Eigendiagnose und Systemüberwachung in Multi-FPGA/Prozessor-Systemen

Diplomverteidigung

Marcel Köhler

s0241122@inf.tu-dresden.de

Technische Universität Dresden

Institut für Technische Informatik

Übersicht

- Aufgabenstellung
- Motivation
- Signalion Digital Radio Module – Multi-FPGA Plattform
- IST und SOLL
- Ansätze zu Diagnosekonzepten in Computern
- Diagnosekonzept für die SDRM – Plattform
- Umsetzung des Konzeptes auf der SDRM-Plattform
- Zusammenfassung und Ausblick

Schwerpunkte der Aufgabenstellung

- **Ausgangspunkt: Grundprinzipien der Eigendiagnose/ Systemüberwachung in Computersystemen**
- **Erarbeitung allgemeiner Lösungsvorschläge zur Eigendiagnose/ Systemüberwachung**
- **Konzept zur Integration einer Lösung für SDRM-Plattform**
- **Implementation, Evaluierung und Test**

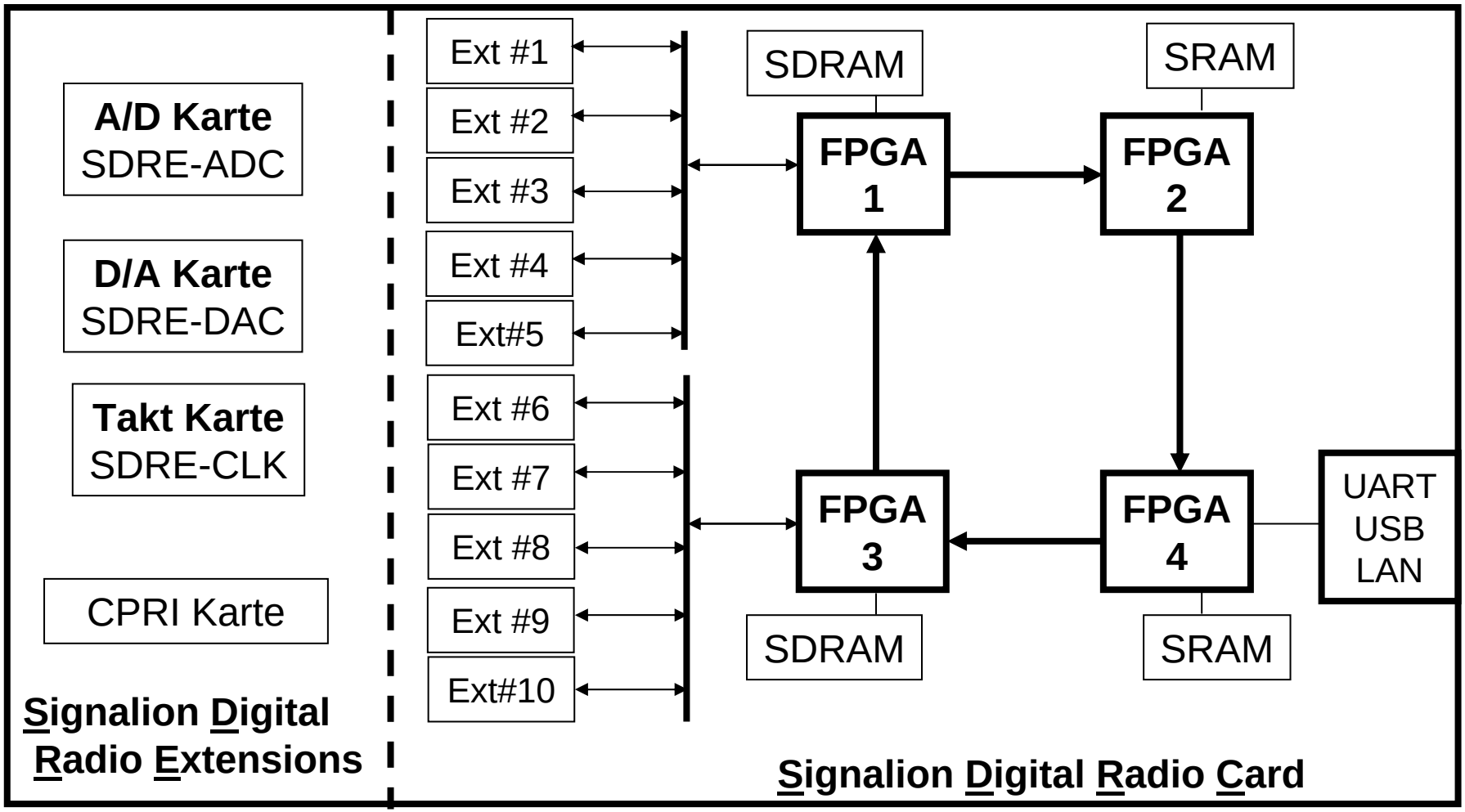
Motivation

- **rasante Entwicklung Computertechnik**
  - stetig steigende Funktionalität und Leistung durch neue Technologien
  - Komplexität der Systeme
- **Sicherstellung des fehlerfreien Systembetriebs**
  - Zuverlässigkeit und Stabilität
  - Diagnose- und Überwachungsfunktionen zur Feststellung des technischen Zustands
  - zielgerichtete und schnelle Beseitigung im Fehlerfall
