Entwicklung eines Mikroprozessorkerns – Forschungsbericht über die zweite Phase ab SS04

Prof. Dr.-Ing. Rainer Bermbach

Fachbereich Elektrotechnik

Tel.: 0 53 31 / 93 93 111

R.Bermbach@FH-Wolfenbuettel.de

1 Einleitung

Das FuE-Vorhaben *Entwicklung eines Mikroprozessorkerns* hatte als Ziel, einen kleinen, flexiblen und frei verfügbaren Prozessorkern in VHDL zu erstellen, der auf preiswerten FPGA-Modulen realisierbar und zusammen mit anderer Hardware implementierbar ist. Die tatsächliche Realisierbarkeit in programmierbarer Hardware ist ein wesentlicher Unterschied zu anderen frei verfügbaren Mikroprozessorkernen, deren Beschreibungen häufig nicht oder höchstens teilweise implementierbar sind. Der Prozessorkern soll mit spezifischen Peripheriekomponenten erweitert bzw. in übergeordnete Systeme eingebunden werden können. Er soll Im Rahmen potentieller weiterer FuE-Vorhaben, Projektarbeiten und Laborversuche zum Einsatz kommen sowie als Anschauungsobjekt für entsprechende Lehrveranstaltungen dienen.

Im ersten Teil des Vorhabens (ab SS03) konnte bereits das grundlegende Ziel erreicht werden. Es entstand ein adaptierbarer, auf preiswerten FPGA-Modulen (mit Xilinx Spartan II) implementierbarer Kern, der mit den Mikrocontrollern der sog. PICmicro Mid-Range Family der Firma Microchip weitestgehend kompatibel ist. Für eine Beschreibung von Struktur, Eigenschaften und Handhabung des Mikroprozessorkerns sei auf [1] verwiesen.

Der zweite Teil des Vorhabens, die Optimierung, hatte zum Ziel, die Verarbeitungsfrequenz (maximale Taktrate) des Prozessors wenn möglich deutlich zu steigern und dabei den Ressourcenverbrauch beizubehalten oder gar zu verringern. Daneben sollten weitere Peripheriekomponenten entstehen, Fehler beseitigt sowie PIC-Software für die Nutzung mit vorhandenen I/O-Modulen standardisiert werden.

Diese Ziele konnten alle erreicht werden. So arbeitet der Mikroprozessorkern jetzt mit über 80 MHz Taktfrequenz (max. 82,6 MHz gemäß Routing-Tool) gegenüber > 25 MHz im ersten Entwurf [3]. Lediglich noch 10 – 15 Signalpfade (je nach Routing-Lauf) benötigen mehr als 10 ns, damit ist das Design nahe der 100-MHz-Grenze. Vergleichbare reguläre PIC-Prozessoren laufen mit maximal 20/24 MHz. Eine probeweise Implementation auf einem FPGA-Board mit einem Baustein der neuen Xilinx Spartan-III-Serie ergab auf Anhieb einen funktionsfähigen Prozessor mit maximalen Taktfrequenzen > 100 MHz. Auch der Ressourcenverbrauch konnte weiter verringert werden.

Im folgenden geht der Bericht auf die verschiedenen durchgeführten Arbeiten ein. Nachdem Kapitel 2 die benutzte Entwicklungsumgebung vorstellt, beschreibt Kapitel 3 die Software für das I/O-Modul, während das folgende Kapitel 4 die Implementierung eines UART-Peripheriemoduls umreißt. Die Umstellung der aktiven Flanken der Block-RAM-Module schildert Kapitel 5. Kapitel 6 setzt sich mit dem Einsatz des Timing-Tools auseinander, eine wichtige Voraussetzung zur Beurteilung der Geschwindigkeit des Kerns. Danach veranschaulichen Kapitel 7 und 8 die Überlegungen und die Vorgehensweise bei der Neugestaltung der ALU bzw. der Einheit DECODE, während Kapitel 9 die erzielten Ergebnisse präsentiert und diskutiert

2 Entwicklungsumgebung

Für die Durchführung des Projekts benötigt man natürlich Hardware, Entwicklungswerkzeuge und Hilfsmittel. Für die Schaltungsentwicklung mit VHDL kam die im Labor für Datentechnik lizenzierte und gepflegte Entwicklungsumgebung Xilinx ISE 6.1i zum Einsatz. Die ISE erlaubt die Entwicklung von der VHDL-Eingabe über Synthese, Mapping, Place & Route bis zum Download in das FPGA. Die Simulation (Pre- und Post-Layout-Simulation) ermöglicht die ISE mit dem Programm ModelSim. Da dessen Komfort und vor allem Arbeitsgeschwindigkeit stark zu wünschen übrig läßt, wurde das ebenfalls lizenzierte Programmpaket Aldec Active HDL 6.2 eingesetzt. Active HDL erlaubt die Einbindung der Xilinx Tools, so daß aus einer einzigen Umgebung heraus gearbeitet werden kann.

Für die Mikroprozessor-Softwareentwicklung mit dem PIC existiert die frei erhältliche IDE (Integrated Development Environment) MPLAB 6.50 von Microchip. Mit ihrer Hilfe erzeugt man aus den Assembler-Sourcen ausführbaren Code im Intel-Hex-Format, den ein eigenes Programm in VHDL-Code zur Initialisierung der für das ROM verwendeten Block-RAMs umsetzt.

