

## **Case Study: Xilinx MicroBlaze Microcontroller**

Stefan Böck (es06m102@technikum-wien.at)

### **Kurzfassung**

In diesem Artikel wird die Architektur des Xilinx MicroBlaze 32bit Mikroprozessors erläutert und mit der Architektur herkömmlicher Mikroprozessoren verglichen. Nach einer kurzen Einführung in die Architektur des Xilinx MicroBlaze Mikroprozessors werden Architekturmerkmale wie Speicherstruktur, Befehlssatz, Pipelining, Cache, Stack, Exception Handling und der Floating Point Unit herausgearbeitet.

Es wird auch auf allgemeine Architektur Probleme, wie zum Beispiel Pipeline Hazards, eingegangen und erläutert wie diese vom Xilinx MicroBlaze Mikroprozessor behandelt und gelöst werden.

Die theoretischen Hintergründe werden nur soweit behandelt inwieweit es für den Vergleich zwischen Xilinx MicroBlaze und herkömmlichen Mikroprozessoren notwendig ist. Ist tieferes theoretisches Hintergrundwissen notwendig wird auf die jeweilige Fachliteratur referenziert.

### **Stichwörter**

Xilinx MicroBlaze, Soft IP Core, Architektur, Befehlssatz, Pipelining, Hazards, Cache, Exception, Stack, Floating Point Unit

## 1. Einleitung

Immer mehr Mikrocontroller werden durch leistungsfähige PLDs wie zum Beispiel FPGAs ersetzt. Immer mehr Firmen steigen in den FPGA Markt ein und bieten ihren Kunden IP Cores für FPGA Bausteine. Diese IP Cores umfassen ein weites Spektrum an Anwendungen wie z.B. RS232 Controller, Ethernet-Controller uvm. Ein großes Feld bilden auch die IP Cores von Mikroprozessoren. Große Firmen wie Xilinx und Altera bieten Mikroprozessor Cores die in ihren FPGAs eingebettet werden können.

Der von Xilinx entwickelte Mikroprozessor Core MicroBlaze ist einer davon. Der MicroBlaze ist herkömmlichen Mikroprozessoren sehr ähnlich, besitzt jedoch auch Besonderheiten. Im folgenden Artikel werden diese Besonderheiten herausgearbeitet und auf die einzelnen Architekturmerkmale der MicroBlaze Familie eingegangen.

Um den Unterschied zwischen herkömmlichen Mikroprozessoren und dem Xilinx MicroBlaze herausarbeiten zu können wurde auf zwei Hauptliteraturquellen Bezug genommen. Informationen über allgemeine Prozessorarchitekturen wurden aus [WST03] entnommen und Informationen bezüglich dem Xilinx MicroBlaze aus[XIL06].

## 2. Xilinx MicroBlaze Architektur Überblick

Der Xilinx MicroBlaze 32bit Mikroprozessor ist ein Reduced Instruction Set Computer (RISC) und wurde für den Einsatz in Xilinx FPGAs entwickelt und optimiert. Dieser wird natürlich immer weiter entwickelt und dadurch gibt es unterschiedliche Versionen. Die aktuelle Version ist v5.00a [XIL06] daher sind die folgenden Erläuterungen immer auf diese Version bezogen.

Die Abb. 1 zeigt das Blockschaltbild des MicroBlaze Prozessors.

