Cache-Speicher für den Softprozessor SIRIUS mit DDR-Interface

Florian Zowislok, M.Eng.
Hochschule Offenburg, Badstraße 24, 77652 Offenburg
florian.zowislok@fh-offenburg.de

Zusammenfassung

Der Cache-Speicher für den Softprozessor SIRIUS ist ein 4-fach assoziativer Cache-Speicher, der mit einem DDR-Interface auf einen externen Speicher zugreifen kann. Er verwaltet und beschleunigt Zugriffe vom Prozessor auf diesen Speicher. Der Cache-Speicher arbeitet intern mit 32 Bit und der doppelten Prozessortaktfrequenz und ermöglicht Systeme mit größeren Speicheranforderungen ohne signifikante Performanceverluste.

Der Cache-Speicher wurde mit der Hardwarebeschreibungssprache VHDL erstellt und mit dem bestehenden Mikrocontrollersystem verbunden. Das Gesamtsystem wurde zunächst simuliert und anschließend mit dem Cyclone III FPGA Starter Kit von Altera, welches ein 32 MB DDR-RAM-Modul zur Verfügung stellt, durch Ausführen eines Testprogramms erfolgreich verifiziert. Für den kompletten Cache-Speicher werden inklusive der Pins für den externen Oszillator und des Reset-Tasters 3805 Logik-Zellen, 27 M9K-Blöcke, 44 Pins und eine PLL benötigt.

1. Das Mikrocontrollersystem

In mehreren vorangegangenen Projekten und Abschlussarbeiten wurde am Institut für Angewandte Forschung ein Mikrocontrollersystem entwickelt. Kernstück dieses Systems ist der 32-Bit SIRIUS-Softcore [1] [2], der zusammen mit verschiedenen Peripherieeinheiten auf einem Cyclone III FPGA emuliert wird. Dieses FPGA stellt zudem 64 KB RAM zur Verfügung. Der SIRIUS Prozessor kann mit der dazugehörigen SIRIUS IDE sowohl in Assembler als auch in C programmiert werden. Die Bedienung des Systems erfolgt mit einem Terminalprogramm, welches auf einem PC ausgeführt wird, der über USB mit dem Mikrocontrollersystem verbunden ist. Mittlerweile verfügt das System auch über eine SD-Karte und einen Touchscreen. Mit steigender Komplexität der Anwendungsprogramme stellt der verfügbare RAM des FPGAs den Flaschenhals dar. Aus diesem Grund soll ein größerer externer RAM verwendet werden. Um die langen Zugriffszeiten des externen RAMs zu kompensieren, sollen die Speicherblöcke des FPGAs als schneller Zwischenspeicher verwendet werden – ein sogenannter Cache. An diesen Cache soll mit einem DRAM-Controller der externe Speicher angebunden werden.

2. Anforderungen

2.1. Zielsystem

Das komplette Mikrocontrollersystem wird in einem Cyclone IIIFPGA der Firma Altera emuliert. Der Cache soll zusammen mit dem DRAM-Controller in das bestehende System integriert werden. Dabei wird der interne RAM durch den Cache ersetzt. Für den Cache können somit die Speicherblöcke des FPGAs verwendet werden, die zuvor für den RAM benötigt wurden.

2.2. Schnittstelle zum Mikrocontrollersystem

In *Abbildung 2.1* ist die ursprüngliche Version des Gesamtsystems dargestellt. Im mittleren Block ist das gesamte Mikrocontrollersystem mit allen Peripherien ohne den RAM zusammengefasst. Der rechte Block zeigt den RAM, der mit mehreren Leitungen an das Mikrocontrollersystem angebunden. Ist. Die Taktfrequenz von einem externen Oszillator wird mit der links oben dargestellten PLL auf die gewünschte Systemfrequenz gesetzt. Die Reset-Logik links unten erzeugt das Reset-Signal beim Einschaltvorgang des Systems bzw. wenn ein externer Resetknopf betätigt wird.

Damit das bestehende System, insbesondere die CPU, so wenig wie möglich verändert werden muss, soll die Ansteuerung des Caches identisch mit der des RAMs sein. Der RAM ist 64 KB groß und verfügt über eine Wortbreite von 16 Bit, womit 15 Bit für die Adressierung der einzelnen Worte benötigt werden. Über eine separate Byte-Enable-Leitung können beim Schreibenvorgang die beiden Bytes des 16-Bit-Wortes ausmaskiert werden. Der Zugriff auf den RAM erfolgt dabei innerhalb von einem Taktzyklus.

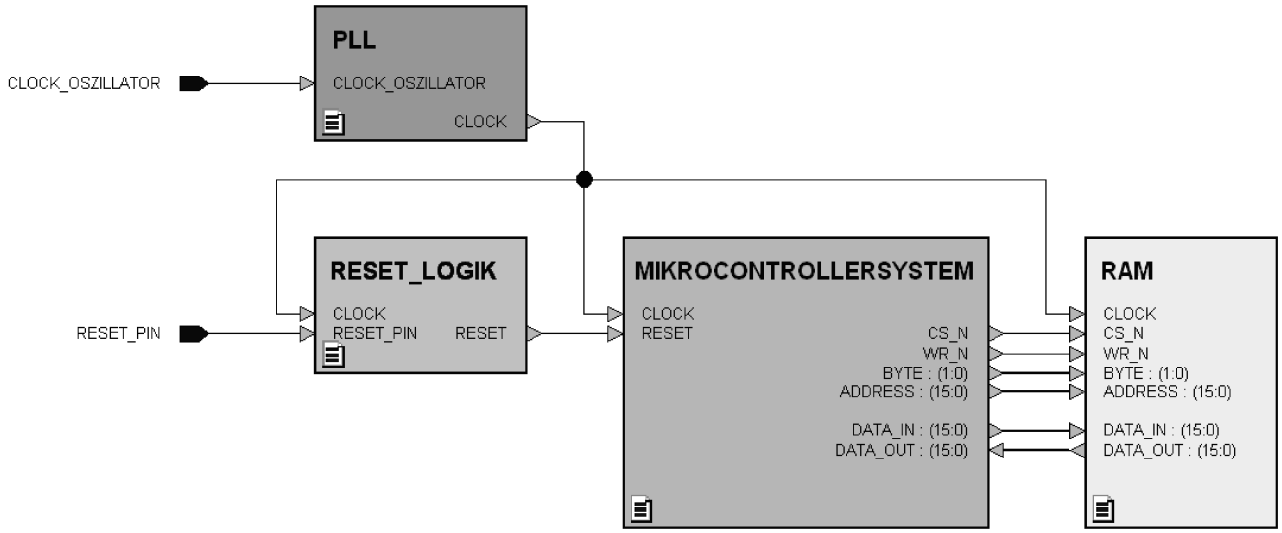

Abbildung 2.1: -Mikrocontrollersystem mit FPGA RAM

Für den Cache-Speicher soll der komplette 32 Bit breite Adressbus verwendet werden, mit welchem das Mikrocontrollersystem intern arbeitet. Zusätzlich wird die Schnittstelle um ein READY-Signal erweitert. Mit diesem Signal ist es möglich, das Mikrocontrollersystem anzuhalten, falls eine Speicheranforderung nicht in einem Taktzyklus bedient werden kann.