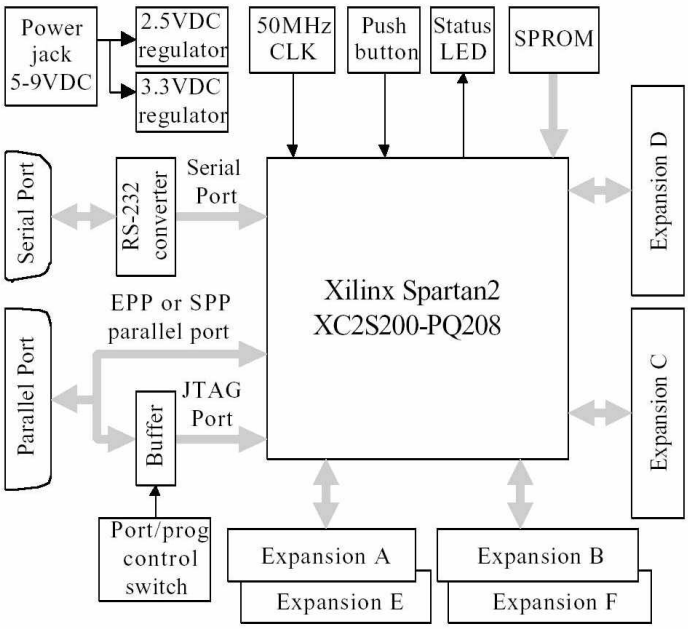

Abb. 2.1 Blockschaltbild des FPGA-Entwicklungsboards

Als Hardwareplattform dienen FPGA-Boards der Firma Digilent (Digilab2), die mit einem Xilinx FPGA des Typs Spartan II (Speed Grade 5) mit 200.000 Gatteräquivalenten be-

stückt sind. Sie verfügen daneben u.a. über Programmierschnittstelle, Takt- und Stromversorgung und an die Boardkanten herausgeführte FPGA-Anschlüsse (s. Abb. 2.1). Abbildung 2.2 zeigt ein Foto eines solchen Boards.

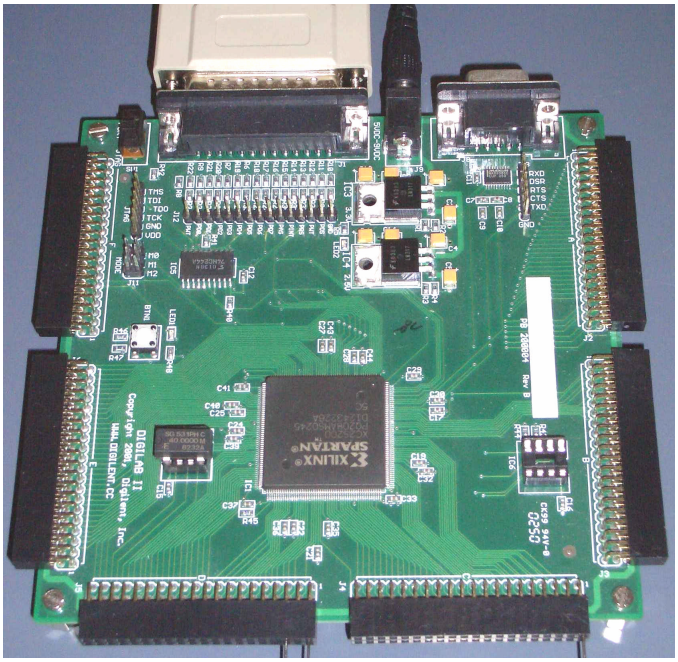

Abb. 2.2 Foto des FPGA-Entwicklungsboards

Möchte man die äußeren Signale und Schnittstellen des implementierten Mikrocontrollers beobachten, erlaubt dies das verwendete Board über die Erweiterungsstecker (Expansion A – F) z.B. per Oszilloskop oder Logikanalysator. Geht es weniger um Meßaufgaben, dafür mehr um direkte Benutzerein- oder -ausgaben, kann man ein I/O-Modul (Digilent DIO2) an das FPGA-Board anstecken.