  - Qualitätssteigerung des Endproduktes

SORBAS und Multi-FPGA Plattform SDRM

- Software Radio Based Access System
- Test Mobile für neuen Mobilfunkstandard LTE (Long Term Evolution)
- Prinzip: Software Defined Radio
- Flexibles System durch Softwareupdates anpassbar

Aufbau der Multi-FPGA Plattform SDRM



Signalion Digital Radio Module (SDRM)

IST und SOLL (1)

- nur minimale Integritätssicherung
- korrekter Download der Konfigurationdaten für FPGA wird angezeigt
- Softwareinkompatibilitäten der FPGA-Implementierungen unberücksichtigt
- Basisinitialisierung für DAC-Karte und CLK-Karte

→ Ursache über Fehlverhalten in Start-/ Betriebsphase schwer diagnostizierbar

IST und SOLL (2)

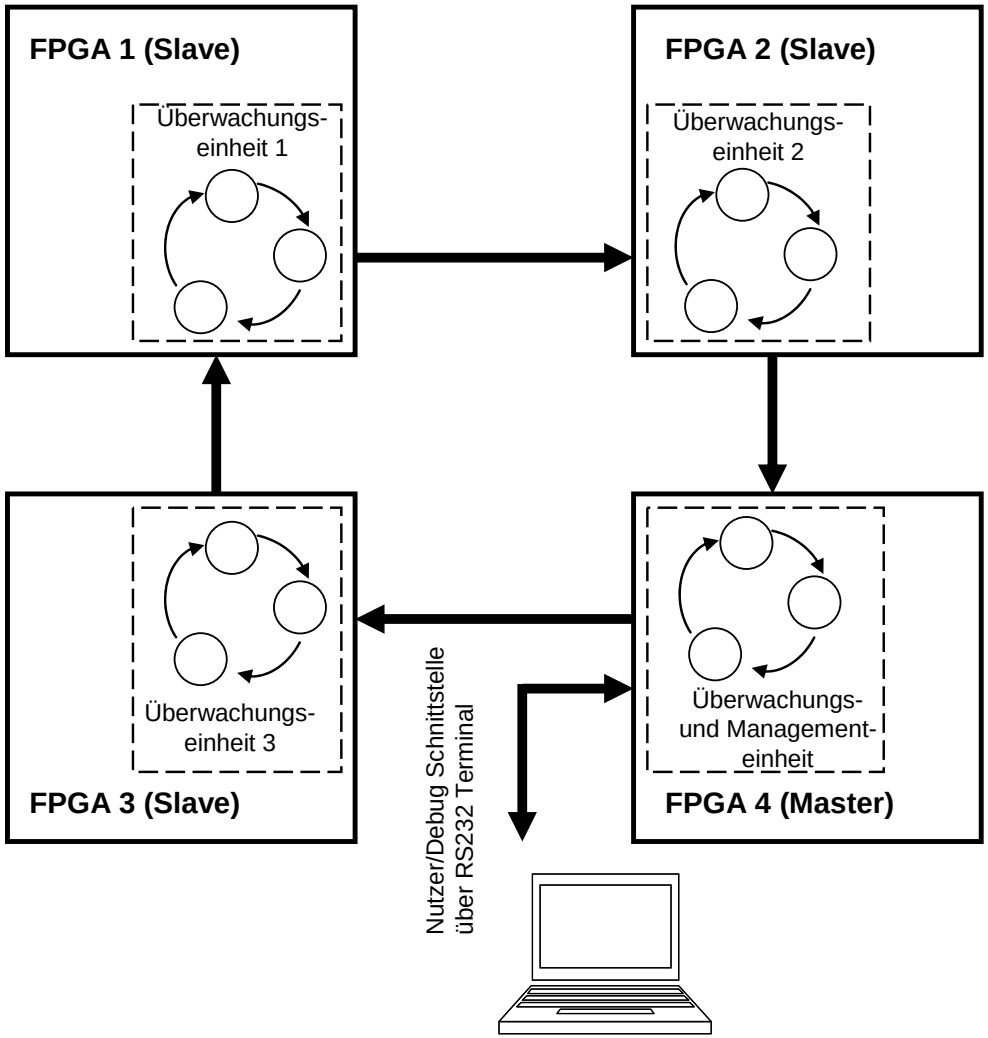
- **Integration der Diagnosefunktionalität in existierendes Application Layer Model**
- **Basisfunktionalität: Entwurf eines Bootvorgangs zur sukzessiven Freigabe von HW-Ressourcen**
  - Kommunikationsstruktur, Softwareversionscheck, Peripherietest
- **Konzeptgestaltung unter Berücksichtigung nachträglicher Erweiterung der Diagnosefunktionalität während der Betriebsphase**
- **Eingrenzung von Fehlerquellen zur schnelleren Behebung**
- **Flexibilität durch automatische HW-Identifikation**

