technische universität
dortmund

Eine Familie von gemeinsamen Speichern für MPSoCs

Abschlussvortrag

Diplomarbeit

Diplomant: David Austin

Betreuer: Prof. Olaf Spinczyk

Matthias Meier

Themenüberblick

- **Einführung**
- Speicher Familie
- Implementierung
- Konfigurierung
- Evaluierung
- Fazit

Einführung

- Kontext der Diplomarbeit:
  - Eine Familie von gemeinsamen Speichern für MPSoCs
  - im Rahmen des LavA-Projekts
  - anpassbar an die Anforderungen der Applikation
- merkmalsgetriebene Entwicklung
  - Konzepte der Produktlinienentwicklung auf Hardware übertragen
  - Konfigurierung des Systems durch XVCL
- betrachtete Speichermodelle
  - Transaktions-Speicher
  - Stream-Speicher
  - üblicher Shared-Memory

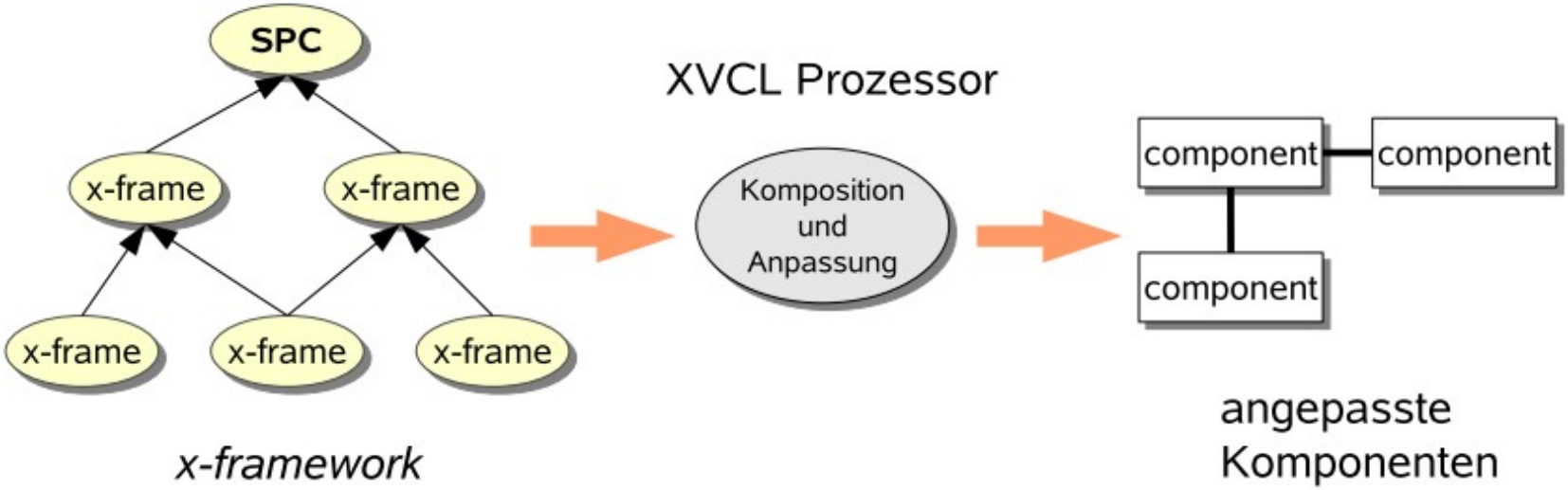
Einführung - Transaktionsspeicher

- TM-System überwacht Speicherzugriffe
  - Read-Set und Write-Set für Speicherzugriffe
    - Lazy oder Eager -Versionsverwaltung
  - automatische Konflikterkennung
    - optimistische oder pessimistische Konflikterkennung
  - automatisches zurücksetzen des Speicherzustandes

- Vorteile:
  - einfacheres Programmiermodell
    - nur ein Sprachmittel, z.B. atomic{}
    - keine unterschiedlichen Locks, die verwaltet werden müssen
  - keine Deadlocks oder Prioritätsumkehr
  - optimistische parallele Ausführung, statt gegenseitigen Ausschluss
  - einfacheres Debugging

Einführung - XVCL

- Auszeichnungssprache zum Konfigurieren von Quellcode
  - basierend auf XML
  - in Java implementiert
  - Quellcode wird in x-frames eingebettet
  - 15 Sprachkonstrukte (XML-Tags)

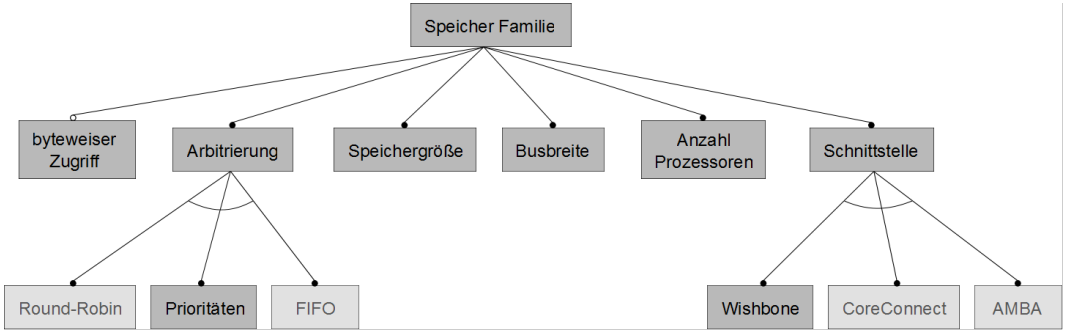


Themenüberblick

- Einführung
- **Speicher Familie**
- Implementierung
- Konfigurierung
- Evaluierung
- Fazit