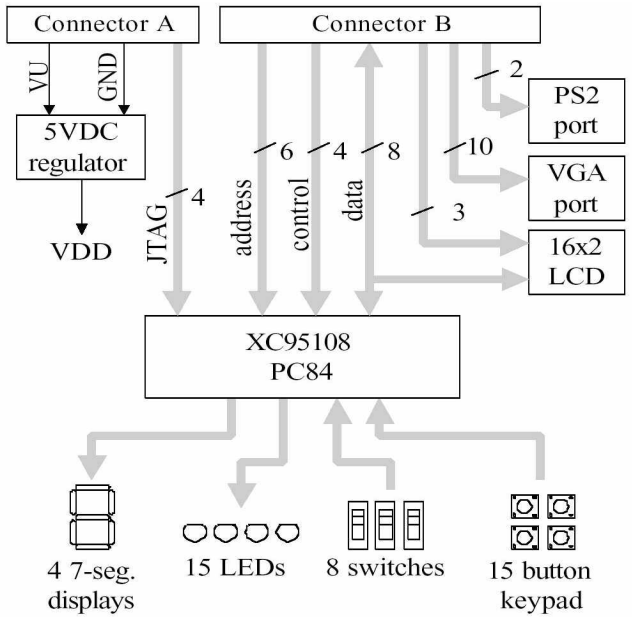

Abb. 2.3 Blockschaltbild des I/O-Moduls

Das I/O-Modul verfügt über ein zweizeiliges LCD, eine vierstellige Siebensegmentanzeige und etliche LEDs, Schalter und Taster (s. Abb. 2.3 und 2.4). Ein auf dem Modul befindliches CPLD realisiert eine asynchrone Busschnittstelle für die Ein-/Ausgabeelemente. Das LCD besitzt die bei solchen Modulen übliche 8+3-Schnittstelle.

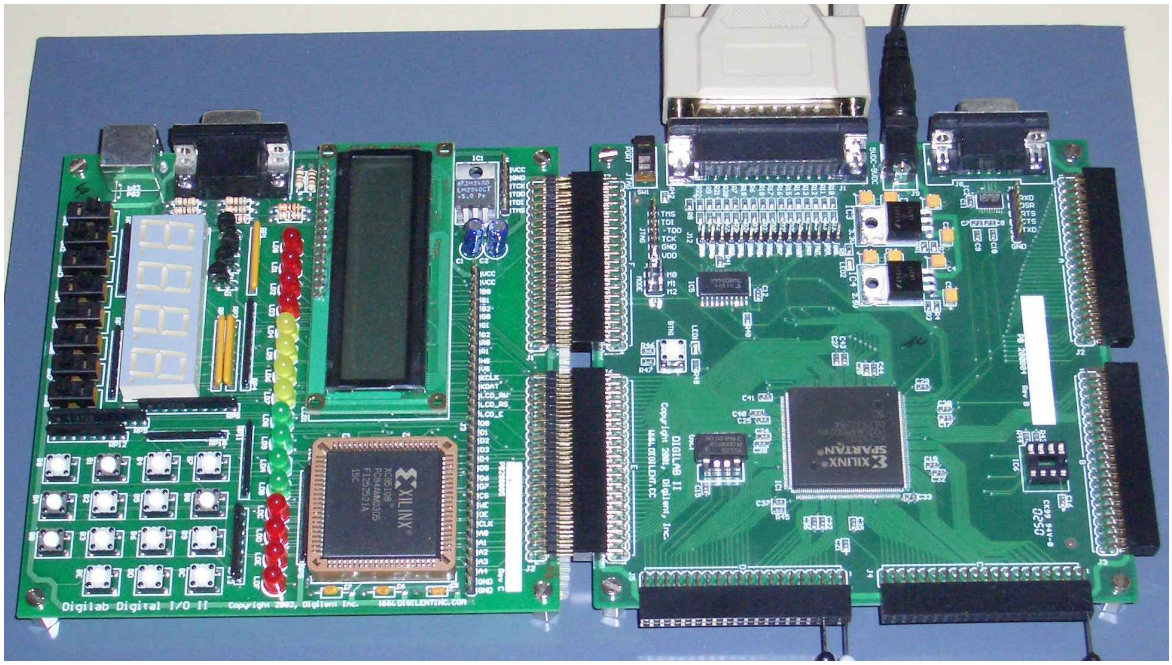

Abb. 2.4 Foto des FPGA-Moduls mit angestecktem I/O-Modul

Der VGA- und der PS2-Port sind auf dem I/O-Modul nur rudimentär ausgeführt. Sie benötigen zusätzlich im FPGA implementierte Hardware. Derzeit ist noch kein VHDL-Code dafür entwickelt worden.

3 PIC-Software zum Betrieb des Digilent I/O-Moduls

Das I/O-Modul wird, wie schon erwähnt, auf die Expansion-Anschlüsse des FPGA-Boards aufgesteckt. Hardwaremäßig bedeutet das einen Anschluß an Ports des implementierten Prozessorkerns. Über diese Ports kann man mit entsprechender (PIC-) Software die Ressourcen des I/O-Moduls bedienen. Die Erstellung von gut strukturierten Funktionen zur Nutzung dieser Ressourcen war ein kleines Unterthema des Projekts.

Es entstanden verschiedene Medium- und Low-Level-Routinen (Assembler-Unterprogramme) zur Ansteuerung der Hardware des I/O-Moduls:

- LCD – init, write instruction, write data, read write position, write string
- 7-Segment – write (digit 1 & 2 und 3 & 4)
- LED – write (LEDs 0 – 7 und 8 – F)
- Switches – read status
- Buttons – read status (mit Entprellung, Taster 0 –7 und 8 - E)
- BCD Counter (vierstellig), BIN Counter (vierstellig)
- Delay-Routinen

In einem Testprogramm wurden alle Routinen in einem übergeordneten Rahmen eingebunden. Durch verschiedene Ausgaben auf den Displays (Text, Counter) und LEDs (Lauflicht), die durch Schalter und Taster beeinflußbar sind, gewinnt man einen schnellen, ersten Anhaltspunkt, ob der implementierte Prozessorkern korrekt läuft oder fehlerhaft ist.