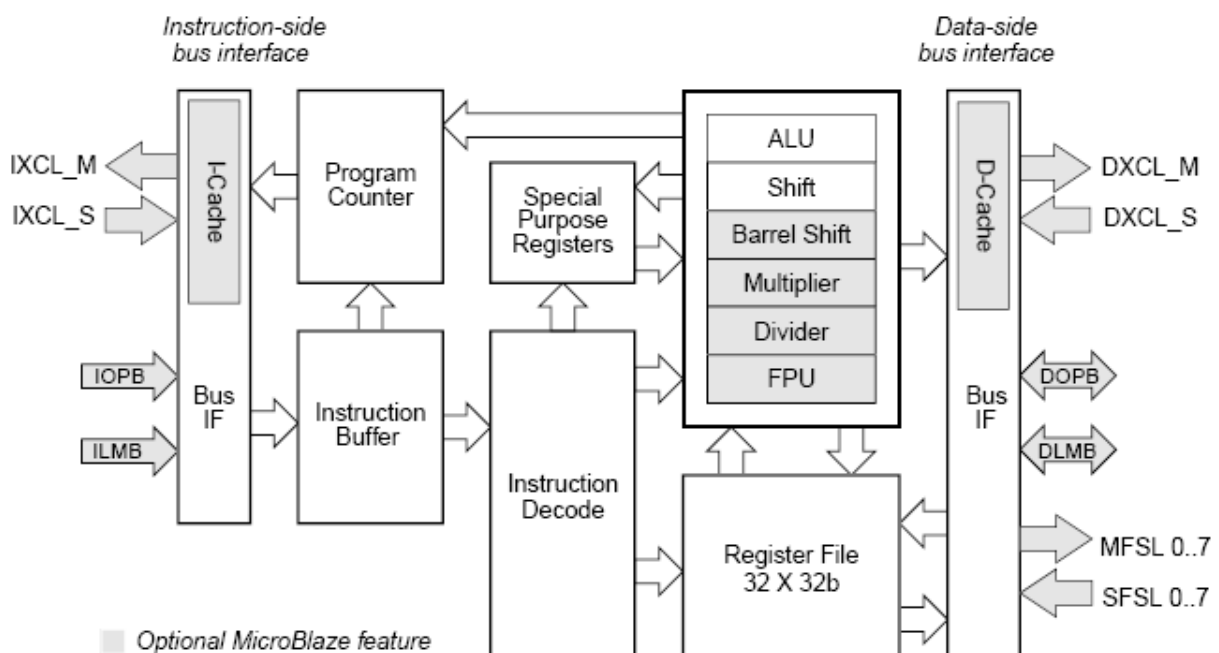


Abb. 1 Blockschaltbild des Xilinx MicroBlaze Prozessors [XIL06]

Wie man aus der Abb. 1 sehen kann ist die Architektur des Xilinx MicroBlaze der eines normalen Mikroprozessors (z.B. Infineon C167CR [INF03]) sehr ähnlich. Die grau hinterlegten Bereiche sind optionale Module die vom Entwickler konfiguriert werden können. Eine genauere Liste der optionalen Module kann aus [Xil06] entnommen werden.

Key Features des Xilinx MicroBlaze sind:

- 32x32bit General Purpose Register
- 32bit Befehlssatz
- 23bit Adressbus

In den weiteren Kapiteln dieses Dokumentes wird nun auf die einzelnen Elemente der Xilinx MicroBlaze Architektur eingegangen und mit denen eines herkömmlichen Mikroprozessor verglichen.

### 3. Speicher Architektur

So wie die meisten Mikroprozessoren besitzt auch der Xilinx MicroBlaze eine Harvard Architektur die einen physikalisch getrennten Daten- und Befehlspeicher besitzt.

Dadurch können gleichzeitig Daten und Befehl aus dem Speicher in die CPU Register geladen werden. Eine klassischen Von - Neumann Architektur hingegen, benötigt hierzu länger, da Daten und Befehl hintereinander aus dem Speicher geladen werden müssen. Weiters könnten bei der Harvard Architektur die Datenwortbreite und die Befehlswortbreite unterschiedlich lang sein, was jedoch beim Xilinx MicroBlaze nicht der Fall ist, da beide 32bit lang sind. Weiters besitzt der Xilinx MicroBlaze auch einen Daten- und einen Befehls Cache auf die in Kapitel 9.2 noch näher eingegangen wird.

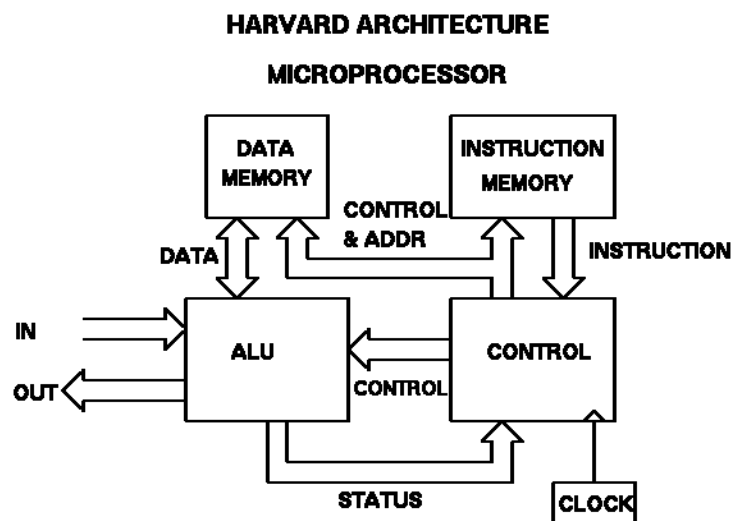


Abb. 2 Harvard Architektur [WST03]

Der Xilinx MicroBlaze Prozessor unterscheidet nicht zwischen Datenzugriff auf I/O und Zugriff auf einen Speicher. Dieser besitzt im Gegensatz zu herkömmlichen Mikrokontrollern, für den Speicherzugriff drei verschiedene Interfaces, die in den folgenden Kapiteln kurz beschrieben werden.

#### 3.1. On-Chip Peripheral Bus (OPB)

Der On-Chip Peripheral Bus wurde von IBM entwickelt. Der Bus besitzt eine Master Slave Architektur. Der Microblaze ist als Master konfiguriert und kann dadurch einen Datenaustausch mit

einem Slave initiieren. Slaves sind meist Peripheriebausteine wie Speicher, usw. Für nähere Informationen über den On-Chip Peripheral Bus können die OPB Spezifikationen [IBM01] von IBM herangezogen werden.

### 3.2. Local Memory Bus (LMB)

Der Local Memory Bus ist ein synchroner Bus der hauptsächlich dazu verwendet wird um auf ein lokales Block Ram zuzugreifen. Der LMB kommt mit nur wenigen Leitungen aus und kann sicherstellen, dass der Zugriff innerhalb nur eines Clock Zyklus geschieht.

So wie auch schon beim OPB geht der Datentransfer vom MicroBlaze aus. Es werden unterschiedliche Transfermodi unterstützt. Die in [Xil06] näher beschrieben werden.

