Karlsruher Institut für Technologie
Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung
Prof. Dr. rer. nat. Wolfgang Karl

Klausur Rechnerstrukturen Wintersemester 2015/16 Musterlösung

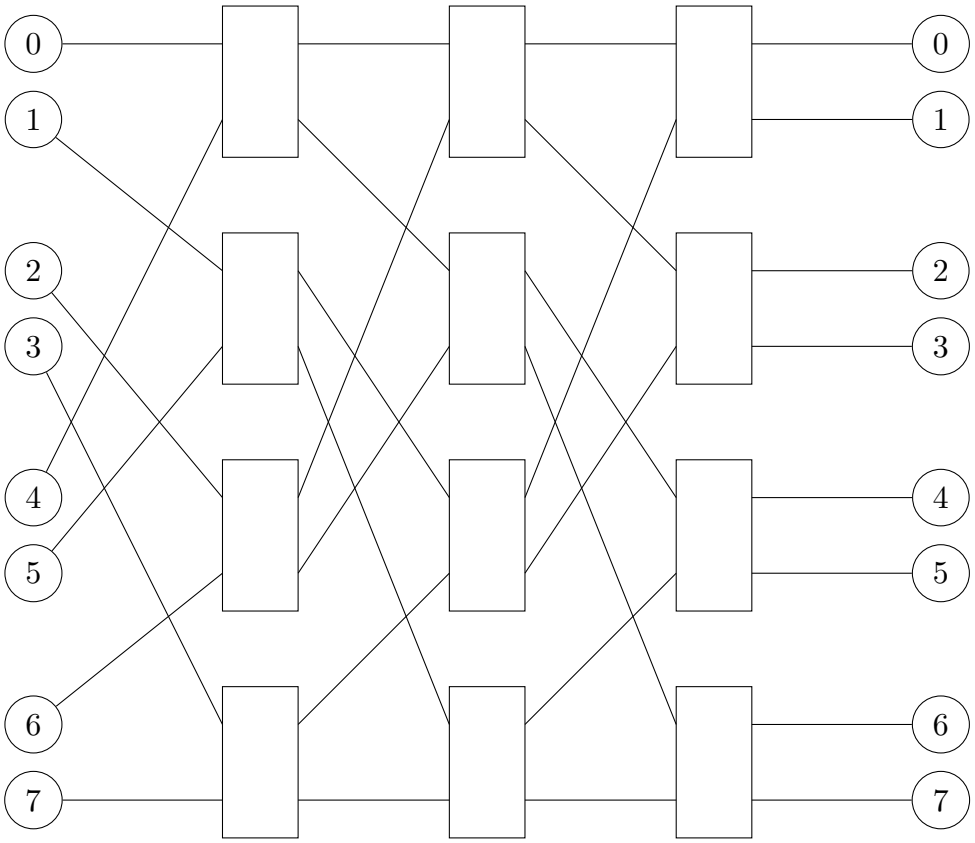
Voraussichtliche Bekanntgabe der vorläufigen Ergebnisse:
April 2016

Aufgabe 1: Verbindungsstrukturen und Fehlertoleranz — 10 P

Verbindungsstrukturen — 6 P

a) Bei vollständigen statischen Verbindungsnetzen ist jeder Knoten direkt mit jedem anderen Knoten verbunden, weshalb die Netzwerkkosten quadratisch mit der Anzahl der Knoten steigt. *1 P*

b) *2 P*



c) Anzahl Switches: $\frac{N \cdot \log_k N}{k}$ *1 P*

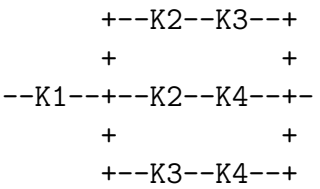
d) *1 P*

- Teurer Verbindungsaufbau, der sich bei kurzen Nachrichten nicht amortisiert
- Blockiert Leitungen während der kompletten Kommunikation, was bei hoher Netzauslastung zu langen Wartezeiten führen kann

e) Beim „store and forward“-Modus wird die Nachricht in jedem Zwischenknoten zunächst komplett in Empfang genommen und vollständig zwischengespeichert. Erst wenn die komplette Nachricht erhalten und gespeichert wurde, wird sie an den nächsten Knoten weiter übertragen. Beim „cut through“-Modus ist dies nicht der Fall. Hier können die einzelnen Teile der Nachricht direkt an den nächsten Knoten weitergeleitet werden. *1 P*

