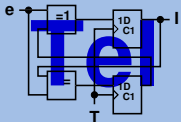


# Konzeption und Realisierung von Test- und Debugtechniken zur Prototypevaluation der grobgranular-rekonfigurierbaren **ARRIVE-Architektur** Diplomverteidigung

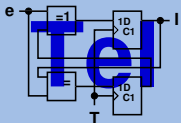
Martin Zimmerling  
mz793134@inf.tu-dresden.de

Institut für Technische Informatik

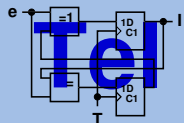


Martin Zimmerling  
Institut für Technische Informatik  
mz793134@inf.tu-dresden.de

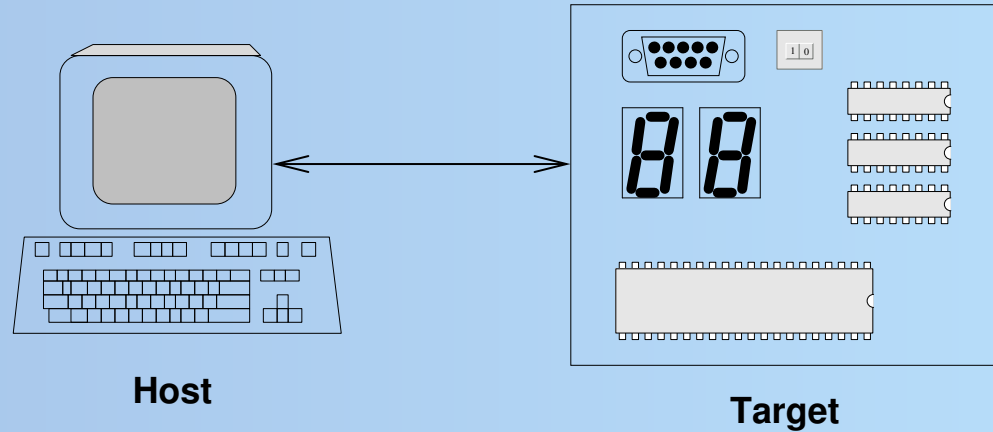
- ◆ Aufgabenstellung
- ◆ Debugging und Prototyping
- ◆ ARRIVE-Architektur
- ◆ Konzept und Realisierung
- ◆ Ergebnisse



- ◆ Implementationsmöglichkeiten aufzeigen zur Inbetriebnahme, Test und Debugging anhand des VHDL-Modells der ARRIVE-Architektur
- ◆ Prototyp-Realisierung auf Basis eines FPGAs
- ◆ Visualisierung mittels einer Test- und Debug-Umgebung
- ◆ Evaluation bzgl. benötigter HW-Ressourcen und Taktfrequenz
- ◆ Vergleich zu Standardzellen-ASIC-Synthese



# Debugging I (Ziele)



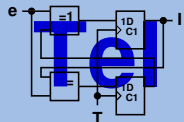
◆ Steuerung

◆ Beobachten

⇒ Finden von Fehlern

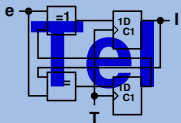
# Debugging II (Anforderungen)

- ◆ Kontrolle des Steuerflusses
  - Starten, Stoppen
  - Singlestepping
  - Befehlszähler setzen
- ◆ Kontrolle des internen Zustands
  - Schreiben/Lesen der Register
- ◆ Rekonfigurierbarer Prozessor  $\Rightarrow$  Konfigurationstabellen
  - Schreiben der Tabellen
  - Lesen der Tabellen (teuer und unnötig!)
- ◆ Prototyp: System-on-Chip  $\Rightarrow$  Speicher
  - Schreiben/Lesen des Speichers



