Fabian Lesniak, Oliver Sander

Hardware/Software Co-Design

Übungsskript

Institut für Technik der Informationsverarbeitung, Karlsruher Institut für Technologie

Kapitel 1: Einleitung

Aufgabe 1.01: Hardware/Software Co-Design

Diskutieren Sie die Vor- und Nachteile der Implementierung eines Systems in Hardware bzw. Software bzgl. folgender Kriterien:

a) Entwurfszeit bzw. time-to-market
b) Performanz
c) Kosten
d) Leistungsverbrauch
e) Wartbarkeit bzw. Änderbarkeit
f) Testbarkeit
g) Sicherheit

Aufgabe 1.02: Architekturen

Überlegen Sie sich Kriterien, welche die Entscheidung der Realisierung einer Spezifikation in Hardware oder Software begünstigen.

a) Was sind die wichtigsten Kriterien im Falle einer Zielarchitektur für
   - Steuerung einer Ampel
   - Mobiltelefon
   - System zur Bildverarbeitung
   - Kraftwerksüberwachung

b) Welche unterschiedlichen Optimierungskriterien machen die Entscheidung Hardware/Software aus im Falle einer
   - Ein-Chip HW/SW-Lösung
   - Board-Level HW/SW-Lösung

   Geben Sie an, für welchen Anwendungs- bzw. Aufgabenbereich die eine bzw. die andere Lösungsvariante Vorteile bzw. Nachteile hat.

c) Für welche Anwendungsbereiche erscheinen Ihnen ASIPs (Prozessoren mit anwendungsspezifischem Instruktionssatz) sinnvoll? Versuchen Sie, für diese Anwendungsbereiche Charakteristika einer optimalen Architektur zu extrahieren und vergleichen Sie diese Anforderungen mit Realisierungen Ihnen bekannter ASIP-Architekturen (z.B. DSPs - Digitale Signalprozessoren).

Kapitel 2: Zielarchitekturen

Aufgabe 2.01: RISC/CISC

Diskutieren Sie die Vor- und Nachteile von RISC bzw. CISC bezüglich

a) Codegröße
b) Pipelining
c) Steuerwerk
d) Compilerunterstützung

Aufgabe 2.02: RISC/CISC

Gegeben ist folgender Assemblercode:

```
a:   word 10
b:   word 20
c:   word

_xyz:
     mov r0, [a]
     add r0, [b]
     mul r0, [b]
     mov [c], r0
```

a) Handelt es sich bei dem Prozessor, der den Assemblercode ausführen kann, um eine RISC oder CISC Maschine? Begründen Sie Ihre Antwort.
b) Was macht der Code?
c) Portieren Sie den Code auf den jeweils anderen Maschinentypen.

Aufgabe 2.03: Pipelining

In der Vorlesung wurde die 5-stufige DLX Pipeline (IF, ID, EX, MEM, WB) behandelt.

a) Handelt es sich um eine RISC oder CISC Pipeline?
b) Erklären Sie den Unterschied zwischen der WB und MEM Stufe?
c) Wie viele Takte muss ein Befehl innerhalb der Pipeline angehalten werden, der das Ergebnis des vorherigen Befehls als Eingabe verwendet?
d) In welcher Pipelinestufe würde dieser Befehl angehalten werden, wenn die Pipeline über eine automatisch Konfliktbehandlung verfügt?

Aufgabe 2.04: Pipelining

Auf einem Prozessor mit einer 5-stufigen DLX Pipeline wird das folgende Programm ausgeführt:

```
(1) load R2,[Var1]       ;Lade den Inhalt von Var1
(2) load R1,[Var2]       ;Lade den Inhalt von Var2
(3) add R1,R2            ;R1 = R1 + R2
(4) load R3,[Var3]       ;Lade den Inhalt von Var3
(5) load R4,[Var4]       ;Lade den Inhalt von Var4
(6) sub R4,R3            ;R4 = R4 - R3
(7) add R1,R4            ;R1 = R1 + R4
(8) store [Res],R1       ;Speichere Resultat nach Res
```

a) Markieren Sie alle Datenabhängigkeiten im ASM Code.
b) Skizzieren Sie den Ablauf der Pipeline. Markieren Sie evtl. auftretende Pausen. Für die Zugriffe auf Variablen können Sie davon ausgehen, dass diese über einen Cache rechtzeitig zur Verfügung stehen. Die Register werden in der Write-Back Phase in der ersten Takthälfte geschrieben und in der ID Phase in der zweiten Takthälfte geladen. Die Pipeline verfügt über einen automatischen Stall-Mechanismus, der die Ausführung so lange anhält bis der Konflikt aufgelöst ist.

#	IF	ID	EX	ME	WB
1	load R2				
2	load R1	load R2			
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

Tabelle 2.1: Programmausführung in der Pipeline

c) Gehen Sie nun davon aus, dass die Pipeline über keine automatische Konflikterkennung verfügt. Füllen Sie den ASM-Code mit NOP Befehlen auf, sodass das Programm korrekt ausgeführt wird.
d) Gehen Sie nun davon aus, dass die Pipeline über eine Forwarding-Technik verfügt, so dass ein berechneter oder geladener Wert direkt an die EX-Phase weitergeleitet werden kann. Markieren Sie in der Lösung von Teilaufgabe c) sämtliche NOPs, die damit eingespart werden können.