4 Implementierung einer UART-Peripheriekomponente

Die als Referenz dienenden PIC-Prozessoren der Mid-Range Family verfügen teilweise über eine serielle Schnittstelle (UART – Universal Asynchronous Receiver Transmitter) für die asynchrone Datenübertragung nach RS232. Bei einigen ist auch eine synchrone Betriebsart (USART) implementiert. Um den Kern mit einer weiteren Peripheriekomponente auszustatten und dabei die Anbindung von neuer Peripherie an den Mikroprozessorkern zu testen, wurde eine kompatible UART-Schnittstelle implementiert. (Der selten benötigte synchrone Part wurde ausgespart.)

Die Schnittstelle besteht aus den Funktionsblöcken Baudratengenerator, asynchroner Sender und asynchroner Empfänger. Diese Struktur findet sich auch in den VHDL-Modulen wieder. Der Baudratengenerator erzeugt aus dem Prozessortakt die benötigten Frequenzen für Sender und Empfänger. Der Empfänger erhält zusätzlich den 16-fachen Takt, um eine gute Synchronisation auf das empfangene Startbit sowie eine Überabtastung des Empfangssignals (Mehrheitsentscheidung) zu erreichen. Bei einem Prozessortakt von 80 MHz ist eine minimale (Norm-) Baudrate von 4.800 Bd möglich. Ist dies in einem Anwendungsfall zu hoch, muß ein entsprechender Vorteiler (z.B. 6- bis 8-Bit-Zähler) vorgeschaltet werden, der im Original-PIC nicht vorhanden ist. Ansonsten entsprechen die Programmierung und das Verhalten des UARTs exakt dem Vorbild [2].

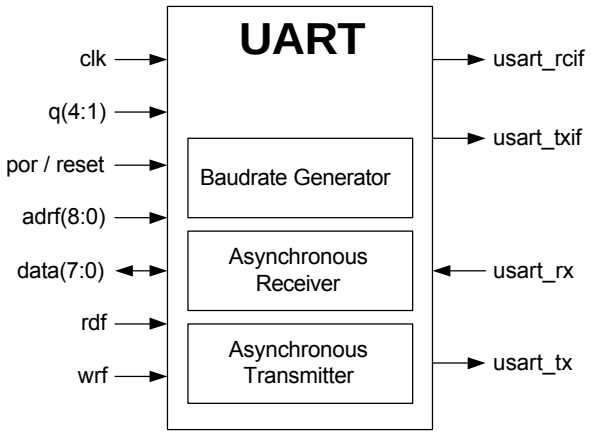

Abb. 4.1 Schnittstellen des UART-Moduls

Die Einbindung in das bestehende Design gestaltete sich problemlos. Die üblichen, äußeren Signale des UART wie Takt, Reset, Adreß- und Datenbus (s. Abb. 4.1, linke Seite des UART) mußten lediglich mit den entsprechenden Prozessorsignalen verbunden werden. Ein Eintrag im UCF (User Constraints File) weist die UART-spezifischen Signale wie Empfangsleitung (usart_rx), Sendeleitung (usart_tx) einem Pin des FPGAs zu. Die speziellen Signale usart_rcif und usart_txif sind die Interruptsignale des UART beim vollständigen Empfang bzw. bei vollständiger Aussendung eines Zeichens. Das Modul INTCON, das für die Interruptverarbeitung zuständig ist, erhält diese Signale und wertet sie aus. Da ent-

sprechende Freigabebits vorgesehen sind, ist INTCON das einzige bestehende Modul, das bei Einbindung des UART adaptiert werden mußte. Um die Flexibilität des Prozessorkerns zu erhalten, wurde die Einbindung des UART in das Gesamtdesign über eine Konstante (USE_UART) im Modul MYPICPACK schaltbar ausgeführt [4].

5 Umstellung der Taktflanke bei Block-RAM-Komponenten

In den Xilinx Spartan II FPGAs stehen bis zu 14 Blöcke synchrones RAM (4 KBit) zur Verfügung. Zwar lassen sich auch die LUTs als RAM einsetzen, die Block-RAMs sind aber sehr flexibel konfigurier- und einsetzbar (4 K x 1 – 256 x 16, single- und dual-ported). Im Design des Mikroprozessorkerns finden Block-RAMs (entsprechend initialisiert) als ROM, als internes RAM (General Purpose Registers – GPR) und als Stackspeicher im Modul PC (Program Counter) Verwendung.

Im bisherigen Design arbeiteten die Block-RAMs (wie von Xilinx vorgesehen) auf der positiven Flanke des Taktsignals, während die Restschaltung auf der negativen schaltete. Dadurch ergaben sich kritische Zeitbedingungen für einige Signale, da teilweise nur eine halbe Taktperiode zur Verfügung stand. Während die Umstellung von ROM und GPR vergleichsweise einfach zu realisieren waren, bedurfte es einiger Änderungen beim Stack im Modul PC.

Bislang zeigte der Stack Pointer auf die nächste freie Stelle im Stack. Durch die Umstellung auf die negative Flanke kommt es dadurch zu Problemen beim „POP“ aus dem Block-RAM. Wird die Schaltung so geändert, daß der Stack Pointer immer auf den letzten Eintrag zeigt und unmittelbar vor dem Schreiben inkrementiert und direkt nach dem Lesen dekrementiert wird, liefert das Block-RAM jederzeit die richtigen Daten für das Füllen des Program Counter.

