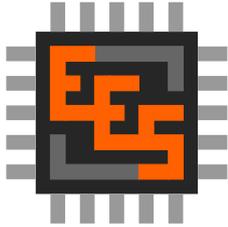




**Hochschule Augsburg**  
University of Applied Sciences

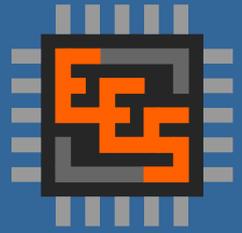


**Arbeitsgruppe**  
**Effiziente Eingebettete Systeme**

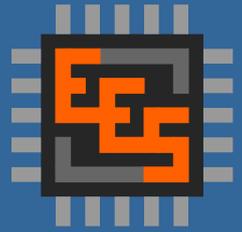
# Optimization of the ParaNut softcore processor for FPGA systems

**Alexander Bahle**



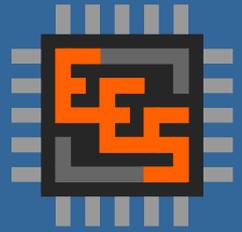


1. Einleitung
2. Status Quo & Optimierungen
3. Evaluation
4. Fazit
5. Weitere Arbeiten



# 1. Einleitung - Motivation

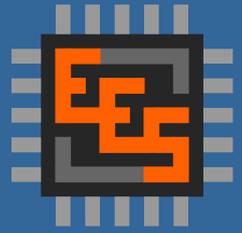
- **Trend zu anwendungsspezifischen Systemen**
    - z.B. Google TPUs, Mobileye EyQ SoC, ...
  - **Viele der Systeme brauchen general purpose Prozessoren**
    - Ausgereifte Toolchains, Software leicht anpassbar, ...
- **Konfigurierbare soft-core Prozessoren**



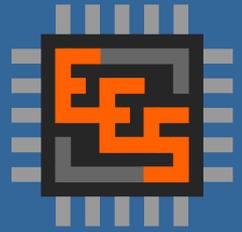
# 1. Einleitung – Ziel der Arbeit

- **Systematische Optimierung des ParaNut**
  - Taktfrequenz: 100MHz als single-core
  - Verbesserte Skalierbarkeit für multi-core Systeme
  - Ressourcenverbrauch reduzieren