### 3.3. Xilinx CacheLink (XCL)

Der Xilinx CacheLink ist im Gegensatz zum LMB ein Hochgeschwindigkeitsbus für den Zugriff auf externen Speicher. Nähere Informationen hierzu siehe [XIL06].

## 4. Datentypen

Der Xilinx MicroBlaze verwendet das Big-Endian Format. In diesem Format wird das höherwertige Byte bzw. Bit (MSB) in die niedrigere Adresse des Speichers bzw. Registers gespeichert. Demzufolge wird das niederwertige Byte bzw. Bit in die höhere Adresse gespeichert siehe Abb. 3.

Byte address	n	n+1	n+2	n+3
Byte label	0	1	2	3
Byte significance	MSByte			LSByte
Bit label	0			31
Bit significance	MSBit			LSBit

Abb. 3 Big-Endian Format [XIL06]

Weiters unterstützt der Xilinx MicroBlaze drei unterschiedliche Wortlängen.

Word (32bit), Half Word (16bit), Byte (8bit)

## 5. Befehlssatz

Um auf die Pipeline Architektur (Kapitel 8) näher eingehen zu können muss zuerst der Befehlssatz des Xilinx MicroBlaze identifiziert werden. Die Klassifizierung des Befehlssatzes wird meist anhand der Speicherung der Befehlsoperanden in der CPU getätigt.

**Stack:** Die Operanden werden implizit im Stack abgelegt.  
**Akkumulator:** Der Operand ist implizit der Akkumulator.  
**Registerset:** Die Operanden sind explizit Register oder Speicherstellen.



Der Xilinx MicroBlaze stellt solche Befehle nicht zur Verfügung, d.h. die Stackprozeduren werden nur vom Xilinx MicroBlaze durchgeführt.

Der Stack wird durch drei Adressen angesprochen:

Stackpointer: Zeigt immer auf die aktuelle Position im Stack. Wird nun ein Element aus dem Stack geholt oder ein Element in den Stack gespeichert wird der Stackpointer inkrementiert oder dekrementiert.

Stack Basis: Bestimmt die Adresse an der der Stack beginnt.

Stack Limit: Bestimmt die Adresse an der der Stack endet.

So wie bei fast allen Mikrocontrollern ist auch beim MicroBlaze der Stackspeicher verkehrt organisiert, d.h. die Stack Basis ist die niederwertige Adresse als das Stack Limit.

Abb. 4 zeigt die Stackorganisation des Xilinx MicroBlaze Mikrokontrollers.

High Address	
	Function Parameters for called sub-routine (Arg n ..Arg1) (Optional: Maximum number of arguments required for any called procedure from the current procedure.)
Old Stack Pointer	Link Register (R15)
	Callee Saved Register (R31...R19) (Optional: Only those registers which are used by the current procedure are saved)
	Local Variables for Current Procedure (Optional: Present only if Locals defined in the procedure)
	Functional Parameters (Arg n .. Arg 1) (Optional: Maximum number of arguments required for any called procedure from the current procedure)
New Stack Pointer	Link Register
Low Address	

Abb. 4 Xilinx MicroBlaze Stackorganisation [XIL06]

Der grau hinterlegte Teil in Abb. 4 zeigt den Stack eine Funktion. Ruft diese Funktion nun eine andere Funktion auf wird der in Abb. 4 weiß dargestellte Stack Frame erzeugt usw.

## 7. Reset, Interrupts, Exceptions und Breaks

Der Xilinx MicroBlaze unterstützt Resets, Interrupts, User Exceptions, Breaks und Hardware Exceptions. Die Priorität ist folgendermaßen gegliedert:

1. Reset
2. Hardware Exception
3. Non-maskable Break
4. Break
5. Interrupt
6. User Exceptions (Vectors)

Der Reset, Interrupts, Exceptions und auch Breaks beeinflussen das Pipelining negativ. In den folgenden Punkten wird beschrieben wie der Xilinx MicroBlaze damit umgeht.

### 7.1. Reset

Wird beim Xilinx MicroBlaze ein Reset durchgeführt, wird die Pipeline gelöscht und der erste Befehl an der Reset Vektoradresse (0x0) geladen. Hierbei wird der Programcounter und andere wichtige Register wie MSR (Machine Status Register) und ESR (Exception Status Register) resetiert. Dies beeinflusst das Pipelining wenig, da die Programmausführung ohnedies von vorne beginnt.

### 7.2. Hardware Exceptions

Kommt es zu einer Hardware Exception wird schon wie beim Reset die Pipeline gelöscht und zu einer Vektoradresse gesprungen. Jedoch wird auch der aktuelle Programcounter vor dem Sprung in ein Special Purpose Register (R17) geladen um nach der Abarbeitung der Exception wieder beim normalen Programm weiterarbeiten zu können.

Es werden Exceptions wie: Instruction Bus Exception, Illegal Opcode Exception, Data Bus Exception, Division durch Null und Exception der Floating Point Unit unterstützt und behandelt.