Aufgabe 2.05: Superskalarität

Gegeben ist eine superskalare Out-of-Order Pipeline, wie sie in der Vorlesung vorgestellt wurde.

a) Was macht der Dispatcher?
b) In welchem Abschnitt der Pipeline findet eine Out-of-Order (OoO) Ausführung und in welchem Abschnitt eine In-Order Ausführung statt?
c) Welche Aufgaben erfüllt die Completion Unit?
d) Wie werden Pipelinekonflikte durch Namensabhängigkeiten verhindert?
e) Wie werden Pipelinekonflikte durch Datenabhängigkeiten verhindert?
f) Warum lohnt sich der ganze Aufwand für eine OoO-Pipeline überhaupt?

Aufgabe 2.06: VLIW

Das folgende Codestück soll auf einem VLIW Prozessor mit drei parallelen Ausführungseinheiten (Execution Units) laufen. Geben eine möglichst effiziente Befehlsverteilung an, wie sie der Compiler durchführen würde. Die Befehle können beliebig umsortiert werden, so lange die Funktionalität identisch bleibt.
Man kann annehmen, dass alle Befehle innerhalb eines Taktes berechnet werden. Ein freies Kästchen ist gleichbedeutend mit einem NOP-Befehl.

```
(1)  add  r1, r2, r3          ; r1 = r2 + r3
(2)  sub  r5, r3, r5          ; r5 = r3 – r5
(3)  ld   r3, [r1]            ; Lade r3 mit [r1]
(4)  mul  r3, r3, r3          ; r3 = r3 * r3
(5)  st   [r5], r3            ; Speichere r3 nach [r5]
(6)  ld   r9, [r7]            ; Lade r9 mit [r7]
(7)  ld   r11, [r12]          ; Lade r11 mit [r12]
(8)  add  r11, r11, r12       ; r11 = r11 + r12
(9)  mul  r11, r11, r9        ; r11 = r11 * r9
(10) st   [r12], r11          ; Speichere r11 nach [r12]
```

a) Nehmen Sie an, dass der Prozessor über drei Ausführungseinheiten verfügt, die jeweils alle Befehle ausführen können.

Slot 1	Slot 2	Slot 3

Tabelle 2.2: VLIW Scheduling

b) Nehmen Sie nun an, dass die ersten beiden Ausführungseinheiten arithmetisch-logische Befehle (add, sub, mul) ausführen können und die dritte für Load/Store Befehle zuständig ist.

Slot 1 (ALU)	Slot 2 (ALU)	Slot 3 (LS)

Tabelle 2.3: VLIW Scheduling mit HW Constraints

Aufgabe 2.07: SIMD

a) Zwei MMX (64-Bit SIMD) Register mm0 und mm1 enthalten die folgenden Werte. Tragen Sie das Ergebnis der beiden MMX-Befehle PADDUSB und PADDW in die Tabelle ein.

PADDUSB führt eine unsigned (U) Addition (ADD) mit Saturation (S) auf Packed-Bytes (B) aus.
PADDW führt eine Addition (ADD) auf Packed-Words (W) aus.

Saturation bedeutet, dass bei einem Überlauf das Ergebnis auf den höchsten oder niedrigsten möglichen Wert gesetzt wird. Bspw. hat ein Unsigned Byte einen Wertebereich von 0x00 – 0xFF. Eine Addition von 0xA0 + 0xA0 ergibt mit Saturation das Ergebnis 0xFF anstatt 0x40.

mm0	10 20 00 80 55 33 FF FF
mm1	01 20 00 80 33 55 00 01
PADDW mm0, mm1	
PADDUSB mm0, mm1	

Tabelle 2.4: SIMD Instruktionen

b) Gegeben ist folgendes Fragment eines C-Programmes:

```
int  i;
char a[8], b[8];    // char = 8-Bit signed integer
for (i = 0; i < 8; i++) {
     if (a[i] > b[i]) {
          b[i] = i;
     }
}
```

Realisieren Sie den C-Code mittels SIMD Assemblerbefehle. Die Registerbreite der SIMD Befehle beträgt 64-Bit und sie können die Register mittels folgender Befehle als Packed-Byte ansprechen. Die Byte-Order (Little oder Big-Endian) sowie Signed/Unsigned kann vernachlässigt werden. Sie können das Ergebnis wahlweise als Assemblerprogramm oder Datenflussgraphen darstellen.

PMOV mm0, 0x0123456789abcdef	Lädt den hexadezimalen Wert 0x0123456789abcdef in Register mm0
POR mm0, mm1, mm2	Bitweises ODER (mm0 = mm1 | mm2)
PAND mm0, mm1, mm2	Bitweises UND (mm0 = mm1 & mm2)
PNEG mm0, mm1	Bitweise Negation
PCMPGTB mm0, mm1, mm2	Packed-Byte Vergleich zwischen mm1 und mm2. Wenn Packed-Byte in mm1 > mm2 ist, wird in mm0 als Ergebnis 0xFF abgelegt, ansonsten 0x00. Beispiel: MM1= 04 70 45 67 09 0A 00 04h MM2= 03 71 00 20 01 0A 02 0Fh PCMPGTB MM0, MM1, MM2 -> MM0= FF 00 FF FF 00 00 00 00h

Tabelle 2.5: SIMD Instruktionen

```
PMOV      mm0, 0x000305080C0D0405  ; a
PMOV      mm1, 0x010703090A0B0200  ; b
```