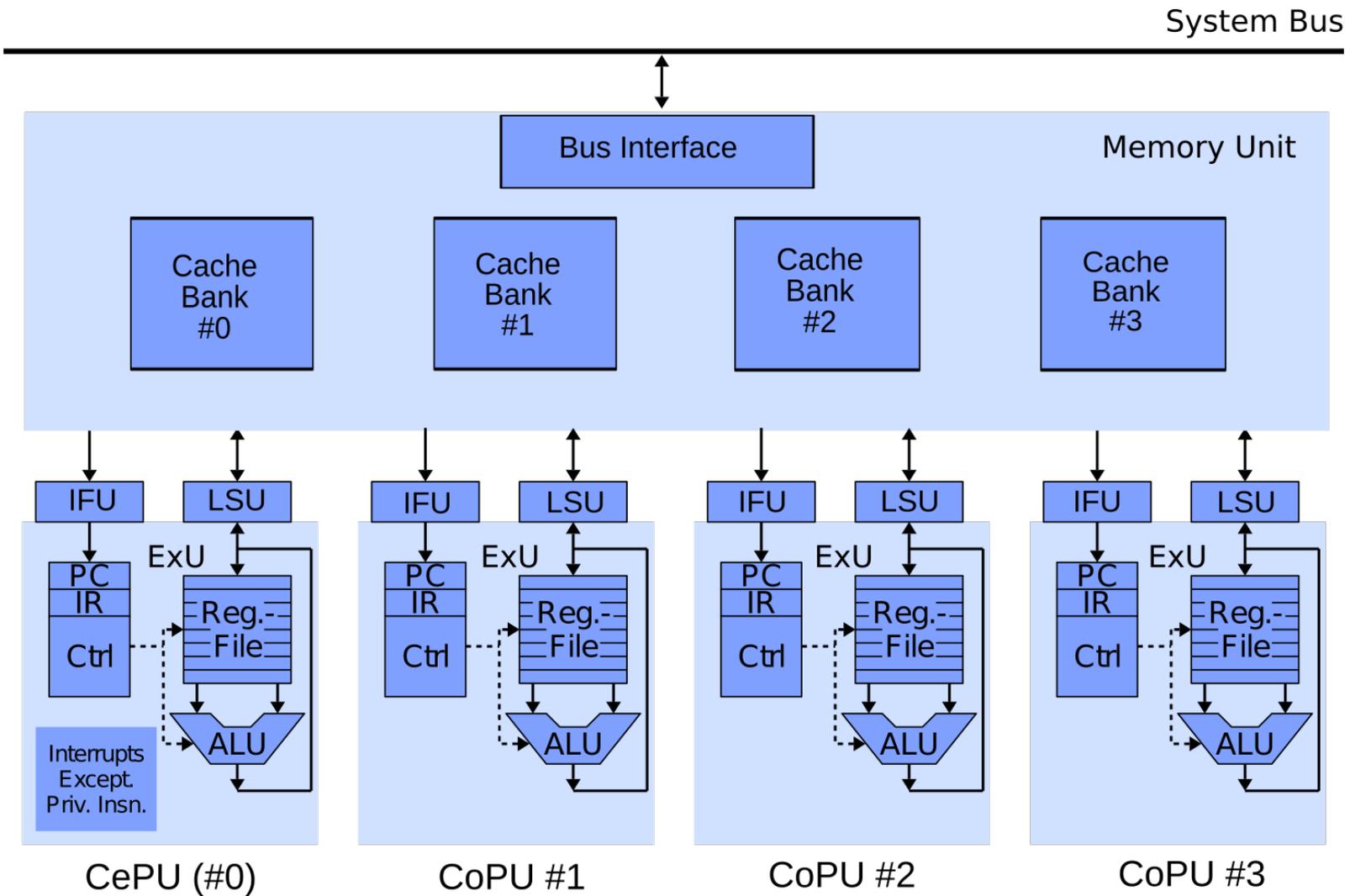
# Agenda

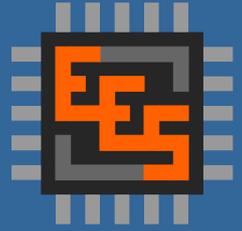


1. Einleitung
2. Status Quo & Optimierungen
3. Evaluation
4. Fazit
5. Nächste Schritte



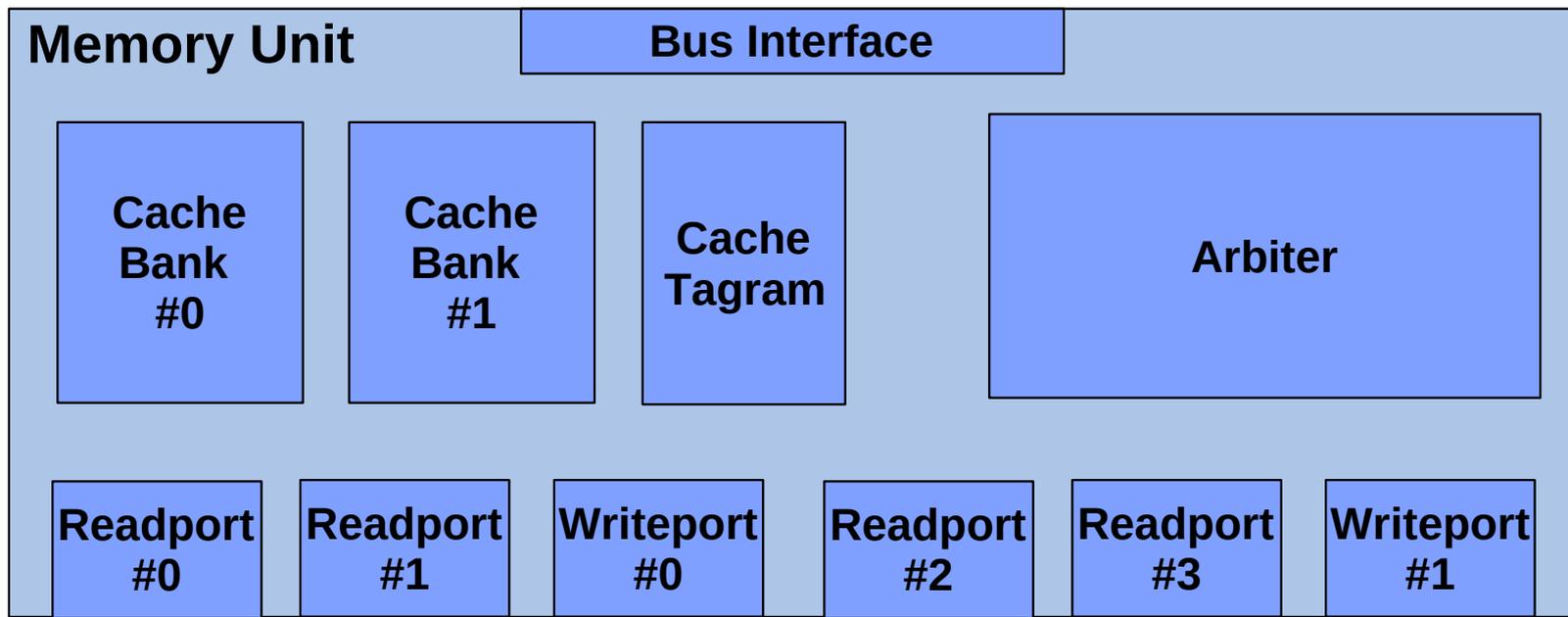
## 2. Status Quo - ParaNut

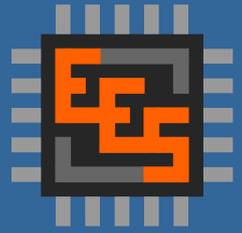




## 2. Status Quo – Memory Unit (MemU)

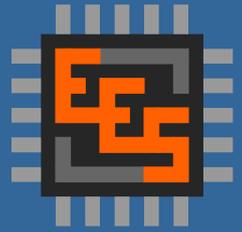
- Kritisch für Performance (Mealy-Automaten, ...)
- Benötigt ca. 30% der Ressourcen
- Beispiel für 2 Kerne & 2 Bänke:





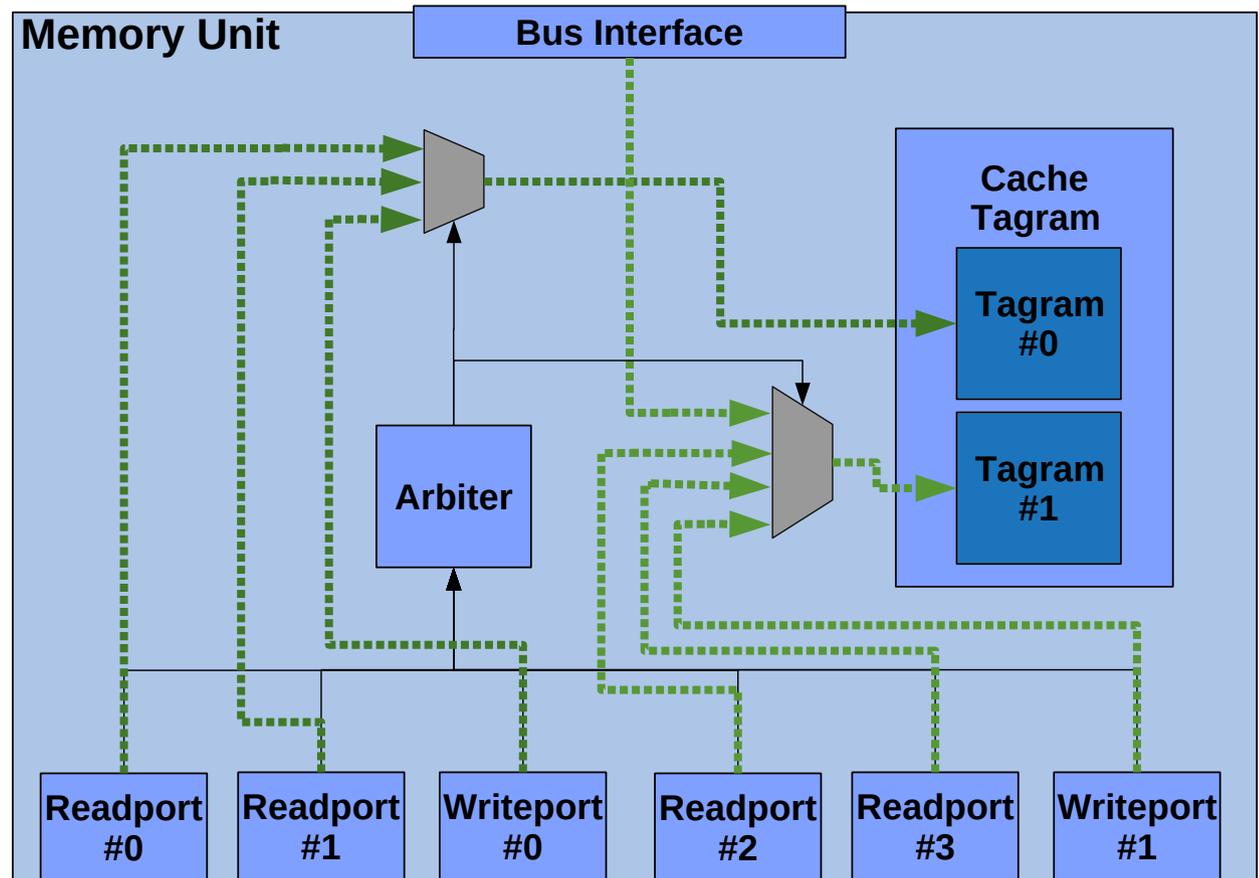
## 2. Optimierungen – MemU Allgemein

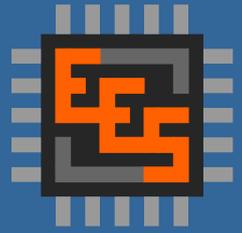
- Regeln für die Mealy-Automaten:
    - Ausgänge von IFU/LSU möglichst aus Registern
    - Eingänge dürfen weitergereicht werden (acknowledge im selben Takt)
    - Ausnahme: Requests zum Arbiter sollten nur vom Zustand abhängen
- Nicht immer umsetzbar



## 2. Status Quo – Cache Tagram

- Replizierter Tagram per CPU
- Adress- und Kontrollsignale in grün



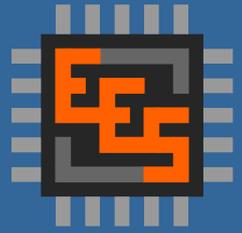


## 2. Status Quo – Cache Tagram

- Tagram Zeile bei 4-facher Assoziativität

Bits	4	4	76	(n)
Desc.	valid(0 to 3)	dirty(0 to 3)	taddr(19)(0 to 3)	(lru)

- Per Tag Eintrag
  - 1 valid Bit
  - 1 dirty Bit
  - 19 Bit Tag Adresse (abhängig von der Konfiguration)
- Es wird immer eine ganze Zeile gelesen/geschrieben
  - Schreiben dauert 2 Takte



## 2. Optimierung – Cache Tagram

- Neue Tagram Zeile bei 4-facher Assoziativität

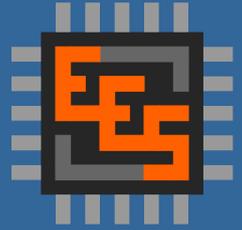
Bits	21	21	21	21
Desc.	v d taddr(19)	v d taddr(19)	v d taddr(19)	v d taddr(19)