### 7.3. Breaks

Es werden zwei Arten von Breaks unterstützt. Software Breaks (intern) und Hardware Breaks (extern). Softwarebreaks werden durch bestimmte Befehle ausgeführt und Hardwarebreaks durch eine steigende Flanke an der Ext\_NM\_BRK Leitung. Bei einem Break wird der aktuelle Befehl noch ausgeführt und erst dann der Sprung durchgeführt. Dadurch ist die Zeit bis der Break abgearbeitet wird nicht immer gleich und ist abhängig vom aktuell ausgeführten Befehl (Latenzzeit).

### 7.4. Interrupts

Der Xilinx MicroBlaze unterstützt einen externen Interrupt. Dieser kann genau wie bei herkömmlichen Mikrokontrollern über das Interrupt Enable Bit (IE) ein- bzw. ausgeschaltet werden. Wie schon bei den Breaks wird beim Eintritt eines Interrupts der aktuelle Befehl, der sich in der Ausführungsstufe der Pipeline befindet, ausgeführt und erst dann der Sprung zum Interrupt Vektor durchgeführt. Während der Ausführung der Interrupt Service Routine wird das IE Bit auf 0 gesetzt, es

ist also kein weiterer Interrupt möglich. Es werden auch alle Interrupts ignoriert, wenn gerade ein Break oder eine Hardware Exception abgehandelt wird (siehe Prioritäten Kapitel 7).

Die Zeit zwischen Eintritt des Interrupts und dem Ausführungsbeginn der Interrupt Service Routine ist genau so wie bei den Breaks von dem aktuell ausgeführten Befehl abhängig.

## 7.5. User Exceptions (Vectors)

Durch einen bestimmten Befehl können User Exceptions ausgeführt werden. Wird dieser Befehl ausgeführt wird an eine bestimmte Vector Adresse gesprungen und der Service Code ausgeführt.

## 8. Pipeline Architektur

### 8.1. Pipeline Allgemein

Um die Befehle ausführen zu können werden diese in Teilschritte zerlegt und in eine so genannten Pipeline geladen. Damit Pipelining überhaupt funktionieren kann müssen diese Teilschritte gleich groß sein. Abb. 5 zeigt eine 4-stufige Pipeline:

- A. Befehlscode laden
- B. Befehlscode decodieren und Daten laden
- C. Befehl ausführen
- D. Ergebnis zurückgeben

Befehlsnummer	Zyklus 1	Zyklus 2	Zyklus 3	Zyklus 4	Zyklus 5	Zyklus 6	Zyklus 7
4				A	B	C	D
3			A	B	C	D	
2		A	B	C	D		
1	A	B	C	D			

Abb. 5 Allgemeine Befehlspipeline

Somit können Befehle parallel zueinander ausgeführt werden was wiederum die Ausführungsgeschwindigkeit erhöht. Pipelining stellt aber auch Probleme dar die nun kurz erklärt werden.

### Strukturprobleme (Struktural Hazards)

Structural Hazards treten immer auf wenn es zu Ressource Problemen kommt. Zum Beispiel wenn der Mikrocontroller nur ein Interface zum Speicher besitzt und in der Pipeline im selben Zyklus vom Speicher gelesen (A) und gleichzeitig geschrieben(D) werden muß. Wie in Abb. 6 zu sehen kann dadurch in Zyklus 4 nicht gleichzeitig gelesen und geschrieben werden und die Pipeline führt einen „Stall“ aus.

Befehlsnummer	Zyklus 1	Zyklus 2	Zyklus 3	Zyklus 4	Zyklus 5	Zyklus 6	Zyklus 7
4				STALL	A	B	C
3			A	B	C	D	
2		A	B	C	D		
1	A	B	C	D			

Abb. 6 Structural Hazards durch Speicherzugriff

### Datenprobleme (Data Hazards)

Data Hazards treten auf, wenn zum Beispiel Registerdaten in einem Befehl beschrieben werden und der darauffolgende Befehl benötigt die Daten auch, bekommt jedoch die falschen, da der erste Befehl die Daten noch nicht zurückgeschrieben hat.

Befehlsnummer	Zyklus 1	Zyklus 2	Zyklus 3	Zyklus 4	Zyklus 5	Zyklus 6	Zyklus 7
...				A	B	C	D
...			A	B	C	D	
SUB R5, R4, R1		A	B	C	D		
ADD R1, R2, R3	A	B	C	D			

Abb. 7 Pipeline Data Hazard

In Abb. 7 ist zu sehen, dass der SUB Befehl die Daten aus dem Register liest (Zyklus 2) bevor der ADD Befehl den richtigen Wert zurückschreibt (Zyklus 4). Der SUB Befehl rechnet also mit dem falschen Wert weiter.

Um dies zu verhindern wird die sogenannte „Forwarding“ Prozedur verwendet. Hierbei werden gewisse Befehle vor andere in die Pipeline gereiht, so dass Data Hazards verhindert werden können. Für nähere Informationen siehe [GMP05].

### Verzweigungsprobleme (Control Hazards)

Control Hazards treten bei bedingten Verzweigungen auf. Da erst nach der Ausführung der Verzweigung klar ist wo die Programmausführung weiter geht muss die Pipeline „gestallt“ werden. Es gibt Möglichkeiten diese Stalls möglich kurz zu halten. Diese sind in [GMP05] beschrieben.