Fehlertoleranz **4 P**

f) Zuverlässigkeitsblockdiagramm: *2 P*



g) Mengen: *1 P*
$B_1 = \{K1\},\ B_2 = \{K2\},\ B_3 = \{K3\},\ B_4 = \{K4\},$
$B_5 = \{K1, K3\},\ B_6 = \{K1, K4\},\ B_7 = \{K2, K3\},\ B_8 = \{K2, K4\}$

h) Zusammenhang: *1 P*
$R(t) = 1 - F_L(t)$ bzw. $F_L(t) = 1 - R(t)$

Aufgabe 2: Low-Power-Entwurf und Leistungsbewertung — 10 P

Low-Power-Entwurf — 5 P

a) Statisch: P_{static}, $P_{leakage}$ Dynamisch: $P_{switching}$, $P_{shortcircuit}$ *2,5 P*
Aufgrund der Miniaturisierung leisten heutzutage Leckströme ($P_{leakage}$) einen wesentlichen Beitrag zur Leistungsaufnahme.

b) Höhere Frequenzen erfordern steilere Taktflanken und damit ein schnelleres Laden von C_{eff} woraus die Notwendigkeit einer höheren Versorgungsspannung resultiert. *1 P*

c) Schaltwarscheinlichkeit und XOR-Gatter: *1,5 P*

$$\begin{aligned}
\mathbb{P}_{Schalt} &= 2 \cdot \mathbb{P}_{Ausgang}(1) \cdot (1 - \mathbb{P}_{Ausgang}(1)) \\
\mathbb{P}_{AusgangXOR}(1) &= \mathbb{P}_{Eingang1}(0) \cdot \mathbb{P}_{Eingang2}(1) + \mathbb{P}_{Eingang1}(1) \cdot \mathbb{P}_{Eingang2}(0) \\
&= \frac{1}{2} \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{2} \\
\mathbb{P}_{Schalt} &= 2 \cdot \frac{1}{2} \cdot \left(1 - \frac{1}{2}\right) = \frac{1}{2}
\end{aligned}$$

Leistungsbewertung — 5 P

d) Prozessor 2: Niedrigere $MIPS$-Werte resultieren bei gleicher Laufzeit in kompakteren Code sowie niedrigeren Schaltfrequenzen und damit geringerem Energieverbrauch *1,5 P*

e) $SPEC_{int}$, weil es sich hierbei um eine per Benchmark (genauer: $SPEC$ ist eine Sammlung von Benchmark-Programmen) ermittelte Leistungsquantifizierung handelt (die Eingangswerte zur Berechnung des Maßes werden also durch Benchmarks ermittelt), die durch den Bezug auf eine Referenzmaschine einen fairen Vergleich zwischen Systemen erlaubt. *1,5 P*

f) *2 P*
- Hardware-Monitore
  - \+ unabhängige physikalische Geräte, dadurch keine Beeinflussung des zu beobachtenden Systems und damit exakte Ergebnisse
  - – aufwändigere Implementierung und Installation
- Software-Monitore
  - \+ einfachere Implementierung

– Beeinträchtigung der normalen Betriebsverhältnisse, da typischerweise im Betriebssystems installiert - damit Ergebnisse weniger exakt

Aufgabe 3: Quantitative Maßzahlen und Pipelining 10 P

Quantitative Maßzahlen 5 P

a) Formel für Amdahls Gesetz: *1 P*

$$T(n) = \underbrace{\frac{1}{n} \cdot T(1) \cdot (1-a)}_{\text{parallel ausführbarer Programmteil}} + \underbrace{T(1) \cdot a}_{\text{sequentieller Programmteil}}$$

mit a = der ausschließlich sequentiell ausführbare Programmanteil und n = die Parallelisierungstiefe (beispielsweise die Anzahl der Prozessoren).

b) Das Verhalten heißt superlinearer Speedup und widerspricht $1 \leq S(n) \leq n$. *1 P*

c) *1 P*
- Einprozessorsystem: Häufig benötigte Daten passen nicht komplett in den Cache bzw. Hauptspeicher
- Mehrprozessorsystem: In der Regel insgesamt mehr Cache- und Hauptspeicher vorhanden. Häufig benötigte Daten passen dadurch in die Caches bzw. Hauptspeicher des jeweiligen Prozessorkerns bzw. Rechenknoten.

d) Maßzahlen: *2 P*
$T(n) = \frac{1}{n} \cdot T(1) \cdot (1-a) + T(1) \cdot a = \frac{4}{5} \cdot \frac{90s}{12} + 90s \cdot \frac{1}{5} = 24s$
$S(n) = \frac{T(1)}{T(n)} = \frac{90s}{24s} = 3,75$
$I(n) = \frac{P(n)}{T(n)} = \frac{576}{24} = 24 \qquad U(n) = \frac{I(n)}{n} = \frac{24}{12} = 2$