Speicher-Familie

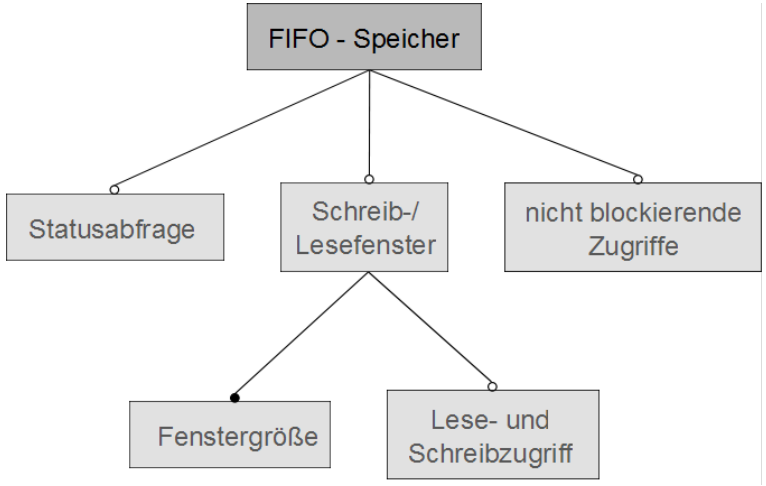
- Gemeinsamkeiten der 3 Speichermodelle
  - Konfigurierbare Größen:
    - Anzahl angeschlossener Prozessoren
    - Busbreiten (8, 16, 32Bit)
    - Speichergröße (Anzahl BlockRAMs)



  - Funktionale Merkmale:
    - Arbitrierung des Busses
    - Bytes einzeln lesen/schreiben (bei Busbreiten > 8Bit)
    - Verwendete Schnittstellenstandard zur CPU / Speicher

Speicher-Familie

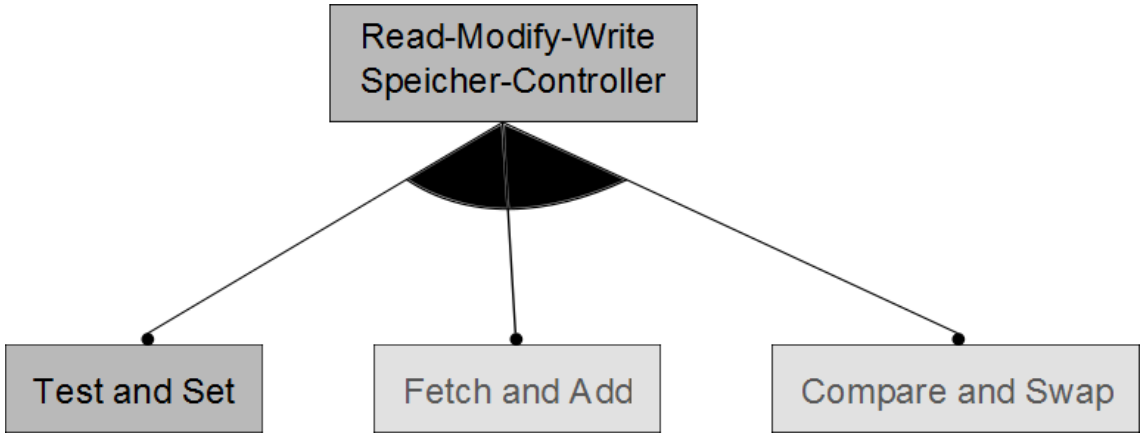
- FIFO-Speicher
  - Statusabfrage:
    - ist Speicher voll/leer?
    - wie viele Bytes sind belegt/frei?
    - war letzter Zugriff erfolgreich?



  - Schreib- /Lesefenster:
    - schreiben/lesen von Objekten > Busbreite
    - FIFO-Speicher nutzen um Daten zu verarbeiten => kein unnötiges kopieren von Daten

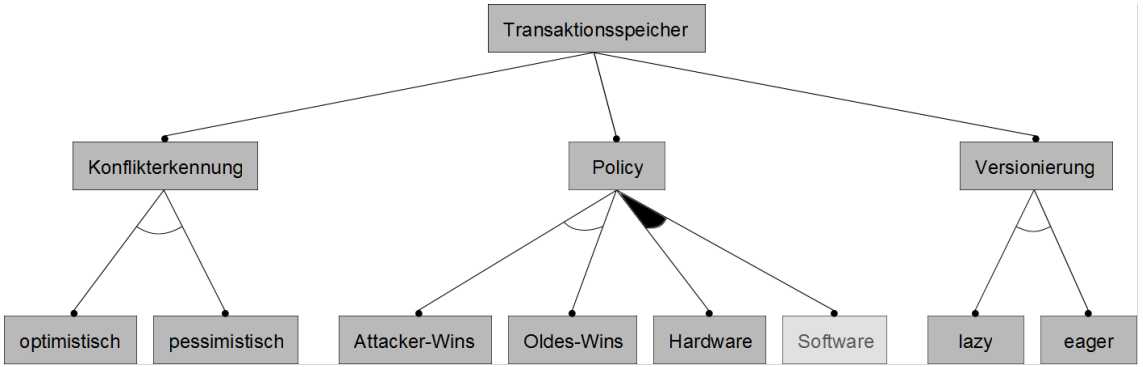
Speicher-Familie

- read-modify-write Speichercontroller
  - ermöglicht Synchronisation zwischen mehreren Prozessoren
    - Test and Set: altes Datum lesen und überschreiben
    - Fetch and Add: Datum lesen, inkrementieren und zurückschreiben
    - Compare and Swap: Datum lesen, mit einem beliebigen Wert auf Gleichheit testen und falls gleich neues Datum zurückschreiben