2.3. Externes Speichermodul

Als externes Speichermodul soll ein geeigneter SD-RAM zum Einsatz kommen, der deutlich mehr Speicher als der interne FPGA-RAM zur Verfügung stellt. Die Größe soll dabei 16 MB oder mehr betragen, um auch zukünftigen Anforderungen gerecht zu werden.

3. Funktionsweise von Cache-Speichern

3.1. Basisinformationen

Um den wiederholten Zugriff auf große Speicher mit langen Zugriffszeiten zu beschleunigen, werden häufig Cache-Speicher eingesetzt. Diese kleinen schnellen Zwischenspeicher enthalten Kopien kleiner Datenbereiche des großen Speichers. Wird auf Daten zugegriffen, die sich nicht im Cache-Speicher befinden, handelt es sich um einen sogenannten Fehlschlag, der als „Miss“ bezeichnet wird. Die Daten müssen zunächst mit einer langen Zugriffszeit aus dem großen Speicher in den Cache geladen werden, bevor darauf zugegriffen werden kann. Bei einem anschließenden Zugriff auf dieselben Daten sind diese bereits im Cache verfügbar. In diesem Fall handelt es sich um einen Treffer, welcher als „Hit“ bezeichnet wird. Die Daten können mit der verhältnismäßig kurzen Zugriffszeit des Caches gelesen oder geschrieben werden. Bei jedem Zugriff kann es somit entweder zu einem Hit oder zu einem Miss kommen. Der prozentuale Anteil wird als Hit-Rate bzw. Miss-Rate bezeichnet. Idealerweise sollte die Hit-Rate möglichst hoch sein. [3]

In der Speicherhierarchie eines Rechnersystems befindet sich der Cache zwischen den CPU-Registern und dem Hauptspeicher. Dies ist in *Abbildung 3.1* dargestellt. Obwohl der Zugriff auf den Hauptspeicher nur über den Cache erfolgt, bekommt der Programmierer davon nichts mit, denn im Gegensatz zum Hauptspeicher kann der Cache nicht direkt adressiert werden. Es handelt sich also um einen versteckten Speicher, der im Verborgenen die Systemperformance erhöht. [4]

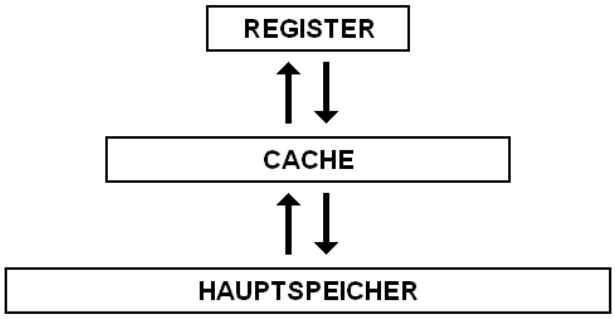

Abbildung 3.1: Speicherhierarchie eines Rechnersystems

3.2. Cache-Organisation

Für Cache-Speicher gibt es drei verschiedene Organisationen. Dabei wird der Cache-Speicher in einen oder mehrere Blöcke unterteilt. Jeder Block enthält die

gleiche Anzahl von Zeilen, wobei jede dieser Zeilen eine bestimmte Anzahl zusammenhängender Datenbytes enthält. Diese liegt üblicherweise im Bereich von 4 bis 128 Bytes. [5]

Besteht der Cache nur aus einem Block, gibt es zwei verschiedene Organisationstypen: „vollassoziativ“ und „direkt verdrahtet“. Bei einem vollassoziativen Cache kann jede Datenzeile aus dem großen Speicher an jeder beliebigen Stelle im Cache abgelegt werden. Bei einem direkt verdrahteten Cache kann jede Zeile, abhängig von der Adresse, nur an einer einzigen Position gespeichert werden. Dadurch vereinfacht sich zwar der Implementierungsaufwand, sobald jedoch nacheinander abwechselnd auf Speicherbereiche zugegriffen wird, die auf dieselbe Position im Cache verdrahtet sind, kommt es jedes Mal zu einem Miss. Um diesem Problem entgegenzuwirken, können mehrere direkt verdrahtete Cache-Blöcke parallel verwendet werden. [5]

Bei n Blöcken spricht man von einem n-fach assoziativen Cache. Jede Adresse ist auch hier mit einer bestimmten Zeile innerhalb eines Blockes verdrahtet, jedoch kann der Block selbst frei gewählt werden. Somit ergeben sich für n Blöcke auch n mögliche Speicherpositionen im Cache. Diese n Speicherpositionen, die mit denselben Adressen verdrahtet sind, werden als Satz bezeichnet. Die Anzahl der Sätze in einem Cache ist gleich der Anzahl der Cache-Zeilen in einem Block. [5] *Abbildung 3.2* zeigt einen n-fach assoziativen Cache mit m Sätzen.

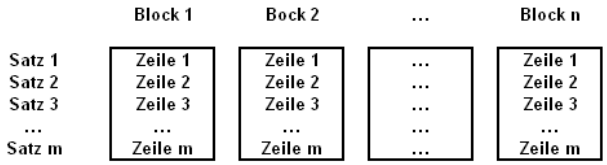

Abbildung 3.2: n-fach assoziativer Cache mit m Sätzen

Für die Entwicklung des Caches war vorgegeben, dass der Cache 4-fach assoziativ ausgelegt sein soll. Damit ist sichergestellt, dass mindestens vier unterschiedliche Datenbereiche gleichzeitig zwischengespeichert werden können. Außerdem stellt dies einen guten Kompromiss aus Effizienz und Implementierungsaufwand dar.

3.3. Adressierung und Steuerung

Bei Cache-Speichern wird die Adresse in drei Adressen unterteilt, die Tag-Adresse, die Selektor-Adresse und die Wort-Adresse, z.B. stellen bei der Adresse „11001001“ die drei MSBs „110“ die Tag-Adresse, die zwei mittleren Bit „01“ die Selektor-Adresse und die drei LSBs „001“ die Wort-Adresse dar. Die Wort-Adresse adressiert ein Wort innerhalb einer Cache-Zeile, die mit der Selektor-Adresse ausgewählt wird. Die zusammenhängenden Daten innerhalb einer Cache-Zeile haben somit immer eine gemeinsame Tag- und Selektor-Adresse, die dazu gehörige Tag-Adresse wird zu jeder Cache-Zeile separat gespeichert. [5]

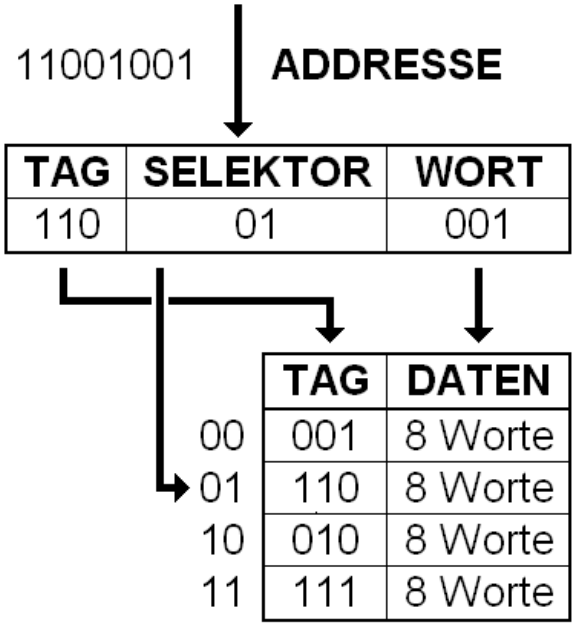

Abbildung 3.3: Adressierung im Cache

In *Abbildung 3.3* ist die Adressierung eines Cache-Blockes dargestellt. Wenn die Tag-Adresse einer über den Selektor ausgewählten Cache-Zeile mit der gespeicherten Tag-Adresse in einem der Blöcke übereinstimmt, kommt es zu einem Hit. Die adressierten Daten sind im Cache verfügbar und können direkt gelesen oder geschrieben werden. Andernfalls kommt es zu einem Miss und die entsprechende Datenzeile muss aus dem großen Speicher in einen Cache-Block nachgeladen werden. Dabei wird auch die entsprechende gespeicherte Tag-Adresse aktualisiert. Welcher Cache-Block dabei gewählt wird, ist im nachfolgenden Kapitel beschrieben.

