Konzeption einer Pipelinearchitektur für einen FPGA-basierten Mikrocontroller – Bericht über die erste Phase im WS2006/2007

Prof. Dr.-Ing. Rainer Bermbach

Fachbereich Elektrotechnik
Tel.: 0 53 31 / 93 93 111
R.Bermbach@FH-Wolfenbuettel.de

1 Einleitung

Moderne Hochleistungsprozessoren erreichen einen Großteil ihrer Verarbeitungsgeschwindigkeit durch den Einsatz von Pipelining in ihrer Architektur. Im Idealfall steigert sich die Rechenleistung um den Faktor der Anzahl der Pipelinestufen. Praktische Probleme, sog. Hasards, begrenzen allerdings diesen Anstieg deutlich. Im Bereich der CPU-Kerne von Mikrocontrollern findet Pipelining bislang kaum Einsatz und wenn, dann nur in Primitivform.

Im Rahmen des beantragten Forschungsprojekts soll eine erste Konzeption für den Einsatz von Pipelining mit einer bestehenden Mikrocontrollerarchitektur entwickelt werden, die als FPGA-basierte VHDL-Implementierung vorliegt [1, 2]. Langfristiges Ziel ist eine deutliche Leistungssteigerung bei vertretbarem Aufwand, der in sinnvollem Verhältnis zum sonstigen Ressourcenbedarf des Mikrocontrollers stehen muß. Für Probleme wie Struktur-, Daten- und Kontrollflußhasards sollen entsprechende Lösungen entwickelt werden.

Die Konzeption soll Basis einer späteren Implementation der Pipeline-CPU und der entsprechenden praktischen Evaluation sein. Aus diesem Grund sollen auch adäquate Strukturen in der Architektur konzipiert werden, die den Test der späteren Implementierung unterstützen. So soll das Konzept auch einen Anschluß der bereits früher entwickelten Debug-Einheit [3] vorsehen.

Die erste Phase konzentrierte sich auf den grundsätzlichen Aufbau der Pipeline, die Anzahl ihrer Stufen und die elementare Struktur sowie die Analyse des Befehlssatzes und der sich daraus ergebenden Anforderungen an die Implementierung der Pipeline.

2 Pipelining bei Mikroprozessoren

In klassischen Prozessordesigns werden die notwendigen Arbeitsschritte sequentiell hintereinander ausgeführt. Benötigt ein Prozessor also z.B. fünf Arbeitsschritte zur Ausführung eines Befehls, so dauert der Befehl i.a. fünf Takte.

Ausgehend von der Idee der Fließbandverarbeitung kann man auch bei Prozessoren eine Parallelverarbeitung einführen. Jede der fünf Stufen der Befehlsverarbeitung wird in der klassischen Realisierung ja nur jeden fünften Takt benötigt, die restliche Zeit ist sie ungenutzt. Lädt man nun jeden Takt einen neuen Befehl, so erhält jede

der Stufen in jedem Takt eine neue Aufgabe und es werden im eingeschwungenen Zustand fünf Befehle gleichzeitig bearbeitet [4, 5].

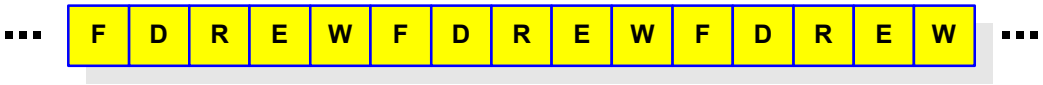

Bild 1: Sequentielle Abarbeitung der Befehlsschritte

In Bild 1 ist die sequentielle Abfolge der Verarbeitung beim klassischen Design angedeutet. Jedes Kästchen (jeweils ein Taktzyklus lang) entspricht einem der fünf Verarbeitungsschritte. In F liest der Prozessor einen neuen Befehl aus dem Programmspeicher (Program Fetch). Im folgenden Schritt decodiert er, was eigentlich wann zu tun ist (Instruction Decode). Für die Ausführung sind meist auch Operanden zu laden – Schritt R (Operand Read), die anschließend verarbeitet werden – Schritt E (Execute). Im letzten Schritt W erfolgt das Zurückschreiben der Ergebnisse (Write Back). Im Bild sind also drei Befehle angegeben.