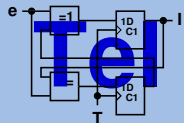
# Anforderungen an das Prototyping

- ◆ Flexibilität
- ◆ Skalierbarkeit
- ◆ Beobachtbarkeit

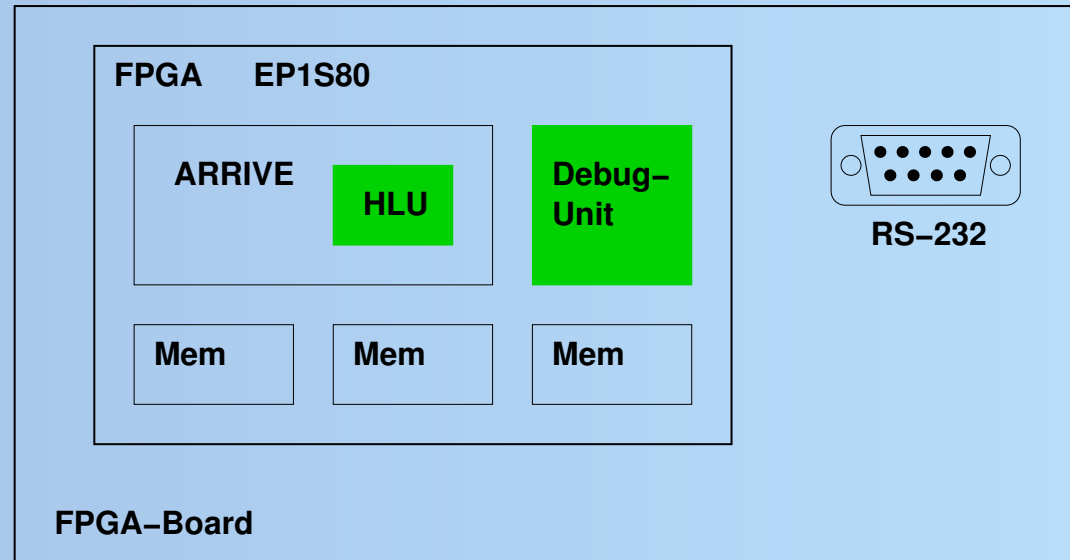


# ARRIVE: Rekonfigurierbare Architekturen

- ◆ grobgranular-rekonfigurierbare Architekturen vs. FPGA, ASIC, Standardprozessor
- ◆ Rekonfiguration in jedem Takt durch Befehlszähler
- ◆ vereint Vorteile der anderen Ansätze:
  - schnelle Implementierungszeit
  - gute Ausnutzung der HW-Ressourcen
  - schnelle und flexible Anpassungsfähigkeit
  - hohe Geschwindigkeit bei geringem Leistungsverbrauch



