Fachhochschule Jena
University of Applied Sciences Jena
Ernst-Abbe-Hochschule Jena

Fachbereich Elektrotechnik und Informationstechnik

Praktikumsbeleg

Prozessordesign

Projektname: hardCORE

eingereicht von	Mario Kellner
Studiengang	SD - embedded Systems
Matrikelnummer	631587
Datum der Abgabe	6. Mai 2017

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Zielstellung

Ziel dieser Projektarbeit für das Modul “Prozessordesign” ist es einen Prozessor in einer RISC-Architektur zu entwerfen und auf einem FPGA zu implementieren. Als Entwicklungsplattform wird das Prototyping-Board BASYS3 von Digilent mit dem FPGA “Xilinx Artix-7 (XC7A35T-ICPG236C)“ verwendet. Das Design wird mithilfe der Entwicklungsumgebung Vivado 214.4 in VHDL erstellt.

Designvorgaben:

- RISC-Architektur
- mindest. 150MHz Taktrate
- (De-)Kodierung eines reduzierten AVR-Befehlssatzes
- 32 8-Bit Register
- 1kByte Datenspeicher
- Pipelinekonzept

2 Beschreibung der Komponenten und wesentlicher Zusammenhänge

2.1 Toplevel

Das Design des Prozessors wurde in mehrere Komponenten gegliedert, die nachfolgend näher erläutert werden.

Um einen höheren Systemtakt verwenden zu können, wird ein mixed-mode-clock-manager (MMCM) aus dem IPC-Katalog genutzt. Das verwendete FPGA-Board stellt einen maximalen Systemtakt von $100MHz$ zur Verfügung, der als Eingangssignal in den MMCM führt. Alle getakteten Komponenten im Design erhalten das Taktsignal vom Ausgang der MMCM.

Um den vorwärtslaufenden Signal- und Datenfluss zu partitionieren und damit eine signifikante Erhöhung des Systemtaktes zu ermöglichen, werden drei Pipelinestufen implementiert:

- instruktion-fetch/instruktion-decode (IF/ID)
- instruktion-exec (IE)
- write-back (WB)

Bei Verwendung von Pipelines entstehen sogenannte Pipeline-Hazards, denen mit zusätzlicher Logik entgegengewirkt werden muss. In den Pipelines ID und IE sind entsprechende Maßnahmen in Form einer Forwarding-Logik (siehe Abschnitt 2.4) vorgenommen worden. Die Forwarding-Logik wird mit Komponenten gleichen Names realisiert.

Bei Sprüngen im Programmablauf wirkt die Sprunglogik (siehe Abschnitt 2.8) sowie der Programmzähler (siehe Abschnitt 2.2) Branch-Hazards entgegen. Hierfür existieren zum Einen zusätzliche Multiplexer im toplevel (ID-pipeline), mit denen ungültige Steuersignale (Schreibzugriff, Sprung- und Stacksteuerung) gehandelt werden. Ausserdem wird mit entsprechenden Steuersignalen die fehlerfreie Arithmetik für den Programmzähler sichergestellt.

Ein *reset* des Prozessors wird erzeugt, indem die fünf Taster auf dem Board gleichzeitig gedrückt werden.