Die neue Schaltungsversion verhält sich makroskopisch betrachtet identisch zur bisherigen und ist kompatibel zum Verhalten der PIC-Prozessoren. Da Microchip keinerlei Angaben über die interne Realisierung macht, ist die neue Lösung in gleicher Weise „kompatibel“ wie die alte.

Durch die Umstellung auf eine gemeinsame (negative) Taktflanke im gesamten Design erhöht sich, wie erwartet, die erreichbare maximale Taktfrequenz beträchtlich.

6 Timing-Analyse

Bislang war das Timing-Analyse-Tool der Xilinx ISE nur übersichtsartig benutzt worden. Für eine Optimierung des Kerns ist ein tieferer Einstieg unumgänglich. Mit dem Analysewerkzeug kann man z.B. zeitliche Vorgaben für alle Pfade, für einzelne, spezifizierte Signalgruppen und für Einzelsignale machen. Gibt man beispielsweise für alle Pfade eine maximale Verzögerungszeit von 10 ns entsprechend 100 MHz Takt an, gewinnt man Aussagen darüber, ob das Design diese Anforderung erfüllt oder falls nicht, welche Pfade um wieviel langsamer sind. Mit diesen Angaben kann eine gezielte Designoptimierung durchgeführt werden.

Probleme entstehen allerdings dadurch, daß das Timing-Tool nicht zwischen tatsächlich genutzten Pfaden und theoretisch möglichen unterscheiden kann. Eine einfache Vorgabe

für alle Pfade führt deshalb zu unübersehbar vielen Verletzungen, von denen die meisten mühsam per manueller Durchsicht ausgeschlossen werden müssen.

Um zu auswertbaren Ergebnissen zu gelangen, müssen also Signalgruppen qualifiziert werden, die andere, geringere Anforderungen an ihre Verzögerungszeiten haben. Diese Pfade nennt man Multi-Cycle Paths. Weiterhin müssen sog. False Paths identifiziert werden, also Pfade, die zwar existieren, aber keine praktische Bedeutung haben. Unter Berücksichtigung der (nicht ganz einfach zu handhabbaren) hierarchischen Beziehungen zwischen den verschiedenen Timing und Grouping Constraints kann so das Design mit dem Timing-Tool analysiert werden.

Die Erstellung eines Diagramms mit den Timing-Beziehungen zwischen den einzelnen Designteilen erlaubte die Identifizierung der verschiedenen Multi-Cycle-Path-Gruppen und der False Paths. Ein Belegung der Multi-Cycle Paths mit Verzögerungszeiten entsprechend n x 10 ns führte letztlich zu auswertbaren Ergebnissen [4].

Die vollständige Qualifizierung der verschiedenen Timing-Pfade mit dem Timing-Analyse-Tool ermöglicht eine schnelle und aussagekräftige Auswirkungsanalyse bei beliebigen Änderungen am Design.

7 Optimierung des Moduls ALU

Da ALU und DECODE die beiden größten Module des Designs sind, bietet es sich an, hier Optimierungen vorzunehmen. Die ALU (Arithmetic Logic Unit) ist für fast die gesamte Datenverarbeitung im Prozessorkern verantwortlich. Die bisherige Version verwendet einen großen, jeweils 8 Bit breiten 10:1-Multiplexer, um die Ergebnisse der zehn verschiedenen Blöcke zum Ausgang durchzuschalten. Auch intern finden einige weitere, 8 Bit breite Multiplexer Einsatz. Da Implementierung von Multiplexern sehr logikaufwendig sind, kann man hier gut ansetzen.

Eine erste Idee, die Multiplexfunktion über Tristate-Puffer zu realisieren, scheidet aufgrund der üblicherweise geringen Geschwindigkeit von tristate-fähigen Schaltungen aus. Betrachtet man die zugrunde liegenden Strukturen in den üblichen FPGAs, so findet man dort kleine RAMs (16 x 1 Bit), die als Look-up Tables (LUT) jede beliebige Logikfunktion mit vier Variablen realisieren können. Eine Zusammenschaltung von acht solcher LUTs kann also zwei 8-Bit-Eingangsvektoren A und B (fast) beliebig verarbeiten zu einem 8-Bit-Ergebnis C. Dabei bleiben zwei Eingänge an den LUTs frei für die Auswahl der gewünschten Funktion (s. Abb. 7.1).

Am einfachsten erkennt man das Potential dieser Struktur bei der Logikverarbeitung innerhalb der ALU. Mit den beiden Auswahlleitungen sel0 und sel1 lassen sich vier verschiedene Verknüpfungen zwischen A und B wählen. Für den Logikteil bieten sich das logische AND, das OR und das EXOR an. Als vierte Funktion liegt die Inversion eines Eingangssignals nahe. (Aus speziellen Gründen wurde tatsächlich die Identität (C = A) gewählt.)

Mit dieser Struktur lassen sich also vier Funktionen verwirklichen, ohne einen einzigen zusätzlichen Multiplexer zu benötigen. In der Vorgängerversion des Designs wurden hierfür vier der zehn Multiplexereingänge verbraucht. Gleichzeitig erhält man mit dieser Lösung eine minimale Implementierung für die gewünschten Funktionen.