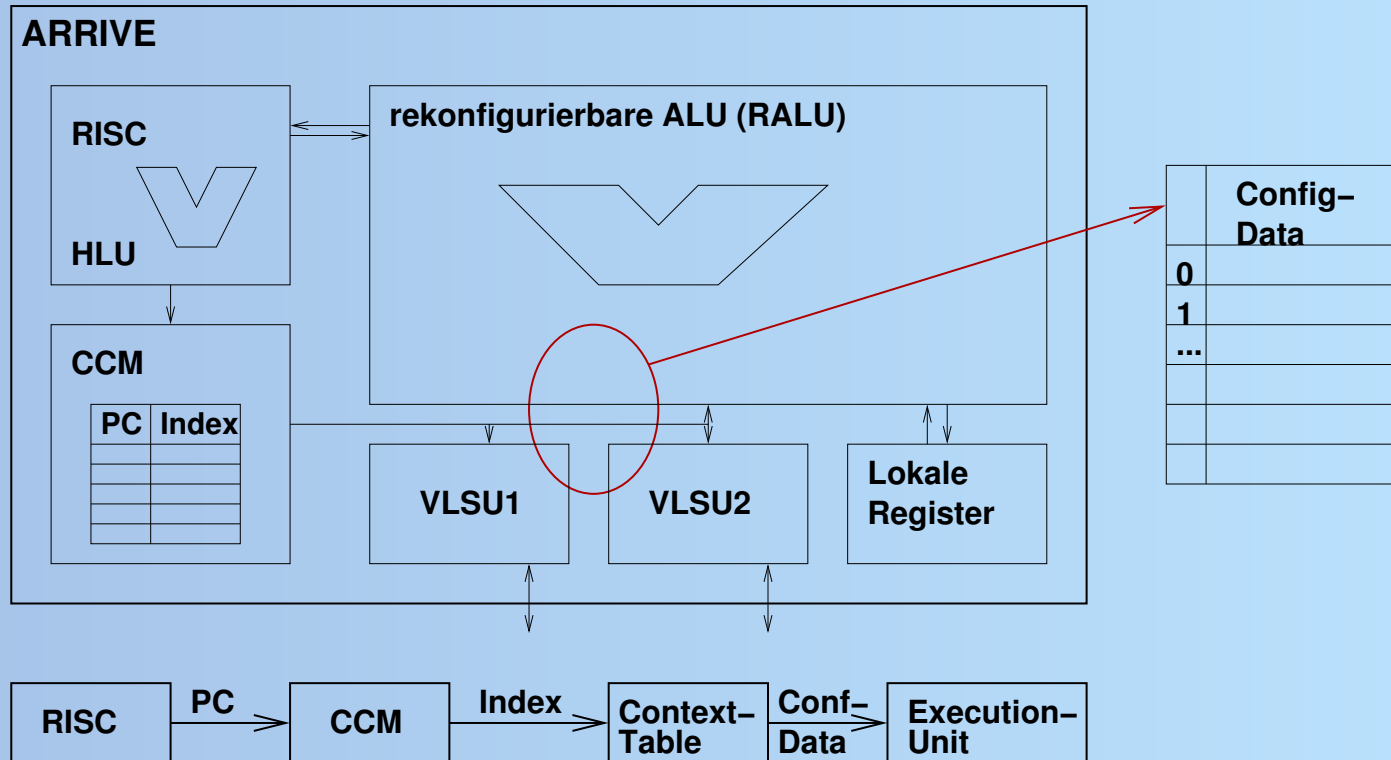
# ARRIVE: FPGA-Board

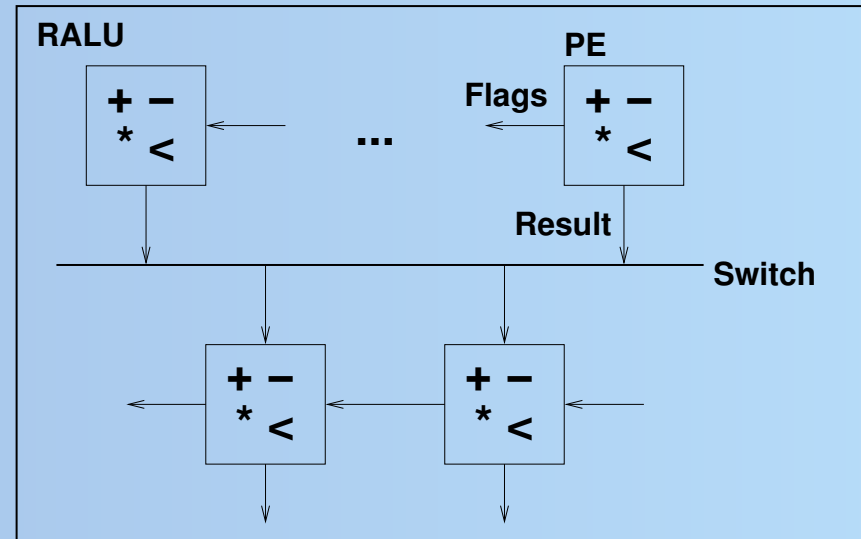


- ◆ RS-232 serielle Schnittstelle
- ◆ Vorteil: Standard, Betriebssystemunterstützung