- Asymmetrischer Block RAM

- 84 Bit breiter Leseport
- 21 Bit breiter Schreibport

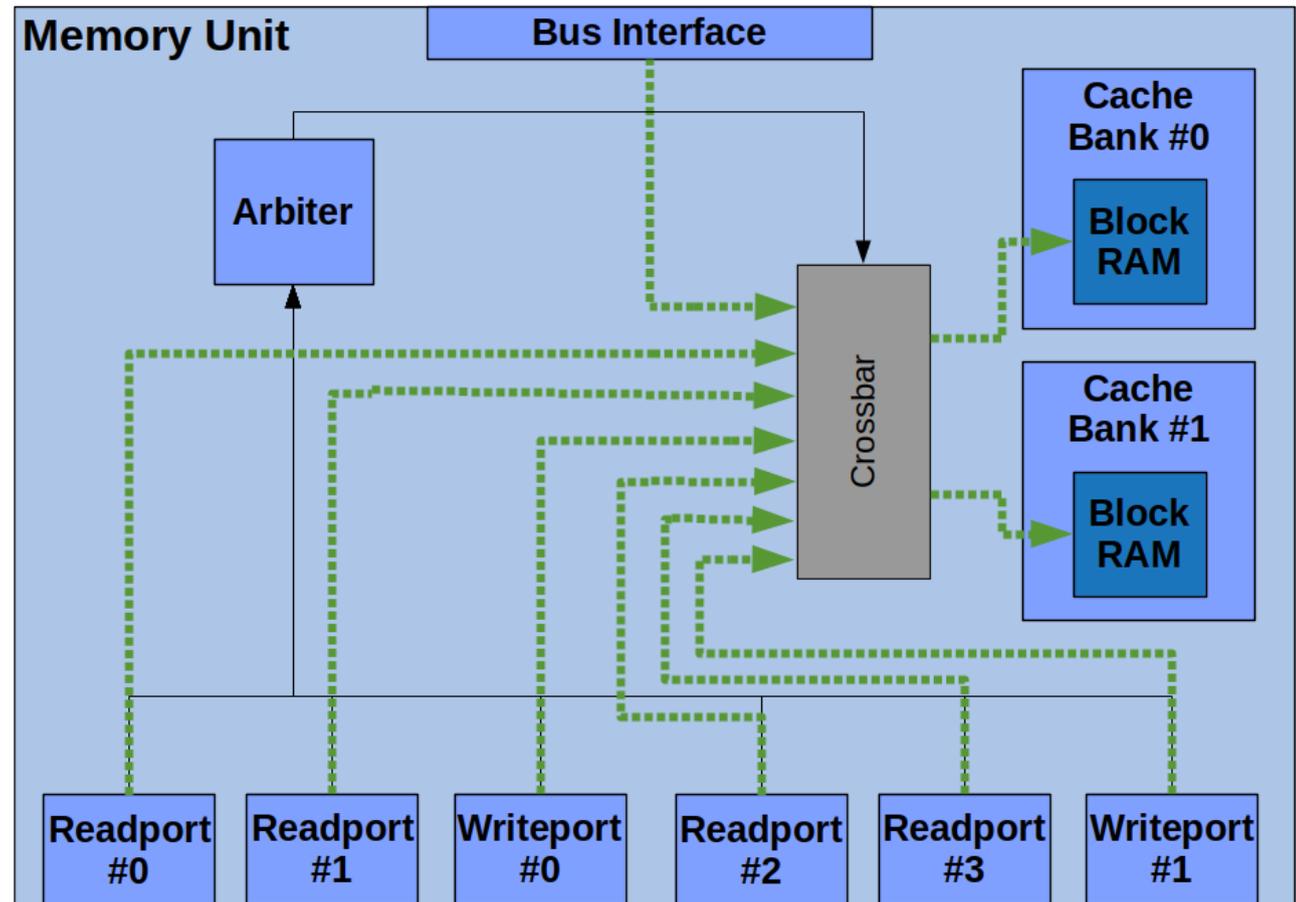
- Es wird immer eine ganze Zeile gelesen aber nur ein Eintrag geschrieben

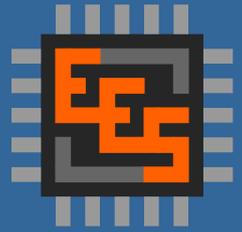
→ Schreiben dauert nur noch 1 Takt



## 2. Status Quo – Cache Bankram

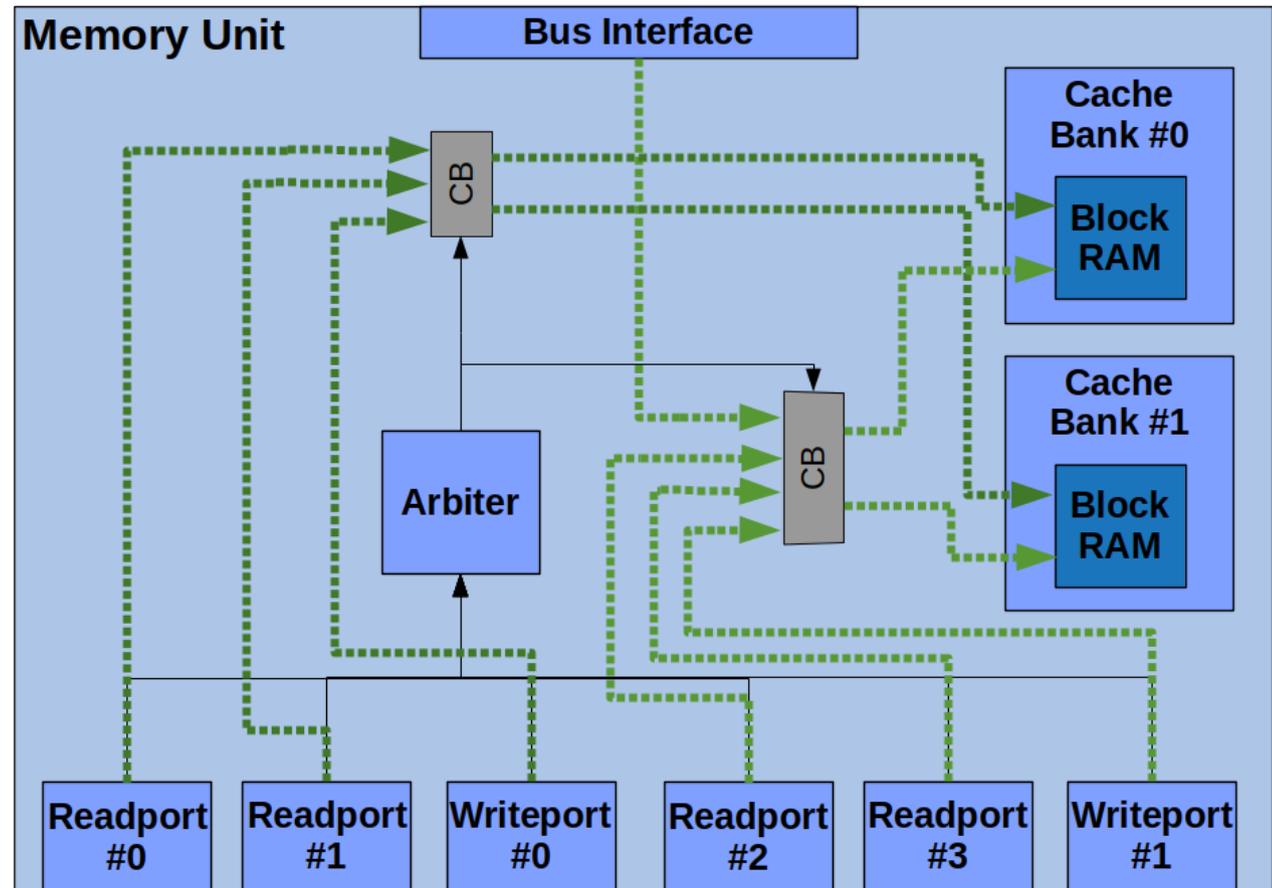
- Alle Ports und Busif sind verbunden (Crossbar)
- Adress- und Kontrollsignale in grün



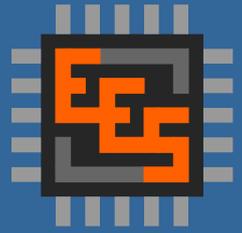


## 2. Status Quo – Cache Bankram

- Nutzung von allen verfügbaren Block RAM Ports reduziert  
Crossbar Größe
- Adress- und Kontrollsignale in grün

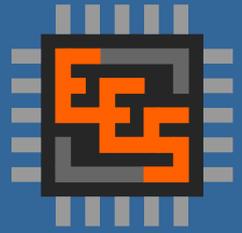


→ Crossbar Switches wachsen stark mit der Anzahl der Kerne



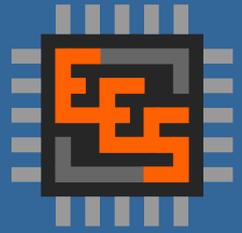
## 2. Status Quo – Cache Bankram

- **Nötige Takte zum schreiben in den Bankram:**
    - Word (4 Byte) – 1 Takt
    - Half-Word (2 Byte) – 2 Takte
    - Byte (1 Byte) – 2 Takte
  - **Ähnliche Ursache wie beim Tagram:**
    - Es wird immer ein ganzes Wort geschrieben
- **Software die mit Bytes/Half-Words arbeitet ist langsamer**



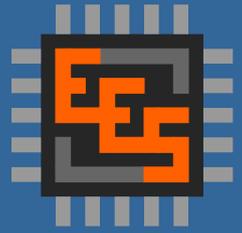
## 2. Optimierung – Cache Bankram

- **Verwenden der Byte Enable Eingänge der Block RAMs**
    - Information ist in den Writeports schon vorhanden
    - Ermöglicht Vereinfachung des WP Automaten
- Lesen und Schreiben dauert 1 Takt, egal ob Word, Half-Word oder Byte**