Aufgabe 2.08: Cache

Gegeben sei ein Cache mit 4 Cache-Zeilen und 4 Bytes pro Cache-Zeile. Zum Vergleich ist der Cache als Direct-Mapped, 2-Wege Assoziativ und Voll-Assoziativ vorhanden. Die Breite der Speicheradresse beträgt 8 Bits.

a) Wie ist Speicheradresse bei den Verschiedenen Cache-Typen aufgeteilt? Pro Bit der Speicheradresse ist ein Kästchen vorhanden. Jedes Bit kann entweder für den Tag (t), Index (i) oder Byteoffset (o) verwendet werden:

Direct-Mapped								
2-Wege Assoziativ								
Voll-Assoziativ								

b) Es werden jetzt nacheinander 6 Lesezugriffe auf den jeweiligen Cache-Typen ausgeführt. Die Speicheradressen der Lesezugriffe sind in Dualform in der ersten Spalte vorhanden. In der 2. Spalte (Hit?) wird ein Cache-Hit markiert. Die restlichen Spalten sind für den Inhalt der 4 Cache-Zeilen vorgesehen. Als Inhalt genügt es, den Tag der Cache-Zeile einzutragen. Als Verdrängungsstrategie wird LRU (Least Recently Used) verwendet.

Adresse	Hit?	Line 0	Line 1	Line 2	Line 3
1000 0011	-	1000			
1000 0100					
1000 1001					
1000 0010					
1001 0010					
1000 1000					

Tabelle 2.6: Direct-mapped cache

Adresse	Hit?	Line 0	Line 1	Line 2	Line 3
1000 0011	-	10000			
1000 0100					
1000 1001					
1000 0010					
1001 0010					
1000 1000					

Tabelle 2.7: 2-Wege assoziativer Cache

Adresse	Hit?	Line 0	Line 1	Line 2	Line 3
1000 0011	-	100000			
1000 0100					
1000 1001					
1000 0010					
1001 0010					
1000 1000					

Tabelle 2.8: Voll-assoziativer Cache

c) Wieviel Speicherplatz verbraucht ein Cache insgesamt? Geben sie ihre Antwort in Abhängigkeit der ihnen bekannten Cache-Parameter an.

d) Eine Matrixmultiplikation wird üblicherweise mit Hilfe von zwei verschachtelten Schleifen implementiert, die über alle Elemente der Matrix laufen (x,y). Wieso gibt es auf einem GPP mit Datencache einen großen Laufzeitunterschied, je nachdem über welche Dimension zuerst (d.h. in der äußeren Schleife) iteriert wird?

Aufgabe 2.09: DSP

Im Folgenden ist eine Implementierung eines FIR-Filters als C-Code gegeben:

```
sum = 0.0;
for (i=0; i<N; i++)
     sum = sum + a[i]*b[i];
```

Ein solcher Code kann auf einem DSP unter Ausnutzung der für solche Domänen angepassten Spezialhardware sehr effizient ausgeführt werden. Im Folgenden ist der Code zu sehen, nachdem er optimiert in Assembler übersetzt wurde:

```
RPTS   N -1                ;Repeat next instruction.
MPYF3  *AR0++%,*AR1++%,R0  ;Multiply…
|| ADDF3  R0,R2,R2         ;… and accumulate.
ADDF   R0,R2               ;Last product accumulated.
```

Der Code nutzt für die Ausführung viele Optimierungen eines DSPs: Es werden zero-overhead loops (RPTS), circular addressing (*AR0++%) und Multiply-Accumulate (||) eingesetzt. Die gesamte MAC-Instruktion wird dabei pipelined ausgeführt und kann daher pro Takt eine Iteration berechnen und aufsummieren.

a) Wie viele Takte benötigt der optimierte Assembler Code für N = 10?
b) Schreiben sie nun das Programm in reinem Assembler ohne alle Spezialbefehle und DSP-Optimierungen. Nutzen sie nur die im Folgenden angegebenen Instruktionen. Wie viele zusätzliche Takte braucht das Programm hierdurch für N = 10?

Für die Aufgabe stehen ihnen die folgenden Assemblerbefehle zur Verfügung. Als Quelle „src“ können sowohl Register als auch konstante Werte direkt genutzt werden. Weiterhin können sie davon ausgehen, dass alle Register bereits mit dem Wert 0 initialisiert sind und die Startadressen der beiden Eingabearrays a[] und b[] bereits in den Registern R1 bzw R2 liegen.

LDF	src, dst	Lädt den Gleitkommawert an der Adresse src in das Register dst
ADDF3	src2, src1, dst	Addiert die Gleitkommawerte aus den Registern src1 und src2 und speichert das Ergebnis in dst
ADDI3	src2, src1, dst	Addiert die Integer Werte aus den Registern src1 und src2 und speichert das Ergebnis in dst
MPYF3	src2, src1, dst	Multipliziert die Gleitkommawerte aus den Registern src1 und src2 und speichert das Ergebnis in dst
CMPI3	src2, src1	Berechnet src1-src2 und setzt das Status bit „Z" auf 1, falls das Ergebnis der Subtraktion 0 ist
BZ	src	Bedingter Sprung zu einem Label oder um einen relativen Wert, sofern im Statusbit „Z" eine 1 steht
BR	src	Unbedingter Sprung zu einem Label oder um einen relativen Wert

Aufgabe 2.10: FPGA

Gegeben ist eine einfache Version eines Xilinx FPGAs bestehend aus CLBs und Switch-Matrizen. Ein CLB enthalt eine LUT mit 8 Zeilen und 3 Eingängen.

a) Realisieren Sie einen Volladdierer mittels zwei CLBs.