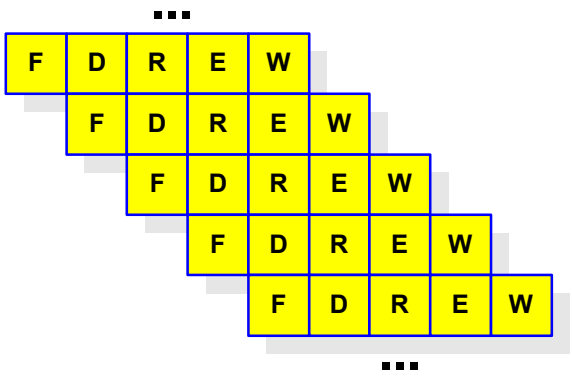

Bild 2: Parallele Befehlsverarbeitung beim Pipelining

Die parallele Verarbeitung in einer Pipelinearchitektur zeigt Bild 2. Da in jedem Takt ein neuer Befehl gelesen (F), decodiert (D), die Operanden gelesen (R), das Ergebnis berechnet (E) und zurückgeschrieben wird (W), sind alle Einheiten ständig beschäftigt, und im Mittel werden in fünf Takten fünf Befehle bearbeitet im Gegensatz zu einem in der sequentiellen Form. Der Geschwindigkeitsgewinn erreicht also den Faktor fünf.

Leider läßt sich diese Leistungssteigerung in der Realität nicht erreichen. Sog. Hasards begrenzen die theoretisch erzielbare Geschwindigkeit.

Strukturhasards entstehen, wenn Ressourcen in verschiedenen Schritten benötigt werden. Da diese nun parallel ablaufen, besteht das Problem, wer darf jetzt die Ressource nutzen und was macht die andere Einheit? Im allgemeinen gibt es nur die Möglichkeit, in der Hardware exklusive Ressourcen vorzusehen, um Strukturhasards aufzulösen, d.h. zusätzliche Hardware wird benötigt. Ein Beispiel ist der Speicher, der die Operanden enthält und die Ergebnisse aufnimmt, was in der Pipelineversion wegen der parallelen Verarbeitung gleichzeitig möglich sein muß. Üblicherweise realisiert man das mit sog. Dual-Port Memories, Speichern, die zwei Schnittstellen besitzen und deshalb gleichzeitiges Lesen und Schreiben erlauben.

Datenhasards können entstehen, da die Verarbeitung der vorangegangenen Daten noch nicht abgeschlossen ist, wenn die nächsten Befehle u.U. bereits mit diesen Daten arbeiten sollen. Speziell sog. Read-After-Write-Hasards (RAW) führen in einer Pipelinestruktur zu Fehlern, wenn sie nicht erkannt und behandelt werden. Betrachtet man den ersten Befehl in Bild 2, so schreibt dieser sein Ergebnis in W in den Speicher. Benötigt der folgende Befehl dieses Ergebnis, so erwartet er es aber bereits in R im Speicher. Ohne weitere Maßnahmen liest der zweite Befehl also den vorigen Wert und nicht das erst in W gespeicherte neue Ergebnis. Diese Fälle müssen daher erkannt und die Pipeline entsprechend gesteuert werden. Es besteht die Möglichkeit, die Pipeline anzuhalten (Pipeline Stall), was Verluste bedeutet, oder sog. Data Forwarding einzusetzen. Letzteres transportiert die am Ende von E des ersten Befehls verfügbaren Daten auf direktem Weg (Bypass) zum Eingang der Verarbeitungseinheiten in E für den zweiten Befehl. Auf diese Weise kann ein Anhalten der Pipeline vermieden werden.