Speicher-Familie

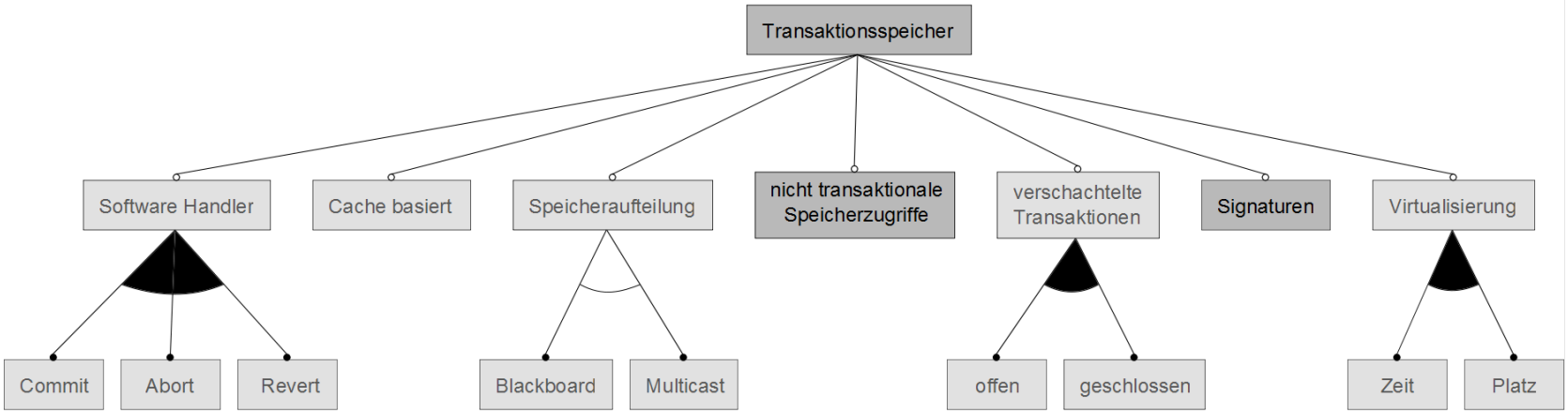
- Notwendige Merkmale des TM-Systems
  - Konflikterkennung:
    - direkt beim Speicher-zugriff
    - im Rahmen des Commits



  - Policy:
    - Vielzahl an Möglichkeiten
    - z.B. Besitzer des Busses (attacker) oder älteste (oldest) Transaktion gewinnt
  - Versionierung:
    - neues Datum im Zwischenspeicher bis Commit
    - neues Datum im Hauptspeicher / altes Datum im Zwischenspeicher

Speicher-Familie

- Optionale Merkmale des TM-Systems
  - Speicherzugriffe außerhalb von Transaktionen:
    - z.B. Teil des Hauptspeichers dient als privater Speicher für CPUs
  - Signaturen des read-/ write-sets für schnelle Überprüfung auf Konflikte

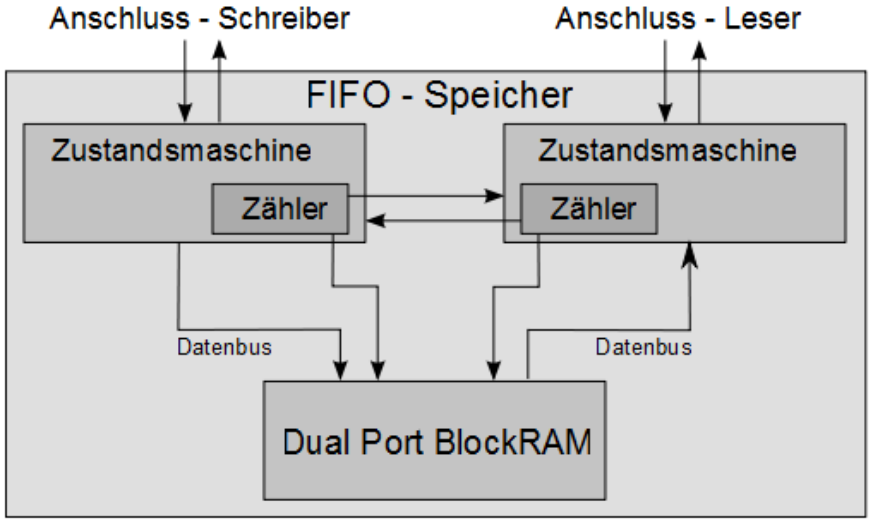


Themenüberblick

- Einführung
- Speicher Familie
- **Implementierung**
- Konfigurierung
- Evaluierung
- Fazit