Ein Volladdierer besitzt drei Eingänge (x, y, c_{in}) und zwei Ausgänge (c_{out}, s). Es werden sämtliche Eingänge addiert und das Ergebnis als 2-Bit Zahl auf die Ausgänge gelegt. So ist s=1 wenn die Summe aller Eingänge ungerade ist und c_{out} =1 wenn die Summe aller Eingänge >= 2 ist.

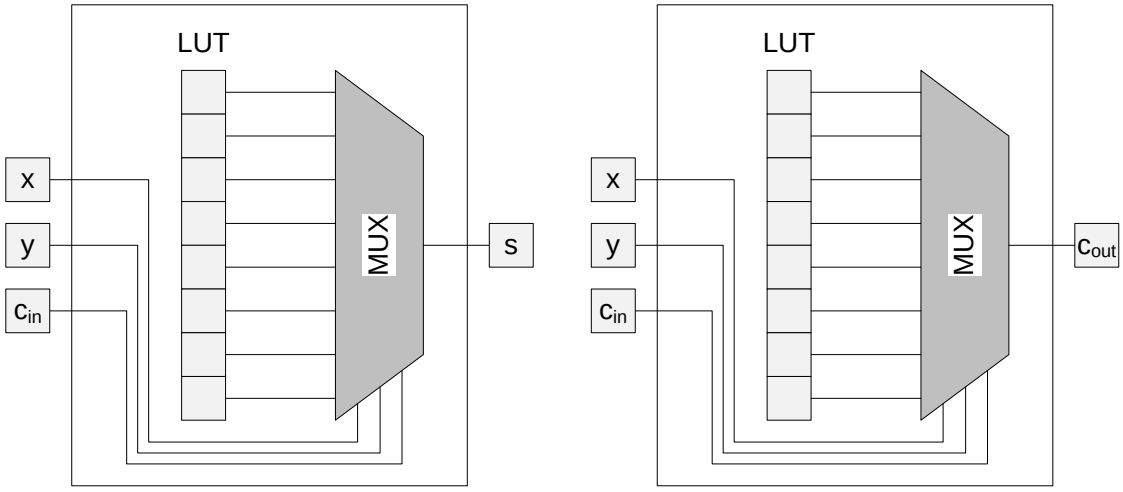
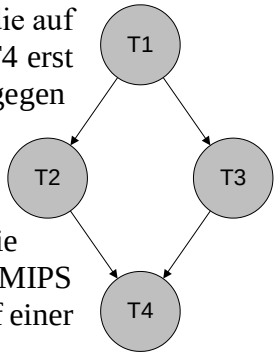


Figure 2.1: LUT Realisierung für einen Volladdierer

Kapitel 3: Schätzung der Entwurfsqualität

Aufgabe 3.01: Design Space, Pareto Punkte

Die rechte Abbildung zeigt einen Task-Graphen mit vier Tasks (T1 … T4), die auf verschiedenen Komponenten ausgeführt werden können. Dabei kann z.B. T4 erst ausgeführt werden, wenn T2 und T3 abgearbeitet wurden. T2 und T3 hingegen können parallel ausgeführt werden.



Die folgende Tabelle zeigt alle verfügbaren Komponenten (MIPS, DSP, FPGA, ASIC), die maximale Anzahl einer Komponente, deren Kosten und die Ausführungsgeschwindigkeit der vier Tasks. Zum Beispiel kostet der MIPS Prozessor 200 Einheiten und kann Task T1 in 5 und T4 in 2ms ausführen. Auf einer Komponente kann jeweils nur ein Task zur gleichen Zeit ausgeführt werden.

Komponente	**Anzahl**	**Kosten [€]**	**Ausführungszeit [ms]**			
			T1	**T2**	**T3**	**T4**
MIPS	1	200	5	-	-	2
DSP	1	100	-	20	18	5
FPGA	1	250	-	12	10	-
ASIC	1	400	-	-	0,8	-

Tabelle 3.1: Eigenschaften der verfügbaren Komponenten

a) Vervollständigen Sie die Ausführungszeit und Kosten der folgenden Tabelle. Sie zeigt sämtliche Realisierungsmöglichkeiten, welcher Task auf welchem Prozessor ausgeführt werden kann.

#	**Tasks**				**Ausführungszeit [ms]**	**Kosten**
	T1	**T2**	**T3**	**T4**		
1	MIPS	DSP	DSP	MIPS	5 + 20 + 18 + 2 = 45	200 + 100 = 300
2	MIPS	DSP	DSP	DSP		
3	MIPS	DSP	FPGA	MIPS		
4	MIPS	DSP	FPGA	DSP		
5	MIPS	DSP	ASIC	MIPS		
6	MIPS	DSP	ASIC	DSP		
7	MIPS	FPGA	DSP	MIPS		
8	MIPS	FPGA	DSP	DSP		
9	MIPS	FPGA	FPGA	MIPS		
10	MIPS	FPGA	FPGA	DSP		
11	MIPS	FPGA	ASIC	MIPS		
12	MIPS	FPGA	ASIC	DSP		

Tabelle 3.2: Ausführungszeit und Kosten für sämtliche Realisierungsmöglichkeiten

b) Tragen Sie die Lösungen aus Aufgabe a) in folgendes Kosten-Zeitdiagramm ein und markieren Sie die Pareto-Punkte.