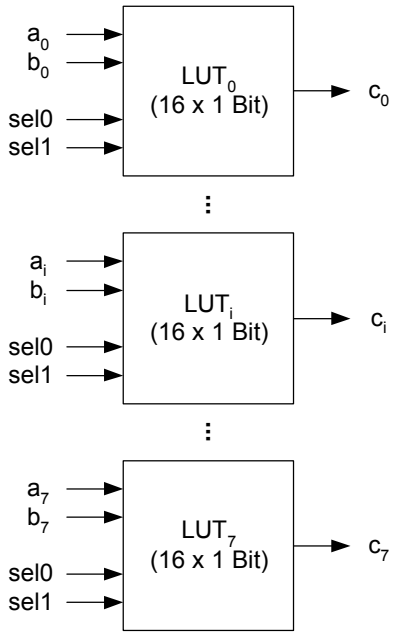

Abb. 7.1 8-Bit-Verarbeitung mit LUT-Struktur

In gleicher Weise läßt sich auch eine LUT-Struktur für die Rotation nach links und rechts implementieren. Da der PIC immer durch das Carry-Bit rotiert, sind hierfür neun LUTs notwendig. Hier bleibt eine Auswahlleitung ungenutzt, die eventuell für andere Funktionen einsetzbar wäre (tatsächlich aber nicht genutzt werden kann). Für die SWAP-Funktion (Vertauschen der unteren vier Bits mit den oberen vier) benötigt man eine weitere LUT-Struktur. Da aber SWAP nur einen Operanden benötigt, kann man das Rotationsergebnis als zweiten Operanden einsetzen und mit einer Auswahlleitung zwischen den beiden Ergebnissen wählen. Damit entfallen wieder drei Multiplexereingänge.

Nach dem gleichen Verfahren läßt sich auch die Arithmetik zu implementieren. Allerdings benötigt die Addition (da keine *reine* bitweise Verarbeitung erfolgt) eine Weiterleitung des Carry-Ergebnisses zwischen den einzelnen LUTs. Solche Anwendungsfälle haben die entsprechenden FPGA-Hersteller vorhergesehen und versorgen die Logikblöcke, in denen die LUTs angeordnet sind u.a. mit schnellen sog. Carry Chains, Schaltungen, die beispielsweise bei Addierern oder Zählern die nötige Weiterleitung des Carrys zum Nachbarblock erlaubt.

Erste gute Ergebnisse lieferte ein von Xilinx als Macro verfügbares Addier-/Subtrahiermodul. Der Einsatz eines solchen Macros würde aber die Übertragung des Designs auf andere FPGA-Architekturen unmöglich machen. Die klassische VHDL-Implementierung der Subtraktion als Addition des Zweierkomplements (*A – B = A + not B +1*) führte zu zusätzlichem Aufwand, da der Compiler nicht erkannte, daß die „*+1*“ einfach in den Beginn der Carry Chain einzuspeisen ist. Weitere Überlegungen führten zu einer anderen Realisierung der Subtraktion in der Form *A – B = not (not A + B)*. Hierfür muß also lediglich der A-Operand invertiert zugeführt und das Ergebnis invertiert werden. Dies können die zur Verfügung stehenden Strukturen einfach realisieren.

Auf die gleiche Weise konnten auch der Eingangsmultiplexer der ALU und weitere benötigte Schaltungsteile vereinfacht werden. Die beschriebenen Maßnahmen reduzierten so u.a. den bisherigen 10:1-Multiplexer am Ausgang auf einen 4:1-Multiplexer für die Ergebnisse der Blöcke ADD/SUB, LOGIC, ROT/SWAP und PASS (Durchreichen des Operan-

den). Insgesamt konnte mit den durchgeführten Optimierungen der Aufwand für die Implementierung der ALU beträchtlich gesenkt werden (s. auch Kap. 9 Ergebnisse).

Ein potentieller Einwand, daß man hier die Optimierung durch zu weitgehenden Zuschnitt auf die zugrundeliegenden Strukturen erzielte und das Design an Flexibilität verlöre, ist in dieser Form unberechtigt. Sämtliche Strukturen sind nach wie vor in Standard-VHDL-Code realisiert. Da die heutigen Compiler vielfach noch nicht in der Lage sind, in abstrakteren Beschreibungen die unterliegende simple Struktur zu erkennen, ist es legitim, einfachere Beschreibungsformen zu verwenden. So führt z.B. eine Vierfach-CASE-Struktur im Code direkt zur Verwendung einer (oder mehrerer paralleler) LUTs, was bei komplexerer Codierung des gleichen Sachverhalts nicht erzielt wird. Natürlich hat der Einsatz solchen Codes auf FPGAs mit anderen Strukturen u.U. ein weniger optimales Ergebnis zur Folge. Da aber mehr als 90% der heute verfügbaren FPGAs mit LUTs arbeiten, ist diese Vorgehensweise sicher legitim.