Implementierung

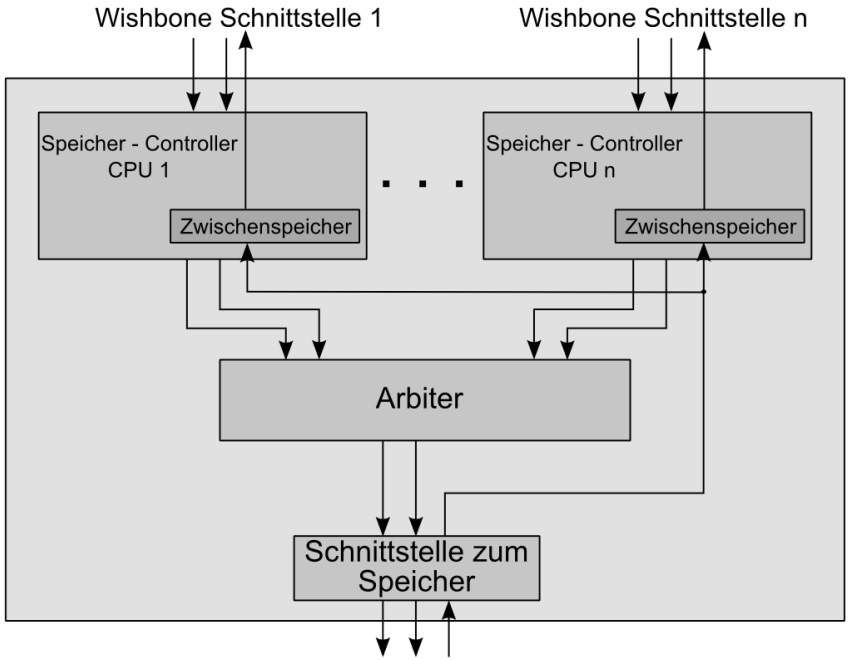
- FIFO-Speicher:
  - ein Schreiber/Leser
  - Zustandsmaschine regelt Wishbone konforme Kommunikation
  - Speicherzugriffe blockieren, falls Speicher voll/leer



  - memory mapped I/O => eine Speicheradresse zum Schreiben/Lesen
  - dual port BlockRAM ermöglichen parallelen Zugriff => Arbiter nicht nötig
  - unabhängige Takte für Schreiber/Leser

Implementierung

- Read-modify-write Speichercontroller:
  - Prioritäten-Arbiter
    - variable Anzahl an Anschlüssen
    - Speichercontroller muss Bus selbstständig wieder freigeben
  - Speicher-Controller pro CPU
    - Wishbone konforme Kommunikation
    - Zwischenspeicher für gelesenes Datum



  - Wishbone Schnittstelle zum Speicher
  - Einleiten einer Test and Set Operation über spezielle Adresse

Implementierung

- Read-modify-write Speichercontroller:
  - Code Beispiel:

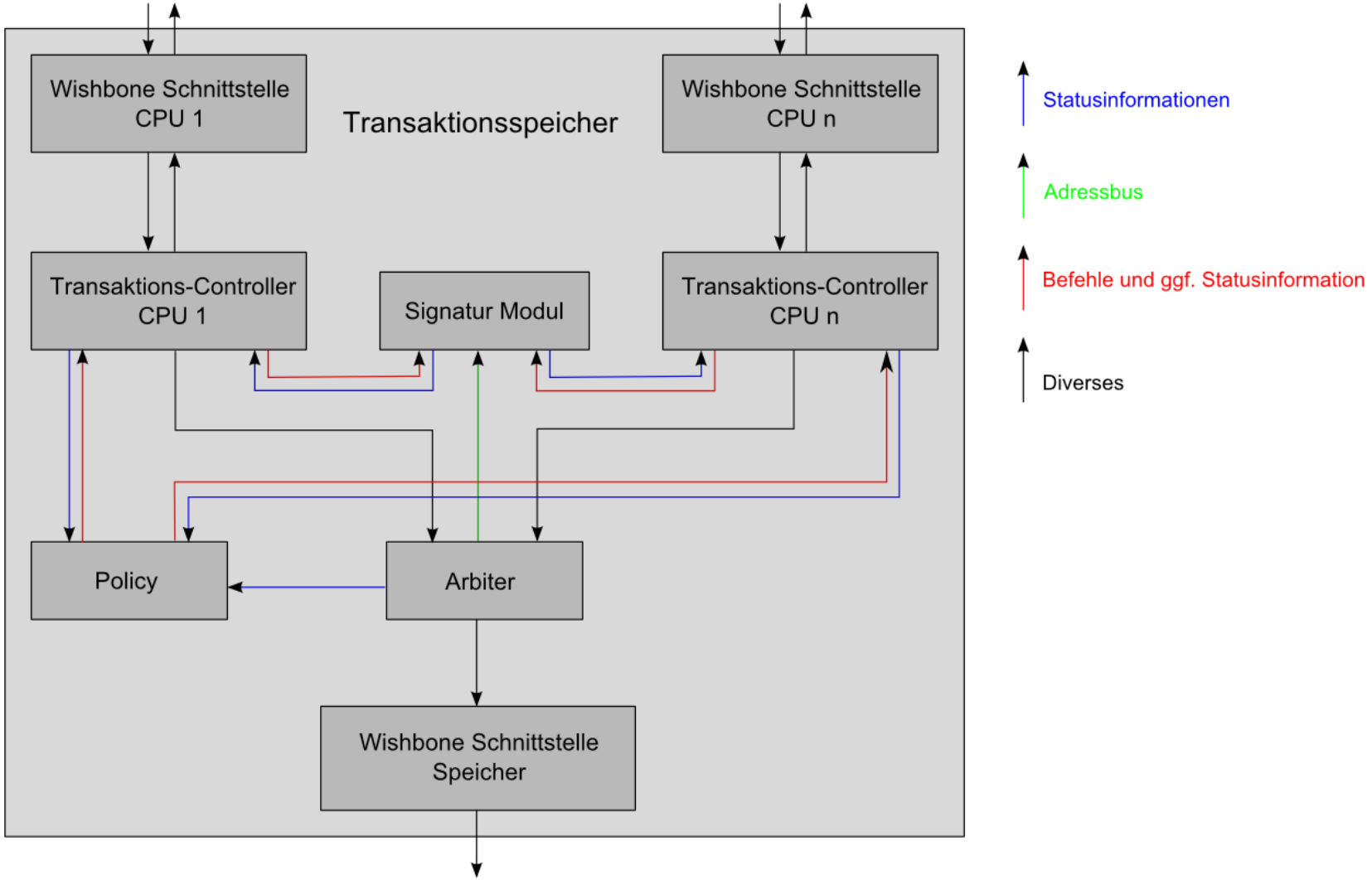
```
// fiktive Adresse des Registers für Test and Set Operationen
#define test_and_set_reg ((volatile int*)(0x1000))

int function test_and_set(int* var, int set_val) {
    *test_and_set_reg = 0;
    *var = set_val;
    return *test_and_set_reg;
}
```