3.4. Verdrängungsstrategien

Beim Nachladen einer Daten-Zeile in einen 4-fach assoziativen Cache kann diese theoretisch in jedem der vier Cache-Blöcke abgelegt werden, wobei jedoch die an der betreffenden Position befindliche Zeile aus dem Cache verdrängt wird. Idealerweise sollte dabei der Cache-Block gewählt werden, dessen über die Selektor-Adresse ausgewählte Zeile am längsten nicht benutzt wird. Diese ist jedoch meist im Voraus nicht bekannt. Aus diesem Grund gibt es verschiedene Verdrängungsstrategien. Es kann beispielsweise eine zufällige, die älteste, die am seltensten genutzte oder

die am längsten nicht genutzte Zeile verdrängt werden. [3]

Der beste Kompromiss zwischen Implementierungsaufwand und Effizienz ist es, die Zeile zu verdrängen, die am längsten nicht genutzt wurde. Dazu wird zu jeder Zeile ein 2-Bit-Zähler als Attribut gespeichert. Dabei darf jede Bitkombination des Zählers in allen vier Cache-Blöcken in den jeweiligen Zeilen mit der gleichen Selektor-Adresse nur einmal verwendet werden. Für die zuletzt verwendete Zeile beträgt die Bitkombination „00“, für die vorletzte „01“, für die am zweitlängsten nichtverwendete „10“ und für die am längsten nicht verwendete Zeile „11“.

3.5. Sicherstellen der Konsistenz

Da der Cache Kopien von Daten enthält, die durch Schreib-Zugriffe verändert werden können, muss die Konsistenz zwischen dem Cache und dem großen Speicher sichergestellt sein. Dazu gibt es mehrere Strategien. Beim „write-through“ werden die Daten direkt in den Hauptspeicher geschrieben und gegebenenfalls auch in den Cache, wenn die Daten im Cache vorhanden sind. Dabei entstehen jedoch auch langsame Schreib-Zugriffe auf den Hauptspeicher, wenn die Daten im Cache verfügbar sind. Dies kann mit der „write-back“-Strategie vermieden werden. Falls die Daten nicht im Cache verfügbar sind, werden sie wie beim „write-through“ direkt im Hauptspeicher verändert. Sind die Daten jedoch im Cache verfügbar, werden sie nur im Cache verändert und die entsprechende Zeile mit dem Attribut „Dirty-Bit“ gleich „1“ gekennzeichnet. Bei dem Verdrängen durch eine andere Datenzeile muss die verdrängte Cache-Zeile dann zuerst in den Speicher zurückgeschrieben werden, falls das Dirty-Bit gesetzt ist. Ist es nicht gesetzt, so kann die verdrängte Zeile verworfen werden. Die „write-allocation“-Strategie funktioniert im Prinzip genauso, jedoch wird im Gegensatz zur „write-back“-Strategie die Datenzeile, in welche geschrieben werden soll, immer zuerst komplett in den Cache geladen und anschließend nur im Cache verändert. [4]

Um die Anzahl der langsamen Zugriffe auf den Hauptspeicher zu minimieren, wird hier die „write-allocation“-Strategie gewählt. Dies hat zur Folge, dass unabhängig davon, ob es sich um einen Lese- oder Schreib-Zugriff handelt, bei einem Miss immer eine Datenzeile vom Hauptspeicher in den Cache geladen werden muss und die verdrängte Zeile dabei gegebenenfalls zurückgeschrieben werden muss.

3.6. Zugriffszeiten

Die Zugriffszeit auf Daten, die sich im Cache befinden, beträgt normalerweise nur einen Taktzyklus. Befinden sich die Daten nicht im Cache, müssen diese aus dem Speicher nachgeladen werden. Da dabei immer eine ganze Zeile geladen wird und gegebenenfalls zurückgeschrieben wird, setzt sich diese Zeit aus der Zugriffszeit und der Transferzeit für die Daten zusammen. Die Zugriffszeit hängt dabei von dem verwendeten Speicher und die Transferzeit noch zusätzlich von der Länge einer Cache-Zeile ab. Werden für die Zugriffszeit beispielsweise 8 und für die Transferzeit 16 Taktzyklen benötigt, dauert der gesamte Nachladevorgang 24 Taktzyklen. [5]

3.7. Beispiel

In diesem Beispiel ist die Funktionsweise des Caches in den *Abbildungen 3.4 bis 3.7* dargestellt. Dazu wird ein Hauptspeicher angenommen, der 32 Datenworte enthält und somit 5 Adressbit benötigt. Der Cache ist 2-fach assoziativ mit 2 Zeilen je Block und 4 Worten pro Zeile ausgelegt. Somit bilden die 2 MSBs des Hauptspeichers die Tag-Adresse, das mittlere Bit die Selektor-Adresse und die 2 LSBs die Wortadresse. Das „C“ in den Abbildungen steht für einen 1-Bit-Zähler, der anzeigt, welche Zeile innerhalb eines Satzes am längsten nicht verwendet wurde. Das „D“ steht für das Dirty-Bit. Zu Beginn sind im Cache schon Daten geladen, wie in *Abbildung 3.4* dargestellt. Die Datenzeile mit der Tag-Adresse „00“ und der Selektor-Adresse „0“ befindet sich beispielsweise in der oberen Zeile des oberen Caches. Wird nun beispielsweise das Wort mit der Adresse „00110“ gelesen, sind diese Daten im Cache verfügbar. Die Selektor-Adresse „1“ (mittleres Bit) wählt die untere Zeile in den Blöcken aus. Im oberen Block stimmt die gespeicherte Tag-Adresse mit der aktuellen Tag-Adresse überein und es kommt zu einem Hit. Mit der Wort-Adresse „10“ wird nun das zweite Wort von links innerhalb der betreffenden Cache-Zeile ausgewählt und kann direkt gelesen werden. Da der Zähler „C“ (COUNTER) dieser Cache-Zeile bereits den Wert „0“ hat, müssen die Zähler innerhalb des Satzes nicht verändert werden.