8 Optimierung des Moduls DECODE

Eine Änderung der ALU zieht immer eine Anpassung der Decode-Einheit nach sich. Da das Modul aufgrund seiner Größe eh Ziel einer Überarbeitung war, konnte diese Hand in Hand mit der Optimierung der ALU einhergehen. Grundüberlegung zur Überarbeitung war eine Untersuchung der Codierung des Befehlssatzes (s. Abb. 8.1).

	F		LW	
	Byte	Bit	Call/Goto	LW
Bit 13 - 12:	00	01	10	11
Bit 11 - 8:				
0000	MOVWF + *	BCF	CALL	MOVLW
0001	CLRF / CLRW			
0010	SUBWF			
0011	DECF			
0100	IORWF	BSF		RETLW
0101	ANDWF			
0110	XORWF			
0111	ADDWF			
1000	MOVF	BTFSC	GOTO	IORLW
1001	COMF			ANDLW
1010	INCF			XORLW
1011	DECFSZ			----------
1100	RRF	BTFSS		SUBLW
1101	RLF			
1110	SWAPF			ADDLW
1111	INCFSZ			

*:+NOP/CLRWDT/
/RETFIE/RETURN
/SLEEP

Abb. 8.1 Befehlscodierung des PIC-Assemblerbefehlssatzes

Das oberste Bit 13 unterscheidet zwischen Befehlen, die mit internen Daten (aus Register File F) und die mit Literalen (LW) arbeiten. Nimmt man Bit 12 hinzu, so ergeben sich die vier Befehlsgruppen Byte-, Bit-, Sprung/Unterprogramm- und Immediate-Befehle (Arbeiten mit einer Konstante). Weiterhin ist leicht ersichtlich, daß bestimmte Befehlsgruppen sehr großzügig codiert sind. Die Nutzung dieser Redundanzen und Gruppierungen beim Entwurf der ALU-Verarbeitungseinheiten hat eine vereinfachte Decodierung der Befehlssteuerung in DECODE zur Folge. Teilweise steuert direkt das Bit aus dem Instruction Register die betreffenden Einheiten der ALU. Die verbleibende Decodierung in DECODE sollte sich

eigentlich einfach gestalten. Leider war auch hier der Compiler nicht in der Lage immer alle passenden Vereinfachungen, insbesondere die Don't Cares, zu erkennen, weshalb die sich ergebenden, üblichen CASE-Strukturen per Hand minimiert und anschließend codiert wurden. Hierunter leidet natürlich die Übersicht im Code, was durch Einfügen der eigentlichen CASE-Konstrukte als Kommentar gemildert wurde.

Insgesamt brachten diese Optimierungen auch im Modul DECODE beträchtliche Einsparungen.

9 Ergebnisse

Bei den Optimierungen interessieren vorrangig die Verbesserungen beim Ressourcenverbrauch sowie bei der erreichten maximalen Taktfrequenz für das Design. Beide werden in den folgenden Unterkapiteln diskutiert.

9.1 Belegung von Logikressourcen

Um die Veränderungen im Ressourcenverbrauch zu beschreiben, kann man auf verschiedene Betrachtungsweisen zurückgreifen. Die sicher naheliegendste ist die Beurteilung des Gesamtbedarfs beispielsweise an LUTs, Flipflops oder Slices. Ein Slice (= ½ Logic Cell) enthält zwei LUTs und zwei Flipflops mit weiterer umgebender Logik u.a. der Carry Chain. Möchte man dabei zu aussagekräftigen Zahlen kommen, ist es wichtig, den jeweiligen Verbrauch nach der Synthese zu betrachten. Im nachfolgenden Map-Prozeß werden die Hardwarestrukturen noch einmal überarbeitet und beispielsweise an die zur Verfügung stehenden Ressourcen angepaßt. Das Routing legt so auch Pfade durch LUTs hindurch, so daß dadurch weitere Ressourcen belegt werden, die mit dem eigentlichen Design nichts zu tun haben. Teilweise verschiebt das Mapping und das Place & Route auch Flipflops in die Ein-/Ausgabebereiche (IOBs), was eine vergleichende Aussage weiter verfälschen würde. Aus diesen Gründen werden im folgenden die Ergebnisse nach der Synthese betrachtet.

Die Abbildung 9.1 gibt die einzelnen Verbräuche sowie ihre Veränderung beim neuen Design an. (Die Ressourcen für das neue Modul UART sind aus Gründen der Vergleichbarkeit hier weggelassen.)

		Vorversion	*Neudesign*	*Verbesserung*	*in Prozent*
ALU	***Slices***	72	41	31	43,1%
	Slice Flipflops	24	16	8	33,3%
	LUTs	135	80	55	40,7%
DECODE	***Slices***	50	32	18	36,0%
	Slice Flipflops	65	43	22	33,8%
	LUTs	88	56	32	36,4%
PC	***Slices***	49	44	5	10,2%
	Slice Flipflops	23	23	0	0,0%
	LUTs	90	82	8	8,9%
MAIN	***Slices***	407	343	64	15,7%
	Slice Flipflops	392	351	41	10,5%
	LUTs	735	593	142	19,3%

Abb. 9.1 Ressourcenverbrauch nach der Synthese [4]

Die Verbesserungen bei ALU und DECODE sind augenfällig. Bei den LUTs benötigt die neue ALU nur noch rund 60% der Vorversion. Auch die Anzahl der Flipflops schrumpfte auf 2/3. Das Modul DECODE benötigt ebenfalls nur noch etwa 2/3 der bisherigen Strukturen.