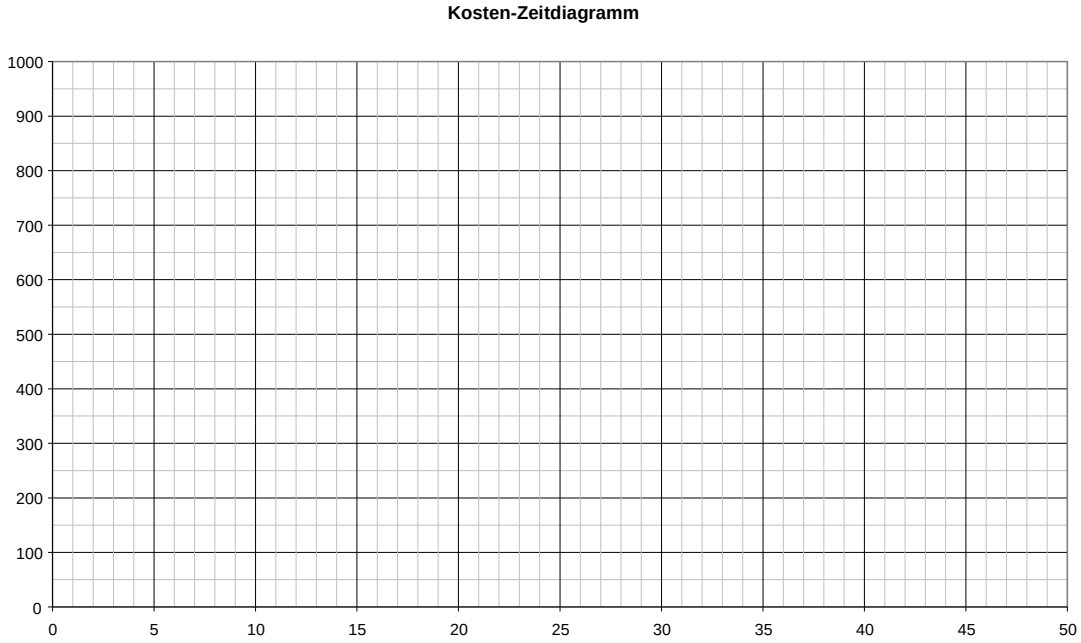


Figure 3.1: Kosten-Zeitdiagramm

c) Was passiert, wenn die Anzahl der Komponenten nicht auf 1 beschränkt ist? Welche zusätzlichen Design-Möglichkeiten ergeben sich? Verändert sich die Menge der Pareto-Punkte?

#	Tasks				Ausführungszeit [ms]	Kosten
	T1	T2	T3	T4		
13						
14						
15						
16						

Tabelle 3.3: Ausführungszeit und Kosten für zusätzliche Realisierungsmöglichkeiten

Aufgabe 3.02: Exaktheit & Treue

In folgender Tabelle sind für vier Entwurfspunkte Metriken und Entwurfsqualität dargestellt, und zwar geschätzte Werte E(D) sowie gemessene Werte M(D).

Entwurfspunkt	E(D)	M(D)
W	112	100
X	128	137
Y	139	121
Z	205	132

Tabelle 3.4: Geschätzte E(D) und gemessene M(D) Werte

a) Bestimmen Sie die Exaktheit (A) des Entwurfspunktes W.
b) Bestimmen Sie die Treue (F) des Schätzverfahrens.

Aufgabe 3.03: Taktschlupf

Gegeben ist eine Menge von funktionalen Einheiten v_k. Die mögliche Taktperiode der Zieltechnologie liegt zwischen 20 und 50 ns.

Funktionale Einheit	k	delay(v_k) [ns]	occ(v_k)
MUL	1	135	9
ADD	2	45	10
SUB	3	55	1

Tabelle 3.5: Eigenschaften der funktionalen Einheiten

a) Was ist der Taktschlupf?
b) Was bringt die Taktschlupfminimierung?
c) Berechnen Sie den *slack* für alle funktionalen Einheiten bei einer Taktperiode von 20ns.
d) Berechnen Sie den mittleren Schlupf (*average slack*) für eine Taktperiode von 20ns.
e) Raten/Überlegen Sie, welche Taktperiode den niedrigsten mittleren Schlupf hat.

Aufgabe 3.04: WCET

Gegeben ist folgendes C-Fragment:

```
weight=volume*density;

if (weight<100) {
     max_index=10;
} else {
     max_index=5;
}

index=1;
additional_weight=0;

do {
     additional_weight+=20;
     index++;
} while (index<=max_index)

weight+=additional_weight;
```

a) Transformieren Sie das C-Fragment in einen Kontrollflussgraphen (CFG).
b) Die WCET (ohne Caches) soll mittels ILP bestimmt werden. Stellen Sie hierfür die Flussgleichungen zur Modellierung der strukturellen Nebenbedingungen (Constraints) auf.
c) Stellen Sie die funktionalen Nebenbedingungen auf.
d) Geben Sie die Zielfunktion an, die durch ILP maximiert wird.

Aufgabe 3.05: Erweiterung WCET

Es wird das erweiterte WCET-Modell zur ILP-Formulierung von Caches betrachtet.

a) Welche Art von Cache kann mit dem Modell modelliert werden?
b) Was ist ein L-Block?
c) Wie lautet die ILP-Zielfunktion der erweiterten WCET Formulierung? Erklären Sie die einzelnen Variablen.
d) Was repräsentiert eine Kante des Cachekonfliktgraphen im CFG? Was darf diese nicht beinhalten?