Ansätze zu Diagnosekonzepten in Computern

- **ähnlich modularer Aufbau wie SDRM**
  - Hauptplatine, Erweiterungen durch Steckkarten, I/O, Speicher…
- **Bootvorgang BIOS & POST**
  - Routinen: Hardwareerkennung, -initialisierung, -test
- **Diagnose in Startphase: POST-Code Ausgabe zur Diagnose**
  - Beep-Code/ Hex-Code/ Sprachausgabe durch spezielle ICs
  - zielgerichtete Beseitigung von Fehlerursachen
- **Diagnose in Betriebsphase: Monitoring-Tools zur Überwachung**
  - Temperaturen, Lüfterdrehzahl, Spannungen
  - meist konfigurierbar, d.h. bei Überschreiten von Toleranzen
    - Fehlermeldung, -protokoll, Fehlerstoppverhalten

Lösungsansätze - Prinzipieller Aufbau

- verteiltes Diagnosesystem
- Master-Slave Prinzip
- Bereitstellung Userinterface über Master
- Management über Master

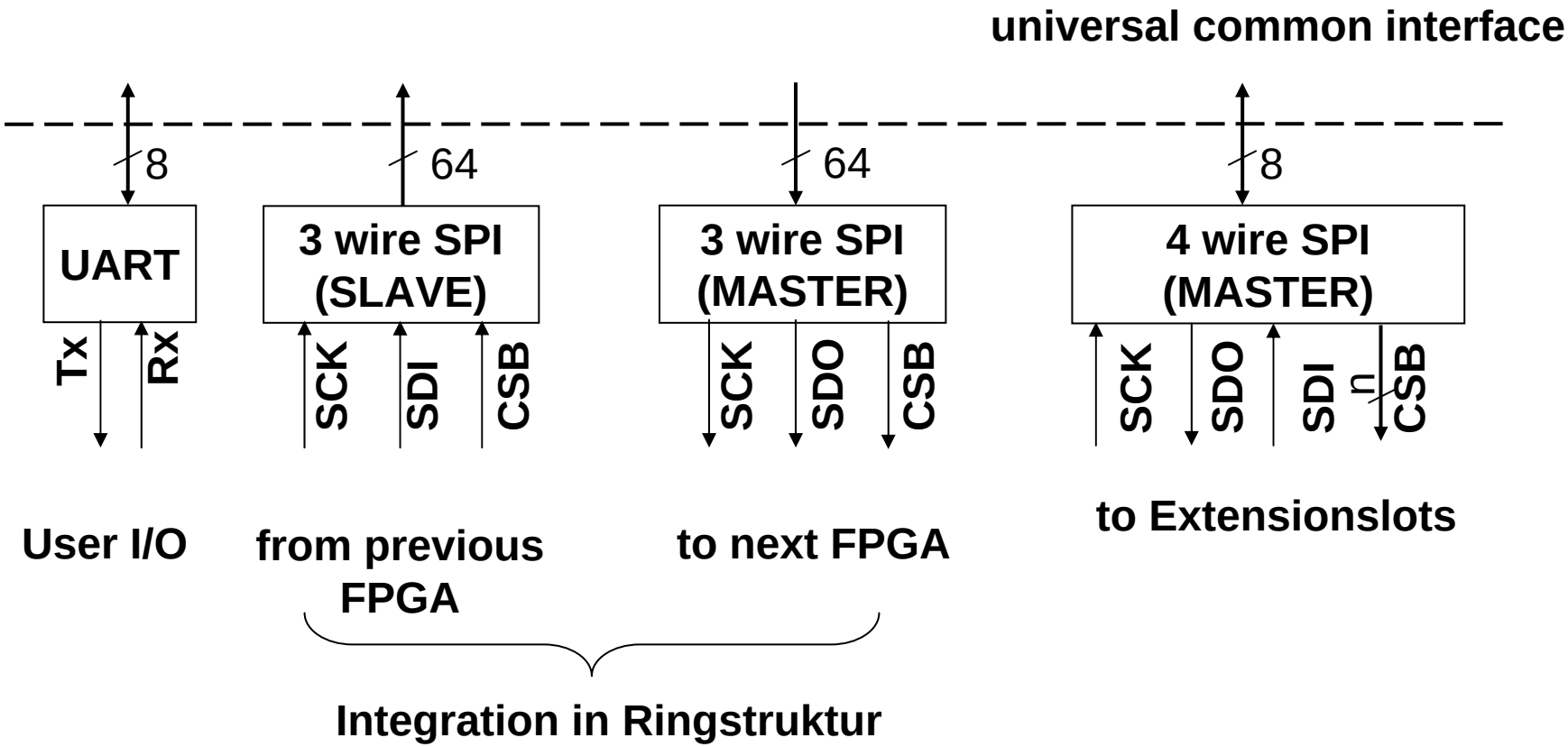


universelles Diagnosekonzept

- **Re-use der HW Komponenten in allen FPGA-Knoten**
- **Erweiterbarkeit – schnelle Anpassung an andere Systeme**
- **Austauschbarkeit des Diagnosekerns zur Anpassung an Zielsystem**
- **Verlagerung der Diagnosefunktionalität in Softwareschicht**
  - Flexible Gestaltung der Algorithmen zur Abarbeitung der Fehlerroutinen
  - Variable Festlegung von Testzeitpunkten: Systemstart, -pausen, in bestimmten Zeitabständen

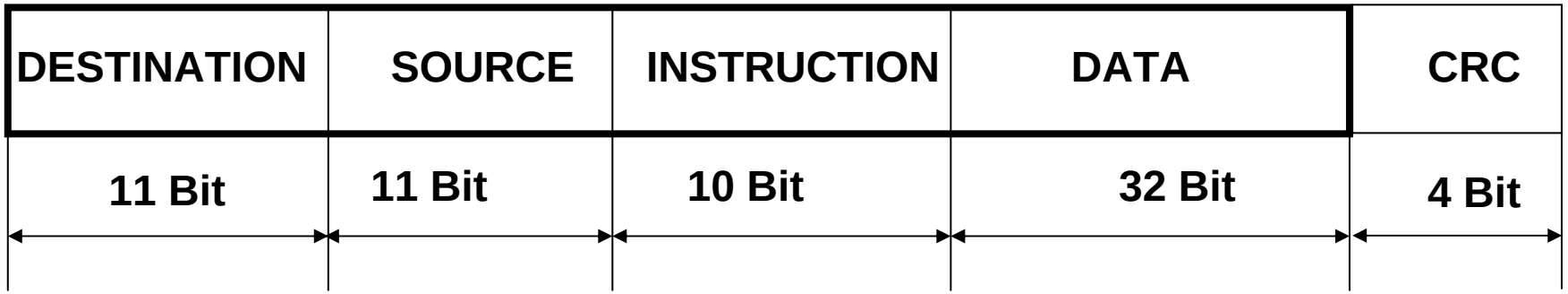
Universelles Diagnosekonzept – einheitliches Systeminterface

