TECHNISCHE UNIVERSITÄT DRESDEN

REKONFIGURIERBARE ARCHITEKTUREN

Robert Rasche

Dresden, 24.05.2011

01 Motivation

Ausgangssituation in eingebetteten Systemen:

- Verarbeitungsleistung ist auf Embedded Prozessor begrenzt

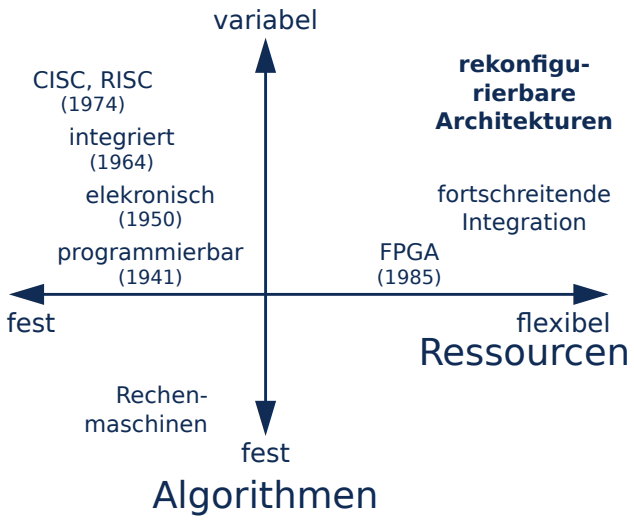
→ Prozessor (General Purpose) ungünstig, um genau eine Aufgabe sehr gut zu erfüllen

- zusätzliche (Logik-) Strukturen zur Unterstützung in Custom-Hardware (ASIC, PLA)

→ Problem: Funktionalität ist fixiert, kaum Anpassung (z.B. neue Versionen) möglich

→ also: rekonfigurierbare Hardware nutzen und Prozessorsystem selbst flexibel gestalten

01 Rekonfigurierbare Hardware



Trend:

- frühe Rechner implementieren genau eine Aufgabe

→ eine Hardware für einen Algorithmus

- programmierbare Universalrechner
- konfigurierbare Hardware
- stetig steigende Integration

→ Platz für rekonfigurierbare Rechnerarchitekturen

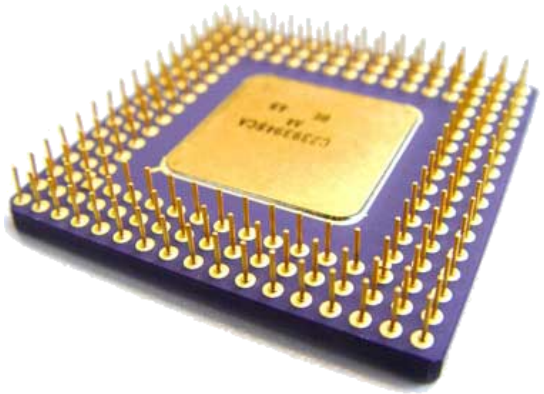
01 Quelle

Struktur und Material für diesen Vortrag:
[1] A Survey on Dynamically Reconfigurable Processors

- → Diskussion wesentlicher Aspekte rekonfigurierbarer Systeme
- → Klassifikationskriterien
- → Verlgeich verschiedener kommerzieller Systeme

02 Plattformen

02 Plattformen



- welche Klassen rekonfigurierbarer Hardware gibt es?
- Aspekte, Vor- und Nachteile
- Eignung für rekonfigurierbares Computing