Implementierung

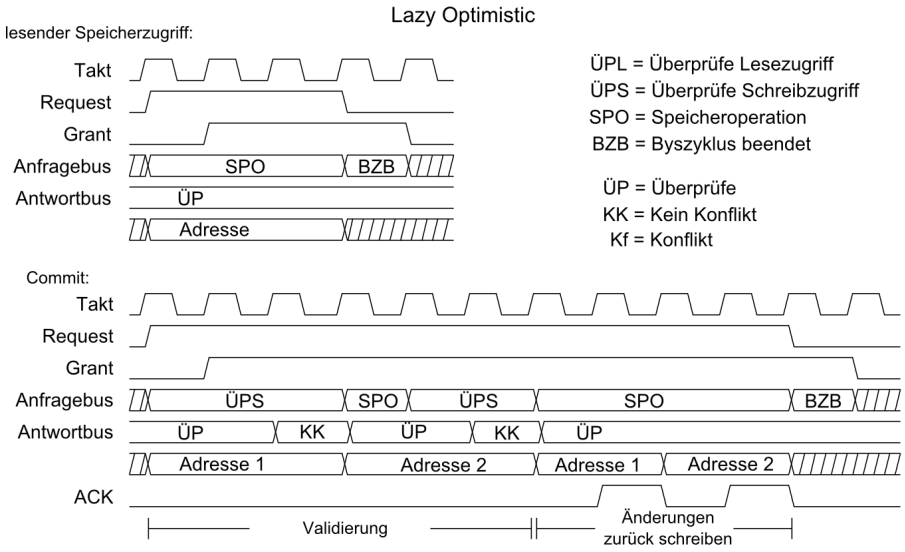
- Transaktionsspeicher:
  - Wishbone Schnittstelle für CPUs und Speicher
    - Register zur Steuerung einer Transaktion (Begin, Commit, Abort)
    - Register zum Auslesen des Status
    - Generierung der Interrupts
  - Transaktions-Controller steuert Speicherzugriffe
    - Backup der alten Werte
    - Anstoßen von Konflikterkennungsmaßnahmen
    - Commit/Rollback
  - Policy löst Konflikte
    - Attacker-Wins: Transaktion die den Bus akquiriert hat gewinnt
    - Oldest-Wins: älteste Transaktion gewinnt
  - Signatur Module prüft auf Konflikte
  - Prioritäten-Arbiter
  - keine Veränderungen an CPU notwendig