## 8.2. Pipeline Xilinx MicroBlaze

Der Xilinx MicroBlaze beinhaltet eine 5 stufige Pipeline. Die 5 Stufen sind:

- A. Fetch (FT)
- B. Decode (OF)
- C. Execute (EX)
- D. Access Memory (MEM)
- E. Writeback (WB)

Für die meisten Befehle benötigt die Pipeline für jede Stufe nur einen Taktzyklus. Daraus folgt das diese Befehle in 5 Taktzyklen abgearbeitet werden können. Es gibt jedoch auch Befehle die mehrer Taktzyklen pro Stufe benötigen. Dies wird durch sogenanntes „Stalling“ der Pipeline berücksichtigt.

	cycle 1	cycle 2	cycle 3	cycle 4	cycle 5	cycle 6	cycle 7	cycle 8	cycle 9
instruction 1	IF	OF	EX	MEM	WB				
instruction 2		IF	OF	EX	MEM	MEM	MEM	WB	
instruction 3			IF	OF	EX	Stall	Stall	MEM	WB

Abb. 8 Stalling der Xilinx MicroBlaze Pipeline [XIL06]

Wie in Abb. 8 ersichtlich wird für Instruktion 2, die insgesamt 3 Taktzyklen für den Speicherzugriff benötigt, die Instruktion mittels „Stall“ verzögert. Instruktion 3 kann daher erst mit Zyklus 8 auf den Speicher zugreifen, nachdem Instruktion 2 mit dem Speicherzugriff fertig ist.

Weiter kann die Verwendung von einem langsamen Speicher zu einer längeren Ausführungszeit führen da der Speicherzugriff dadurch langsamer ist. Dadurch würde die Effektivität der Pipeline stark beeinflusst werden. Um diesem Effekt entgegenzuwirken besitzt der Xilinx MicroBlaze sogenannte Prefetch Buffer. Befindet sich nun die Pipeline in einem Stalling Zustand, muss die Befehlsausführung nicht gestoppt werden, sondern es werden die Befehle in diesen Prefetch Buffer gespeichert. Beendet nun die Pipeline den Stalling Zustand können die Befehle direkt aus dem schnellen Prefetch Buffer gelesen werden und nicht aus dem langsamen Speicher.

## Pipelining von Verzweigungen (Branches)

Normalerweise werden die Befehle in der Fetch und Decode Stufe beim Erreichen einer Verzweigung gelöscht. Danach wird die Fetch Stufe mit dem Befehl an der Verzweigungsadresse neu geladen. Die MicroBlaze Pipeline benötigt hierzu insgesamt 5 Taktzyklen. 3 für die Ausführung der Abzweigung und 2 um die Pipeline neu zu füllen. Um diese Latenzzeit zu minimieren wurden für bestimmte Verzweigungsbefehle sogenannte „Delay Slots“ eingeführt.

Hierzu werden nicht Fetch und Decode Stufe beim Erreichen einer Verzweigung gelöscht, sondern nur die Fetch Stufe. D.h. der Befehl der sich in der Decode Stufe („Delay Slot“) befindet wird noch fertig ausgeführt. Dadurch wird das Neubefüllen der Pipeline von 2 Taktzyklen auf einen reduziert.

Meist gibt es diese Verzweigungsbefehle mit und ohne „Delay Slot“.

z.B.: Bedingte Verzweigung wenn rA und rB gleich sind:

ohne Delay:    **bne rA,rB**  
mit Delay:       **bned rA,rB**

## 9. Cache

### 9.1. Cache allgemein

Der Cache Speicher dient dazu dem Mikroprozessor einen sehr schnellen Speicher zur Verfügung zu stellen um Daten bzw. Befehle möglichst schnell aus dem Speicher holen zu können. Der Cache ist aus Kostengründen meist eher klein im Gegensatz zum Hauptspeicher, der groß aber dafür langsamer ist. In Abb. 9 ist zu sehen, dass sich der Cache zwischen Mikroprozessor und Hauptspeicher befindet.

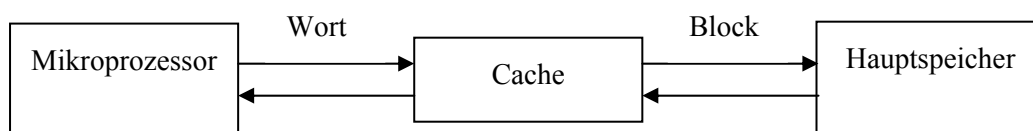


Abb. 9 Cache und Hauptspeicher [WST03]

Möchte nun der Mikroprozessor aus dem Speicher lesen wird zuerst im Cache nachgesehen, ob sich das angeforderte Wort im Cache befindet. Ist dies nicht der Fall wird ein Speicherblock aus dem

Hauptspeicher gelesen und in den Cache übertragen. Erst dann liest die CPU aus dem Cache die angeforderten Daten. Oder die Daten werden gleichzeitig zum Mikroprozessor und in den Cache transportiert.