02 CPLD, FPGA

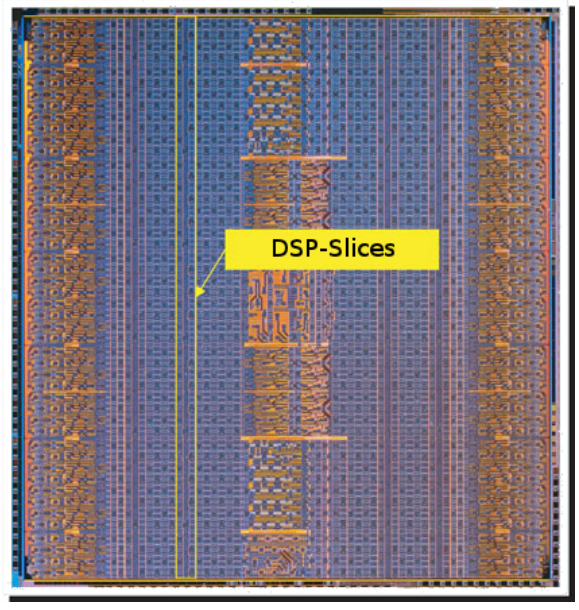
- feingranulare konfigurierbare Logik
- sehr große Anzahl an Einheiten (CLBs)
- Switch-Matrizen zum Verbinden der Einheiten

→ sehr großes Bit-File zur Konfiguration

- unhandlich bei Abbildung von Prozessoren

→ z.B. Microblaze-Kern

02 FPGA + dedizierte Blöcke



chipdesignmag.com

nicht nur feingranulare Logik sondern häufig benötigte Blöcke für

- Arithmetik (wortweise Verarbeitung)
- Signalverarbeitung

→ v.a. Multiplikation, MAC

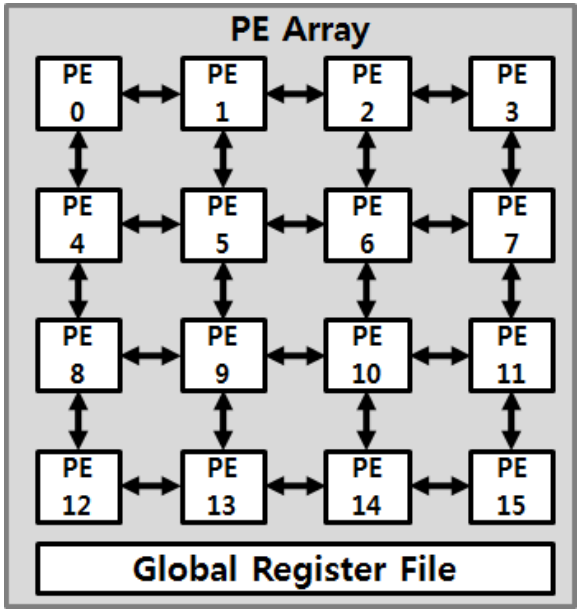
- Registerfiles
- verteilter SRAM

02 FPGA + IP-Core

integrierter Prozessorkern, umgeben von rekonfigurierbaren Blöcken

- verschiedene optimierte IP-Cores (ARM, PowerPC, AVR)
- Tools für die Hardware-Software Cosynthese
- auch andere Einheiten als Hard-Core: (Ethernet, PCI, ...)
- auch Mehrkernsysteme

02 CGRA, rDPA, PE-Arrays



[4]

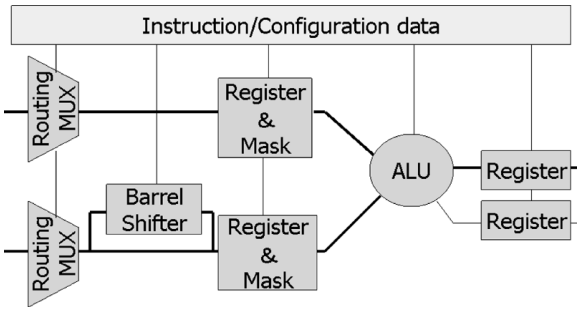
'Coarse Grain Reconfigurable Architecture' oder 'reconfigurable Data Path Array'

- Verarbeitungseinheiten für Bitgruppen und ganze Worte (4bit, 8bit, ... , 32bit)
- üblicherweise ALUs und Wort-Register als Zellen
- wortweises Routing, Routing zu Nachbarzellen

→ nur Bruchteil an Konfigurationsdaten ggü. FPGA. (je nach Wortbreite $^1/_{10} \ldots {}^1/_{100}$)

→ ineffizient für Bit-Operationen und spezifische Logik

02 PE Arrays – Aufbau einer Zelle



[?]