Implementierung



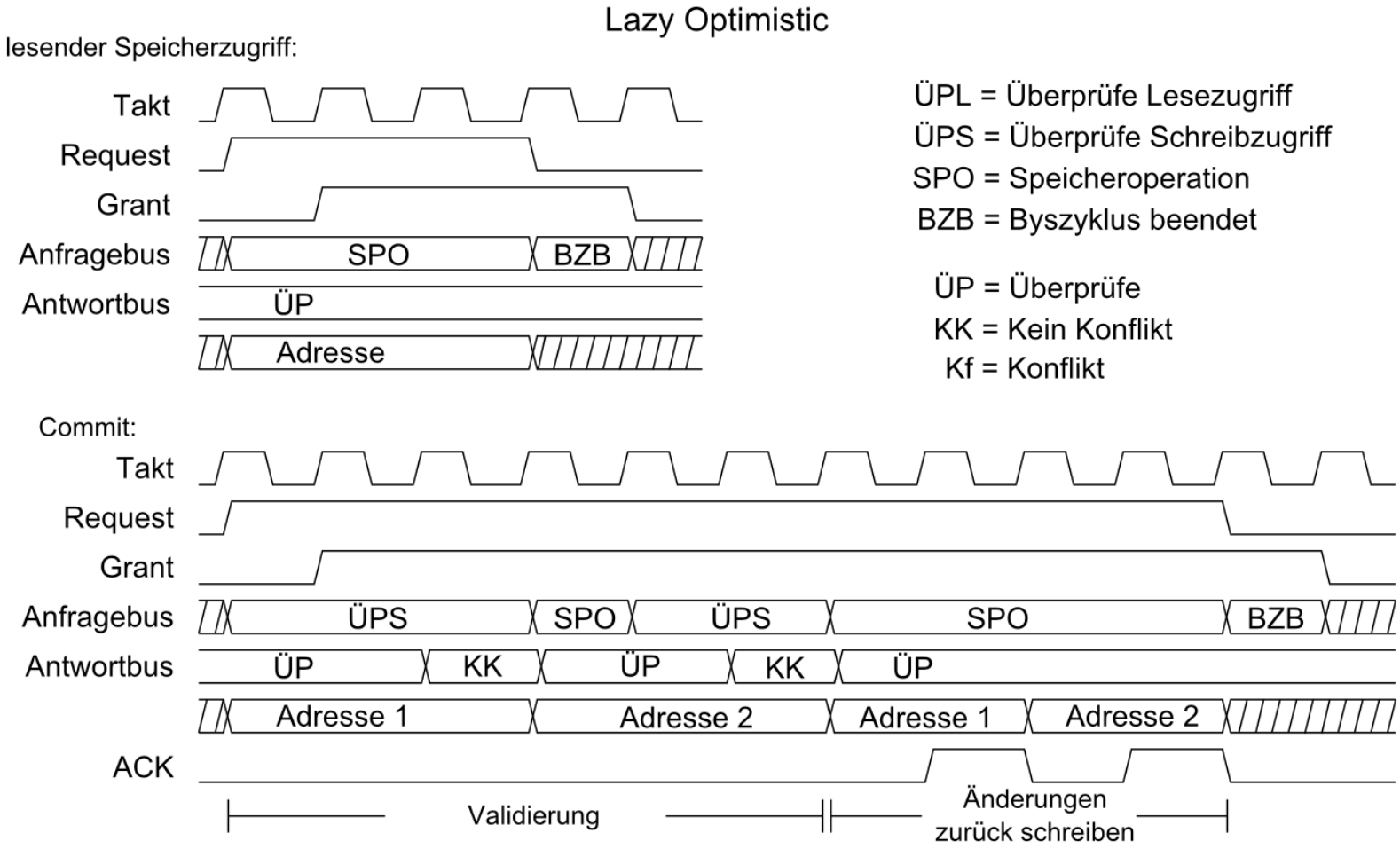
Implementierung

- Transaktionsspeicher:
  - Transaktions-Controller (LO, LP, EP), Policy und Signatur Modul austauschbar
    => flexibles Kommunikationsschema nötig
  - zwei grundlegende Aktionen:
    - Speicherzugriff (keine Aktion der anderen Transaktions-Controller)
    - Prüfung einer Adresse auf Konflikte
  - Aktionen können parallel stattfinden



Implementierung

- Transaktionsspeicher
  - Kommunikationsschema



Implementierung

- Transaktionspeicher:
  - memory mapped I/O
    - Hauptspeicher und Register zur Steuerung von Transaktionen
  - Assembler-Routinen zum Sichern/Wiederherstellen des CPU Zustandes
  - Interrup-Handler behandeln auftretende Konflikte
  - drei Makros für Programmierer
    - TM_BEGIN()
    - TM_COMMIT()
    - TM_ABORT()

Themenüberblick

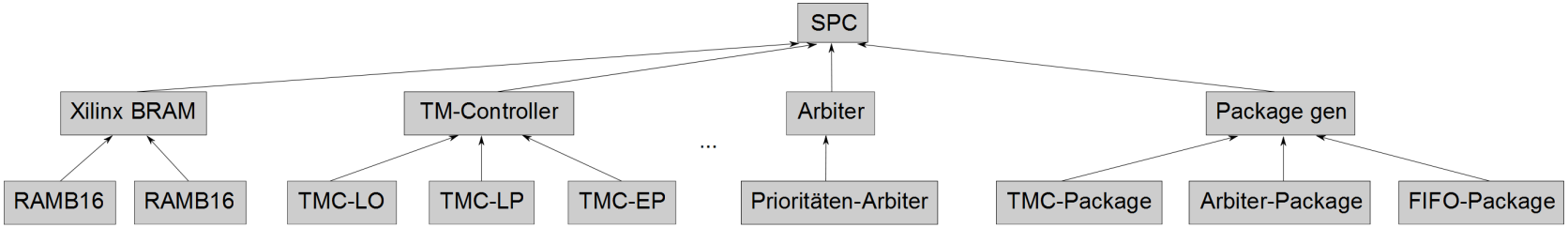
- Einführung
- Speicher Familie
- Implementierung
- **Konfigurierung**
- Evaluierung
- Fazit

Konfigurierung

- VHDL-Sprachmittel:
  - alternative Architekturen
  - Generics und generate-Anweisungen:
    - Daten-/Adressbusbreite
    - Größe des Speichers/Zwischenspeichers
    - Anzahl Anschlüsse
    - byteweiser Zugriff
    - Speicherzugriffe außerhalb von Transaktionen
- XVCL:
  - Instanziierung mit passenden Generics
  - Generierung angepasster Architekturen
    - TM-Controller: LO, LP, EP
    - Policy: attacker wins, oldest wins
    - Top-Entity

Konfigurierung

- XVCL-Hierarchie:



- SPC adaptiert alle benötigten x-frames / bereitet Variablen vor
- x-frames der 2. Ebene sorgen für einmalige Generierung
- x-frames der 3. Ebene generieren angepasste VDHL-Dateien

Themenüberblick

- Einführung
- Speicher Familie
- Implementierung
- Konfigurierung
- **Evaluierung**
- Fazit

Evaluierung

- Konfigurierbarkeit der Speicher-Familie
  - Wenige Gemeinsamkeiten der Speichermodelle
    => wenige Komponenten konnten wiederverwendet werde
    - RAM-Implementierung
    - Arbiter
  - Transaktionsspeicher bietet viele Konfigurationsmöglichkeiten
    - Policy
    - Versionierung
    - Konflikterkennung
    - ..
  - Austauschen von Komponenten im TM-System erfordert Kommunikationsschema
    => Kompromiss zwischen Flexibilität und Performance