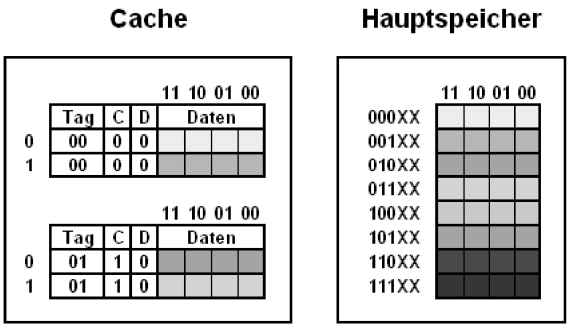

Abbildung 3.4: Aufbau des Beispiel-Caches

Als nächstes wird in Adresse „01001“ geschrieben. Die Daten sind in der oberen Zeile des unteren Cache-Blocks verfügbar und können direkt geschrieben wer-

den. Dabei muss der Zähler innerhalb des Satzes verändert und das Dirty-Bit der betreffenden Cache-Zeile gesetzt werden, wie in *Abbildung 3.5* dargestellt. Die veränderten Daten werden durch ein „X“ markiert und sind nun nicht mehr mit dem Hauptspeicher konsistent.

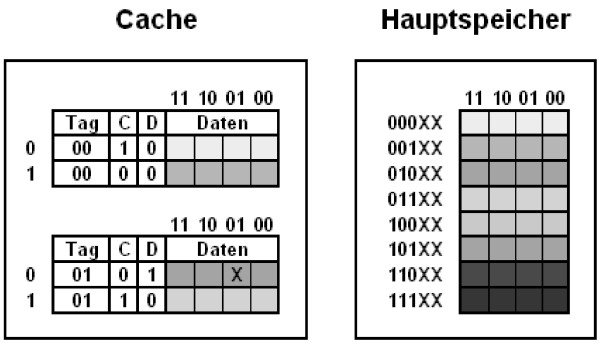

Abbildung 3.5: Beispiel-Cache nach dem Schreib-Zugriff

Der nächste Lese-Zugriff erfolgt auf Adresse „10010“. Über die Selektor-Adresse werden die jeweils oberen Cache-Zeilen ausgewählt, wobei es in keinem der Blöcke zu einer Übereinstimmung der Tag-Adressen kommt. Die angeforderten Daten müssen also zunächst in den Cache geladen werden. Der Zähler „C“ zeigt an, dass die Cache-Zeile des oberen Blocks innerhalb des Satzes am längsten nicht verwendet wurde. Da keine Daten verändert wurden (Dirty-Bit = 0), kann die Cache-Zeile verworfen werden und die benötigte Zeile aus dem Speicher geladen werden. Dies ist in *Abbildung 3.6* dargestellt. Beim Nachladevorgang wird die neue Tag-Adresse in den Tag-Speicher geschrieben und das Dirty-Bit gegebenenfalls auf „0“ gesetzt. Die Daten sind nun im Cache verfügbar und können gelesen werden. Dabei wird der Zähler der Cache-Zeile auf „0“ gesetzt und der anderen Zähler des gleichen Satzes auf „1“ gesetzt.

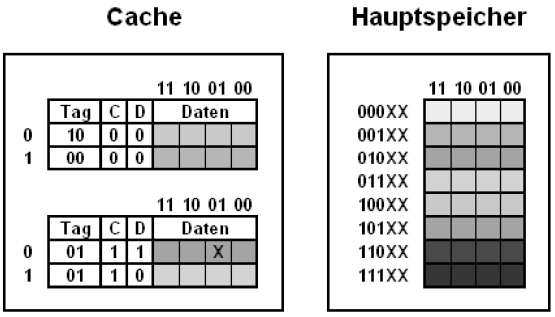

Abbildung 3.6: Nachladen in den Beispiel-Cache

Der nächste Schreib-Zugriff wird mit 11000 adressiert. Wieder sind die oberen Cache-Zeilen selektiert und es kommt zu keiner Übereinstimmung der Tag-Adresse, weshalb die angeforderten Daten aufgrund des Zählerstandes in die obere Zeile des unteren Cache-Blocks geladen werden müssen. Hier ist jedoch das Dirty-Bit gesetzt, womit die Cache-Zeile zunächst in Hauptspeicher geschrieben werden muss. *Abbildung 3.7* zeigt den Cache und den Hauptspeicher, nachdem die alte Cache-Zeile in Hauptspeicher geschrieben, die neue Datenzeile geladen und das adressierte Wort geschrieben wurde.

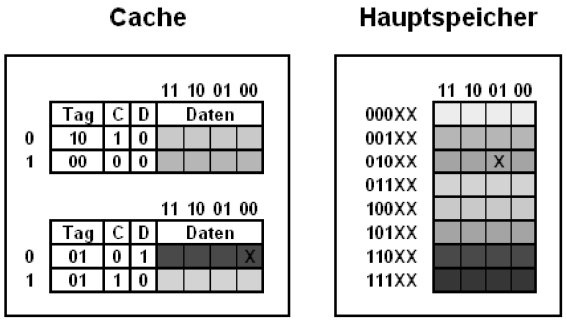

Abbildung 3.7: Beispiel-Cache nach dem Rückschreiben in den Hauptspeicher

4. Aufbau des kompletten Caches

4.1. Daten- und Adressbus

Wie in *Kapitel 2.2* beschrieben, soll der Cache mit der kompletten Busbreite von 32 Adressbit angesteuert werden, damit die Speichergröße des externen Speichers bis zu einer Größe von maximal 4 GB erweitert werden kann. Obwohl für die Schnittstelle zum Mikrocontrollersystem ein 16-Bit-Datenbus vorgesehen ist, soll der Cache-Speicher intern mit einem 32-Bit-Datenbus arbeiten, damit das Mikrocontrollersystem später ohne großen Aufwand ebenfalls auf einen 32 Bit breiten Datenbus umgestellt werden kann.

4.2. Dimensionierung der Cache-Blöcke

Die Größe eines Cache-Blockes wird durch zwei Parameter bestimmt, die Länge einer einzelnen Cache-Zeile, sowie die Anzahl von Zeilen innerhalb eines Blockes. Das Produkt aus diesen beiden Parametern ergibt die Größe eines Blockes.

Da der Cache-Speicher zukünftig auch in einem ASIC zum Einsatz kommen soll, bei dem die Speicherzellen den größten Teil der Chipfläche benötigen, sollte die Größe des Caches möglichst gering gehalten werden. Andererseits können mit einem großen Cache auch mehr Daten zwischengespeichert werden, was die Hit-Rate und somit die Systemperformance erhöht. Eine Cachegröße von insgesamt 16 KB zuzüglich der benötigten Speicher für die Tag-Adresse und Attribute zu jeder Cache-Zeile sind das Maximum für den

geplanten ASIC. Somit stehen für jeden der vier Cache-Blöcke 4 KB Datenspeicher zur Verfügung.

Auch bei der Länge der Cache-Zeilen muss ein Kompromiss gefunden werden. Die Daten innerhalb der Cache-Zeile stellen das Abbild aus einem zusammenhängenden Speicherbereich des externen DRAMs dar. Beim Nachladen einer Cache-Zeile aus dem DRAM muss diese immer komplett als Burst gelesen oder geschrieben werden. Beim Zugriff auf den DRAM wird dabei vor jedem Burst eine gewisse Latenzzeit benötigt. Um möglichst wenig dieser Latenzzeiten zu verursachen, sollte die Zeilenlänge möglichst groß gewählt werden. Beim Zugriff auf einzelne Bytes, die nicht innerhalb einer Zeile liegen, kommt es dabei jedoch dazu, dass lange Zeilen aus dem DRAM geladen werden müssen, obwohl immer nur ein einzelnes Byte benötigt wird. In diesem Fall würde eine kurze Zeilenlänge die Systemperformance erhöhen. Die ideale Zeilenlänge kann letztendlich nur über Benchmarks ermittelt werden und hängt stark von der Programmierung der Anwendung ab. Für dieses Cache-System wird eine Zeilenlänge von 64 Byte verwendet. Ein 4-KB-Cache-Block setzt sich somit aus 64 Zeilen zusammen.