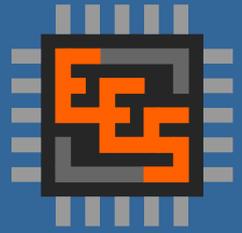
## 2. Status Quo – Arbiter

- **Arbiter ist ein Schaltnetz**
  - + schnelle Reaktion
  - Abhängigkeiten – vor allem bei multi-core Systemen
- **Gegliedert nach zu arbitrierender Ressource:**
  - Bus Interface
  - Cache Linelock
  - Cache Tagram
  - Cache Bankram
- **Bestimmung von Priorität konfigurierbar**
  - Round-Robin – alle x Takte
  - Pseudo-Zufällig



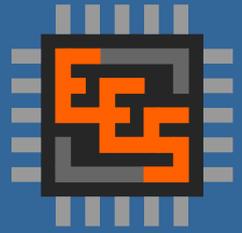
## 2. Status Quo – Arbiter - Buslf

- **Eingänge:**
  - Alle Readports
  - Alle Writeports
- **Pro EXU fixe Priorisierung:**
  - Daten Readport
  - Befehls Readport
  - Daten Writeport
- **Globale Priorisierung über konfiguriertes Verfahren**



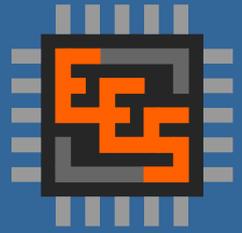
## 2. Status Quo – Arbiter – Cache Linelock

- **Eingänge:**
  - Bus Interface
  - Alle Writeports
- **Writeport Linelocks sind exklusiv**
- **Buslf Linelock und 1 Writeport Linelock ist möglich**
  - Adressen müssen unterschiedlich sein
- **Globale Priorisierung über konfiguriertes Verfahren**



## 2. Status Quo – Arbiter – Cache Bankram

- **Eingänge:**
  - Bus Interface
  - Alle Readports
  - Alle Writeports
- **Pro Bankram und Port ein Reader oder Writer**
- **Per EXU fixe Priorität (Daten Read, Befehl Read, Daten Write)**
- **Globale Priorisierung über konfiguriertes Verfahren**

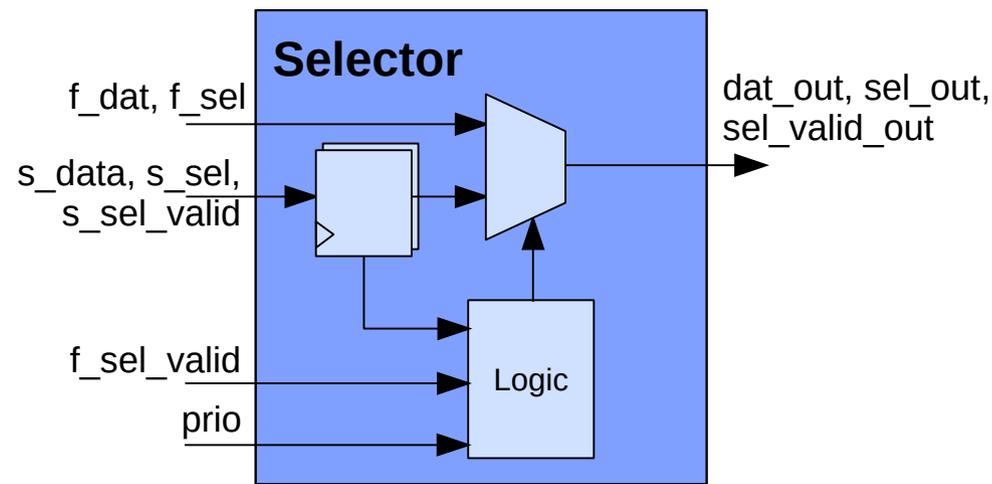


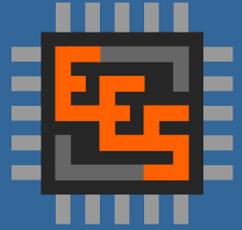
## 2. Optimierung – Arbitrer

→ Ziel: Abhängigkeiten auflösen

- Definition eines Selector Moduls:

- Schnelle  $f\_*$  Eingänge → Antwort im selben Takt möglich
- Langsame  $s\_*$  Eingänge durch Register → Mindestens 1 Takt Verzögerung

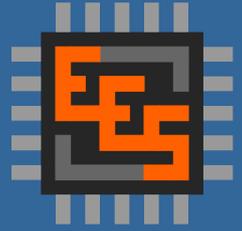




## 2. Optimierung – Arbitrer

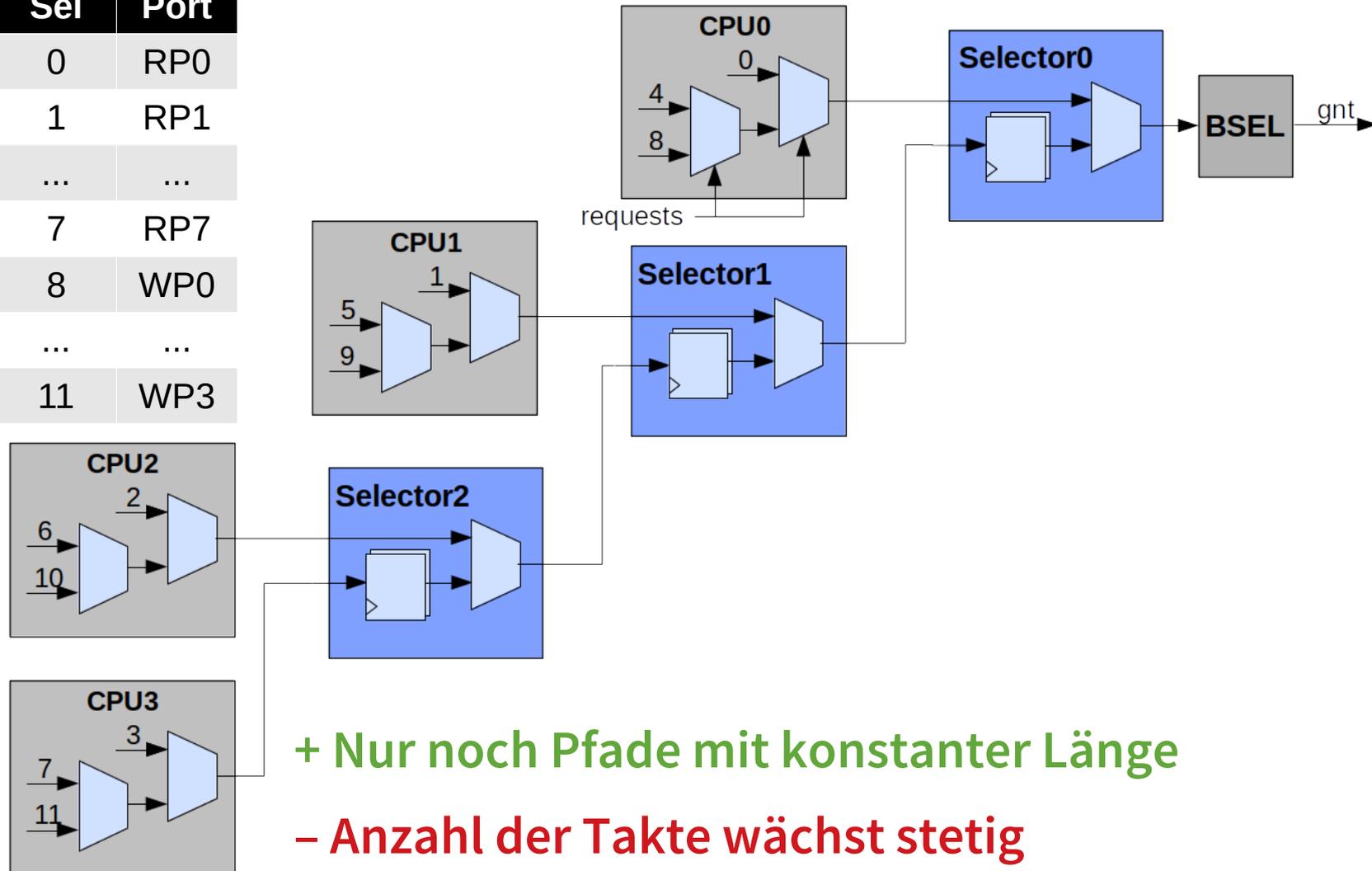
- Eingänge/Ausgänge:

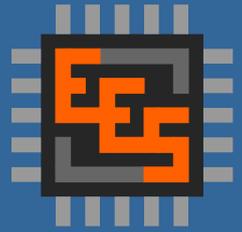
Name	Beschreibung
*_sel sel_out	Index (bspw. CPU#) der Ressource, die ein grant erhalten soll.
*_sel_valid sel_valid_out	Gibt an ob der Index und die Daten valide sind (entspricht dem request).
*_dat dat_out	Optionale Möglichkeit Daten (bspw. Adressen, Befehle) direkt mit zu Arbitrieren/Auszuwählen.
prio	Wert um zu bestimmen, ob der schnelle Eingang oder der langsame Eingang Priorität hat.



