informatiCup 2008 • Aufgabe 2

DOPI for CMOS

Einführung

CMOS-Bausteine (Complementary Metal Oxide Semiconductor, dt. komplementärer Metall-Oxid-Halbleiter) sind Halbleiterbauelemente, die für integrierte Schaltkreise benutzt werden. Der große Vorteil dieser Logikfamilie ist, dass Strom (von der Versorgungsspannung zur Masse) nur im Umschaltmoment fließt. Die Stromaufnahme bzw. die Verlustleistung einer CMOS-Schaltung ist also – abgesehen vom wesentlich kleineren Kriechstrom – hauptsächlich von der Umschalthäufigkeit (Taktfrequenz) abhängig.

Unter den Annahmen, dass 1) alle Gates (dt. Logikgatter) die gleiche Lastkapazität C haben 2) an allen Gates die gleiche Versorgungsspannung V_{dd} anliegt und 3) alle Gates mit der gleichen Taktfrequenz f betrieben werden, ist die elektrische Leistungsaufnahme eines CMOS-Schaltkreises

$$P = C \times {V_{dd}}^2 \times f \times n,$$

wobei n die Anzahl an Transitionen (Umschaltung des Eingangssignals eines Gatters von 0 nach 1 und 1 nach 0) pro Taktzyklus im gesamten Schaltkreis ist. Durch die Verwendung moderner Fertigungstechnologien und geschickter Schaltkreisentwürfe sind die Versorgungsspannung V_{dd} und die Lastkapazität C in integrierten Schaltungen bereits optimiert. Auch die Frequenz f ist in den meisten getakteten Schaltungen konstant. Die einzige Möglichkeit, die Leistungsaufnahme einer Schaltung noch weiter zu senken, ist, die Anzahl der Transitionen zu verringern. Hierbei kann man sich folgenden Sachverhalt zu Nutze machen.

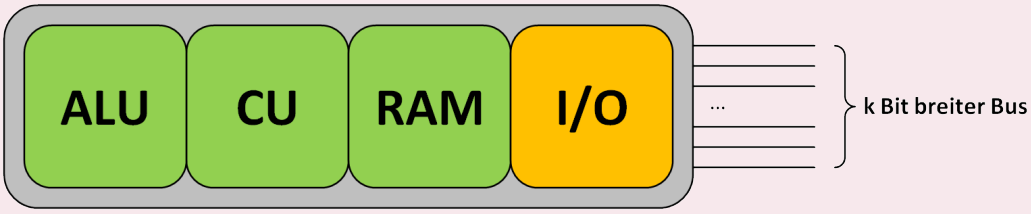

Abbildung 1: Aufbau einer CPU

Eine CPU (Central Processing Unit, dt. Hauptprozessor) besteht, wie in der obigen Abbildung dargestellt, aus der ALU (Arithmetic Logic Unit, dt. Rechenwerk), der CU (Control Unit, dt. Steuerwerk), dem RAM (Random Access Memory, dt. Hauptspeicher) und einem I/O-Teil (Ein-/Ausgabe-Teil), der den Bus ansteuert. Tatsächlich haben nicht alle Gatter in solch einer CPU die gleiche Lastkapazität. Üblicherweise sind die Lastkapazitäten – und damit die Leistungsaufnahme – des I/O-Teils einer Schaltung um Größenordnungen höher als die des internen Teils wie z.B. dem Rechen- und Steuerwerk eines Mikroprozessors ($C_{I/O} \gg C_{intern}$). Typisch für heutige Schaltkreise ist außerdem, dass der I/O- und der interne Teil einer Schaltung in etwa die gleiche Leistung aufnehmen. Entsprechend folgt, dass $n_{intern} \gg n_{I/O}$.

Das Einsparen einiger weniger Transitionen beim I/O kann der Leistungsaufnahme einiger tausend Transitionen im internen Schaltungsteil entsprechen! Transitionen des I/O-Teils einer Schaltung können eingespart werden, indem die I/O-Daten *kodiert* werden. Für die Berechnung dieser Kodierung können, wie das folgende Beispiel zeigt, problemlos $n_{I/O} \times \log n_{I/O}$ Transitionen des internen Schaltungsteils aufgewendet werden.