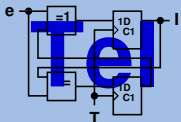
# ARRIVE: Architektur



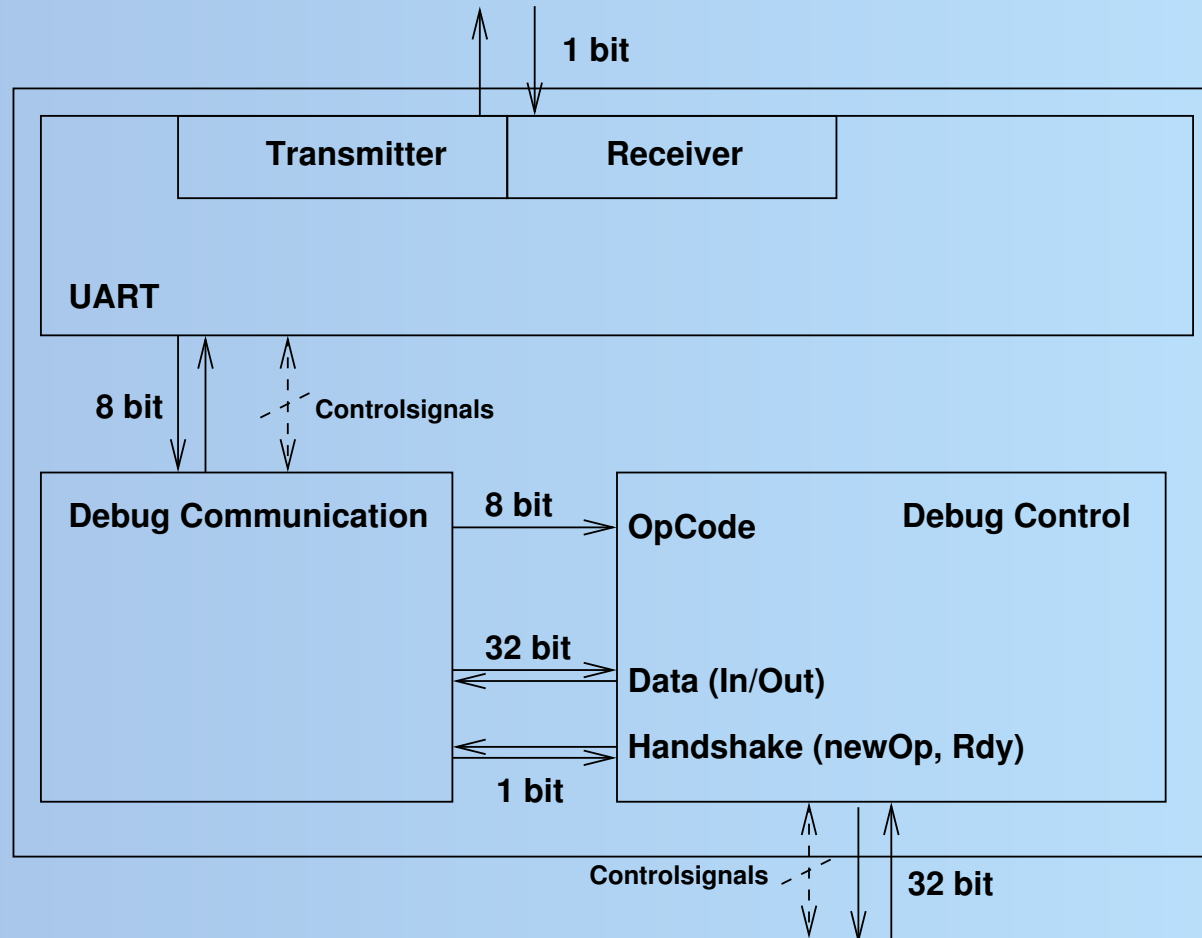


- ◆ Paramterierbar; zwei typische Varianten
  - 16-Bit PE, 1 Cluster à 4x4 PE ⇒ 16 PE
  - 8-Bit PE, 2x2 Cluster à 4x4 PE ⇒ 64 PE

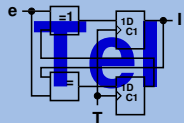
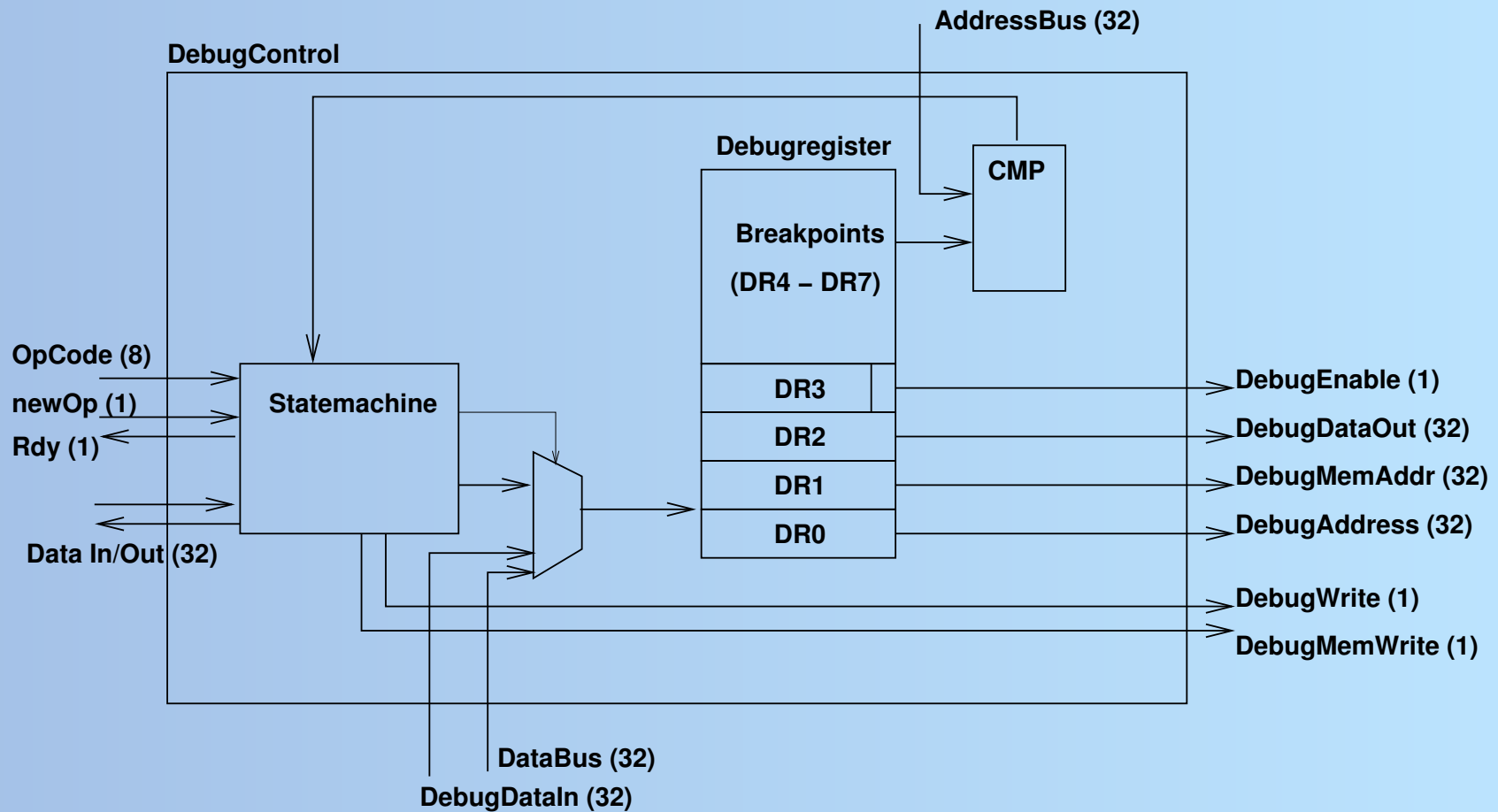
- ◆ RISC-Prozessor
  - 17 Register (R0 - R15, Flags)
  - Hardware-Loop-Unit
- ◆ ARRIVE-Komponenten
  - 8 Lokale Register
  - CCM (32-64 Einträge)
  - VLSU Contexttables (2)
  - Switch-Matrizen Contexttables (4)
  - RALU Contexttables (16 - 64)
  