## 2. Optimierung – Arbitrer - Buslf

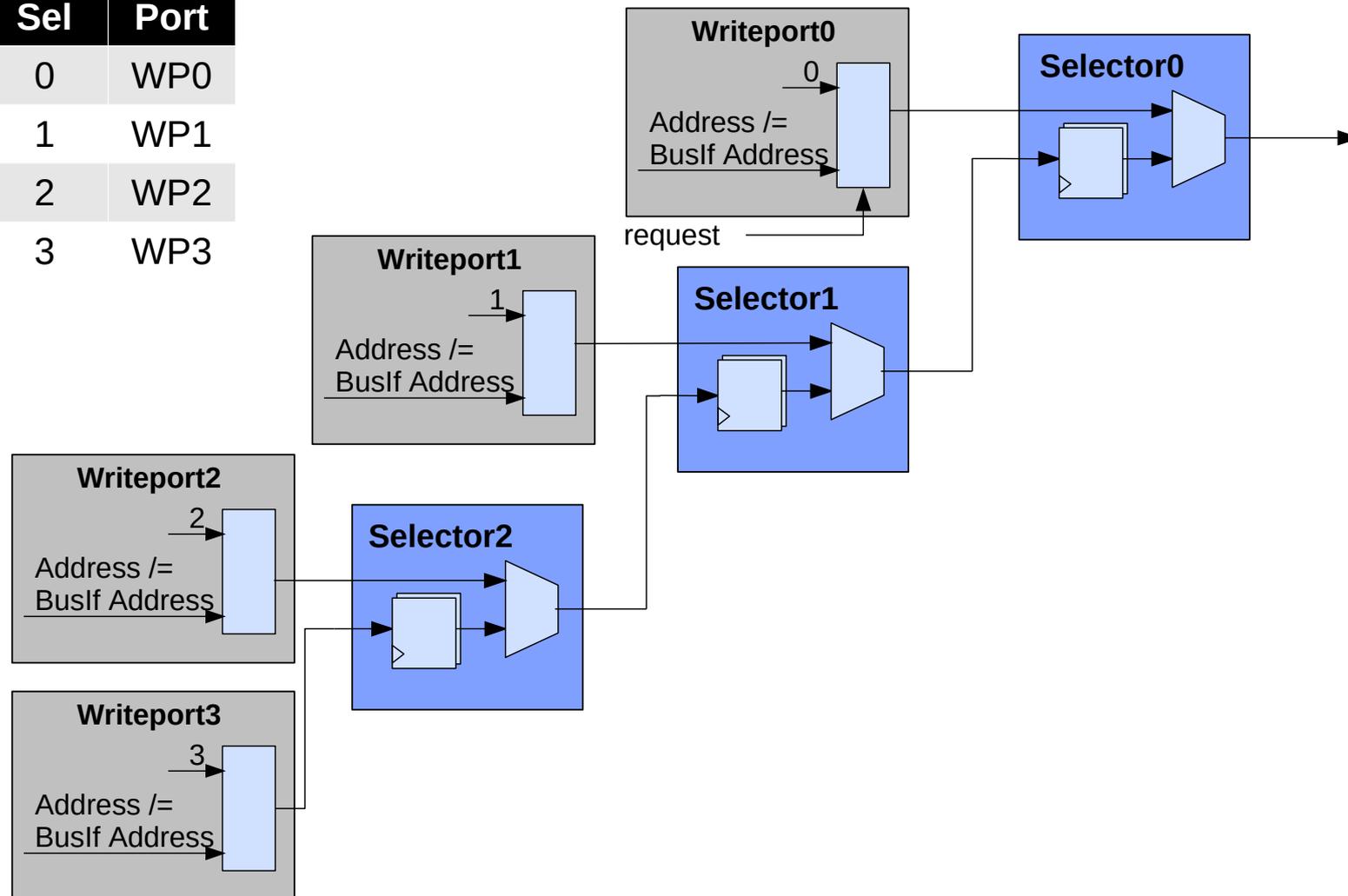
Sel	Port
0	RP0
1	RP1
...	...
7	RP7
8	WP0
...	...
11	WP3