Relativ komplexe Einheiten für wortbreite Verarbeitung.

- verschiedenste Ausführungen
- Shifter, Mask, ALU, Register
- RAM-Zellen
- Routing als Teil der Zelle
- homogene oder heterogene Anordnung (Schachbrett, Streifen) im Array

02 PE Arrays – Aufbau einer Zelle

M	PE	M	PE	M	PE	M	PE
PE	M	PE	M	PE	M	PE	M
M	PE	M	PE	M	PE	M	PE
PE	M	PE	M	PE	M	PE	M

M	PE	M	PE	M	PE	M	PE
M	PE	M	PE	M	PE	M	PE
M	PE	M	PE	M	PE	M	PE
M	PE	M	PE	M	PE	M	PE

[?]

Relativ komplexe Einheiten für wortbreite Verarbeitung.

- verschiedenste Ausführungen
- Shifter, Mask, ALU, Register
- RAM-Zellen
- Routing als Teil der Zelle
- homogene oder heterogene Anordnung (Schachbrett, Streifen) im Array

02 Prozessor Arrays

Einfache Prozessoren und verteiltes RAM im Array

- jede Prozessorzelle hat eigene Instruktionsspeicher und eigenen Instruktionspointer
- benachbarte RAM-Blöcke addressierbar
- Umkonfiguration über die Software der Prozessorzelle

02 Zusammenfassung

Spielraum für konfigurierbare Hardwareplattform

- immer: Array-Struktur für die konfigurierbare Hardware
- unterschiedliche Granularität und Komplexität der Zellen (FPGA bis Prozessorarray)
- dedizierte optimierte Einheiten (Multiplizierer bis Ethernet)
- verteiltes RAM im Array
- Host-Prozessor

→ System on a Programmable Chip (SoPC)

03 Rekonfiguration

03 Rekonfiguration

Wünschenswert: Umkonfiguration des Systems zur Laufzeit

- Hardware wird zu verschiedenen Zeiten der Ausführung mit gerade passender Konfiguration betrieben

→ eine Verarbeitungseinheit (Zelle, Block) wird für mehrere Aufgaben genutzt

→ bessere durchschnittliche Auslastung der Chipfläche

- ermöglicht auch adaptive Hardware

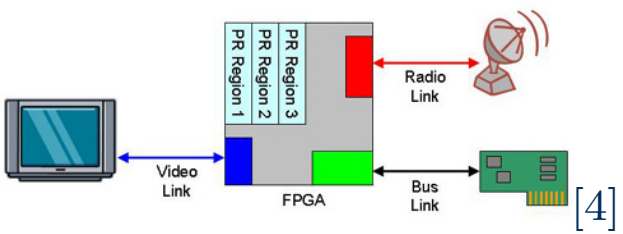
03 Rekonfiguration

Aber: Konfiguration ist (daten)aufwändig

- FPGA Konfigurationsfile sehr groß
  (z.B. Virtex4 'XC4VFX60': 20 MiB [3])
- zeitaufwendige Umkonfiguration
  über Bitstrom (ms bis s) oder wortweise (ms)
- gesamter Baustein währenddessen angehalten
  oder in Reset-Modus

$\rightarrow$ für on-the-fly Rekonfiguration ungeeignet

03 Partielle Rekonfiguration 1



[4]

Besser: Partielle Rekonfiguration: nur Teile des Bitfiles neu schreiben

- differentiell: gezielt kleine Bereiche verändern (einzelne Bits oder Worte)

