Übung zur Vorlesung Einführung in Computer Microsystems

Prof. Dr-Ing. A. Koch
Jaco Hofmann, MSc.

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 15
Übungsblatt 4

Diese Übung hat die Implementierung eines „direct-mapped-cache“ in Bluespec zum Inhalt. Zuerst wird eine Testumgebung mit RAM und entsprechenden Hilfsmodulen erstellt. In dieser wird der Cache implementiert und schließlich getestet.

Diese Übung erfordert ein erhöhtes Maß an eigenständiger Leistung. Wenn Sie bei der Bearbeitung der Aufgaben Probleme haben können Sie gerne das Forum oder die Sprechstunde nutzen. Zusätzlich werden eine Woche vor der Besprechung der Aufgabe Teile des Lösungsvorschlags veröffentlicht um Ihnen die Bearbeitung der Kernaufgabe (Cache implementieren) zu vereinfachen.

Aufgabe 4.1 Cache

Ein „direct-mapped-cache“ ist die einfachste Form eines Caches. Jede Speicheraddresse des übergeordneten Speichers kann an genau einer Position des Caches abgelegt werden.

Zusätzlich speichert der Cache immer eine Cache-Line die aus mehreren Worten besteht anstatt nur ein Wort zu speichern. Dies wird gemacht um räumliche Lokalität zwischen nahe im Speicher liegenden Daten auszunutzen. Eine typische Cache-Line besteht aus 16 Wörtern. Bei einem Byte-adressierten Speicher mit 4 B pro Wort ergeben sich somit 64 B pro Cache-Line. Dementsprechend benötigt man mindestens 6 bit um die Position eines Bytes in der Cache-Line eindeutig zu bestimmen.

Der Cache hat eine festgelegte Anzahl an Plätzen die „Slot“ genannt werden. Jede Speicheradresse kann, wie oben erwähnt, in genau einen dieser Slots geladen werden. Für dieses Beispiel nehmen wir an der Cache habe 128 Slots. Dementsprechend benötigt man 7 bit um alle Slots eindeutig kodieren zu können.

Angenommen die Speicheraddresse besteht aus 32 bit dann werden die unteren 6 bit genutzt um das Wort in der Cache-Line zu spezifizieren, die darüber liegenden 7 bit spezifizieren den Slot und die restlichen 19 bit der Adresse sind das sogenannte Tag.

Wird eine Speicheranfrage an den Cache gestellt wird das Tag der in dem zur Speicheranfrage gehörenden Slot mit dem Tag der Speicheranfrage verglichen. Sind diese gleich gibt es einen so genannten Cache-Hit und das Wort mit dem zur Speicheranfrage gehörenden Offset wird zurückgegeben. Passen die Tags nicht zusammen gibt es einen Cache-Miss und die entsprechende Cache-Line muss aus dem höherliegenden Speicher geladen werden.

Der zu implementierende Cache soll dabei die Schreibstrategien „write-back“ und „write-allocate“ implementieren. Das heißt ein Schreibzugriff wird erst lokal im Cache ausgeführt und erst zurück in den höherliegenden Speicher geschrieben wenn die Cache-Line verdrängt wird. Dementsprechend muss vor dem Schreiben die korrekte Cache-Line im Cache vorliegen.

Weitere Informationen finden Sie in den GDI 3 (`https://www.ra.informatik.tu-darmstadt.de/lehre/gdi-iii/`) Folien unter „Speicherhierarchie 1-3“.

Aufgabe 4.1.1 RAM

Implementieren Sie einen Byte-adressierbaren RAM mit 1 048 576 B Größe. Verwenden Sie dafür das `BRAM` Modul aus der letzten Übung. Die Adressbreite soll 32 bit betragen. Geben Sie immer an Wortgrenzen angeordnete Wörter zurück (Anfrage für Adresse $0x03$ würde das Wort, das an Speicherstelle $0x00$ startet, zurückgeben). Die Wortbreite soll 32 bit betragen. Verwenden Sie einen RAM mit Byte-Enable, damit Sie wählen können, dass nur Teile von einem Wort geschrieben werden.

Überlegen Sie sich ein sinnvolles Interface (z.B. `ClientServer` mit entsprechenden Anfragen und Rückgaben).

Zusatzaufgabe: Implementieren Sie das Modul generisch für beliebige Wortbreiten.

Aufgabe 4.1.2 Hilfsmodule

Schreiben Sie zwei Hilfsmodule zum Umgang mit Cache-Lines. Das erste Modul soll den in der vorherigen Aufgabe implementierten RAM nutzen und Cache-Lines mit 512 bit zur Verfügung stellen. Das zweite Modul soll auf der einen Seite Anfragen mit Wortbreite annehmen und auf der anderen Seite Anfragen mit Cache-Line Breite ausgeben.

Zusatzaufgabe: Implementieren Sie die Module generisch für beliebige Wort- und Cache-Line-Breiten.

Aufgabe 4.1.3 Direct-Mapped-Cache

Implementieren Sie den oben beschrieben Cache mit 128 Slots.

Zusatzaufgabe: Implementieren Sie den Cache generisch für eine beliebige Anzahl Slots.

Aufgabe 4.1.4 Testen

Erstellen Sie eine Testbench, die die obigen Module nutzt und zusammenschaltet. Testen Sie verschiedene Speicherzugriffsmuster mit und ohne Cache. Mögliche Muster sind zum Beispiel:

a) Aufsteigendes Lesen und Schreiben für verschiedene Anzahl von Wörtern

b) Aufsteigendes Lesen und Schreiben mehrmals auf die selben Wörter

c) Lesen und Schreiben aller n Wörter für $n = 4, 8, 16, 32$

d) Zufälliger Zugriff auf Speicherelemente

e) Spalten- oder zeilenweise Matrixmultiplikation

Vergleichen Sie die Zugriffszeit pro Element für die verschiedenen Zugriffsmuster. Was stellen Sie fest, wenn Sie die selben Zugriffsmuster auf den Poolrechnern implementieren und die Laufzeiten vergleichen?

Zusatzaufgabe: Testen Sie die obigen Speichermuster mit verschiedener Cache-Line Größe und Anzahl von Slots.