4.3. Adressierung eines Cache-Blockes

Vom 32-Bit-Adressbus werden theoretisch die untersten 6 Bit zum Adressieren der 64 Bytes innerhalb einer Zeile verwendet. Da der Cache intern jedoch mit 32-Bit-Worten arbeitet, werden je 4 Bytes zu einem Wort zusammengefasst. Somit bleiben die untersten 2 Bit ungenutzt. Mit den übrigen 4 Bit können die daraus resultierenden 16 Worte innerhalb einer Cache-Zeile adressiert werden. Die darüber liegenden 6 Bit des Adressbusses werden als Selektor-Adresse zum Auswählen der Cache-Zeile verwendet. Die höchsten 20 Bit vom Adressbus ergeben somit die Tag-Adresse, die neben den Attributen zu jeder Cache-Zeile gespeichert wird. In *Abbildung 4.1* ist die Adressierung eines Cache-Blockes dargestellt.

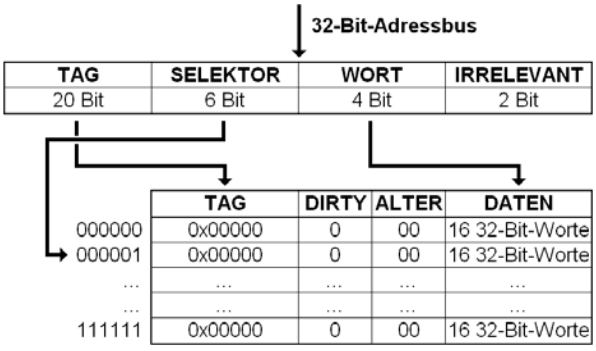

Abbildung 4.1: Adressierung eines Cache-Blockes

4.4. Aufbau des gesamten Cache-Systems

Abbildung 4.2 zeigt das gesamte Cache-System. Es besteht aus den vier Cache-Blöcken und einer Steuereinheit, die die Zugriffe auf die einzelnen Cache-Blöcke verwaltet. Mit dem im Cache enthaltenen DRAM-Controller kann die Steuereinheit auf den externen DRAM zugreifen. Die Kommunikation mit dem Mikrocontrollersystem erfolgt ebenfalls über die Steuereinheit.

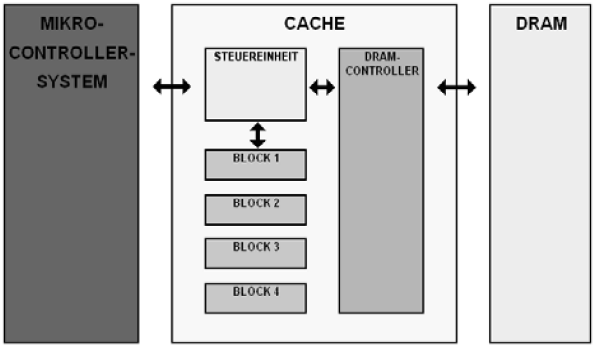

Abbildung 4.2: Aufbau des Cache-Systems

5. Auswahl der Hardware

Bevor mit der Implementierung des Systems begonnen werden kann, muss festgelegt werden, welche Hardware verwendet werden soll, da der DRAM-Controller an das externe Speichermodul angepasst werden muss. Um den Aufwand möglichst gering zu halten, wird zunächst nach einer fertigen Lösung gesucht. Dazu bietet Altera ein preiswertes Entwicklungsboard an, das Cyclone III FPGA Starter Kit, welches in *Abbildung 5.1* dargestellt ist.

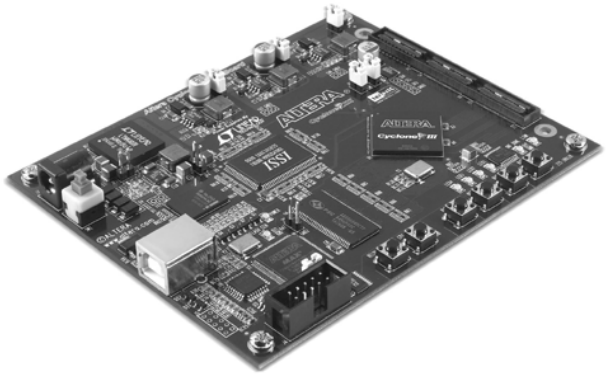

Abbildung 5.1: Cyclone III FPGA Starter Kit

Das Entwicklungsboard stellt zahlreiche Hardwarekomponenten zur Verfügung. Die relevanten Elemente sind im Folgenden aufgelistet:

- Cyclone III EP3C25 FPGA

- 32 MB DDR-RAM
- USB-Blaster zum Programmieren des FPGAs
- 50 MHz Oszillator
- Jeweils 4 Taster und LEDs
- HSMC-Anschlussleiste für Zusatzplatinen

5.1. DDR-RAM

Bei DDR-RAMs handelt es sich um dynamische Speicher mit großen Kapazitäten, sogenannte DRAMs. Um diese Kapazitäten auf kleinen Raum zu erhalten, werden die Datenbit mit Kondensatoren gespeichert. Da sich die Kondensatoren jedoch über unerwünschte Leckströme entladen, geht der Speicherinhalt nach kurzer Zeit verloren. Aus diesem Grund müssen die Daten im Speicher zyklisch mit einem sogenannten „Refresh“ aktualisiert werden. Im DRAM sind die Speicherzellen zeilenweise angeordnet.

Um Adressleitungen zu sparen, wird beim Zugriff über die gleichen Leitungen zuerst die Zeilenadresse und dann die Wortadresse mit denselben Leitungen übertragen. Durch den internen Aufbau dauert der Zugriff auf DRAMs deutlich länger als auf statische RAMs, jedoch können die zusammenhängenden Daten innerhalb einer Zeile als “Burst“ ausgelesen werden. Dabei werden nach der relativ langen Zugriffszeit für die erste Speicherzelle die folgenden mit jedem Taktzyklus gelesen bzw. geschrieben.