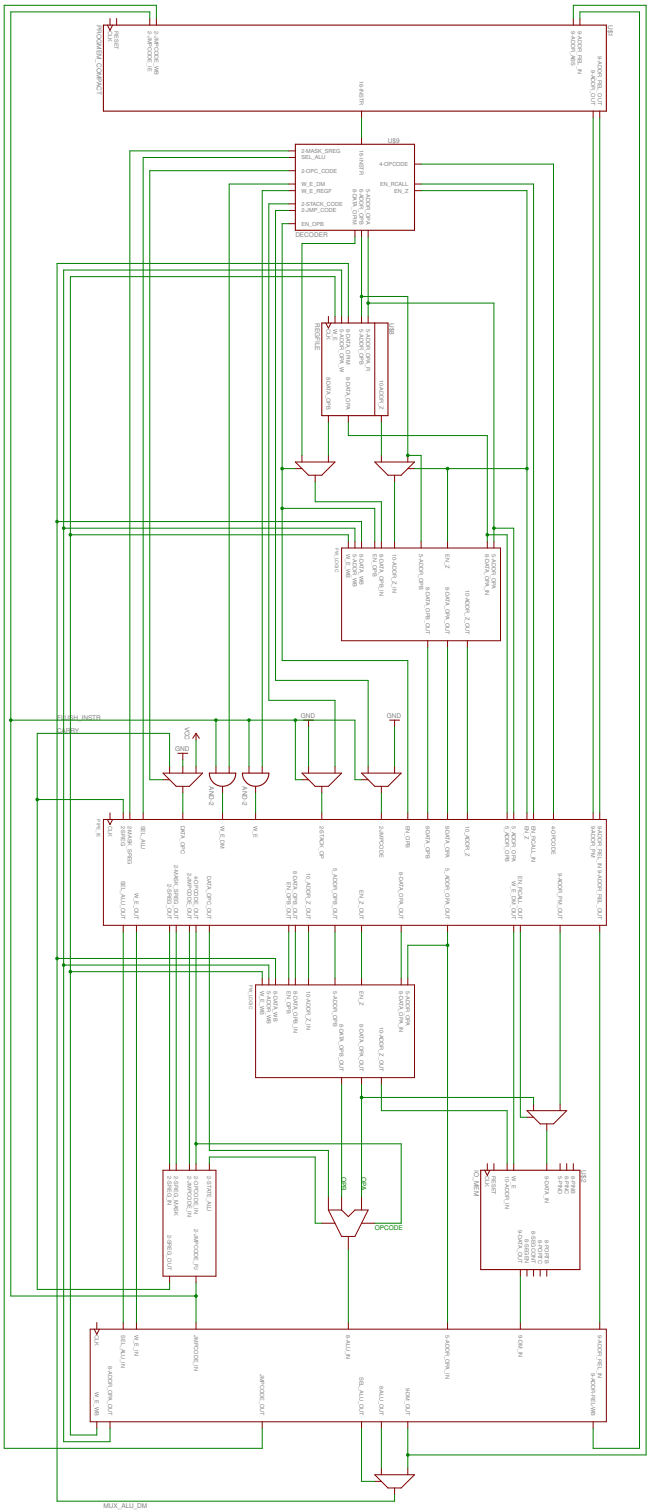

Abbildung 1: schematic toplevel

2.2 Programmzähler

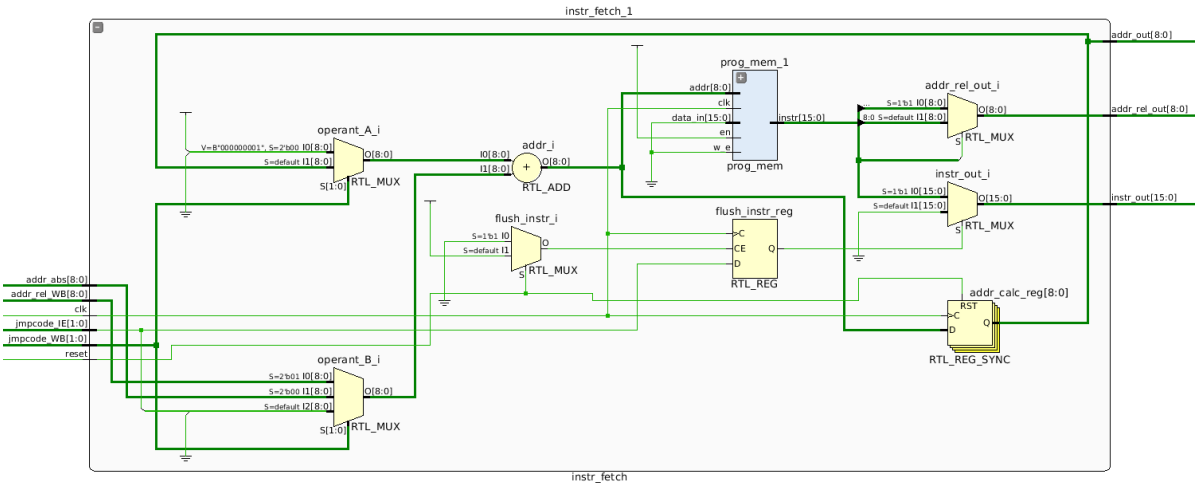

Abbildung 2: schematic Programmzähler

Der Programmzähler adressiert anhand zweier Operanten den Programmspeicher, der als Unterkomponente enthalten ist und als Block-Ram deklariert wird. Bei der Synthese wird der Programmspeicher aber als LUT-RAM gemapped, da dies laut Synthese-Log für das Signallaufzeitverhalten im vorliegenden Design günstiger ist.

Die Komponente stellt die Pipeline IF/ID dar.

In Abhängigkeit der Sprungkodierung aus der Writeback-Stufe (jmpcode_wb) werden die Operanten vorbereitet.

Hierbei werden vier Fälle unterschieden:

- absoluter Sprung
- relativer Sprung
- Inkrement
- Flush (Sprungvorbereitung)

Im Falle eines Sprungs wird ein Takt zuvor ein NOP als Instruktion erzwungen (flush-Signal aus IE). Bei jedem Takt wird eine relative Sprungadresse aus der aktuellen Instruktion dekodiert, unabhängig davon ob es sich tatsächlich um eine Sprunginstruktion handelt. Diese wird durch alle Pipelines gereicht. Des Weiteren wird die aktuelle Adresse für evtl. rcall-Instruktionen ausgegeben und bis zum Datenspeicher durchgereicht.

