    ...  
    my_ro->do_that();  
    ...  
}  
...
```

## Semantik:

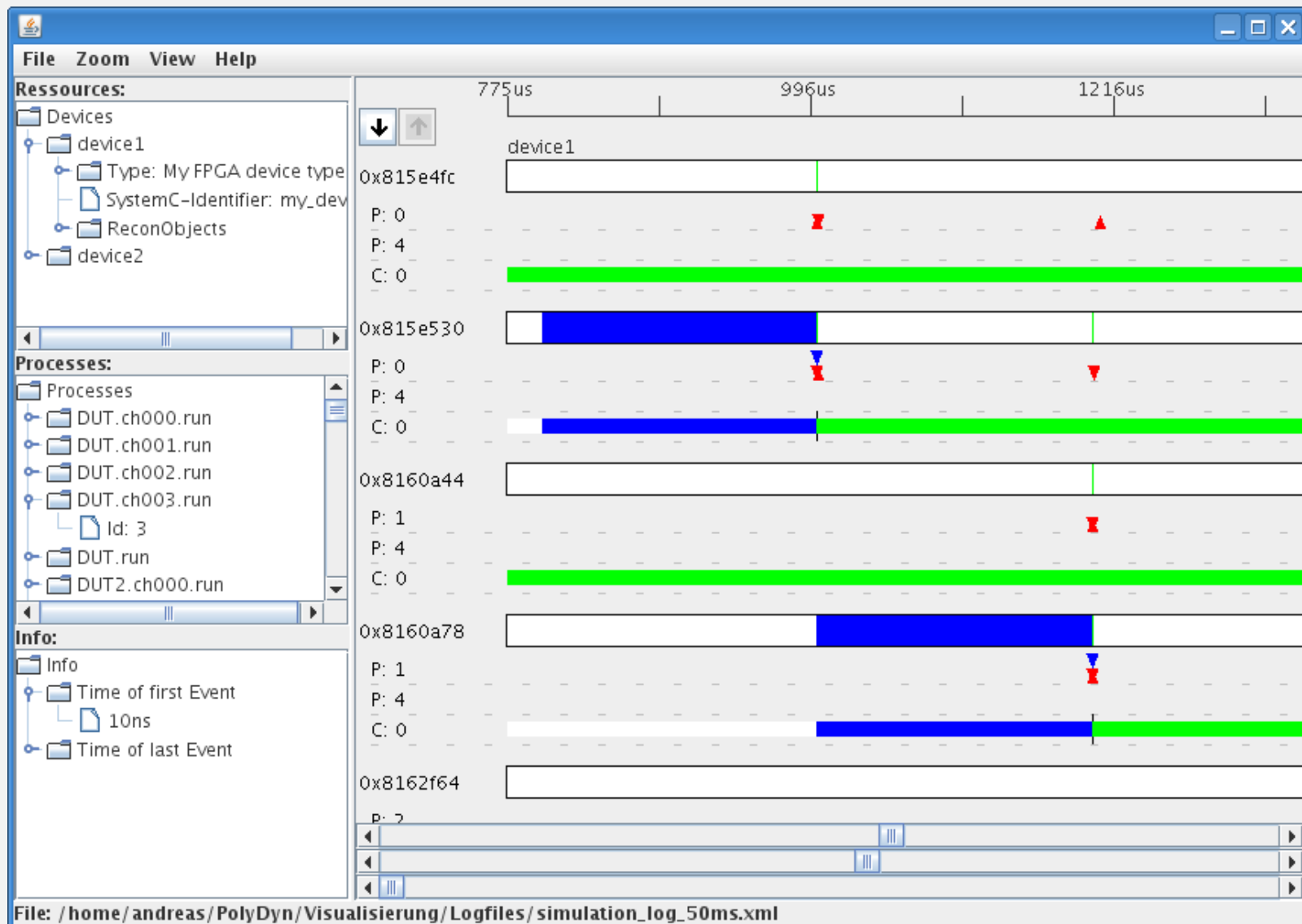
„Behalte Zugriffsberechtigungen für alle an my\_ro gebundenen Kontexte nach einem Zugriff. Gebe sie zurück:  
a) beim Zugriff auf einen anderen Kontext  
b) beim Verlassen des Blocks.“

- Kein Zugriff anderer Prozesse auf das Objekt, solange der Prozeß im Block ist
- Zugriffe werden schneller, da keine Zugriffsrechte mehr anzufordern sind

# Simulationsmodell nach Synthese



# Visualisierung



# Gliederung

- Motivation
- Gewählter Ansatz
- Schlaglichter auf Erweiterungen und Neuerungen
- **Zusammenfassung**

# Flächeninformationen

Prototypenboard Xilinx ML401 mit Virtex 4 LX-25

Beispiel eines Wellenformgenerators, Flächenbedarf:

Zugriffskontroller 1	0,10%
Zugriffskontroller 2	0,10%
Rekonfigurationskontroller (applikationsabh.Teil)	1,00%
Rekonfigurationskontroller (plattformabh.Teil)	1,30%

TODO: Flächenbedarf durch Veränderung der Nutzerprozesse ist hieraus noch nicht ersichtlich.

# Neue Publikationen

- Ralph Görgen, Frank Oppenheimer, Andreas Schallenberg, Wolfgang Nebel  
*Analyse und Optimierung von dynamisch rekonfigurierbaren Systemen mittels Ereignisvisualisierung*  
Tagungsband des 11. ITG/GMM/GI-Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen"
- Schallenberg, Andreas *OSSR+R: Modelling and Simulation Self-Reconfigurable Systems*, EDAA / DATE PhD Forum 2008
- Fernando Herrera, Eugenio Villar, Philipp A. Hartmann. *Specification of HW/SW adaptive Embedded Systems in SystemC*.  
In Proc. of the Forum on specification and Design Languages 2008, FDL08. Stuttgart. Germany
- Ralph Görgen, Frank Oppenheimer, Wolfgang Nebel, *Adaptive Scheduling of Dynamic Reconfiguration in Stream-Based Applications*  
Proceedings - ReCoSoC '08 (submitted)

# Ergebnisse PolyDyn

- Modellierungskonzept für DPR
  - High-Level
    - Vermeidung „unnötiger“ Spezifikationsarbeiten
- Alle Elemente simulierbar
- Synthese für grundlegende Elemente
- Debugging:
  - Trace-Visualisierung
  - Post-Synthese Simulation

Vielen Dank für Ihre Aufmerksamkeit