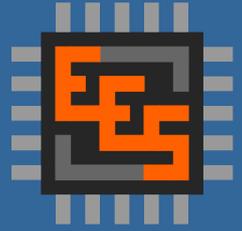




## 2. Optimierung – Arbiter - Linelock

Sel	Port
0	WP0
1	WP1
2	WP2
3	WP3

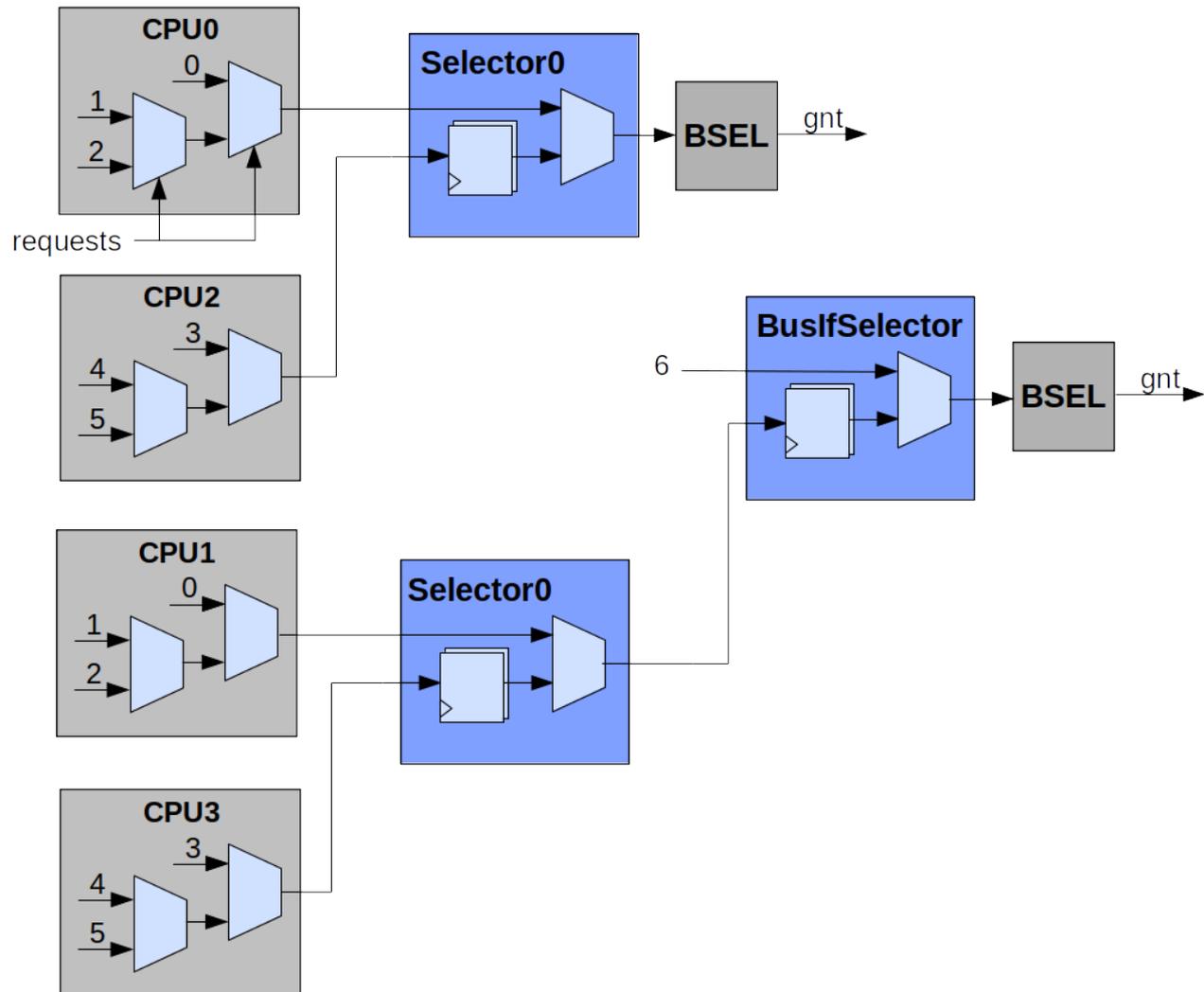


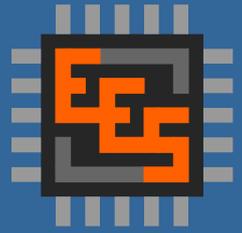


# 2. Optimierung- Arbiter – Cache Bankram

Sel	Port
0	RP0
1	RP4
2	WP1

Sel	Port
0	RP1
1	RP5
2	WP1
3	RP3
4	RP7
5	WP3
6	BusIf

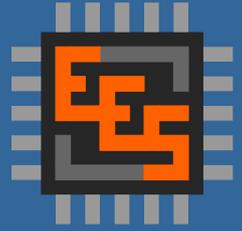




## 2. Optimierung- Arbiter – Probleme

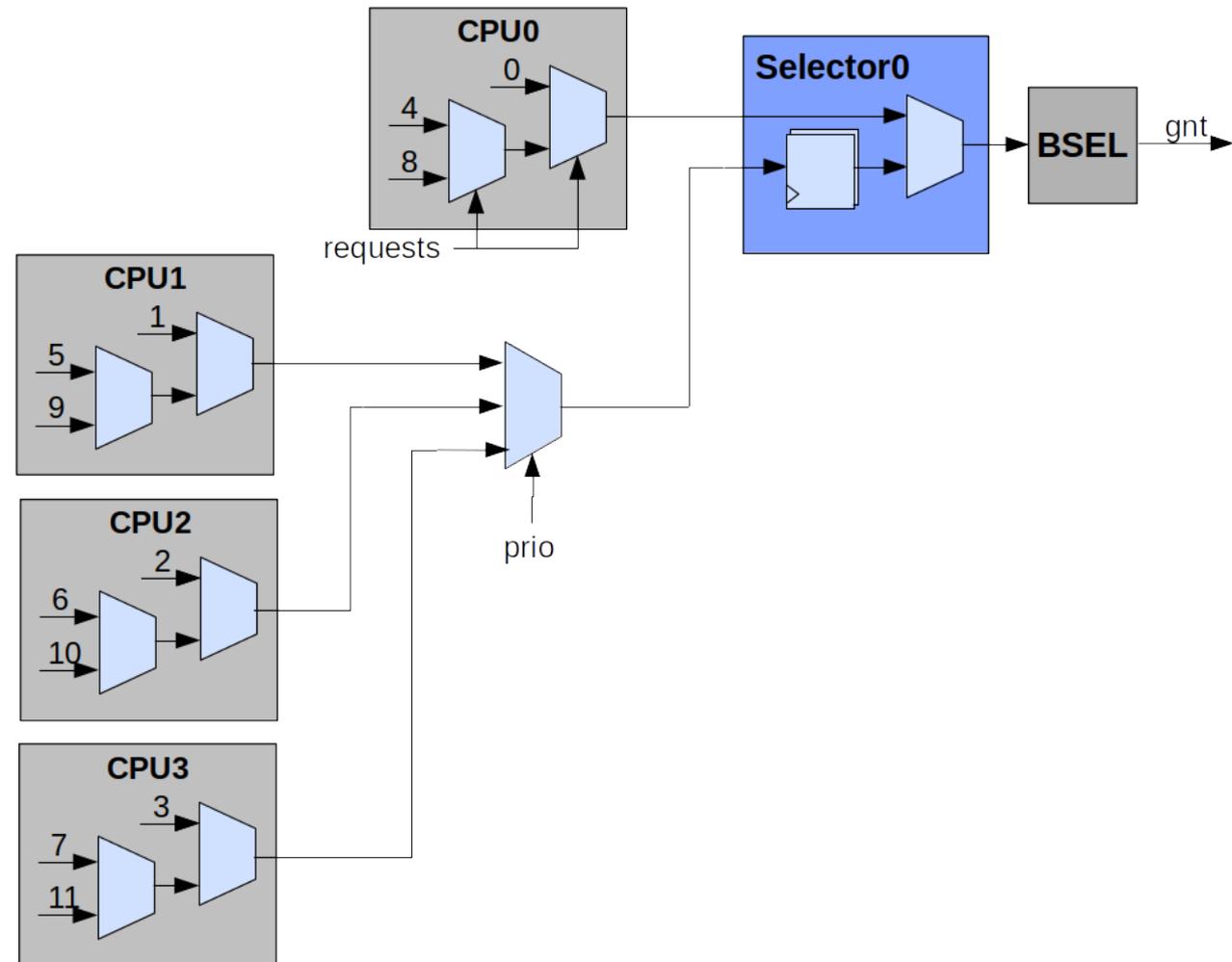
- Probleme bei 8 Kernen
- Bei der Buslf und Linelock Arbitrierung entstehen lange Verzögerungen (bis zu 7 Takte)
- Race Condition im Falle, dass:
  - request nach einem grant zurückgenommen wird
  - in dieser Zeit wieder ein request gesetzt wird

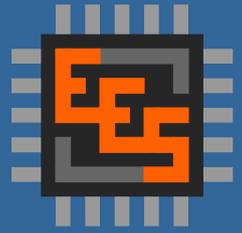
→ Außerhalb der Masterarbeit gelöst



## 2. Optimierung- Arbiter – Probleme

Sel	Port
0	RP0
1	RP1
...	...
7	RP7
8	WP0
...	...
11	WP3

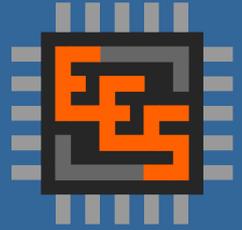




## 2. Optimierung- Arbiter – Probleme

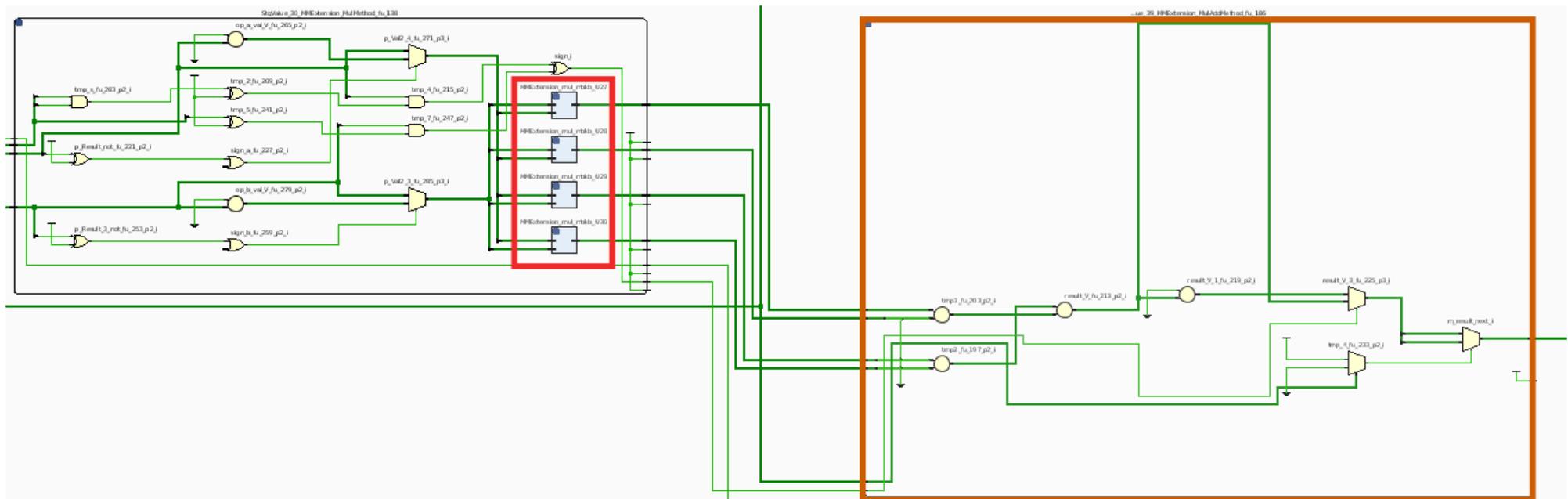
- Auswirkungen:

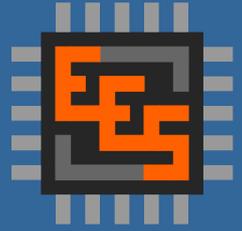
Cores	MHz	Ratio to single core	Ratio to before
1	100	1.00	2.00
2	85	0.85	2.13
4	75	0.75	2.27
8	40	0.40	2.00



## 2. Status Quo – EXU

- **Multiplication aus der M-Extension:**
  - Rot: Vier 16x16 Multiplikationen
  - Orange: Addieren und Auswählen