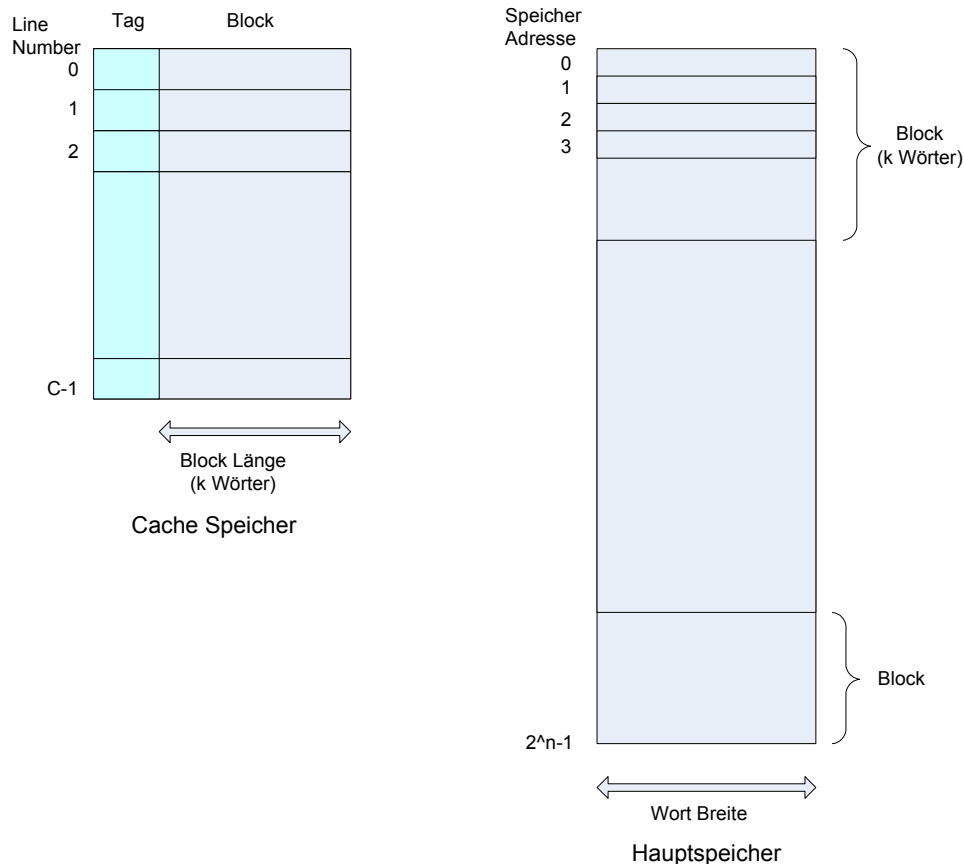


Abb. 10 Cache und Hauptspeicher Struktur [WST03]

Wie in Abb. 10 ersichtlich besitzt der Hauptspeicher  $2^n$  adressierbare Datenwörter. Diese sind in Blöcke aufgeteilt mit jeweils  $k$  Datenwörtern. Der Cache besteht aus  $C$  Zeilen. Jede Zeile enthält einen Block des Hauptspeichers, also  $k$  Datenwörter. Da der Cache viel kleiner als der Hauptspeicher ist können nie alle Blöcke gecached werden. Aus diesem Grund besitzt jede Zeile im Cache eine Tag Nummer um feststellen zu können welche Blöcke sich im Cache befinden. Weiters muss es bestimmte Algorithmen geben die bestimmen welche Blöcke in den Cache geladen werden sollen. Diese werden Mapping Algorithmen genannt und sollen jetzt kurz beschrieben werden.

### Cache Mapping Algorithmen

Es gibt drei verschiedene Cache Mapping Algorithmen: Direkt, Assoziativ und Set Assoziativ. Da der Xilinx MicroBlaze nur Direkt Mapping unterstützt wird nur dieser Algorithmus näher erklärt.

Direktes Mapping ist der einfachste Algorithmus. Hierbei wird jeder Block des Hauptspeichers immer in dieselbe Zeile des Cache Speichers geladen. Dies vereinfacht zwar den Algorithmus, jedoch wenn der Mikrocontroller auf Daten zugreifen will die sich in Blöcken befinden die wiederum in derselben Zeile des Caches gespeichert werden, muss die Cachezeile zwischendurch neu geladen werden.

## 9.2. Cache Xilinx MicroBlaze

Wie in Abb. 1 ersichtlich besitzt der Xilinx MicroBlaze zwei Cache Module. Einerseits den Daten- andererseits den Befehls-cache. Diese können vom Benutzer getrennt ein- bzw. ausgeschaltet werden. Als Cache Algorithmus wird Direkt Mapping verwendet, wie in 9.1 beschrieben

### Befehls-cache

Wenn der Befehls-cache aktiv ist wird der Speicher in zwei Segmente aufgeteilt: ein Segment in dem gecached wird und ein Segment in dem nicht gecached wird. Das Cacheable Segment wird über zwei Register gesetzt: C\_ICACHE\_BASEADDR und C\_ICACHE\_HIGHADDR