Aufgabe 3.06: Erweiterung WCET

Gegeben ist ein CFG mit vier Basis-Blöcken bestehend aus jeweils zwei L-Blöcken. Auf der rechten Seite sind die Cache-Zeilen der L-Blöcke angegeben. So verwenden z.B. B2,1 und B3,1 beide die zweiten Cache-Zeile und stehen somit im Konflikt.

a) Formulieren Sie die strukturellen Constraints aus den Flussgleichungen des CFGs.
b) Formulieren Sie die allgemeinen Bedingungen, die für das erweiterte WCET ILP-Modell notwendig sind
c) Zeichnen Sie den Cachekonfliktgraphen für die zweite Cache-Zeile.
d) Geben Sie die aus dem CCG ableitbaren strukturellen Constraints der zweiten Cache-Zeile an.
e) Geben Sie die strukturellen Constraints für die Cache-Zeilen 0, 1, 4 und 5 an.

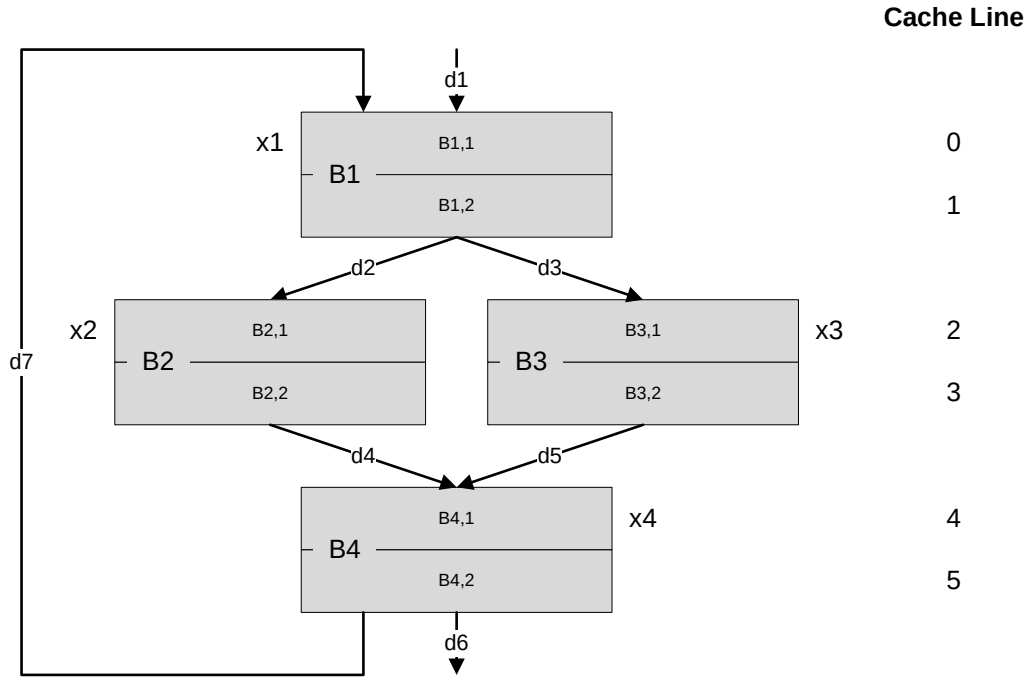


Figure 3.2: Kontrollflussgraph (CFG) mit Cache-Line Mapping

Aufgabe 3.07: Profiling

Gegeben ist folgender Kontrollflussgraph mit Gewichten:

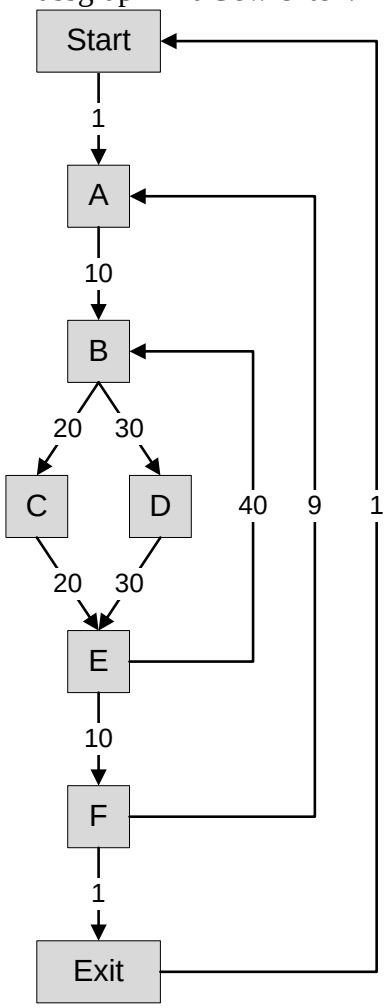


Figure 3.3: Kontrollflussgraph

a) Was ist der Unterschied zwischen WCET und Profiling? Welches findet zur Laufzeit bzw. Compilezeit statt?
b) Welches Graphenproblem wurde in der Vorlesung bei Profiling vorgestellt und gelöst?
c) Bestimmen Sie den maximal aufspannenden Baum (MAB) des Kontrollflussgraphen. Sie können in der Zeichnung die Kanten, die zum MAB gehören, mit einem X markieren. Der Kruskal-Algorithmus zum Berechnen des MAB lautet wie folgt:

Führe den folgenden Schritt so oft wie möglich aus: Wähle unter den noch nicht ausgewählten Kanten von G (dem Graphen) die längste/schwerste Kante, die mit den schon gewählten Kanten keinen Kreis bildet.

d) Auf welchen Kanten werden jetzt die Zähler für das Profiling platziert? Markieren Sie diese Kanten.
e) Warum wird in diesem Verfahren der Maximale und nicht der Minimale Aufspannende Baum verwendet?
f) Wie können die restlichen Zählerwerte aus den platzierten Zählern bestimmt werden?