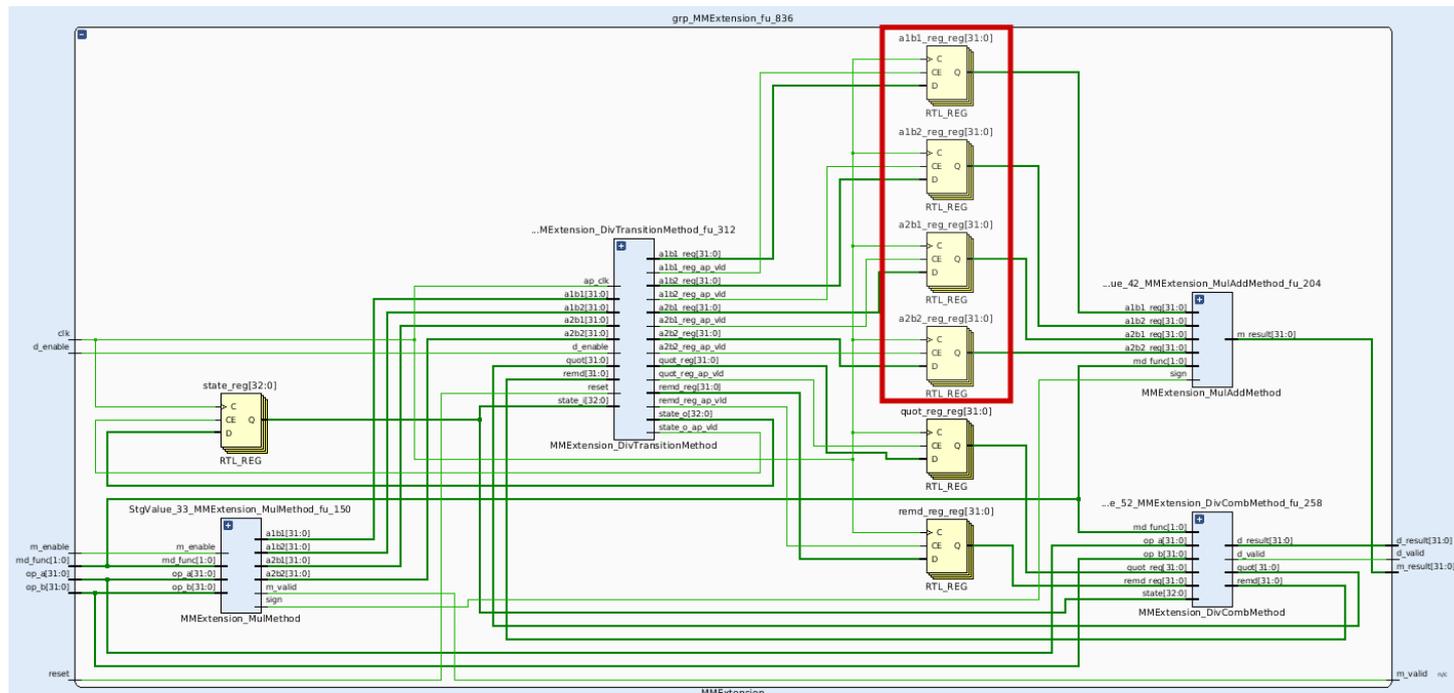


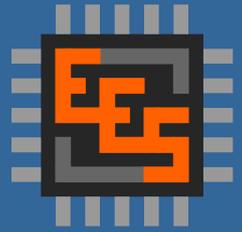


## 2. Optimierung – EXU

### • Multiplikation aus der M-Extension:

- Automat der EXU ist noch nicht optimal (3 Takte)
- Retiming durch Register nach der Multiplikation

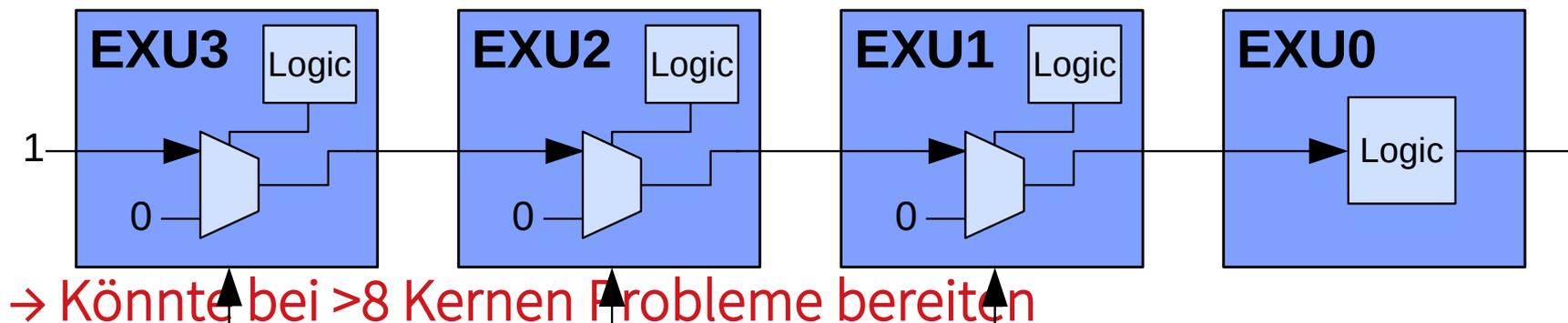


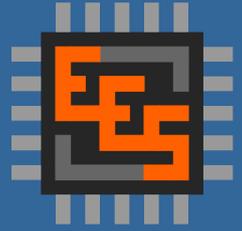


## 2. Status Quo – EXU

- **Linked Mode Synchronisierung:**

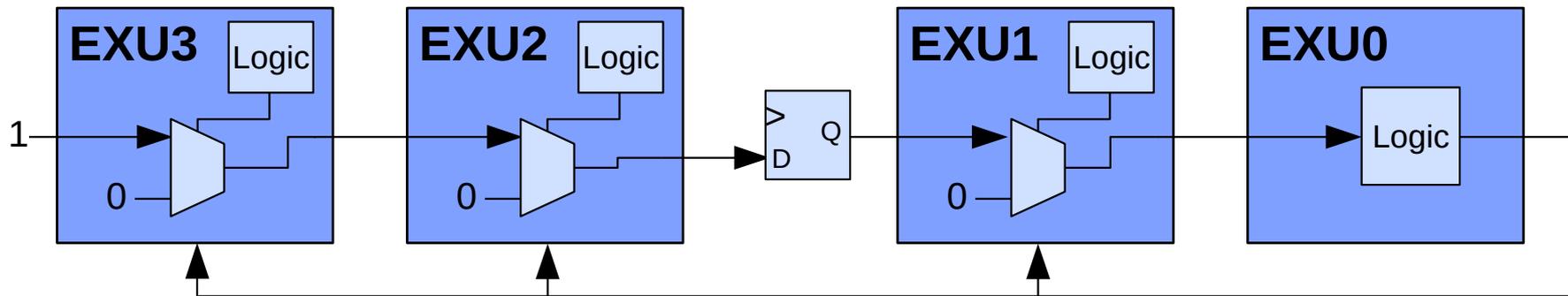
- Daisy Chain: Entweder der Eingangswert oder 0 wird weitergegeben
- Wert von EXU0 aktiviert die anderen EXUs wieder

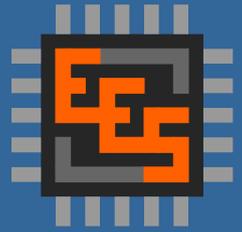




## 2. Optimierung – EXU

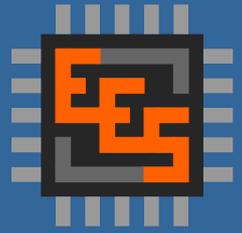
- **Linked Mode Synchronisierung:**
  - Daisy Chain nach x Kernen unterbrechen um konstante Verzögerung zu erhalten





## 2. Status Quo – LSU

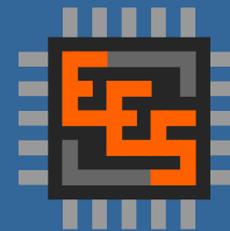
- **Wichtig für die Performance:**
  - Jeder Lese- oder Schreibzugriff geht durch die LSU
  - Kritisch: „write buffer hit detection“ bei Lesezugriff
  - Atomare Befehle benötigen extra Multiplexer an den Ausgängen zum Writeport
  - Verbesserungswürdige HLS Ergebnisse



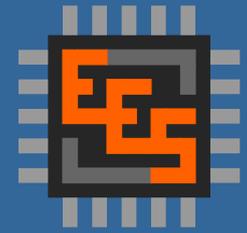
## 2. Optimierung – LSU

- **Refactor der LSU:**
  - Alle Ausgänge zum Writeport nur noch aus Registern
  - Auch Cache-Befehle (Flush, ...) gehen in den Write Buffer
  - Register für atomare Befehle auf den Readport umverteilt
  - HLS Ergebnisse verbessert (weniger Redundanz)