- SPI – seriell synchrones Interface

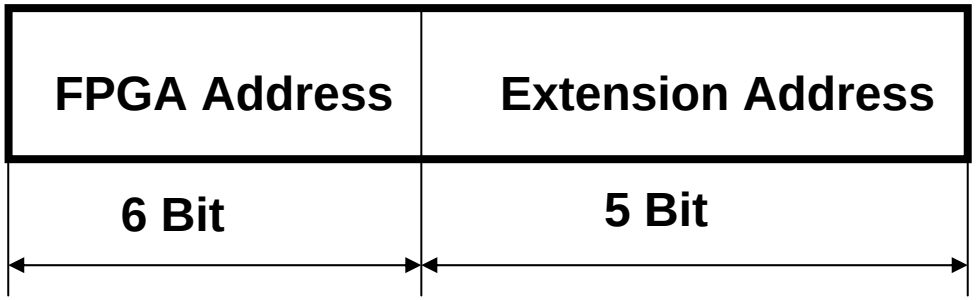


paketorientierte Kommunikation

- Paketstruktur
- vor Übertragung CRC Schutzkodierung



- Adressierung



Diagnosekonzept – Startphase (Boot)

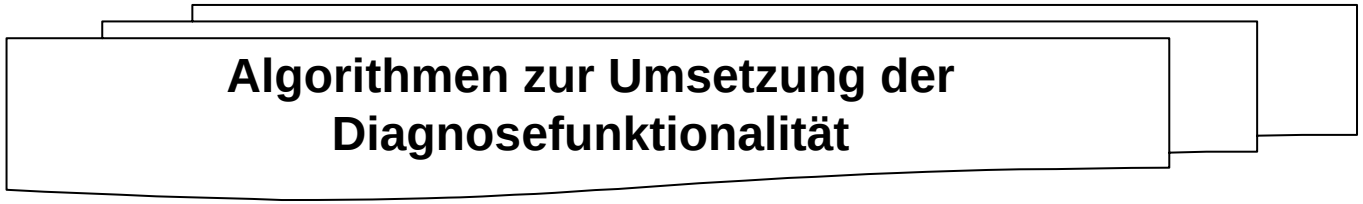
- **Sicherstellung der Betriebsbereitschaft aller Funktionseinheiten**
- **umfasst einfache Tests und Hardwareinitialisierung**

1. **Test der Ringstruktur**
2. **FPGA Test (Softwareversion) und Freigabe**
3. **Scan Erweiterungskarten, Initialisierung, Auslesen Typ/Version/Status und Freigabe**
4. **Report/ Protokollierung**

Diagnosekonzept - Betriebsphase

- **Abfrage im laufenden Betrieb umfasst:**
- **Statusabfragen HW**
- **Versionsabfrage der SW der Module auf FPGA**
- **Auslösen bestimmter Tests**
- **Abruf Fehler-Protokoll über User-Interface**
- **Einspielen von Updates für Erweiterungskarten**

Möglichkeiten zur Umsetzung des Konzeptes



Variante 1:

Finite State Machine
- Abbildung direkt in Hardware
- neue Synthese bei Änderung
- geringe Flexibilität

Variante 2:

Verwendung von Mikrocontrollern
- Verwendung IP-Cores (Soft-IP)
- Ressourcenverbrauch besser abschätzbar
- Erweiterung oder Änderung von Algorithmen durch schnellen Tausch des Programmcodes