→ z.B. Datenpfad anders routen

- modular: ein reservierter Bereich wird mit neuem Bitfile geladen

→ neuer Hardwareblock für eine neue Aufgabe

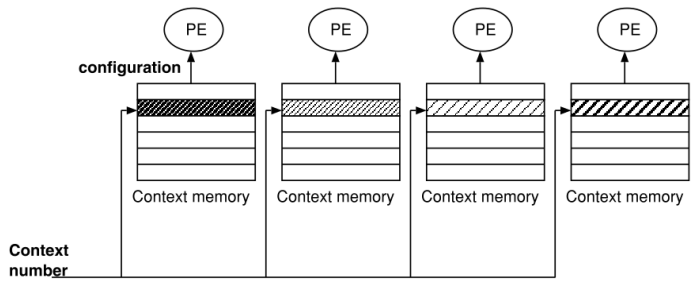
→ Rest der Konfiguration kann weiterlaufen

03 Partielle Rekonfiguration 2

- Konfigurationsdaten liegen im System (on chip)
- Ausliefern relativ schnell möglich (10μs)
- aber: Verarbeitung währenddessen zumindest teilweise unterbrochen

$\rightarrow$ nicht schnell genug!

03 Konfigurationskontexte 1



[1]

Kontextbasierte Rekonfiguration: mehrere Konfigurationen für jede Einheit im Hintergrund

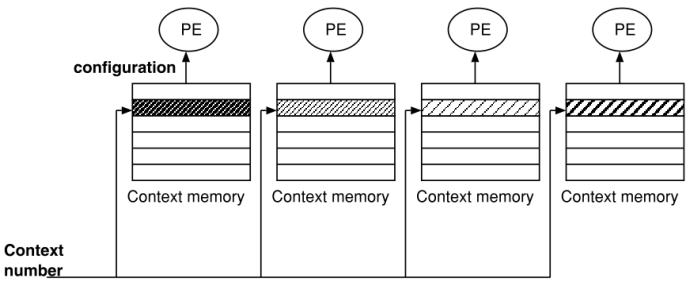
- (globales) Kontextsignal: legt fest, welche davon verwendet wird

→ sehr schnell (1 Takt), keinerlei Unterbrechung nötig

- Konfigurationsspeicher ist über die Einheiten verteilt
- Anzahl und Größe der Kontexte bestimmt benötigte SRAM-Größe pro Zelle

→ effizient für Coarse-Grain und PE-Arrays

03 Konfigurationskontexte 2



[1]

Kontexte können als Instruktionen für eine Einheit betrachtet werden:

- Umkonfiguration in jedem Verarbeitungsschritt (Takt)

→ vollkommen anderes Verhalten der Einheit (andere Instruktion)

- Konfigurationsspeicher ist Mikroprogrammspeicher
- Auswahl eines Kontextes entpricht Ausführung einer Instruktion

04 Hostprozessoren

04 Hostprozessor

Ein Prozessorkern zur Kontrolle des rekonfigurierbaren Teils

- leistungsfähiger 'General Purpose' Kern
- Erweiterung der Möglichkeiten des Hosts mit konfigurierbarer Logik
- Array ist 'Beschleuniger' für den Prozessor

04 Integration der Beschleunigerfunktionen

Möglichkeiten:

- Erweiterung des Befehlssatzes des Hosts um spezielle Instruktionen

→ Array wird als Ausführungseinheit für diese konfiguriert und verwendet

- Einsatz als 'Coprozessor': auf Tasks von Host warten

→ komplett unabhängige Ausführung des Tasks und Rückschreiben des Ergebnisses

04 Datenaustausch mit Hostprozessor

verschiedene Möglichkeiten, Daten zwischen Array und Host zu vermitteln:

- Zugriff auf gemeinsamen Hauptspeicher (oder Cache)

→ Matrix und Host arbeiten (kooperativ) parallel

- über Systembus des Host-Datenpfades
- Austauschregister: vom Prozessor und von einigen Einheiten des Arrays aus zugreifbar

05 Programmierung

05 Programmiermodelle

Problem: Wie Funktionalität auf Software (für den Host) und Hardwarebeschleuniger (Matrix) aufteilen?