Pipelining 5 P

e) *1,5 P*
- Strukturkonflikte (alternativ: Ressourcenkonflikte)
- Datenkonflikte
- Steuerkonflikte

f) *1,5 P*
- Strategie 1: Pipeline stall (Anhalten der Pipeline)
- Strategie 2: Pipeline bubble (Einfügen eines Leerzyklus)
- Folge: Leistungseinbußen

g) *2 P*
- Durchsatzobergrenze (IPC$\leq$ 1 oder CPI$\geq$ 1), Lösung: Nebenläufigkeit, parallele Pipeline
- Ineffiziente Pipeline, Lösung: spezialisierte Pipelines
- „Pipeline Stall“-Strategie, Lösung: verteilte Ausführungs-Pipelines

Aufgabe 4: Hardware-Entwurf 10 P

a) Festverdrahtetes Steuerwerk: *2 P*

- Ein solches Steuerwerk wird meistens über einen endlichen Zustandsautomaten realisiert.

Mikroprogrammierbares Steuerwerk:

- Die Steuerung des Datenpfades wird über ein Mikroprogramm realisiert, dass auf einem Miniaturcomputer ausgeführt wird. Hierfür werden Kontrollbefehle, die den Datenpfad steuern, in einer Tabelle gespeichert.

b) Festverdrahtetes Steuerwerk: *3 P*

- Realisierung in Hardware
  - Realisierung als Hardware-Tabelle, wobei ein Eintrag anhand des aktuellen Zustands, Teile der Instruktion und Informationen des Datenpfades bzw. des Speichers ausgewählt wird.
- Reduzierung von Hardware-Kosten
  - Durch Ausnutzung von Redundanz kann auf Kosten der Adressierungslogik der benötigte Speicher für die Tabelle reduziert werden. Solche Speicher werden auch programmed logic arrays (PLA) genannt.
- Optimierungsmöglichkeiten für Leistungssteigerungen
  - Reduzierung der durchschnittlichen Anzahl von Zuständen entlang einer Instruktion reduziert die Anzahl von Zyklen pro Instruktion (CPI)

Mikroprogrammierbares Steuerwerk:

- Realisierung in Hardware
  - Als Minicomputer mit Mikroprogrammspeicher und Adressauswahllogik, die anhand von Informationen des Datenpfades sowie des Mikroprogrammspeichers die nächste Mikroinstruktion auswählt. Die Startadresse eines Mikroprogramms wird anhand der Assemblerinstruktion ausgewählt.
- Reduzierung von Hardware-Kosten
  - Durch Kodierung von Bits im Mikroprogrammspeicher, verschiedene Instruktionsformate nutzen,..
- Optimierungsmöglichkeiten für Leistungssteigerungen
  - Reduzierung der Anzahl von Zyklen pro Instruktion (CPI) durch Erhöhung der Parallelität, d.h. mehre Operationen in einer Mikroinstruktion enthalten

c) Sichten in VHDL: *2 P*

- Strukturelle Beschreibung

  Darunter versteht man die Beschreibung einer Skizze, die angibt, welche Komponenten zur Realisierung eines Bausteins/Schaltung benutzt werden und wie diese geeignet untereinander als auch mit den Ein-und Ausgaben verbunden sind.

- Funktionale Beschreibung/Verhaltensbeschreibung

  Bei dieser Art der Beschreibung wird der Baustein durch nebenläufige Prozesse, die die Belegungen der Signale funktional bestimmen, beschrieben. Somit beschreiben Prozesse, wie Signale sich in Abhängigkeit anderer Signale verhalten.

d) Entity-Beschreibung: 0,5 P Benennung, 0,5 P Syntax *3 P*
Architecture-Beschreibung: 1 P Benennung, 1 P Syntax

```
ENTITY blinklicht is
     PORT(
         clk : in std_logic;
         reset : in std_logic;
         led : out std_logic
     );
END blinklicht;

ARCHITECTURE structure of blinklicht is
    COMPONENT counter
    GENERIC(countMax : positive);
    PORT(
        clk : in std_logic;
        out : out std_logic
    );
    END COMPONENT
    COMPONENT DCM
    PORT(
        clkin_in : IN std_logic;
        rst_in : IN std_logic;
        clkdv_out : OUT std_logic;
    );
    END COMPONENT;

    signal clk_int : std_logic;
BEGIN
    Inst_DCM : DCM
    PORT MAP(
        clkin_in => clk,
        rst_in => reset,
        clkdv_out => clk_int,
    );
    Inst_counter : counter
    GENERIC MAP (countMax => 25000000)
    PORT MAP(
        clk => clkIn,
        out => led
    );
END structure;
```