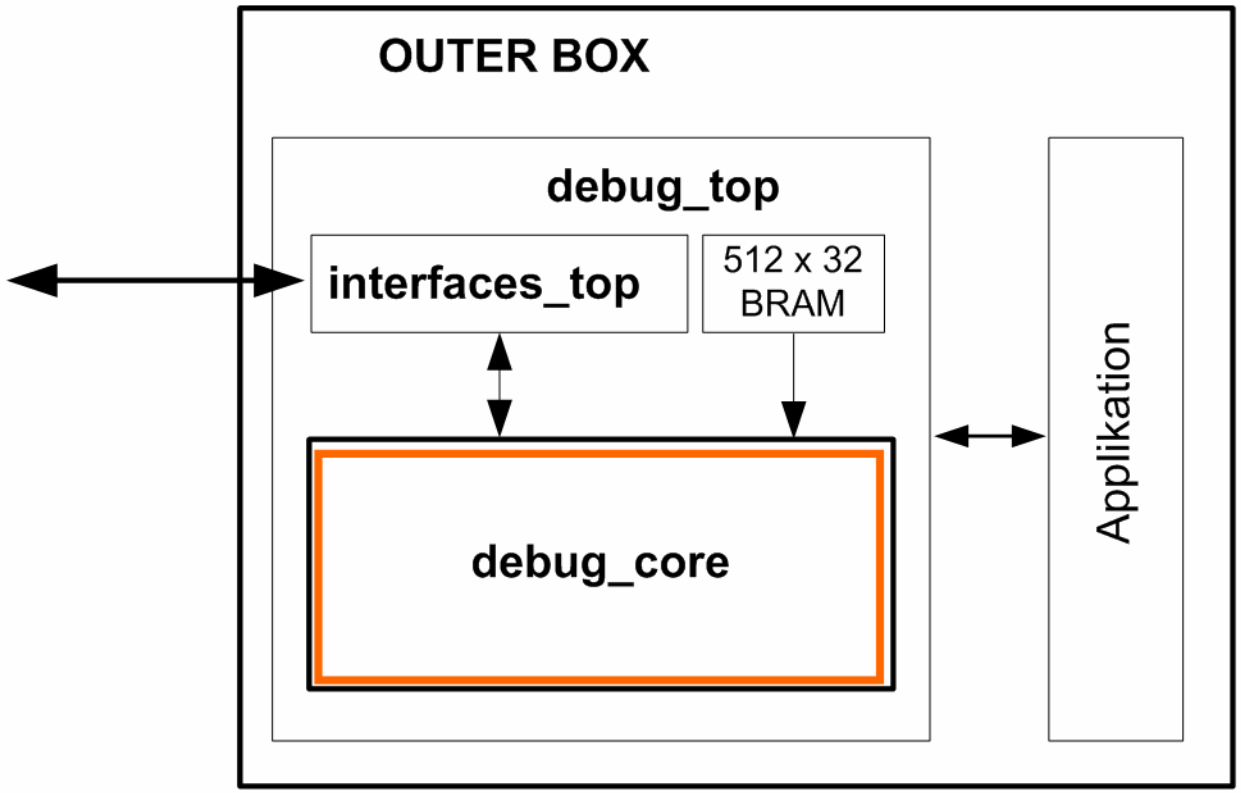
→ **PicoBlaze Ansatz:**

- 8-Bit Mikrocontroller, max. 1024 Befehle Programmspeicher
- Umsetzung in Assembler

→ **MicroBlaze Ansatz:**

- 32-Bit Mikrocontroller, 64 KB Programmspeicher, erweiterbar
- Umsetzung in C-Code