Aus $V_{dd} = 1V$, $f = 1MHz$, $C_{intern} = 10$, $n_{intern} = 1000$, $C_{I/O} = 1000$ und $n_{I/O} = 10$ folgt

$$P = 10 \times 1 \times 1 \times 1000 + 1000 \times 1 \times 1 \times 10 = 20000.$$

Wird durch Kodierung $n_{I/O}$ halbiert und n_{intern} um 35 (d.h. $10 \log 10$) vergrößert so ist

$$P_{kodiert} = 10 \times 1 \times 1 \times 1035 + 1000 \times 1 \times 1 \times 5 = 15350 \approx \tfrac{3}{4}P.$$

Wie könnte nun solch eine Kodierung aussehen? Wieder kann man einen Sachverhalt aus der Praxis ausnutzen. In vielen Anwendungsfällen, wo große Datenmengen übertragen werden, spielt die Reihenfolge in der die Daten übertragen werden keine Rolle. Ein Beispiel hierfür ist das Übertragen von Paketen im Internet. Entprechend ist es zulässig die zu übertragenden Datenwörter vor der Übertragung in eine Reihenfolge zu bringen, für die die einzelnen Leitungen des Datenbusses möglichst selten umgeschaltet werden müssen.

Die Kodierung, die hier verwendet werden soll, ist das sogenannte *Data Ordering with Inversion*. Das entsprechende NP-vollständige *Data Ordering Problem with Inversion* lautet wie folgt.

Gegeben ist eine Menge von n Wörten w_1, w_2,..., w_n gleicher Länge k, die ohne Beachtung der ursprünglichen Reihenfolge über einen Bus der Breite k übertragen werden sollen. Zusätzlich steht eine Leitung für ein Invertierungssignal zur Verfügung. Bestimme 1) für jedes Wort ob es invertiert oder nicht invertiert übertragen werden soll und 2) die Reihenfolge, in der die Wörter über den Bus übertragen werden sollen, so dass die Anzahl der Transitionen auf dem Bus minimal ist (eine Umschaltung des Invertierungssignals zählt ebenfalls als Transition).

Unter der *Invertierung* eines Binärwortes versteht man das Ersetzen jedes einzelnen Bits durch sein Komplement ($0 \rightarrow 1$, $1 \rightarrow 0$).

Die Anzahl der Transitionen beim Übertragen der Wörter w_1 und w_2 der Länge k entspricht der *Hammingdistanz* $d(w_1, w_2) = \sum_{i=1}^{k} w_{1_i} \oplus w_{2_i}$ der beiden Wörter ($\oplus$ ist die logische Exklusiv-Oder-Verknüpfung: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$).

Aufgabenstellung

Entwickeln Sie für die erste Runde ein Programm, das für eine gegebene Menge an Binärwörtern das *Data Ordering Problem with Inversion* löst. Beachten Sie folgende Hinweise.

(a) Die Eingabedaten für Ihr Programm sind als Textdateien in folgendem Format gegeben. In der ersten Zeile steht die Anzahl der Wörter, in der zweiten Zeile deren Länge (alle Wörter sind gleich lang). Es folgen die Binärwörter kodiert als Folge von mindestens einer und höchstens 1000 Nullen und Einsen.

```
1 3
2 4
3 0000
4 1100
5 0001
```

(b) Geben Sie Ihre Lösung als Textdatei wie folgt aus. In der ersten Zeile steht die Anzahl der Transitionen, wenn die Wörter in der Originalreihenfolge übertragen werden, gefolgt von der Anzahl der Transitionen für die kodierte Übertragung. Es folgen die Datenwörter in der berechneten Reihenfolge. Jedem Datenwort vorangestellt ist ein 'S' oder ein 'I', wenn das Datenwort unverändert bzw. invertiert übertragen wird.

```
1 5
2 3
3 S0000
4 S0001
5 I0011
```

(c) Implementieren Sie für das Data Ordering Problem with Inversion eine exakte Lösung, die zu einer minimalen Anzahl Transitionen führt.

(d) Implementieren Sie zwei Heuristiken Ihrer Wahl, deren Berechnung möglichst schnell erfolgt bei einer möglichst geringen Anzahl Transitionen (Moderne PCs verfügen über Mehrkernprozessoren. Können Ihre Heuristiken davon profitieren?).

(e) Erstellen Sie zehn Testfälle, von denen Sie annehmen, dass Ihre Heuristiken im Vergleich zu denen anderer Teams gute Ergebnisse liefern.

(f) Dokumentieren Sie Ihre Lösungsideen und die algorithmische Umsetzung.

In der zweiten Runde soll eine graphische Oberfläche erstellt werden, die folgende Funktionalitäten unterstützen soll.

(a) Auswahl einer Eingabedatei und der anzuwendenden Lösungsstrategien.

(b) Darstellung eines quantitativen Vergleichs der ausgewählten Lösungsstrategien.

(c) Visualisierung der Güte einer verwendeten Lösungsstrategie (z.B. Darstellung der Übertragung der Binärwörter als fließender Farbverlauf oder Animation).

Erstellen Sie eine Bedienungs- und Installationsanleitung. Schicken Sie uns Ihre Implementierung, Anleitungen sowie einen Bericht über die von Ihnen gemachten Entwurfsentscheidungen in der Software-Entwicklung.