- Zuordnung:
  Was wird Software, was Hardware?
- Automatisierung:
  Was ist die 'beste' Lösung?
- Beschreibung:
  Programmier- oder HW-/Verhaltenspache?

05 Hardware-Software-Codesign

Hardware- und Softwareteile der Anwendung werden vom Programmierer abgestimmt

- oft verschiedene Beschreibungsformen gemischt
- 'saubere' Lösungen, aber nicht immer optimal
- intensive Tests notwendig

05 Hardware-Software-Cosynthese

Partitionierung und Synthese erfolgen automatisch

- einheitliche High-Level Beschreibung (z.B. C, Fortran)
- Erkennen von Parallelismus im Datenflussgraphen des Programmes
- Automatisch Synthese von Konfigurationen für kritische parallelisierbare Abschnitte (komplexe Ausdrüke, Schleifenkerne)
- Erzeugen von passendem Maschinencode, der die synthetisierten Einheiten ausnutzt

05 Online-Synthese

Erzeugen von geeigneten Hardwarekonfigurationen, während das Programm läuft

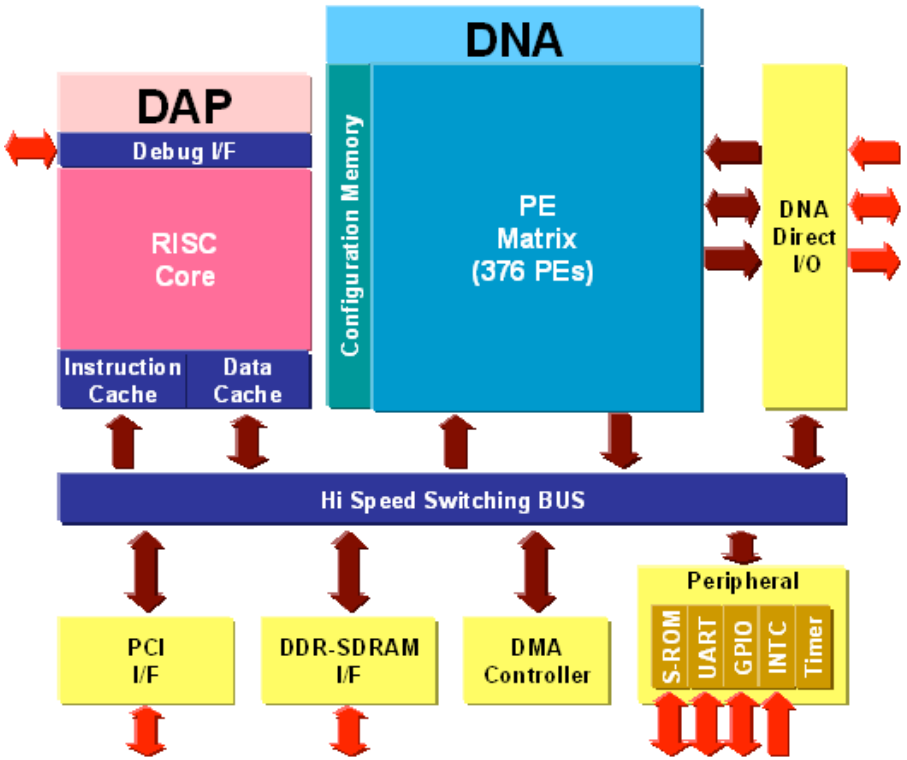
- Profiling der laufenden Anwendung

→ Erkennen laufzeitkritischer Abschnitte

- Synthese auf dem Zielsystem selbst
- Synthesejob läuft, wenn Hostprozessor frei ist
- benötigt synthesefähige Repräsentation des Programms (z.B. Bytecode)