Implementierung - Architektur



- **universelle Austauschbarkeit des Controller-Kerns**

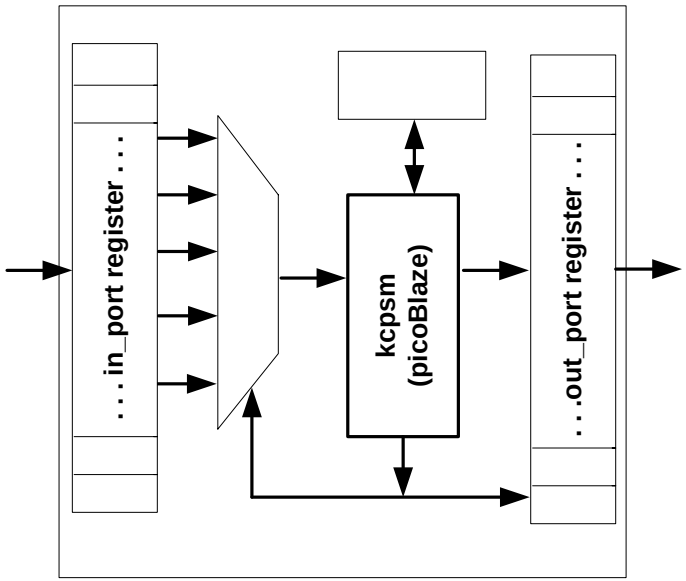
→ Anpassung der Diagnosefunktionalität an Zielsystem

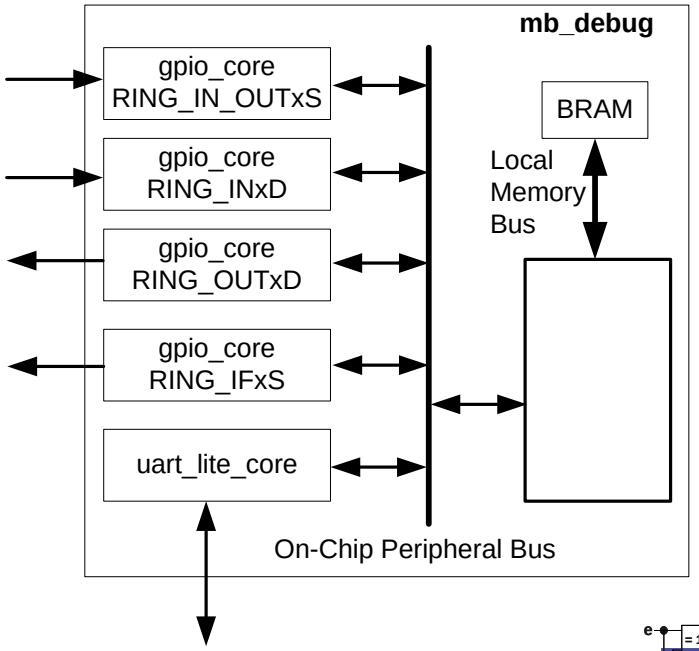
Implementierung - Diagnosekern

- **Umsetzung der Diagnosefunktionen in Software**

Lösungsvariante 1:
- Xilinx PicoBlaze in allen Diagnoseeinheiten
- Umsetzung in Assembler

Lösungsvariante 2:
- Xilinx MicroBlaze
- Austausch des Diagnosekerns im Master-FPGA
- Umsetzung in C-Code





debug_ctrl

Implementierung - Erweiterungskarten

- **Integration der Erweiterungskarten in Bootprozess**
- **Erweiterung des existierenden C-Codes für ATmega8 Mikrocontroller**
  - jeder Kartentyp stellt eindeutige EXT_ID bereit
  - Verbindung zur SDRC über Software-SPI
  - Interface interruptgesteuert
- **Bootablauf**
  - ATmega8 initialisiert/ konfiguriert kartenspezifische ICs
  - Bootstatus der Karte setzen
  - Freigabe der Interrupts
  - Diagnoseabfrage kann erfolgen

Implementierung - Protokollierung

Mit PicoBlaze Core

- **Ausgabe der mitgeschnittenen Pakete**
- **Maschinenlesbares Protokoll**

…

10 00 87 C1 00 00 00 C3

10 00 87 E1 00 00 00 1A

10 00 87 F1 00 00 00 01

…

Mit MicroBlaze Core

- **Pakete werden im Core ausgewertet**
- **Ausgabe erfolgt in ASCII-Kodierung**

…

Extension Bus Test…

Extension ID Code: 0xC3

SDRE-ADC at FPGA#1 EXT#2 detected.

SW ID is: 0x1A

Card Status ok.