Aufgabe 5: Caches und Sprungvorhersage — 10 P

Caches — 6 P

a) *4 P*

Proz.	Aktion	Proz. 1		Proz. 2		Proz. 3	
		Zeile 1	Zeile 2	Zeile 1	Zeile 2	Zeile 1	Zeile 2
	init	-	-	-	-	-	-
1	rd 1	1/E					
3	rd 3					3/E	
2	rd 1	1/S		1/S			
3	wr 3					3/M	
1	rd 3		3/S			3/O	
2	wr 3		3/I		3/M	3/I	
3	rd 4					4/E	
1	rd 4		4/S			4/S	
2	rd 2			2/E			
3	rd 2			2/S			2/S

b) Harvard-Architektur *1 P*

c) Mit dem Dirty-Bit wird eine im Cache per Schreibzugriff modifizierte Zeile markiert, die sich noch nicht im Hauptspeicher befindet und somit bei einer Verdrängung in den Hauptspeicher zurückgeschrieben werden muss. *1 P*

Sprungvorhersage — 4 P

d) *4 P*

	Globaler Prädiktor	S1 Vhs.	S1 Sprung	Globaler Prädiktor	S2 Vhs.	S2 Sprung
1	(**WT**, WT)	T	NT	(**WNT**, WT)	**NT**	NT
2	(**SNT**, WT)	NT	T	(WNT, **WT**)	T	NT
3	(**WNT**, WNT)	NT	T	(WT, **WNT**)	NT	T
4	(WT, **WT**)	**T**	NT	(**WT**, WNT)	T	NT

Aufgabe 6: Parallelrechner und Speicherarchitekturen — 10 P

a) *2 P*

- Single instruction, single data, Beispiel: Einkernprozessor mit sequentieller Ausführung
- Single instruction, multiple data, Beispiel: Vektorprozessor
- Multiple instruction, single data
- Multiple instruction, multiple data, Beispiel: Parallelrechner

b) *3 P*

- Nein. Superskalare Prozessoren können theoretisch auch „in order“ parallel arbeiten, wenn nur solche Befehle parallel ausgeführt werden, deren Ausführung auch „in order“ abgeschlossen wird.
- Nein. Befehle können, soweit es ihre Abhängigkeiten zulassen, auch auf einer skalaren Architektur beliebig verschoben werden. Dies bringt in den meisten Fällen allerdings keinen Vorteil mit sich. Durch das Vorziehen von Ladebefehlen könnte sich zum Beispiel ein Vorteil ergeben.

c) Beispiel für mögliche Antworten: *3 P*

- Superskalarität
  - \+ Kein Mehraufwand für Programmierer oder Compiler
  - – Komplexere Hardware
- Vektorverarbeitung
  - \+ Im Vergleich zu VLIW kompakterer Code
  - – Nur datenparallele Aufgaben lassen sich parallelisieren
- VLIW
  - \+ Im Vergleich zu Superskalararchitektur einfache Hardware
  - – Programmierer oder Compiler müssen entsprechenden Code erstellen

d) *2 P*

- Erklärung Unterschied:
  - – Bei einer „Uniform Memory Access (UMA)“-Architektur greifen alle Prozessoren über dieselbe Verbindungsstruktur auf einen oder mehrere Speicher zu. Für alle möglichen Prozessor-Speicher-Kombinationen ergibt sich dadurch dieselbe Zugriffslatenz.

    Vorteil: Der Zugriff auf alle Daten im Speicher ist gleich schnell. Dadurch ist die Programmierung einfacher.
  - – Bei einer „Non-uniform Memory Access“-Architektur greifen die einzelnen Prozessoren über unterschiedliche Wege auf einen oder mehrere Speicher zu. Zum Beispiel können einzelne Speicher mit bestimmten

Prozessoren verbunden sein und andere Prozessoren können erst über einen oder mehrere Zwischenschritte durch ein Verbindungsnetzwerk auf einen Speicher zugreifen, was die Speicherlatenz entsprechend erhöht.

Vorteil: Wenn die oft benötigten Daten der Prozessoren jeweils komplett in den lokalen Speichern liegen, können die Prozessoren ihren schnellen lokalen Speicher größtenteils exklusiv nutzen. Dadurch kann eine kürzere Anwendungslaufzeit erreicht werden.

- Bevorzugte Architektur: NUMA. Unter der Voraussetzung, dass die benötigten Daten größtenteils im Speicher mit der niedrigsten Latenz gehalten werden, ist die Gefahr geringer, dass im Gegensatz zu UMA-Architekturen das gemeinsame Verbindungsnetzwerk zum Flaschenhals wird.