Kapitel 4: Hardware/Software Partitionierungsverfahren

Aufgabe 4.01: Hierarchical Clustering

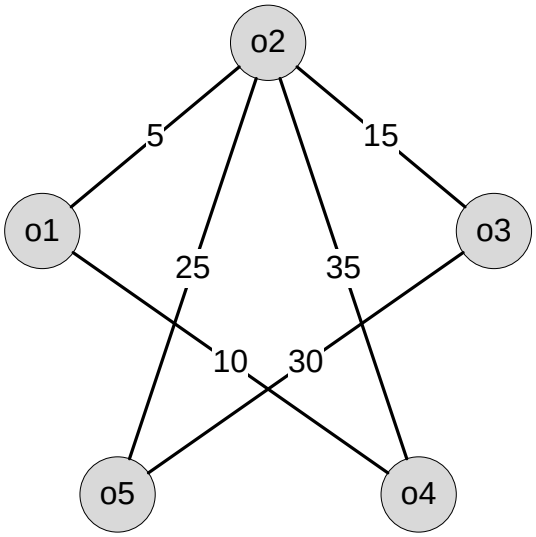


Figure 4.1: HC Partition

Führen Sie das Hierarchical Clustering Verfahren anhand des gegebenen Graphen durch. Bei der Verschmelzung der Knoten soll die Maximum-Metrik angewandt werden, d.h. das Kantengewicht einer neuen Kante entspricht dem Maximum der vorherigen Kantengewichten.