  - PE-Register (16 - 64)



# Debugereinheit: Übersicht

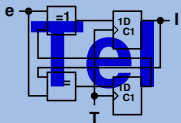


# Einheit DebugContol



# Debugcontrol: Funktionen

- ◆ getrennte Adressräume (Speicher, Zustand)
- ◆ jeder Adressraum 4G x 32 Bit
- ◆ 1 Bit zur Steuerung des Kontrollflusses
- ◆ HW-Breakpoints
- ◆ Auto-Inkrement der Debugregister

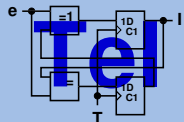


# Debugereinheit: Steuerbefehle

Nur 9 Befehle:

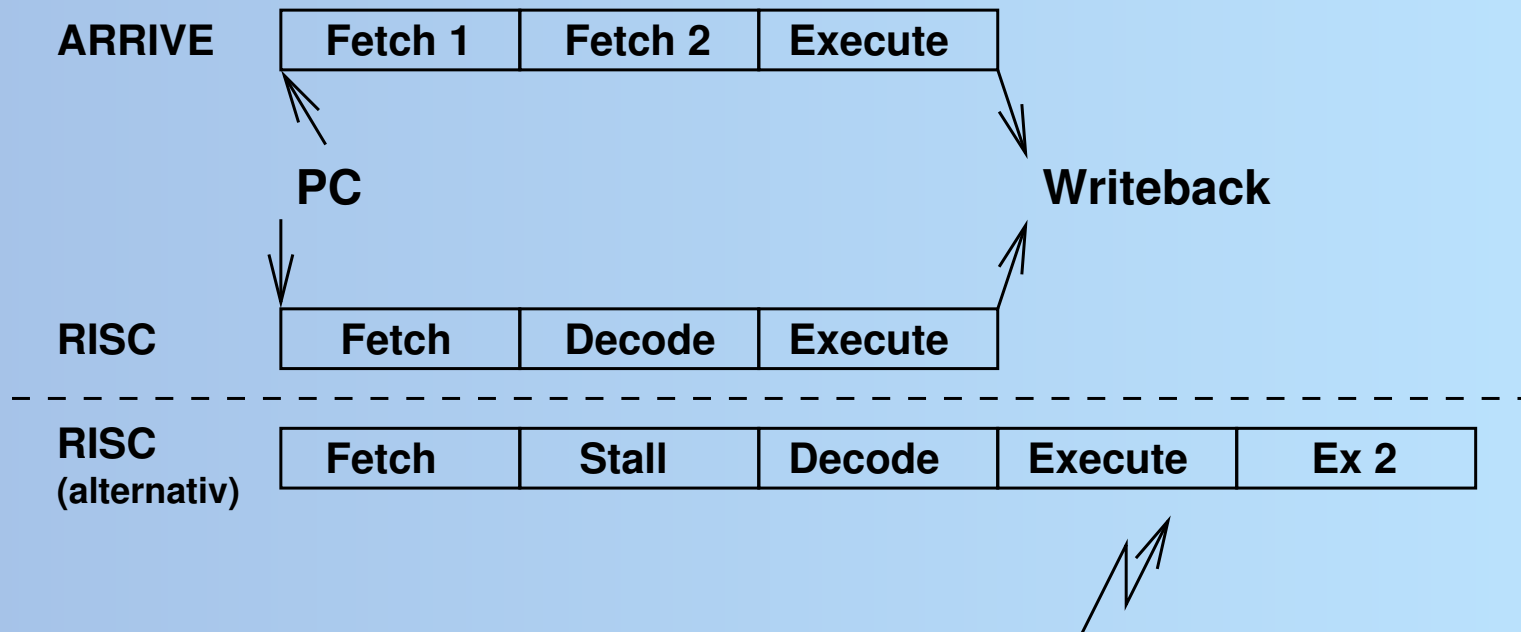
Befehlskodierung	Befehl
10001-Reg	SetDebugRegister (+32 bit)
11001-Reg	GetDebugRegister (-32 bit)
001-I-0xxx	WrMem[Inc]
011-I-0xxx	RdMem[Inc]
000-I-0xxx	WrReg[Inc]
010-I-0xxx	RdReg[Inc]
00001-000	STOP
00001-001	STEP
00001-111	RUN