Die dritte und „unangenehmste“ Gruppe von Pipelinefehlern sind die Kontrollflußhasards. Sie entstehen durch Sprünge und andere Arten von Verzweigungen (Unterprogrammaufrufe, Unterprogrammrückkehr, Interrupts). Ist in Bild 2 der erste Befehl eine Verzweigung, so ist dies frühestens am Ende von D, bei berechneten Sprungzielen oder bedingten Sprüngen u.U. erst am Ende von E bekannt. Bis dahin sind bereits ein bis drei Befehle in der Pipeline bearbeitet worden, die aber von der falschen Adresse gelesen wurden. Sie müssen verworfen und die Pipeline mit neuen Befehlen von der Programmstelle des Sprungziels gefüllt werden. Jeder Sprung hat daher einen Neuanlauf der Pipeline zur Folge, so daß im Beispiel erst nach fünf Takten wieder pro Takt ein Befehl beendet wird. Viele Kontrollflußverzweigungen reduzieren den Pipelinegewinn also drastisch. Abhilfe ist nur schwer und meist nur auf statistischem Wege möglich (Sprungvorhersage etc.), weshalb insbesondere bei den sehr langen Pipelines moderner Hochleistungsprozessoren immenser Aufwand getrieben wird, um den Gewinn durch die Pipeline nicht wieder zu verlieren.

3 Grundstruktur der Pipeline des Mikrocontrollers

Die bestehende Architektur des Mikrocontrollers [1, 2] ist weitgehend kompatibel zur PICmicro Mid-Range Family der Firma Microchip [6]. Sie weist ein rudimentäres Pipelinekonzept auf, wie Bild 3 zeigt.

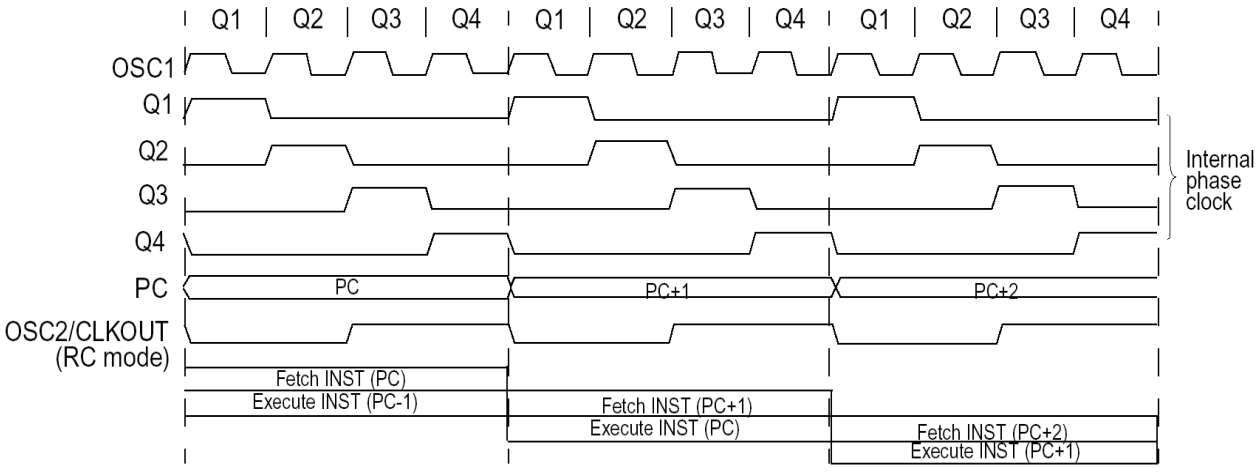

Bild 3: Befehlsverarbeitung des PIC [6]

In den ersten vier Takten liest sie den Befehl ein (Fetch) und in den folgenden vier führt sie die eigentliche Verarbeitung durch (Decode in Q1, Read Operands in Q2, Execute in Q3 und Write Back in Q4). Es sind also immer zwei Befehle in der Verarbeitung.