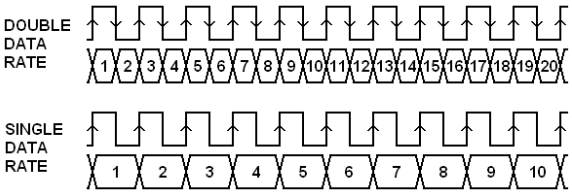

Abbildung 5.2: DDR im Vergleich zu SDR

Die Besonderheit bei DDR-RAM liegt darin, dass die Daten sowohl mit steigender als auch mit fallender Flanke übertragen werden, womit die doppelte Daten-Rate von herkömmlichen DRAMs erreicht wird, bei denen die Daten nur mit der steigenden Taktflanke übertragen werden. Dies ist in *Abbildung 5.2* dargestellt. Bei dem verwendeten DDR-RAM-Modul handelt es sich um einen Speicherchip mit einem 16 Bit breiten Datenbus. Der Speicherchip kann theoretisch im Bereich zwischen 83 und 200 MHz betrieben werden, jedoch unterstützt der verwendete DRAM-Controller im Cyclone III FPGA nur Frequenzen bis maximal 167 MHz. Somit ergeben sich bei den unterschiedlichen Betriebsfrequenzen theoretisch die folgenden maximalen Datenraten:

83 MHz * 16 Bit * 2 (DDR) = 317 MB/s
100 MHz * 16 Bit * 2 = 381 MB/s
133 MHz * 16 Bit * 2 = 507 MB/s
150 MHz * 16 Bit * 2 = 572 MB/s
167 MHz * 16 Bit *2 = 637 MB/s

Durch den zyklischen Refresh und vor allem den hohen Zugriffszeiten werden in der Praxis jedoch nur Datenraten erreicht, die bei etwa 50% dieser Angaben liegen. Die Speichergröße des verwendeten DDR-RAMs ist mit 32 MB auch für zukünftige Anforderungen ausreichend.

5.2. Weitere Hardware-Komponenten

Da das Mikrocontrollersystem in der bisherigen Version den Flash-Speicher M25P40 der Firma ST zum Booten des Systems benötigt, soll dieser an das System angeschlossen werden. Dazu wird eine Zusatzplatine der Firma Bitec verwendet, die über die HSMC-Anschlussleiste mit dem Entwicklungsboard verbunden wird. 158 Anschlüsse des FPGAs sind direkt auf dem Board verfügbar, sowie Versorgungsspannungen von 3,3 und 12 Volt. *Abbildung 5.3* zeigt Vorder- und Rückseite der Zusatzplatine.

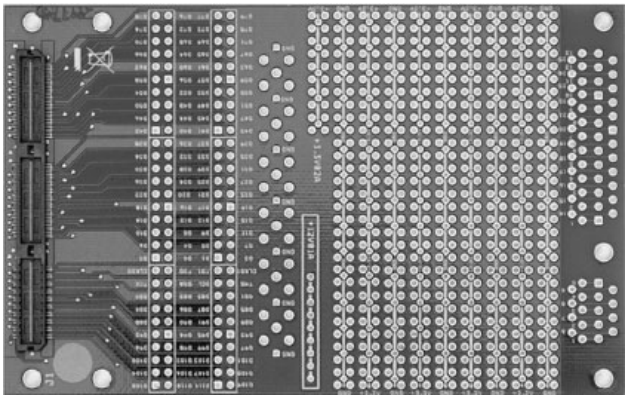

Abbildung 5.3: Zusatzplatine für das Cyclone III FPGA Starter Kit

Da die Versorgungsspannung des FPGAs 2,5 Volt beträgt und der Flash mit den 3,3 Volt der Zusatzplatine betrieben wird, wird zum Konvertieren der Signale der Pegelwandler ADG3308 von Analog Devices eingesetzt. Des Weiteren wird zur Tonausgabe ein Piezo-Lautsprecher verwendet.

6. Implementierung in VHDL

6.1. Verwendete Entwicklungssoftware

Für die Implementierung und Simulation des VHDL-Codes werden die Programme HDL Designer und ModelSim verwendet, die im FPGA Advantage Softwarepaket von Mentor enthalten sind. Die VHDL-Codes für FPGA-spezifische Funktionsblöcke werden mit dem in der Quartus II Software von Altera enthaltenen MegaWizzard Plug-In Manager generiert.

6.2. Altera MegaCore Funktionen

Um das auf dem Entwicklungsboard vorhandene DDR-Speichermodul mit dem Cyclone III FPGA ansprechen zu können, wird die Altera MegaCore Funktion „DDR SDRAM High Performance Controller“ verwendet. Altera MegaCore Funktionen sind optimierte Funktionsblöcke für häufig verwendete Anwendungen, welche mit dem MegaWizzard Plug-In konfiguriert werden können.

6.3. Festlegung der Taktfrequenzen

Bei dem bisherigen Mikrocontrollersystem erfolgte ein Speicherzugriff innerhalb von einem Taktzyklus. Diese Bedingung soll auch beim Cache erfüllt werden, sofern die angeforderten Daten im Cache verfügbar sind. Bei einem Schreib-Zugriff auf den Cache müssen jedoch zuerst die Tag-Adressen des angeforderten Speicherbereichs mit den gespeicherten Tag-Adressen in den selektierten Zeilen der Cache-Blöcke verglichen werden, bevor die Daten in den richtigen Block geschrieben werden können. Dafür sind zwei Taktzyklen erforderlich. Aus diesem Grund wird für den Cache die zweifache Taktfrequenz des Mikrocontrollersystems gewählt. Für einen Taktzyklus des Mikrocontrollersystems ergeben sich somit zwei Taktzyklen im Cache

6.4. Übersicht über das Gesamtsystem

Das gesamte Cache-System, in *Abbildung 6.1* dargestellt, ist in sieben verschiedene Einheiten unterteilt. Die Anschlussports links bilden die Schnittstelle zum Mikrocontrollersystem, die Ports auf der rechten Seite die Schnittstelle zum DRAM-Modul.

Das Kernstück bildet die Steuereinheit, die direkt und über den Multiplexer mit den vier Cache-Blöcken verbunden ist. Während des Zugriffs auf den externen Speicher mit dem DRAM-Controller übernimmt Nachladeeinheit die Steuerung. Die Anpassung zwischen den unterschiedlichen Datenbusbreiten von Mikrocontrollersystem und Cache erfolgt mit dem Datenbus-Wrapper. Die Reset-Logik wurde vom übrigen Mikrocontrollersystem in den Cache verlegt, da damit direkt der DRAM-Controller angesteuert wird..