⇒ vollständige Kontrolle über Target



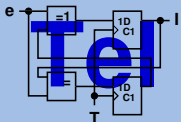
- ◆ Anhalten an Befehls Grenzen
- ◆ Jeder geholte Befehl wird ausgeführt!

Pipelining:





- ◆ Sammlung an Werkzeugen
  - Oberflächen (Textmodus, Java)
  - Programmlader für TADL-Programme
  - diverse andere Programme (Speicher-, Registerdump, ...)
- ◆ Debugsoftware einfach zu erstellen  $\Rightarrow$  Flexibilität
- ◆ Programme bisher nur bedingt parametrierbar (nur zur Compile-Zeit)



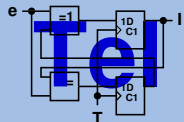
# Oberfläche (Text-Modus)

```
F1) UPDATE          F6) RUN F7) STEP F8) VIEW F10) QUIT          Stopped
<< Reg >>  << Memory >>  << CCM >>  << RALU >>  << Array Out >>

ArmReg-----LocalReg-----BreakPoints-----
R00 00000005 R08 00000000      L00 00120009      BP00 00000018 *
R01 00090001 R09 00000000      L01 0012001B      BP01 00000000
R02 00000000 R10 00000000      L02 00000000      BP02 00000000
R03 00000000 R11 00000000      L03 FFFFFFFF      BP03 00000000
R04 00000000 R12 00000000      L04 00000000
R05 00000000 R13 00000000      L05 00000000
R06 00000000 R14 00000003      L06 00000000
R07 00000000 R15 0000000C      L07 00000000
  Flags 00000010

CFMU-----00
  Start      End      Counter
00 00000008 00000014 00000002 ##
01 0000000C 00000010 00000002 <=
02 00000000 00000000 00000000
03 00000000 00000000 00000000

UP, DOWN, LEFT, RIGHT) Move F4) Insert
```



# Oberfläche (graphisch)

The screenshot displays a debugger interface with several panels:

- MainMem:** Memory dump showing addresses from 0x0 to 0x3c and their corresponding values.
- VLSUL:** Memory dump showing addresses from 0x0 to 0x3c and their corresponding values.
- ARM:** Register window showing registers R0 through R15 and their values.
- LocalReg:** Register window showing registers R0 through R7 and their values.
- HLD-Stack:** Stack window showing start and end addresses, counters, and active status.
- ARRIVE Control:** Control panel with buttons for RUN, STOP, STEP, and EXIT, and a table for breakpoints.

Mem	Value
Mem[0x0]	e3a00005
Mem[0x4]	e2800002
Mem[0x8]	e2800005
Mem[0xc]	eafffffb
Mem[0x10]	0
Mem[0x14]	0
Mem[0x18]	0
Mem[0x1c]	0
Mem[0x20]	0
Mem[0x24]	0
Mem[0x28]	0
Mem[0x2c]	0
Mem[0x30]	0
Mem[0x34]	0
Mem[0x38]	0
Mem[0x3c]	0

Mem	Value
Mem[0x0]	fb837a69
Mem[0x4]	82edc3b6
Mem[0x8]	74906865
Mem[0xc]	314adc2d
Mem[0x10]	6f9b32bc
Mem[0x14]	bc3b1b35
Mem[0x18]	8631b732
Mem[0x1c]	f44db698
Mem[0x20]	ad6ea947
Mem[0x24]	24737bad
Mem[0x28]	f9dc7a91
Mem[0x2c]	cc7b8f79
Mem[0x30]	7a270c5c
Mem[0x34]	2c971cfb
Mem[0x38]	87ff4218
Mem[0x3c]	3863ec37