Ausgehend von dieser Arbeitsteilung bei der sequentiellen Architektur bietet es sich für die Pipelineversion des PIC an, mit einer fünfstufigen Pipeline (F D R E W) zu arbeiten (vgl. Bild 2). Auch erscheint eine solche Arbeitsteilung grundsätzlich balanciert: Die einzelnen Stufen einer Pipeline sollten möglichst gleich lange Verzögerungen besitzen, da sich dadurch die höchste Performance erreichen läßt. Detailliertere Aussagen zur Balancierung lassen sich erst mit tatsächlichen Implementierungen und entsprechenden Messungen gewinnen. Erst dann könnte sich auch die Notwendigkeit ergeben, die Pipeline weiter zu unterteilen, d.h. die Anzahl der Stufen zu erhöhen.

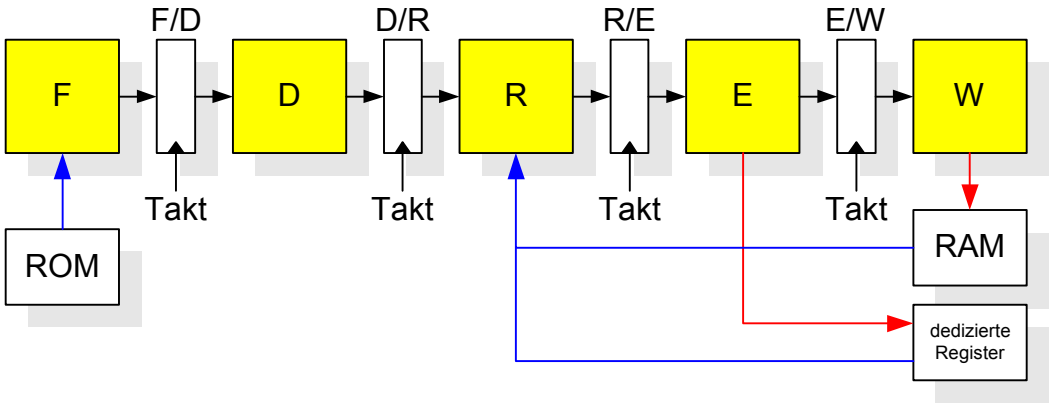

Bild 4: Elementarstruktur der Pipelineversion

Den grundsätzlichen Aufbau der sich ergebenden fünfstufigen Pipeline zeigt Bild 4. Da in jedem Takt in jede Stufe neue Informationen übernommen werden, sind zwischen den Stufen Register eingefügt. Sie sind nach ihrer Position benannt, d.h. das Register zwischen F und D heißt F/D, das zwischen D und R D/R usw. Aufgabe der Register ist es, sämtliche Daten und Steuerinformationen aus der vorhergehenden Stufe für alle nachfolgenden Stufen zu speichern. Beispielsweise liefert D ja sämtliche Steuerinformationen für alle nachfolgenden Stufen, auch wenn sie erst einige Takte später benötigt werden. So müssen die Schreibadresse und das Schreibsignal für W bereits in D/R gespeichert werden und mit jedem Takt über R/E und E/W an W weitergeleitet werden. Weiterhin stellt z.B. R über das Register R/E der Stufe E die gelesenen Operanden zur Verfügung. Die Register sind also unterschiedlich breit, manche Signale gelangen nur zur Nachbarstufe, während andere über mehrere Register zur entsprechenden Stufe gelangen.

Die auftretenden Hasards machen spezielle Vorkehrungen notwendig. Aufgrund der Harvard-Architektur und des eigenen PC-Inkrementers schon in der Originalversion treten Strukturhasards nur beim RAM auf, d.h. dieses muß als Dual-Port-Speicher ausgeführt sein.