Evaluierung

- XVCL als Konfigurationswerkzeug für VHDL
  - Nachteile:
    - höherer Entwicklungsaufwand (Quellcode wird in XVCL eingebettet)
    - Auswerten komplexer Bedingungen umständlich
    - Variablen können nicht in adaptierten x-frames verändert werden
    - XML besser für automatische Verarbeitung geeignet
    - keine explizite Repräsentation des Konfigurationsraums
  - Vorteile:
    - Generierten Dateien umfassen nur benötigten Code => bessere Wartbarkeit
    - Vermeidung von doppeltem Code
    - einfache Konfigurierung durch Variablen

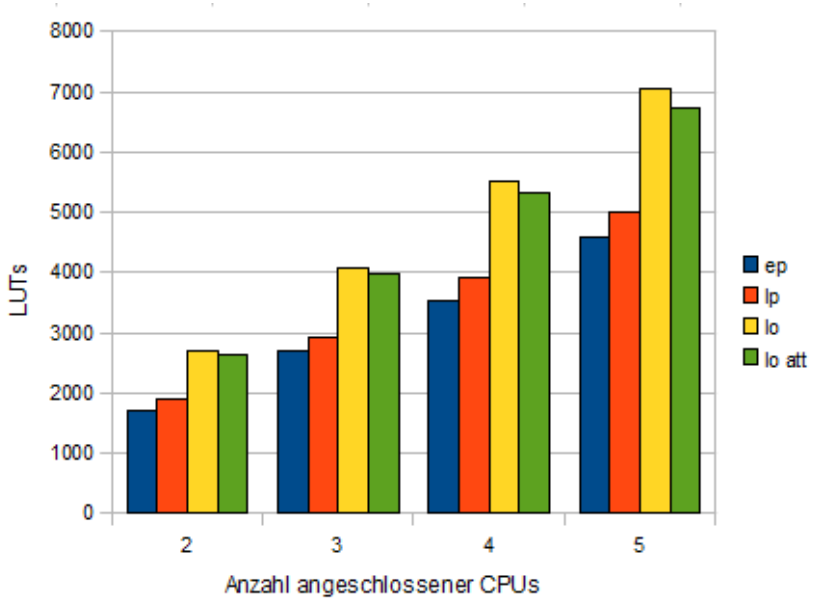
Evaluierung

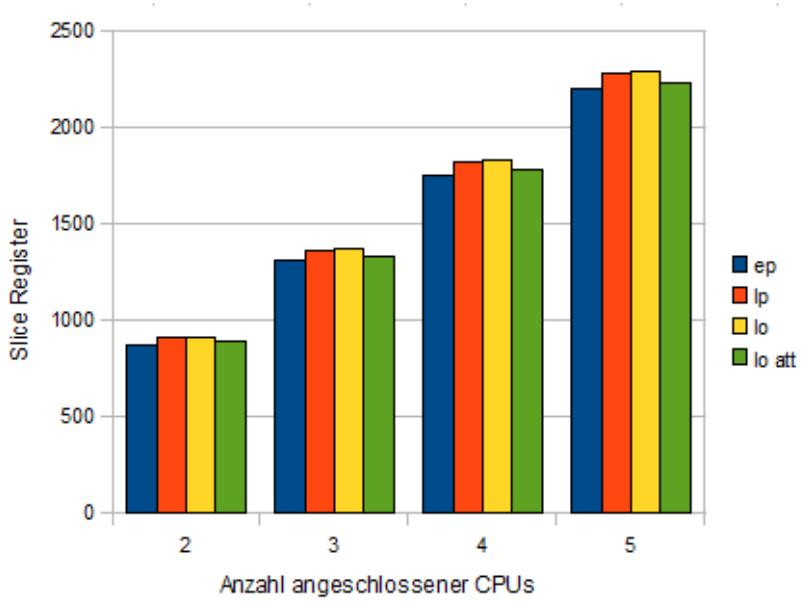
- Ressourcenverbrauch
  - FIFO-Speicher und read-modify-write Speicher-Controller

Speichergröße	Busbreite	LUTs	Register	Max. MHz
4kB	8	38	24	111
	16	38	24	111
	32	38	24	111
20kB	8	38	24	111
	16	38	24	111
	32	38	24	111
4kB	8	132	76	111
	16	132	76	111
	32	132	76	111
20kB	8	132	76	111
	16	132	76	111
	32	132	76	111