Reg	Value
R0	c
R1	0
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0
R13	0
R14	0
R15	0

Reg	Value
R0	0
R1	0
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0

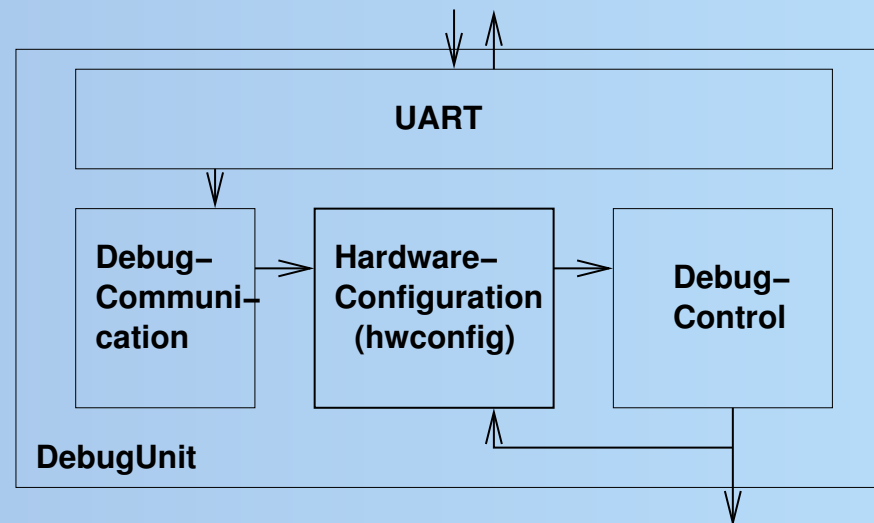
Startaddress	Endaddress	Counter	active
0	0	0	
0	0	0	
0	0	0	
0	0	0	

BP	Address	Enabled
BP0	0	<input type="checkbox"/>
BP1	0	<input type="checkbox"/>
BP2	0	<input type="checkbox"/>
BP3	0	<input type="checkbox"/>

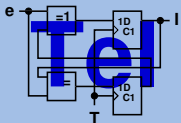
# Debugereinheit: Erweiterung

Ziel: erweiterte Unterstützung des Debuggings

- ◆ Memory-Tracing
- ◆ komplexe Debug-Befehle bzw. Ausnutzung der Debugereinheit durch RISC-Kern



- ◆ Umsetzung des Konzepts funktioniert

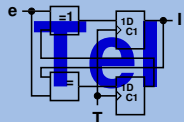


# Ergebnisse (Algorithmen)

Testalgorithmen:

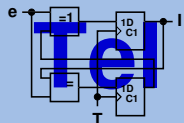
Algorithmus	Arrive-Variante	Ladezeit	Resultat
IIR-Filter	8-Bit	1895 ms	läuft korrekt
FIR-Filter	8-Bit	1045 ms	läuft korrekt
Viterbi-Decoder	8-Bit	1953 ms	läuft korrekt
DCT8	8-Bit	1080 ms	läuft korrekt
DCT8x8	8-Bit	1129 ms	läuft korrekt
Turbo-Decoder	16-Bit	7270 ms	läuft korrekt

- ◆ Einschränkung: 16-Bit Variante nicht kompatibel zu Prosim
- ◆ FFT nicht lauffähig (eingeschränkter Befehlssatz des RISC)



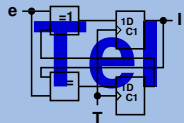
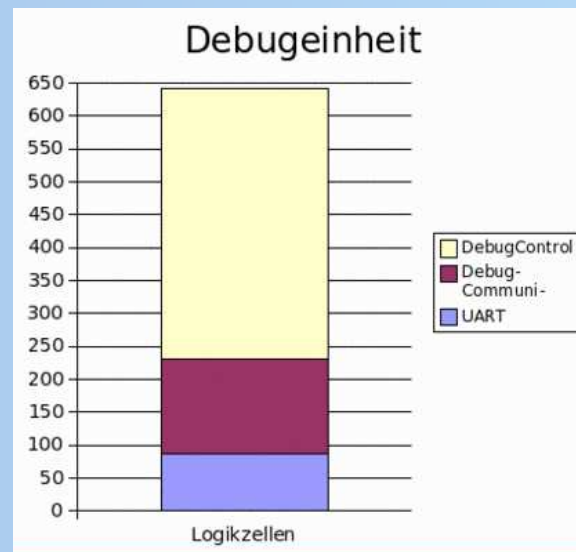
# Benchmark-Ergebnisse