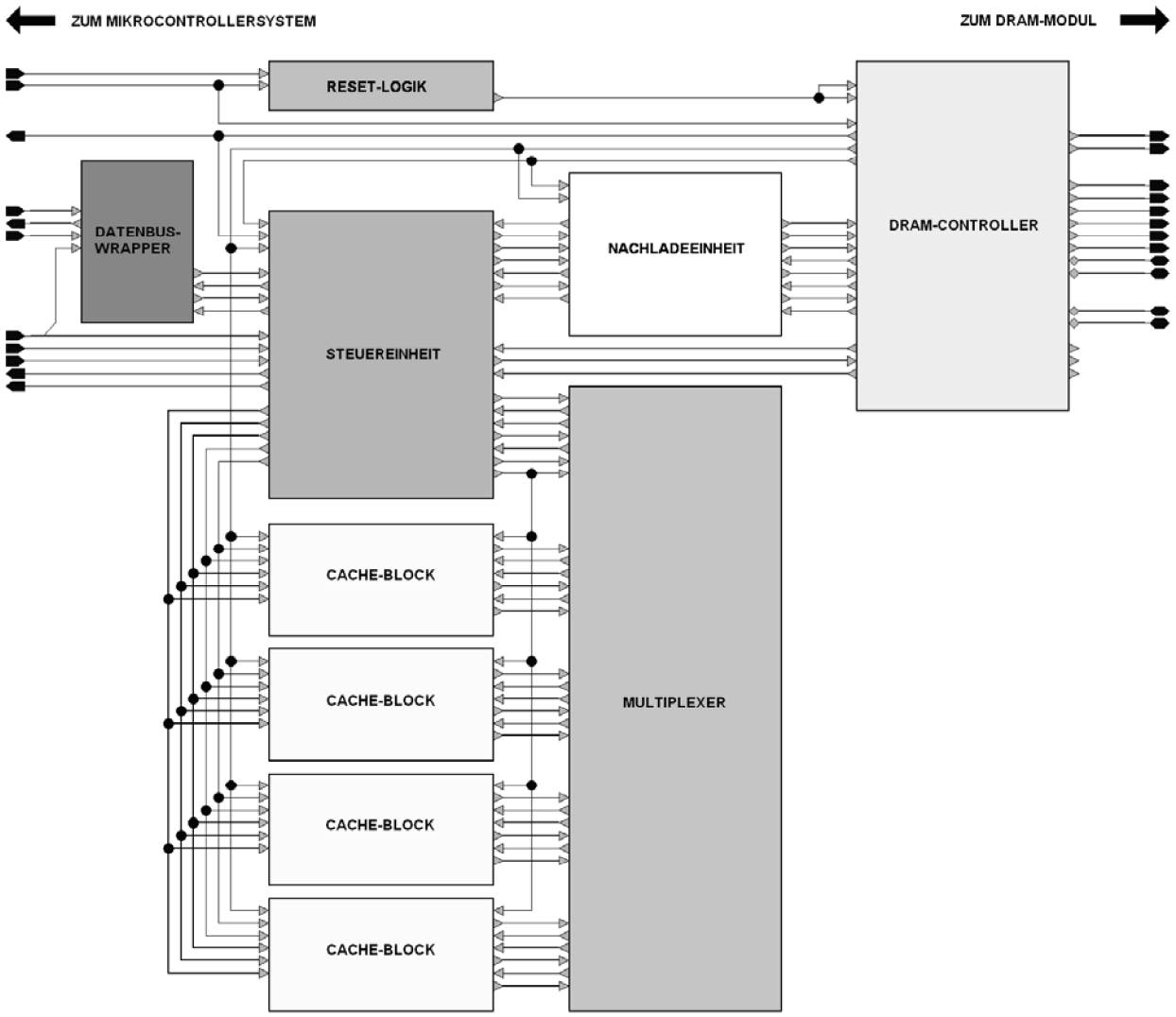

Abbildung 6.1: Übersicht über das Gesamtsystem

6.5. Steuereinheit

Die Steuereinheit ist das Kernstück des Caches. Abgesehen von der Reset-Logik ist die Steuereinheit mit allen Einheiten des Caches direkt verbunden. Die komplexe Cache-Steuerung wird durch einen einfachen Zustandsautomaten realisiert. Da die Schnittstelle zum Mikrocontrollersystem nur mit der halben Taktfrequenz des Caches betrieben wird, benötigt die Steuereinheit beide Taktsignale. Durch den DRAM-Controller ist zwar sichergestellt, dass die beiden Taktsignale synchronisiert sind, aber es ist nicht bekannt, ob zum Zeitpunkt der steigenden Taktflanke vom schnelleren Cache-Takt das langsamere Taktsignal eine steigende oder fallende Flanke aufweist. Aus diesem Grund wird zum Zeitpunkt der fallenden Flanke des Cache-Taktes der Zustand des langsamen Taktes in dem internen Signal „SYS_CLK_LEVEL“ gespeichert. Ist „SYS_CLK_LEVEL“ zum Zeitpunkt der steigenden Taktflanke des Cache-Taktes gleich „0“, so ist die Taktflanke des langsamen Taktes ebenfalls steigend. Dies ist in *Abbildung 6.2* veranschaulicht.

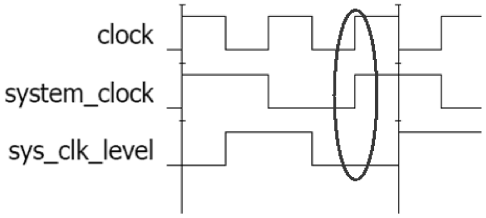

Abbildung 6.2: Phasensynchronisation des Takts

Der Zustandsautomat hat fünf Zustände, „RESET“, „LEERLAUF“, „TAG LESEN“, „LESEN/SCHREIBEN“ und „NACHLADEN“. Nach dem Einschalten befindet sich der Zustandsautomat im „RESET“-Zustand und der DDR-RAM enthält zufällige Daten. Deshalb können die ebenfalls zufälligen Daten im Cache-Speicher als gültiges Abbild des DDR-RAMs angenommen werden. Beim Booten des Systems wird zunächst der

untere Speicherbereich beschrieben. Um das unnötige Nachladen aus dem DDR-RAM von zufälligen Daten zu vermeiden, werden die Daten im Cache als Abbild des unteren Speicherbereichs angenommen. Dazu müssen im „RESET“-Zustand die Tag-Adressen und Attribute der Tag-Speicher korrekt initialisiert werden.

Mit den vier Cache-Blöcken, die je 4 KB Daten zwischenspeichern, können die unteren 16 KB Daten abgebildet werden. Dies entspricht dem Adressraum von 0x00000000 bis 0x00003FFF. Dies ergibt für die Cache-Blöcke folgende Adressbereiche:

Block 0: 0x00000000 bis 0x00000FFF
Block 1: 0x00001000 bis 0x00001FFF
Block 2: 0x00002000 bis 0x00002FFF
Block 3: 0x00003000 bis 0x00003FFF

Die Tag-Adressen innerhalb eines Blockes sind für alle Zeilen identisch und unterscheiden sich nur in den zwei LSBs gegenüber den anderen Blöcken. Die oberen 18 Bit sind dabei immer gleich „0“. Das Attribut für das Alter einer Cache-Zeile kann beliebig gewählt werden, jedoch muss dabei sichergestellt sein, dass bei den Cache-Zeilen mit gleicher Selektor-Adresse jeder Zählerwert nur einmal verwendet wird. Zur Vereinfachung wird das Attribut entsprechend den zwei LSBs der Tag-Adressen initialisiert. Das Dirty-Bit wird immer auf „0“ gesetzt. Somit ergeben sich für die Tag-Speicher der Cache-Blöcke die in *Abbildung 6.3* gezeigten Werte für die Initialisierung.

	TAG-ADRESSE	ALTER	DIRTY
BLOCK_0	00000000000000000000	00	0
BLOCK_1	00000000000000000001	01	0
BLOCK_2	00000000000000000010	10	0
BLOCK_3	00000000000000000011	11	0

Abbildung 6.3: Initialisierung der Tag-Speicher

Nach der Initialisierung wechselt er in den „LEERLAUF“-Zustand. Sobald während einer steigenden Taktflanke des langsamen System-Takts der Cache mit dem Chip-Select-Signal angesteuert wird, wechselt er in den „TAG LESEN“-Zustand, in welchem die Tag-Adressen aus den adressierten Cache-Zeilen ausgelesen werden. Danach wird in den „LESEN/ SCHREIBEN“-Zustand gewechselt. In diesem Zustand werden die Tag-Adressen in den Cache-Blöcken miteinander verglichen und gegebenenfalls ein Hit-Signal erzeugt. Kommt es zu einem Hit, können die Daten aus dem entsprechenden Cache-Block gelesen oder geschrieben werden. Bei einem Miss wird in den „NACHLADEN“-Zustand gewechselt, in dem die benötigte Datenzeile aus dem DDR-RAM nachgeladen wird. Nach dem Nachladevorgang wird mit steigender Taktflanke wieder in den „TAG LESEN“-Zustand und darauf folgend in den „LESEN/SCHREIBEN“-Zustand gewechselt. Da die nachgeladenen Daten nun auf jeden Fall im Cache verfügbar sind, kommt es zu einem Hit. In diesem Fall wird mit steigender Taktflanke des System-Taktes wieder in den „TAG LESEN“-Zustand gewechselt, wenn der Cache mit dem Chip-Select-Signal erneut angesteuert wird. Ansonsten kehrt der Zustandsautomat in den „LEERLAUF“-Zustand zurück, bis ein erneuter Cache-Zugriff erfolgt. Das Zustandsdiagramm ist in *Abbildung 6.4* dargestellt.