06 Beispiel

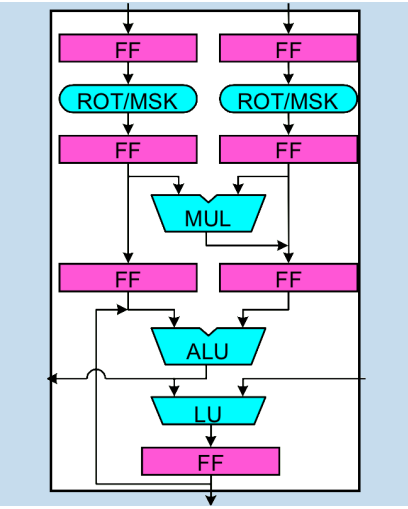
06 IPFlex DAP/DNA-2 (2004)



[2]

- Host-Array-Verbindung: über Systembus
- I/O-Controller direkt mit dem Array verbunden
- Konfigurationsspeicher nahe am Array (4 Kontexte)
- 576 KiB Speicher im Array

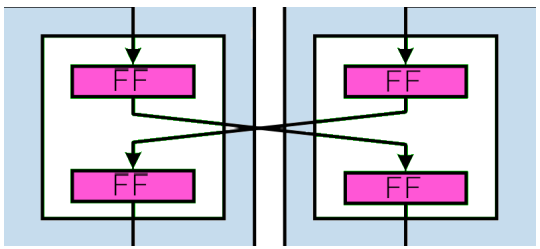
06 DAPDNA-2 – Processing Element



[2][4]

- verschiedene Elemente vorhanden
- ALU-PE: 16 bit ALU und Multiplizierer
- Interconnect-PE: am Rand eines Segments
- Routing innerhalb Segment über zwei Matrizen
- 6 Segmente zu je 8x8 PEs

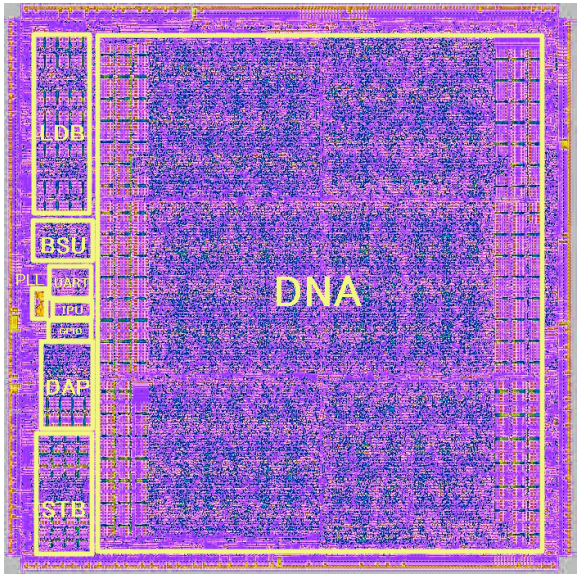
06 DAPDNA-2 – Processing Element



[2][4]

- verschiedene Elemente vorhanden
- ALU-PE: 16 bit ALU und Multiplizierer
- Interconnect-PE: am Rand eines Segments
- Routing innerhalb Segment über zwei Matrizen
- 6 Segmente zu je 8x8 PEs

06 DAPDNA-2 – Processing Element



[2][4]

- verschiedene Elemente vorhanden
- ALU-PE: 16 bit ALU und Multiplizierer
- Interconnect-PE: am Rand eines Segments
- Routing innerhalb Segment über zwei Matrizen
- 6 Segmente zu je 8x8 PEs

06 DAP/DNA-2 – Toolchain

- eigene C-artige Programmiersprache
- eigene Entwicklungsumgebung
- grafisches Design der Algorithmen möglich
- MATLAB/Simulink - Integration

07 Fragen und Abschluss

 H. Amano A Survey on Dynamically Reconfigurable Processors , IEICE TRANS. COMMUN., VOL.E89–B, NO.12, 2006

 Tomoyoshi S. A Dynamically Reconfigurable Processor with 376 32-bit Processing Elements HotChips17 16 Aug 2005

 Virtex-4 FPGA Configuration User Guide , Xilinx, UG071 (v1.11) June 9, 2009

 und ja: auch und vor allem google und wikipedia für manche Bilder