Für Datenhasards muß u.a. die Pipeline angehalten werden können, allerdings nicht die gesamte Verarbeitung, sondern nur alle Befehle ab dem, der noch auf das Zurückschreiben eines Ergebnisses wartet. Die entsprechenden Register erhalten solange kein Enable und behalten ihre Werte (Ausnahme R/E). Die Befehle davor müssen weiter laufen, damit Ergebnisse erzeugt und gespeichert werden können. Im Re-

gister R/E werden allerdings die Steuersignale für ein NOP (No Operation) gespeichert, damit die bereits durchgeführte Operation nicht nochmals erfolgt und das Ergebnis verfälschen würde.

Kontrollflußhasards werden in D erkannt. Der folgende Befehl, dessen Befehlscode schon in das Instruction Register gelesen wurde, wird durch den Code für NOP (No Operation) ersetzt. Mit dem neuen Wert des Program Counter liest F des nächsten Befehls den jetzt gültigen Befehl an der neuen Adresse. Bei bedingten oder berechneten Sprüngen ist das Vorgehen etwas komplizierter.

4 Befehlssatzanalyse

Der Befehlssatz des Mikrocontrollers besteht aus 35 Befehlen [6]. Wie schon angedeutet, müssen bei bestimmten Befehlen oder Befehlsfolgen Maßnahmen ergriffen werden, damit die Pipeline keine falschen Werte erzeugt. Grundsätzlich müssen alle Arten von Datenhasards und alle möglichen Kontrollflußhasards erkannt werden.

4.1 Betrachtung von Datenbefehlen

Datenhasards treten auf, wenn ein Wert gelesen wird (Auslösebefehl), der von einem vorhergehenden Befehl (Bedingungsbefehl) noch nicht geschrieben wurde. Eine Analyse aller Befehle bzw. Befehlskombinationen liefert eine Reihe verschiedener, möglicher Datenhasards (s. Tabelle 1).

Tabelle 1: Mögliche Datenhasards [7]

Nummer	Bezeichnung	auslösende Befehlsfolge(n)
1.1	Adressierungshazard	`write RP / access F direct`
1.2.1	Adressierungshazard	`write FSR / access F indirect`
1.2.2		`write IRP / access F indirect`
2.1.1	RAW-Hazard	`write F / read F`
2.1.2	RAW-Hazard	`write FF / read FF`
2.2	RAW-Hazard	`write W / read W`
3.1.1	Flag-Hazard	`rotation / rotation`
3.1.2	Flag-Hazard	`calculation / rotation`
3.1.3	Flag-Hazard	`write STATUS / rotation`
3.2.1	Flag-Hazard	`rotation / read carry`
3.2.2	Flag-Hazard	`calculation / read flag`
3.2.3	Flag-Hazard	`write STATUS / read flag`
4.1.1	Sprunghazard	`write PCLATH / CALL`
4.1.2		`write PCLATH / GOTO`
4.2	Sprunghazard	`write PCLATH / write PCL`
5.1.1	Peripheriehazard	`write PORTx / read PORTx`
5.1.2		`write TRISx / read PORTx`
5.1.3		`write PORTB / read INTF`
5.1.4		`write TRISB / read INTF`
5.2	Peripheriehazard	`write TXREG / read TXIF`

In Gruppe 1 sind sog. Adressierungshasards aufgeführt. Sie entstehen durch Eigenheiten der PIC-Architektur. Der Zugriff auf das interne RAM erfolgt in vier Bänken.

Welche Bank aktuell adressiert ist, wählen zwei Bits (RP) im Statusregister aus. Ein Ändern dieser Bits und ein folgendes Lesen im RAM führt folglich zu einem Hasard, wenn der Schreibbefehl für die RP-Bits noch nicht abgeschlossen ist. Bei der indirekten Adressierung des RAMs enthält das Bit IRP das MSB der Adresse und Register FSR die anderen Adreßbits. Ein noch nicht abgeschlossener Schreibbefehl auf IRP oder FSR erzeugt folglich einen Hasard bei anschließender indirekter Adressierung.