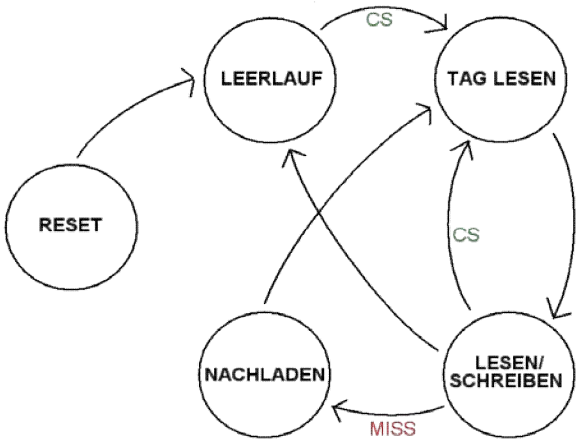

Abbildung 6.4: Zustandsautomat

7. Simulation

Zur Simulation des Cache-Systems wird das Programm ModelSim verwendet. Dabei kann das Verhalten der Schaltung genau analysiert werden. Damit die Eingangssignale nicht für jeden Taktzyklus einzeln festgelegt werden müssen, wird das Cache-System zusammen mit dem kompletten Mikrocontrollersystem simuliert. Sollte es im Cache-Speicher zu einem Fehler kommen, ist dies meist leicht am Fehlverhalten des Mikrocontrollersystems zu erkennen.

Bei der simulierten Systemfrequenz von 50 MHz erfolgt der Zugriff auf den Cache gemäß der Anforderung innerhalb eines System-Taktes. Dies entspricht einer Zugriffszeit von 20 ns. Beim Nachladen einer Datenzeile aus dem DDR-RAM werden für den gesamten Datenzugriff 23 Systemtakte benötigt, womit sich eine Zugriffszeit bei einem Miss von 460 ns ergibt. Falls dabei eine Cache-Zeile zuvor in den RAM zurückgeschrieben werden muss, dauert der Zugriff 35 Systemtakte, womit sich die Zugriffszeit auf 700 ns verlängert. Wird während des Nachladevorgangs ein Refresh des DDR-RAMs vom DRAM-Controller ausgelöst, verlängert sich der Zugriff um weitere 4 System-Takte bzw. 80 ns.

8. Verifikation im FPGA

In der Simulation werden die Durchlaufzeiten der Gatter nicht berücksichtigt. Um herauszufinden, ob die erstellte Schaltung mit der gewünschten Frequenz betrieben werden kann, wird diese im Cyclone III FPGA des Entwicklungsboards verifiziert. Dazu muss zunächst das fertige Design der Software Quartus II kompiliert werden. Dies erfolgt in mehreren Schritten.

Im ersten Schritt wird der VHDL-Code analysiert und in eine Gatternetzliste umgewandelt. Im nächsten Schritt müssen die Pins des FPGAs konfiguriert werden. Dazu wird die beim Entwicklungsboard mitgelieferte Pin-Beschreibung verwendet. Anschließend werden mit dem „Fitter“ die benötigten Gatter des FPGAs verdrahtet. Zuletzt erstellt der Assembler eine Konfigurationsdatei, die in den FPGA geladen werden kann. Nach dem Kompilieren kann der Software entnommen werden, welche Ressourcen des FPGAs für den Cache benötigt werden. Diese sind in *Abbildung 8.1* aufgelistet.

	Logik-Zellen	M9K-Blöcke	Pins	PLLs
CACHE_TOP	3805	27	44	1
CACHE_CONTROL	154	0	0	0
DDR_RAM_LOADER	73	0	0	0
DDR_CONTROLLER	2523	3	42	1
CACHE_MUX	131	0	0	0
CACHE_BLOCK (0)	226	6	0	0
CACHE_BLOCK (1)	225	6	0	0
CACHE_BLOCK (2)	225	6	0	0
CACHE_BLOCK (3)	226	6	0	0
RESET_LOGIC	33	0	1	0
SIRIUS_WRAPPER	20	0	0	0
OSZILLATOR	0	0	1	0

Abbildung 8.1: Verwendete Ressourcen des FPGAs

Um das Cache-System zu testen, wurde mit der SIRIUS IDE ein Programm geschrieben, das zuerst den kompletten DDR-RAM beschreibt und anschließend ausliest, wobei die gelesenen Daten überprüft werden. Während des Vorgangs kommt es dabei auch dazu, dass der Programmcode in den DDR-RAM geschrieben und wieder ausgelesen wird. Nach einem erfolgreich durchlaufenen Test wird eine bestimmte Melodie abgespielt.

9. Technische Daten

- 16 KB Cache-Speicher 4-fach assoziativ
- Wahlweise 16- oder 32-Bit-Datenbus
- Maximal 32-Bit-Adressbus (bis zu 4 GB Speicher adressierbar)
- DDR-RAM-Controller
- 100 MHz Betriebsfrequenz
- PLL zur Erzeugung der Taktsignale
- 50 MHz Taktgeber für die Systemfrequenz
- Resetlogik für das Mikrocontrollersystem
- Lese- und Schreib-Zugriffe auf Daten im Cache in einem System-Takt (20 ns)
- Lese- und Schreib-Zugriffe auf Daten im Hauptspeicher in 23 Takten ohne „write-back“ (460 ns) bzw. 35 Takten mit „write-back“ (700 ns)

10. Quellen

[1] Dirk Jansen, Nidal Fawaz, Daniel Bau, Marc Durrenberger: „A Small High Performance Microprocessor Core SIRIUS For Embedded Low Power Designs, Demonstrated in a Medical Mass Application Of an Electronic Pill (ePille®)“

Embedded System Design Topic, Techniques&Trends, p. 363-372 2007, USA Hanser Verlag, Deutschland, ISBN 978-0-387-72257-3

[2] Marc Durrenberger: „Überarbeitung eines 32 Bit Prozessor-Kerns mit Optimierung der ALU für die Synthese in eine 0.35 µm CMOS Bibliothek und Erprobung in einem FPGA“

Diplomarbeit, Hochschule Offenburg

[3] Deutschsprachige Wikipedia: „Cache“

http://de.wikipedia.org/wiki/Cache

[4] Bernd Becker, Paul Molitor: „Technische Informatik“

Oldenbourg Verlag München Wien, ISBN 978-3-486-58650-3

[5] John L. Hennessy, David A. Patterson: „Rechnerarchitektur“

Vieweg Verlag, ISBN 3-528-05173-6

[6] „DDR and DDR2 SDRAM High Performance Controller User Guide“

http://www.altera.com/literature/ug/ug_ddr_sdram.pdf