Evaluierung

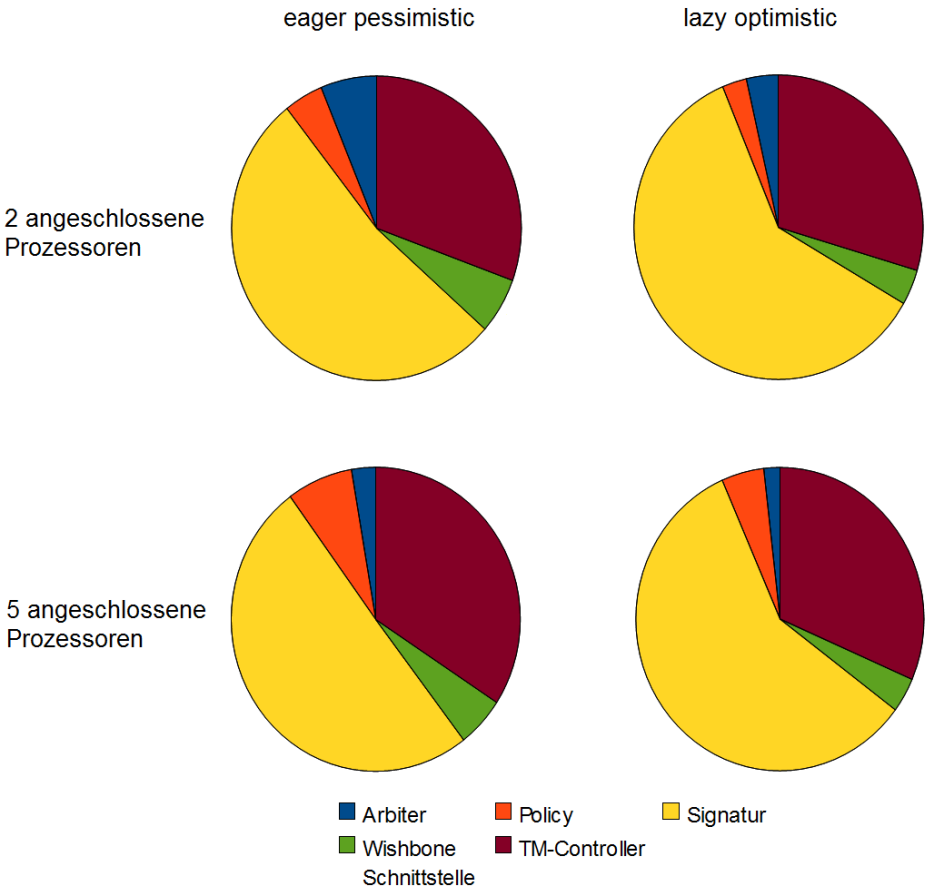
- Ressourcenverbrauch
  - Transaktionsspeicher





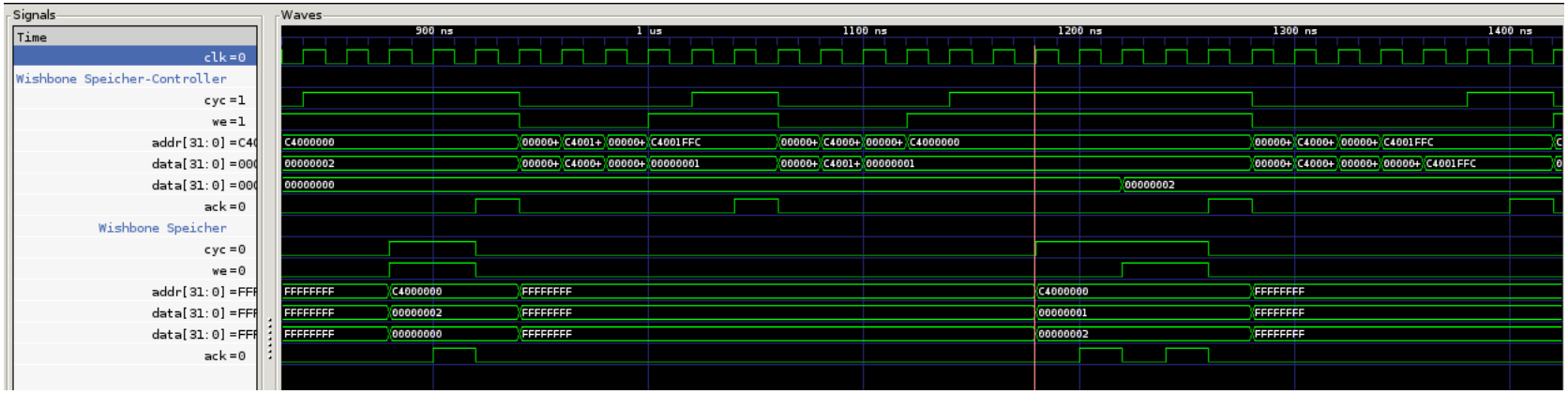
Evaluierung

- Ressourcenverbrauch
  - Komponenten im Verhältnis zum gesamten Transaktionsspeicher



Evaluierung

- Laufzeiteigenschaften
  - FIFO-Speicher:
    - 2 Takte pro Schreib-/Lesezugriff
  - Read-modify-write Speicher-Controller:
    - abhängig vom Speicher
    - 5 Takte + Latenz des Speichers für schreiben/lesen
    - 7 Takte + Latenz des Speichers für Test and Set



Evaluierung

- Laufzeiteigenschaften
  - Transaktionsspeicher

	Lesezugriff	Speicherzugriff	Commit	Revert
EP	6	9	2	Abhängig vom Write-Set
LP	Abhängig vom Write-Set	6	Abhängig vom Write-Set	2
Lo	Abhängig vom Write-Set	3	Abhängig vom Write-Set	2

Themenüberblick

- Einführung
- Speicher Familie
- Implementierung
- Konfigurierung
- Evaluierung
- **Fazit**

Fazit

- Implementiert
  - FIFO-Speicher
  - Test and Set Speicher-Controller
  - Transaktionsspeicher
    - Lazy Optimistic, Lazy Pessimistic, Eager Pessimistic
    - Policy: attacker-wins, oldest-wins
- Speichermodell übergreifend nur wenige Komponenten wiederverwendbar
- XVCL ergänzt VHDL
  - Erweitern von VHDL Komponenten
  - Automatische Generierung benötigter Komponenten
  - Aber: zum Teil umständlich umzusetzen

Fragen

Zeit für Fragen?