Gruppe 2 enthält die klassischen Read-After-Write-Hasards, d.h. noch nicht abgeschlossener, schreibender Zugriff auf eine Adresse in RAM, dedizierten Registern oder dem W-Register des PIC und anschließendes Lesen. Ganz ähnlich verhält es sich mit den sog. Flag-Hasards der Gruppe 3. Die Flags des Prozessors werden bei Rotationsbefehlen und bei Zugriff auf das Statusregister gelesen. Vorherige Änderungen durch Berechnungen, Rotationsbefehle oder Schreibzugriff auf das Statusregister, die noch nicht gespeichert sind, verursachen Hasards.

In Verbindung mit den Kontrollflußbefehlen CALL, GOTO und direktem Schreiben des unteren Teils (PCL) des Program Counter entstehen Hasards (Gruppe 4), wenn die oberen Bits des Program Counter (PCLATH) vorher geschrieben und noch nicht gespeichert wurden. Weiterhin verursachen bestimmte Folgen von Peripheriezugriffen auf Ports und serielle Schnittstelle mögliche Datenhasards (Gruppe 5).

Interessant ist bei all diesen Hasards, welchen Abstand die einen potentiellen Hasards bedingenden Befehle von den tatsächlich auslösenden haben können. Eine genauere Untersuchung zeigt, daß die Befehlskombinationen in Tabelle 1 Abstände von 0 bis 2 besitzen, d.h. einige Hasards treten aufgrund ihrer speziellen Konstellation in der Architektur praktisch gar nicht auf (Abstand 0), während in anderen Fällen die beiden aufeinanderfolgenden Befehle betrachtet werden müssen (Abstand 1). Bei Abstand 2 wirkt sich der Bedingungsbefehl auf den nächsten und den übernächsten Befehl aus. Die tatsächlich möglichen Hasards und ihr Abstand müssen bei der Erkennung der Hasards einfließen und entsprechend lange Pipeline Stalls oder Steuerung des Data Forwarding erzeugen. Für letzteren Fall müssen also Ergebnisse von maximal zwei Befehlen vorher gespeichert werden, um sie bei Bedarf direkt der Execution-Einheit (ALU) zuzuführen. Dafür sind weitere Multiplexer am Eingang erforderlich. Bislang erfolgt die Auswahl am ALU-Eingang zwischen Werten aus dem RAM und Konstanten aus dem Instruction Register. Bei Einsatz von Data Forwarding muß zusätzlich zwischen einem der beiden letzten Ergebnisse oder dem Inhalt des Statusregisters gemultiplext werden.

4.2 Kontrollflußbefehle

In der Gruppe 5 (s. Kapitel 4.1) waren schon mögliche Probleme mit Kontrollflußbefehlen betrachtet worden. Zusätzlich erzeugen diese Befehle auch alle Kontrollflußhasards. Der PIC-Befehlssatz kennt die folgenden Verzweigungsbefehle:

- GOTO – unbedingter Sprung
- CALL – Unterprogrammaufruf
- RETURN, RETLW, RETFIE – Rückkehr aus Unterprogramm/Interrupt
- INCFSZ, DECFSZ, BTFSC, BTFSS – bedingte Sprünge (Conditional Jump)
- Schreiben auf PCL (PC low) – berechneter Sprung (Computed Jump)
- Interrupt – ähnlich wie CALL auf Adresse 0004h

Die Erkennung dieser Befehle erfolgt immer in Schritt D (Decode). GOTO, CALL und Interrupt laden den Program Counter neu und in der Pipeline muß das Instruction Register mit NOP geladen werden, da der zuvor gelesene Befehl ja von einer nicht mehr gültigen Adresse stammt. Ähnliches gilt für Return-Befehle.

Wie schon unter 5.1 erwähnt, verursachen vorhergehende Zugriff auf PCLATH (obere Bits des Program Counter) eventuelle Hasards, die mit Stalls abgefangen werden müssen.