Aufgabe 4.02: Tabu-Search

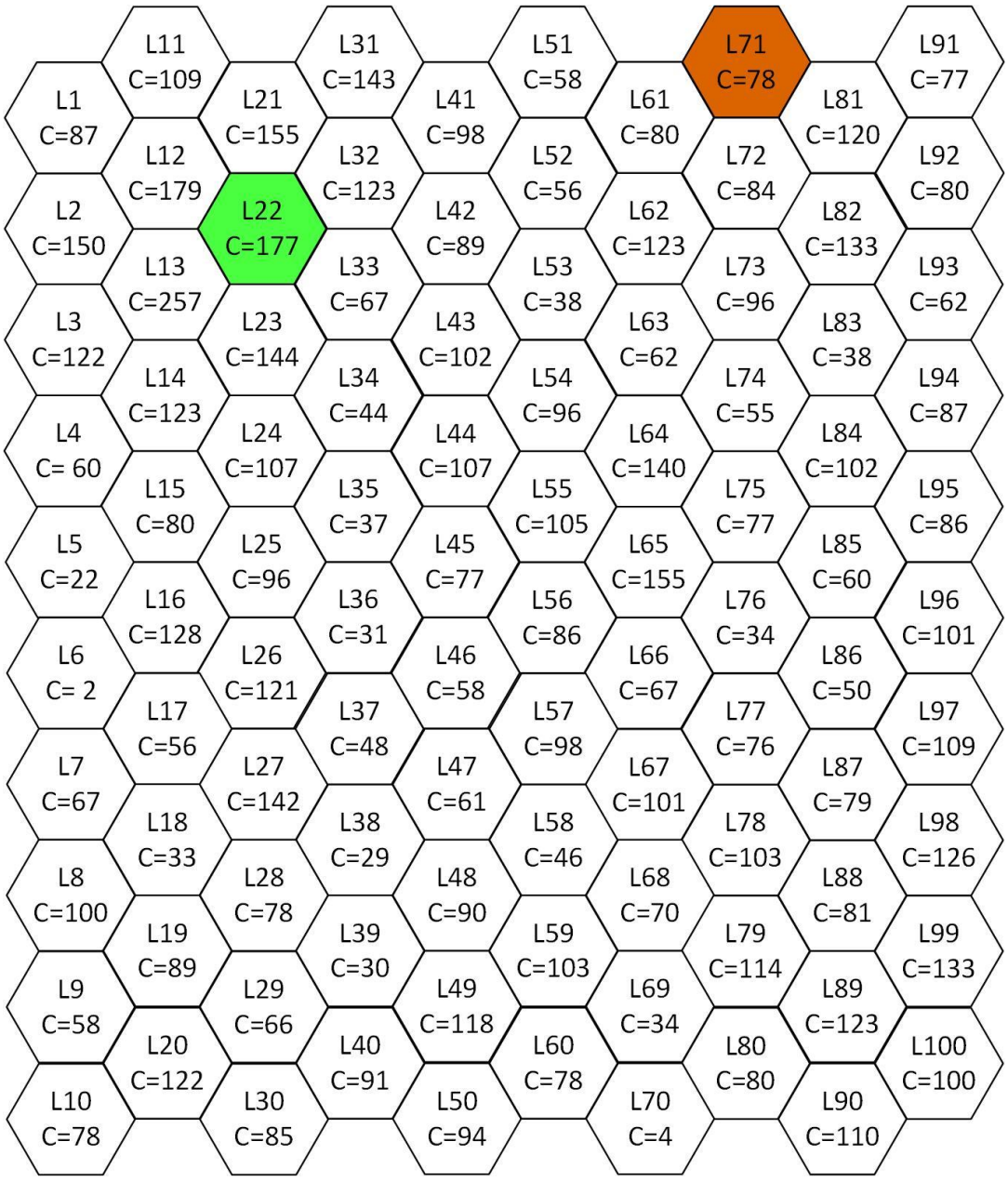


Figure 4.4: Tabu-Search

a) Führen Sie den Tabu-Search Algorithmus auf der in Figure 4.4. gegebenen Lösungsmenge jeweils für die zwei unterschiedlichen Startpunkte „L22“ und „L71“ aus. Der Algorithmus soll mit der Stop-Condition $C < 5$ beendet werden und verfügt über eine Tabu-Liste der Länge 8.
b) Kann das Verfahren Hill-Climbing?
c) Findet das Verfahren die global beste Lösung?

Aufgabe 4.03: Kernighan-Lin

Gegeben sei der unten abgebildete ungerichtete Graph G=(V, E) mit 6 Knoten und 8 Kanten. Für die Kanten ist jeweils eine Kostenfunktion definiert, wobei in diesem Fall die Kosten jeder Kante gleich 1 sind. Die erste Partitionierung ist durch die gestrichelte Linie gegeben.

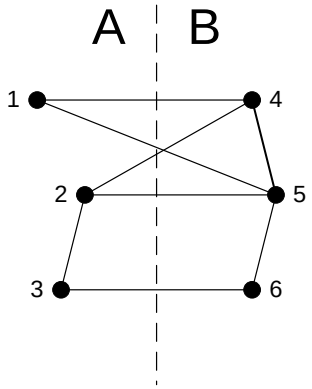


Figure 4.2: KL Partition

a) Berechnen Sie den anfänglichen Gain sämtlicher Knoten und tragen Sie diesen für Schritt 1 in der Gain-Tabelle ein.

Knoten	**Gain**		
	Schritt 1	**Schritt 2**	**Schritt 3**
1			
2			
3			
4			
5			
6			

Tabelle 4.1: Gain

Schritt	**Knotenpaar**	**Change**	**Cutsize**
0	-	0	5
1			
2			
3			

Tabelle 4.2: Ergebnis / tatsächliche Vertauschungen

b) Wie berechnet sich der Gewinn aus dem Vertauschen zweier Knoten i und j?

c) Vertauschen Sie sämtliche Knoten virtuell und berechnen Sie den jeweiligen Gewinn in der folgenden Tabelle. Markieren Sie das Knotenpaar, das am Ende wirklich vertauscht wird und tragen Sie das Ergebnis bei Schritt 1 in der Ergebnistabelle ein.

i	j	Gewinn
1	4	
2	4	
3	4	
1	5	
2	5	
3	5	
1	6	
2	6	
3	6	

Tabelle 4.3: Gewinn Schritt 1

d) Durch das Vertauschen ändern sich die Gain-Werte. Ergänzen Sie entsprechend den Gain von Schritt 2 in Tabelle 4.1.

e) Für einen kompletten Durchlauf von KL müssen Teilaufgabe c) und d) jeweils für Schritt 2 und 3 wiederholt werden.

f) Welche Vertauschungen werden am Ende eines kompletten Durchlaufs durchgeführt?

i	j	Gewinn

Tabelle 4.4: Gewinn Schritt 2

i	j	Gewinn

Tabelle 4.5: Gewinn Schritt 3

Aufgabe 4.04: Fiduccia-Mattheyses

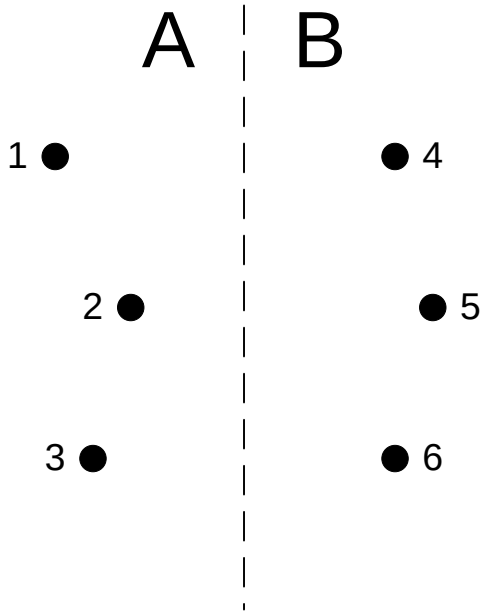


Figure 4.3: FM Partition

a) Fiduccia-Mattheyses verwendet im Gegensatz zu Kernighan-Lin Hyperkanten in einem Hypergraphen. Wandeln Sie den Graphen aus Aufgabe 4.03 in einen Hypergraphen um. Wie groß ist der Cutsize?
b) Welche Netze werden in diesem Graphen als kritische Netze bezeichnet?
c) Berechnen Sie den Gain g(i) sämtlicher Zellen.
d) Welche Lösungen findet FM, die KL nicht findet?

Zelle	Gain g(i)
1	
2	
3	
4	
5	
6	

Tabelle 4.6: Gain sämtlicher Zellen