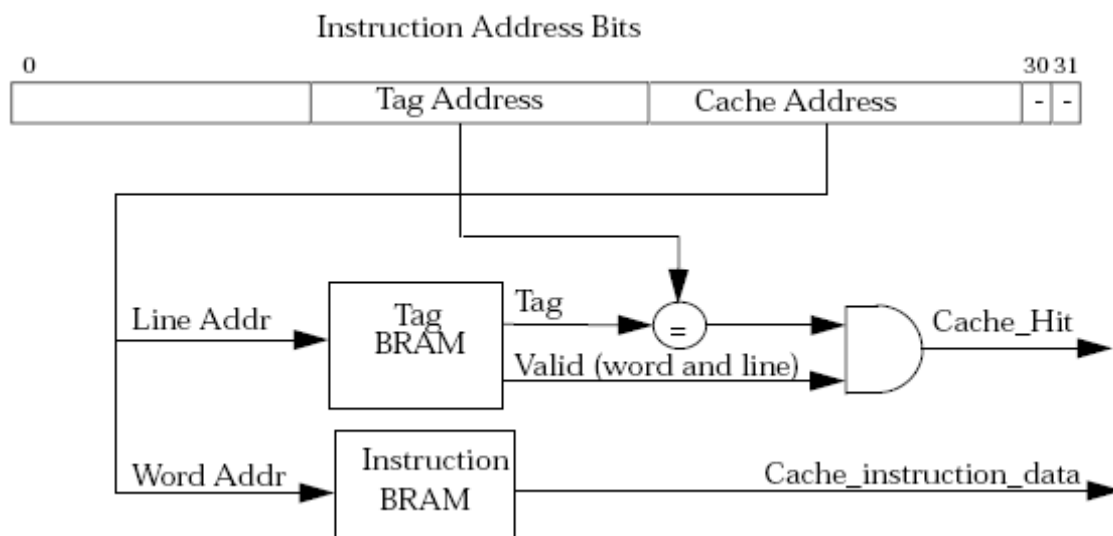


Abb. 11 Befehls-cache Organisation [XIL06]

Wie Abb. 11 zeigt besteht die Befehlsadresse des Cacheable Segment aus jeweils einer Cache- und einer Tag Adresse. Das Cacheable Segment kann zwischen 2kB und 64kB groß sein, also kann die Adresse zwischen 11 und 16bit lang sein. Die Cache Adresse und die Tag Adresse zusammen soll die Adresse des Cacheable Segmentes nicht überschreiten. D.h. wenn das Cacheable Segment 64kByte (16bit Adressen) groß ist und der Cache 4kB (12bit Adressen) groß ist bleiben für die Tag Adresse noch 4bit (16-12bit = 4bit) übrig.

Wird nun ein Befehl vom Mikroprozessor aus dem Speicher angefordert, wird geprüft ob die Befehlsadresse im Cacheable Segment liegt. Ist dies nicht der Fall wird das Request an den OPB (On-Chip Peripheral Bus, siehe 3.1) oder LMB (Local Memory Bus, siehe 3.2) weitergegeben.

Ist der aktuelle Befehl jedoch im Cacheable Segment wird im Tag Speicher nachgesehen ob der Befehl bereits gecached wurde oder nicht. Ist dies nicht der Fall wird der Befehl über das Xilinx CacheLink Interface in den Cache geladen und danach weiter verarbeitet.

### Datencache

Der Datencache des Xilinx MicroBlaze ist dem Befehls-cache sehr ähnlich. Jedoch unterstützt dieser im Gegensatz zum Befehls-cache ein Write-Through Protokoll. Werden nun vom Mikrokontroller Daten in den Speicher geschrieben, werden diese vom Cache Controller über das Xilinx Cache Link

Interface in den externen Speicher geschrieben. Wenn die Zieladresse auch im Cache vorhanden ist, wird dieser Wert auch neu beschrieben andernfalls bleibt der Cache unberührt.

Werden Daten vom Mikrokontroller aus dem Speicher gelesen, funktioniert es genauso wie beim Befehls-cache. Sind die Daten im Cache werden diese aus dem Cache geholt. Andernfalls werden diese über das Xilinx CacheLink Interface angefordert. Solange die Daten nicht in der Pipeline angekommen sind wird diese natürlich „gestallt“ siehe Kapitel 8.2.

## 10. Floating Point Unit (FPU)

Der Xilinx MicroBlaze besitzt wie in Abb. 1 ersichtlich zusätzlich zu den Cache Modulen auch noch eine optionale Floating Point Unit die vom Benutzer konfiguriert werden kann. Die Floating Point Unit oder Gleitkommaeinheit dient, wie der Name schon sagt, zur Verarbeitung von mathematischen Operationen bzw. Gleitkommazahlen. Allgemeine FPU's verarbeiten Operationen wie die Grundrechnungsarten, Logarithmus-, Wurzel- und Potenzrechnungen, trigonometrische Funktionen und Vergleiche von Zahlen (größer, kleiner, kleiner-gleich, usw.). Die Xilinx MicroBlaze FPU ist bei weitem nicht so leistungsfähig und verarbeitet nur folgende Operationen:

Grundrechnungsarten:

Addition, Subtraktion, Multiplikation und Division

Vergleiche:

Kleiner, gleich, kleiner-gleich, größer, ungleich, größer-gleich

Runden:

Runden auf die nächste ganze Zahl

Weitere Besonderheit:

Eine Zahl  $n$  die sich im Wertebereich  $(1.17549 \cdot 10^{-38} > n > 0)$ ,  $(0 > n > -1.17549 \cdot 10^{-38})$  befindet kann nicht mehr erkannt werden. Werden Operanden die sich in diesem Bereich befinden verwendet wird ein Errorflag im FSR Register gesetzt. Wird durch eine Operation ein Ergebnis in diesem Wertebereich geliefert, wird dieses auf 0 gesetzt und ein weiteres Flag im FSR gesetzt. Diese Besonderheit der MicroBlaze FPU ist jedoch meist nicht von Bedeutung, da sich diese Ungenauigkeit nur bei Werten in der Potenz  $10^{-38}$  auswirkt. (siehe Wertebereich)

## 11. Schlussfolgerungen

Wie man erkennen kann ist der Xilinx MicroBlaze herkömmlichen Mikroprozessoren sehr ähnlich, könnte also durchaus als Alternative zu herkömmlichen Mikrocontrollern herhalten. Jedoch besitzt der Xilinx MicroBlaze kaum Peripherie wie man es von Mikrocontrollern gewohnt ist. D.h. Um wirklich mit dem Xilinx MicroBlaze arbeiten zu können müssen entweder zusätzliche IP Cores verwendet werden, die jedoch teuer sein können, oder man muss die Peripherie selbst entwickeln, dass wieder rum mehr Zeit in Anspruch nimmt. Dies sollte bei der Entscheidung für oder gegen den Xilinx MicroBlaze auf jeden Fall beachtet werden.

## 12. Glossar und Abkürzungsverzeichnis

Central Processing Unit (CPU): Die CPU ist das Kernstück jedes Computers. Er steuert alle Geräte und Peripherie Elemente wie zum Beispiel Speicher uvm.

Field Programmable Gate Array (FPGA): Stammt aus der Familie der PLDs. Er beinhaltet logische Komponenten und programmierbare Verbindungen zwischen diesen Komponenten. Durch Programmierung der Verbindungen können unterschiedlichste Funktionen implementiert werden.

Last-In-First-Out (LIFO): Der Begriff LIFO wird meist bei Speichern verwendet. Er bezeichnet die Reihenfolge wie die Daten in den Speicher geladen werden und wie wieder herausgeholt werden. Den LIFO Speicher kann man mit einem Papierstapel vergleichen. Der Zettel der als Erster hingelegt wurde, wird als Letzter wieder weggenommen.

Programmable Logic Device (PLD): Elektronische Bauteile für integrierte Schaltungen. Anders als logische Gatter, die eine bestimmte Funktion haben, bekommen PLD erst durch ihre Programmierung eine benutzerdefinierte Funktionalität.

Reduced Instruction Set Computer (RISC): Ein Computer mit einem RISC Befehlssatz verzichtet auf komplexe Befehle und ist daher schneller in der Ausführung. Auch ein Interrupt wird schneller behandelt, da der aktuell ausgeführte Befehl nicht unterbrochen werden darf.

## 13. Literatur

- [IBM01] IBM On-Chip Peripheral Bus Spezifikation, Revision Datum: 01.04.2001, Publikationsnummer: SA-14-2528-02, [http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/9A7AFA74DAD200D087256AB30005F0C8/\\$file/OpbBus.pdf](http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/9A7AFA74DAD200D087256AB30005F0C8/$file/OpbBus.pdf)
- [WST03] William Stallings, Computer Organisation and Architecture, Sixth Edition, ISBN: 0-13-049307-4
- [XIL06] Xilinx MicroBlaze Prozessor Reference Guide, Version 6.0 Juni 2006, [http://www.xilinx.com/ise/embedded/mb\\_ref\\_guide.pdf](http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf)
- [GMP05] Computer Architecture Tutorial by Gurpur M. Prabhu, <http://www.cs.iastate.edu/~prabhu/Tutorial/title.html>
- [INF03] Infineon C167CR 16bit Mikrokontroller User Manual V3.2, 01.05.2003 [http://www.infineon.com/upload/Document/cmc\\_upload/documents/008/066/c167cr\\_um\\_v3.2\\_2003\\_05.pdf](http://www.infineon.com/upload/Document/cmc_upload/documents/008/066/c167cr_um_v3.2_2003_05.pdf)

## 14. Fragen

- (1) Welche Befehlsarchitektur besitzt der Xilinx MicroBlaze Mikroprozessor?
- (2) Welche allgemeinen Probleme treten beim Pipelining auf? Erklären Sie diese in Stichwörtern. Was sind mögliche Lösungen dafür?
- (3) Wie erfolgt der Zugriff auf Speicher beim Xilinx MicroBlaze Mikroprozessor?
- (4) Welche Arten von Cache Mapping gibt es? Welche Art wird vom Xilinx MicroBlaze verwendet?
- (5) Nach welchen Kriterien wird der Befehlssatz klassifiziert? Wie kann man den Befehlssatz des Xilinx MicroBlaze klassifizieren?
- (6) Wie erkennt der Xilinx MicroBlaze ob ein Befehl oder Daten aus dem Cache oder aus dem Hauptspeicher geladen werden können?
- (7) Angenommen es kommt in der Pipeline des Xilinx MicroBlaze zu einem „Stall“. Wie wird verhindert, dass durch langsamen Zugriff auf externen Speicher noch mehr Zeit vergeht bis sich der nächste Befehl in der Pipeline befindet?
- (8) Aus welchem Grund besitzt die Floating Point Unit des MicroBlaze Mikroprozessors eine Ungenauigkeit in kleinen Wertebereichen.