Bei einem bedingten Sprung oder einem Computed Jump ist erst am Ende von E (Execute) bekannt, ob bzw. wohin gesprungen wird. Entsprechend gehen zwei oder drei Taktzyklen durch Anhalten der Pipeline verloren.

Besondere Sprungbehandlung wie Sprungvorhersage u.ä. ist derzeit nicht konzipiert. Wenn die Pipeline sonst vollständig implementiert und getestet ist, können sich Arbeiten hierauf richten.

5 Zusammenfassung

Der erste Teil des Forschungsprojekts konzentrierte sich auf den grundsätzlichen Aufbau der Pipeline für den PIC-kompatiblen Mikrocontroller. Die Anzahl ihrer Stufen wurde angepaßt an die bisherigen Verarbeitungsschritte auf fünf festgelegt, was eine prinzipielle Balancierung der Pipeline verspricht. Da in der sequentiellen Architektur bereits ein rudimentäres Pipelining implementiert ist, ergibt sich durch die geplante Struktur eine maximale Geschwindigkeitssteigerung um den Faktor vier (statt fünf). Durch die Hasards und die geringere Taktgeschwindigkeit aufgrund der zusätzlichen Hardware werden praktisch eher Faktoren im Bereich von zwei bis drei erwartet.

Ausgehend von der Stufenanzahl wurde die elementare Struktur der Pipeline festgelegt sowie die entsprechenden Register zwischen den Stufen zur Speicherung der weitergeleiteten Daten- und Steuerinformationen. Zusätzliche Funktionen zum teilweisen Anhalten der Pipeline und Einfügen von NOPs wurden vorgesehen.

Die Analyse des Befehlssatzes lieferte die grundsätzlichen Informationen, welche Befehle Daten- bzw. Kontrollflußhasards auslösen können. Die verschiedenen Datenhasards wurden klassifiziert und die möglichen Abstände von Bedingungs- und Auslösebefehlen eruiert, woraus sich die Anforderungen an die Implementierung der Pipeline bzw. einer Hasarderkennung ergeben. Dazu passend wurden Erfordernisse und prinzipielle Strukturen für Pipeline Stalls sowie eine Data Forwarding Unit aufgestellt.

Konzeptionelle Arbeiten zum Anschluß der Debug-Einheit wurden auf später verschoben. Dafür konnten schon erste Probeimplementierungen (ohne Performance Tests oder Timing-Untersuchungen) angegangen werden, die optimistisch für die folgende Phase des Projektes stimmen.

Literatur

[1] Bermbach, R.: *Entwicklung eines Mikroprozessorkerns*. Forschungsbericht, Fachhochschule Braunschweig/Wolfenbüttel, 2003..

[2] Bermbach, R.: *Entwicklung eines Mikroprozessorkerns – Bericht über die zweite Phase*. Forschungsbericht, Fachhochschule Braunschweig/Wolfenbüttel, 2004.

[3] Bermbach, R.; Kupfer, M.: *Development of a debug module for a FPGA-based microcontroller*. IFAC workshop on programmable devices and embedded systems. Brünn, Tschechien, 2006.

[4] Patterson, D.; Hennessy, J.: *Rechnerorganisation und –entwurf: Die Hardware/Software-Schnittstelle*. 3. Aufl. München u.a.: Elsevier Spektrum Akademischer Verlag, 2005

[5] Bermbach, R.: *Einsatz von Pipelining bei Mikroprozessoren*. Vorlesungsskript zur Veranstaltung Rechnerarchitekturen II, Fachhochschule Braunschweig/Wolfenbüttel, 2006.

[6] Microchip Technology Inc.: *PICmicro™ Mid-Range MCU Family Reference Manual DS33023A*. Chandler, USA, 1997.

[7] Breustedt, P.: *Analyse und Behandlung von Datenhazards in der Pipeline eines VHDL-Mikrocontrollers*. Diplomarbeit Fachhochschule Braunschweig/Wolfenbüttel, 2007.