…

Ergebnisse

- Welche Fehler werden entdeckt?
- Diagnosesystem zum Auffinden von Low-Level Fehlern:
- Fehler beim Laden der FPGA-Konfiguration
- inkompatible Prozessor-/ Verarbeitungseinheiten
- falsch platzierte Erweiterungskarten
- Fehlen einer Takt Referenzkarte (Bedingung für Applikation)
- Ausfall einer Funktionseinheit

Bewertung der Implementierung

- **Ressourcenverbrauch Slave FPGAs:**

	FPGA 1	FPGA 2	FPGA 3
Slices	500 (0,7%)	431 (0,6%)	511 (0,8%)
Slice FFs	730 (0,5%)	615 (0,5%)	713 (0,5%)
4-input LUTs	759 (0,6%)	570 (0,4%)	659 (0,5%)

- **Ressourcenverbrauch Master FPGA:**

FPGA 4	PicoBlaze	MicroBlaze
Slices	499 (0,7%)	2679 (4%)
Slice FFs	704 (0,5%)	1977 (1%)
4-Input LUTs	717 (0,5%)	3842 (3%)

- **Taktfrequenz des Entwurfs: 100 MHz**

- **Gesamtdauer des Bootprozesses t_{boot} = 39,4 ms / 478 ms**

Zusammenfassung und Ausblick

- **Diagnosekonzept in x86-Computern untersucht**
- **Entwicklung eines universellen Diagnosekonzepts**
- **Basis – Diagnosefunktionalität in Form eines Bootprozess**
- **Integration des universellen Diagnosekonzepts auf SDRM-Plattform + Umsetzung eines Bootalgorithmus**
- **Anwendbarkeit des entwickelten Konzeptes**
  - Produktion
  - Support beim Kunden
  - Erweiterbarkeit auf komplexere Systeme
  - Einfache Erweiterung um weitere Überwachungsfunktionen

Vielen Dank für Ihre Aufmerksamkeit!

Auslastung der Programmspeicher

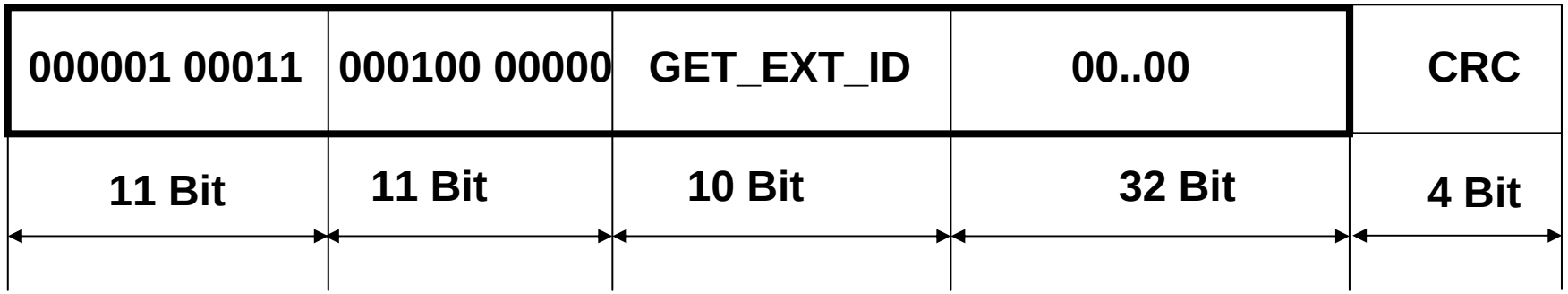
- **Kapazität der Programmspeicher:**
  - picoBlaze: 18 kbit (nicht erweiterbar)
  - MicroBlaze: 64 kByte (erweiterbar)

	Auslastung
FPGA 1	**15%**
FPGA 2	**11%**
FPGA 3	**18%**
FPGA 4 mit picoBlaze	**99%**
FPGA 4 mit MicroBlaze	**18%**

- **AVR ATmega8: 1338 Bytes /8 KB (16 %)**

paketorientierte Kommunikation - Beispiel

- Abfrage des Kartentyps in Erweiterungssteckplatz 3 am FPGA 1



- Anwortpaket