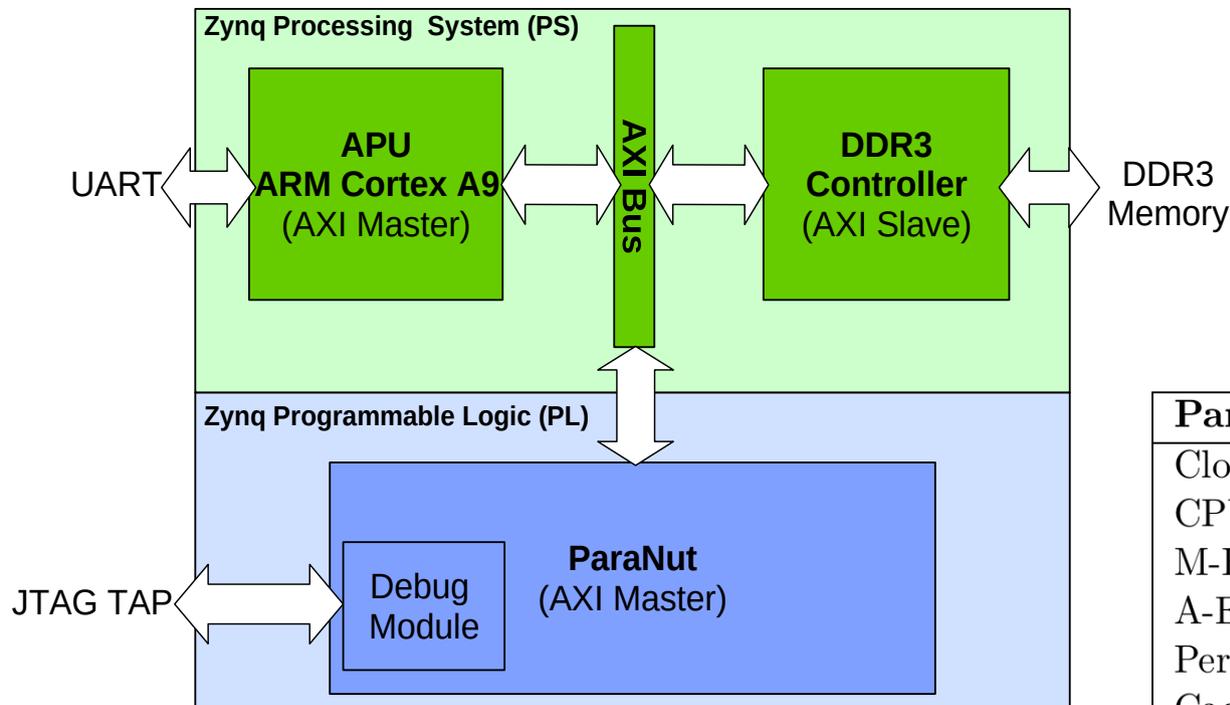
→ Kritische „write buffer hit detection“ noch da



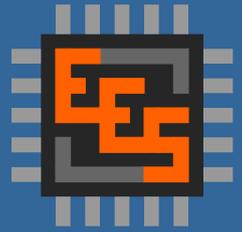
1. Einleitung
2. Status Quo & Optimierungen
3. Evaluation
4. Fazit
5. Weitere Arbeiten



# 3. Evaluation - System

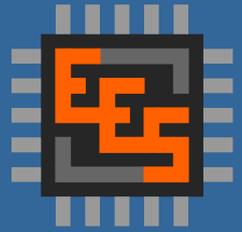


Parameter	Value
Clock Speed	20 ... 100MHz
CPU Cores	1 ... 8
M-Extension	✓
A-Extension (lr.w and sc.w)	✓
Performance Counter Enable	1
Cache size	32 kB
Cache sets	512
Cache line size	16 Bytes (4 Banks)
Cache associativity	4 ways
Cache replacement strategy	LRU
Instruction buffer size (IFU)	4 words
Write buffer size (LSU)	4 words
MEMU arbitration	7 (256 cycles)



### 3. Evaluation - Übersicht

Maßnahme	Auswirkung
Regeln für die Mealy-Automaten	0 Takte im Idealfall
Tagram Write in 1 Takt	-1 Takt
Bankram Write in 1 Takt	-1 Takt bei Byte/HW Writes
Arbiter Selector Module	0 bis n-1 Takte
Retiming der Multiplikation	(1 Takt)
Refactor der LSU	0 Takte



### 3. Evaluation - Ressourcen

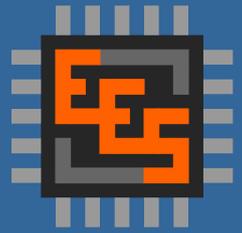
- Ressourcenverbrauch zur Embedded World 2020:

Cores	Slice LUTs	Slice FFs	Slices	Increase
1	7,139	3,759	2,504	1.00
2	12,433	6,094	4,185	1.67
4	23,599	10,712	7,265	2.90
8	44,393	19,536	12,473	4.98
Freedom E310 (1 tile)	6,713	4,131	2,139	

- Ressourcenverbrauch jetzt:

Cores	Slice LUTs	Slice FFs	Slices	Increase	Ratio to 6.2
1	5,463	3,649	1,984	1.00	0.79
2	9,626	5,905	3,506	1.77	0.84
4	16,900	10,462	5,799	2.92	0.80
8	31,381	19,113	9,147	4.61	0.73

→ Im Ø 21% weniger Ressourcen

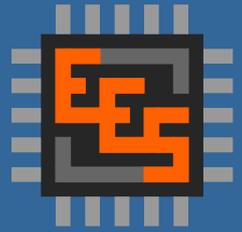


### 3. Evaluation - Timing/Takt

- Mögliche Taktfrequenzen (EW2020, jetzt):

	Cores	MHz	Ratio to single core	Ratio to before
Before	1	50	1.00	
	2	40	0.80	
	4	33	0.66	
	8	20	0.40	
After	1	100	1.00	2.00
	2	90	0.90	2.25
	4	80	0.80	2.42
	8	45	0.45	2.25

→ Taktfrequenz mindestens verdoppelt



## 3. Evaluation - Benchmarks

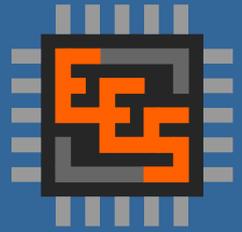
- CoreMark Benchmark EW2020:

Processor	Cores	CoreMark/MHz	Speedup
ParaNut	1	0.87	1.00
	2	1.73	2.00
	4	3.45	3.97
	8	6.59	7.59
MicroBlaze [EEMBC 2019]	1	1.90	
HiFive Unleashed [EEMBC 2019]	1	2.01	

- CoreMark Benchmark jetzt:

Processor	Cores	CoreMark/MHz	Speedup	Ratio to <b>6.5</b>
ParaNut	1	0.85	1.00	0.98
	2	1.35	1.58	0.92
	4	2.21	2.60	0.75
	8	-	-	-

→ 2% bis 25% Verlust

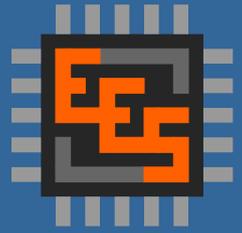


### 3. Evaluation - Benchmarks

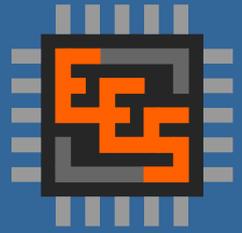
- Dhrystone Benchmark (EW2020, jetzt):

	<b>Dhrystones/MHz</b>	<b>Ratio</b>	<b>Dhrystones</b>	<b>Ratio</b>
Before (50MHz)	909.09	1.00	45,454.5	1.00
After (100MHz)	909,09	1.00	90,909.1	2.00

→ Kein Verlust feststellbar bei doppelter Taktfrequenz



1. Einleitung
2. Status Quo & Optimierungen
3. Evaluation
4. Fazit
5. Weitere Arbeiten



## 4. Fazit

- Ziel der 100MHz Taktfrequenz erreicht
- Skalierbarkeit verbessert → Taktfrequenz mindestens verdoppelt
- Ca. 20% weniger Ressourcen
- Vorbereitungen für Änderungen in zukünftigen Arbeiten

# Vielen Dank

**Alexander Bahle**

**`alexander.bahle@hs-augsburg.de`**

**Gundolf Kiefer**

**`gundolf.kiefer@hs-augsburg.de`**