Interessant erscheint, daß die Änderungen am Stack zur Anpassung der Taktflanke ebenfalls zu einer Ersparnis im Modul PC von etwa 10% führte. Bezogen auf das Gesamtdesign (= Modul MAIN) sind die (prozentualen) Änderungen nivelliert, nicht so drastisch, fallen aber mit 10 % bis knapp 20 % doch nicht unwesentlich aus. (In gewissem Maße tragen auch Verbesserungen am Watchdog Timer zu diesem Ergebnis bei. Sie wurden nötig, da dieser nicht mehr bei den Taktfrequenzen um 80 MHz arbeitete und deshalb ebenfalls umgestellt wurde.)

Setzt man die im Design verwendeten Slices, Flipflops und LUTs ins Verhältnis zu den im FPGA verfügbaren (XC2S200) ergibt sich der Ausnutzungsgrad des FPGAs. Dieser fiel bei den Slices von 17,3 % auf 14,6 %, bei den Flipflops von 8,3 % auf 7,5 % und bei den LUTs von 15,6 % auf 12,6 %. Wie schon erwähnt liegen diese Werte für die tatsächliche Implementation höher, da auch Logic Cells für Routingzwecke eingesetzt werden. Inklusive UART liegt der Ausnutzungsgrad jeweils etwas höher als beim alten Design ohne UART.

Insgesamt kann man sagen, daß der Ressourcenbedarf für das Design sehr gut optimiert wurde. Mit Ausnahme der Ports gibt es keinerlei fixe Positionierungen auf dem Chip, was üblicherweise noch zu wesentlichen Einsparungen (durch günstigeres Routing) führt. Ziel war es ja, einen reinen „Software“-Kern zu erstellen, der sich auf den üblichen Strukturen günstig und ohne Beschränkungen implementieren läßt.

9.2 Steigerung der maximal möglichen Taktfrequenz

Allein die Umstellung des Designs auf eine einheitliche Taktflanke brachte den „Löwenanteil“ der Geschwindigkeitssteigerung. Dank der Timing-Qualifizierung sämtlicher Pfade auch für das alte Design erzielte dieses bereits ca. 37 MHz maximale Taktfrequenz (gegenüber ersten Abschätzungen von > 25 MHz noch in der letzten Berichtsperiode). Die Flankenumstellung führte das Design in den Bereich von ca. 71 MHz.

Alle Timing-Angaben beziehen sich auf FPGAs des Speed Grade 5, d.h. die langsamere der beiden verfügbaren Geschwindigkeitsklassen der Spartan II FPGAs. Mit Speed Grade 6 erreicht man natürlich entsprechend schnellere Implementierungen.

Die weiteren Optimierungen an Struktur und Detailimplementierung ergaben eine Steigerung auf maximal 82,6 MHz Taktfrequenz (Verzögerungszeit 12,11 ns). Nur noch etwa 10 – 15 Pfade wiesen Verzögerungszeiten größer 10 ns auf, d.h. das Design ist nahe der 100-MHz-Grenze. Bei den Untersuchungen zeigte sich eine starke Abhängigkeit des Ergebnisses von der Qualität der Place & Route-Läufe. Versuche ergaben maximale Frequenzen von 69,4 – 82,6 MHz. Interessant an den sich ergebenden Verzögerungszeiten der kritischen Pfade ist, daß der Logikteil mit etwa 4 ns relativ konstant blieb und der Anteil durch das Routing je nach Durchlauf zwischen ca. 8 und 10 ns variierte.

Überhaupt beträgt der Routing-Anteil an der Verzögerung mittlerweile etwa 66 – 72 %. Damit sind weitere Verbesserungen der Logikstruktur wenig sinnvoll, da sie nur zu einem Bruchteil eingehen. Dies zeigt sich auch drastisch durch die probeweise Implementation

auf einem FPGA-Board mit einem Spartan III, der Nachfolgegeneration von Spartan II. Diese Chips werden mit kleineren Strukturgrößen (90 nm) gefertigt und besitzen deshalb geringere Chipflächen. Die dadurch auch kürzeren und weniger verlustbehafteten Routing-Pfade ermöglichten auf Anhieb einen Prozessor, der mit 100 MHz Takt lief. Die völlig problemlose Portierung auf eine neue Architektur spricht ebenfalls für die erreichte Leistungsfähigkeit des Designs.

Literatur

[1] Bermbach, Rainer: *Entwicklung eines Mikroprozessorkerns*. Forschungsbericht (Kurzfassung) Fachhochschule Braunschweig/Wolfenbüttel, Fachbereich Elektrotechnik, September 2003.

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

[3] Cramm, Ingmar: *Entwicklung eines PIC-kompatiblen Mikrocontrollerkerns in VHDL*. Diplomarbeit Fachhochschule Braunschweig/Wolfenbüttel, Fachbereich Elektrotechnik, Juli 2003.

[4] Andreas, Volker: *Optimierung eines PIC-kompatiblen VHDL-Mikrocontrollerkerns*. Diplomarbeit Fachhochschule Braunschweig/Wolfenbüttel, Fachbereich Elektrotechnik, August 2004.