	ARM	ARRIVE		
		8-bit PE	16-bit PE	
FIR Filter	9.5	0.5	0.25	cycles/tap
IIR Filter	59.7	3	2	cycles/biquad
FFT Radix-2	99608	8739	5100	cycles/512pt
Viterbi Decoder	212	11	12	cycles/symbol
Turbo Decoder	4648	–	216	cycles/symbol
DCT	5213	170	136	cycles



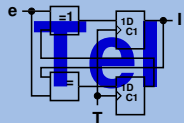
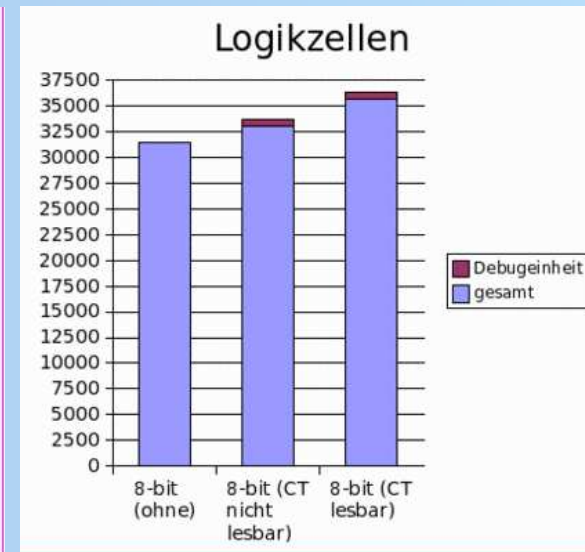
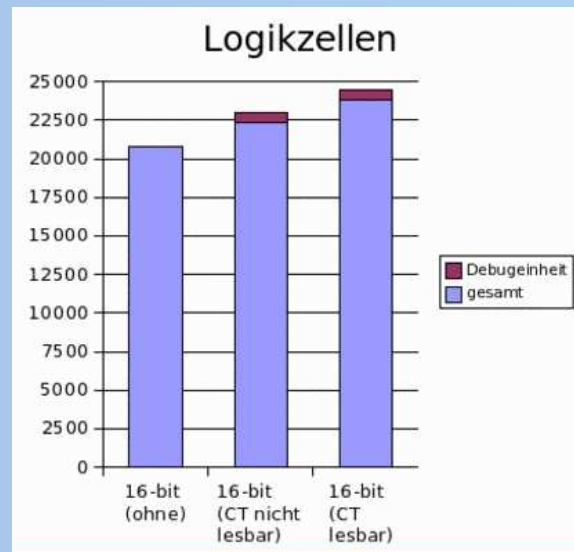
# Ergebnisse (Logikzellen I)

- ◆ FPGA EP1S80: ca. 80000 Logikzellen (LC)
- ◆ Debugereinheit: 643 Logikzellen
- ◆ vgl. RISC-Kern: 3100 Logikzellen
- ◆ konstanter Anteil des Gesamtmehraufwands: etwa 1500 LC



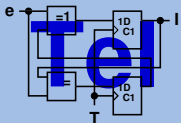
# Ergebnisse (Logikzellen II)

- ◆ Logikzellen 16-bit: +7.3% +14.4%
- ◆ Logikzellen 8-bit: +4.8% +13.4%
- ◆ Geschwindigkeit bleibt gleich (11.6 MHz bzw. 6.2 MHz)
- ◆ kritischer Pfad durch RALU





- ◆ Vorstellung ARRIVE als Beispiel für eine rekonfigurierbare Architektur
- ◆ Erfüllung der Debuganforderungen (Flexibilität, Skalierbarkeit, Beobachtbarkeit) in der Realisierung
- ◆ Implementationsdetails (stoppen RISC-Kern, Entwurf und Realisierung der HLU) wurden nicht genannt
- ◆ ARRIVE-Prototyp konnte durch Debugereinheit erfolgreich auf FPGA in Betrieb genommen und getestet werden
- ◆ Standardzellen-ASIC-Synthese leider nicht möglich gewesen



- ◆ Verbesserung der Debugwerkzeuge
  - verbessertes Parsen der TADL-Programme
  - Parametrierung der Oberfläche zur Startzeit oder Laufzeit
  - Parametrierung der Werkzeuge zur Nutzung mit anderen Architekturen
- ◆ Erweiterung des Mikroprogrammsteuerwerks
- ◆ Standardzellen-ASIC-Synthese