Der Programmspeicher ist auf 512 16-Bit-Instruktionen begrenzt.

2.3 Decoder

Die Instruktion vom Programmzähler wird dekodiert, um entsprechende Steuerkodierungen für arithmetisch-logische Operationen, Sprungtypen, Konstanten oder Stackoperationen zu erzeugen. Ausserdem werden Steuersignale erzeugt, um Operanten für arithmetische Operationen, Adressierung des Datenspeichers sowie Schreibzugriffe auf das Registerfile und den Datenspeicher zu aktivieren.

2.4 Forwarding Logik

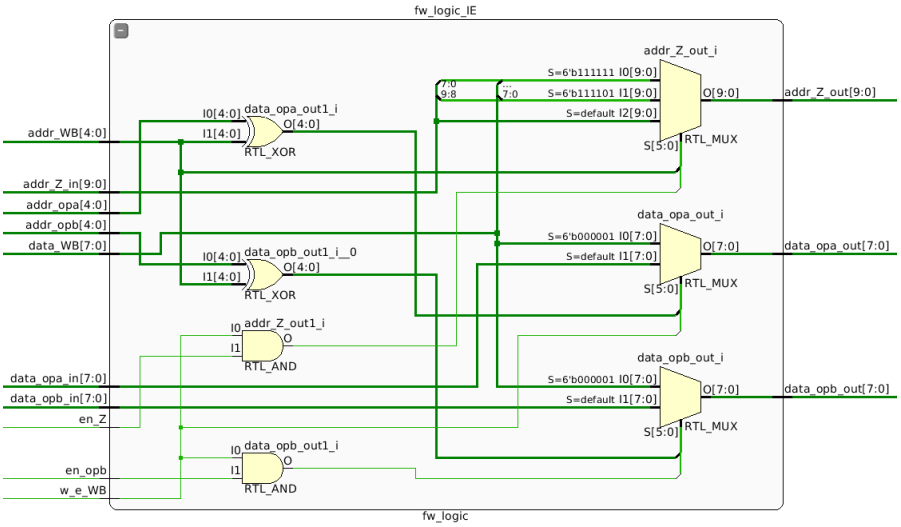

Abbildung 3: schematic Forwarding Logik

Mit der forwarding-Logik wird die Datenintegrität sichergestellt bzw. werden Pipeline-Hazards verhindern.

Sobald Register in der writeback-Pipeline vorliegen, die in der IE-Pipeline verändert wurden und in den Pipelines IF/ID sowie IE zur weiteren Verarbeitung vorliegen, müssen diese aktualisiert werden. Hierfür wird die Adresse des zu aktualisierenden Registers der forwarding-Logik als Vergleichswert zugeführt.

Sowohl in der intruction-fetch als auch in der intruction-execute-Pipeline werden jeweils vier Operanten (OPA, OPB, ZH, ZL) weitergeleitet. In beiden Pipelines muss die Adresse des überschriebenen Registers mit den Adressen aller Operanten verglichen werden. Bei Übereinstimmung einer Operantenadresse mit der writeback-Adresse wird der Dateninhalt nur aktualisiert, wenn der Operant auch aktiv ist. Hierfür wird ein zusätzliches Enablesignal je Operant benötigt.

Das Verhalten der Forwarding-Logik ist in Listing 1 als Pseudocode beschrieben.

```
if (w_e_rf == aktiv AND adresse_wb == adresse_OPA)
  OPA <= data_writeback

if ( (w_e_rf AND en_opB) == aktiv AND adresse_wb == adresse_OPB)
  OPB <= data_writeback

if ( (w_e_rf AND en_Z) == aktiv AND adresse_wb == 0x31)
  ZH <= data_writeback

if ( (w_e_rf AND en_Z) == aktiv AND adresse_wb == 0x30)
  ZL <= data_writeback
```

Listing 1: Pseudocode forwarding-logik

Um möglichst wenig Schaltstufen zu verwenden, werden die Vergleichswerte zwischen Operantenadresse und writeback-Adresse mit den jeweiligen Enablesignalen des Operanten verknüpft und dem entsprechenden Multiplexer als Eingang zugeführt.

2.5 Registerfile

Das Registerfile dient als nicht-persistenter Speicher für die Verarbeitung von Registerinformationen. Es existieren 32 8-bit Register, wobei die zwei höchstwertigen Register für die Adressierung des Datenspeichers dienen können (Z-Register). Das Auslesen von Registerinformationen geschiet ungetaktet bzw. nebenläufig über Adressierung vom Dekoder. Ein verändertes Register wird nach dessen Adressierung (decode) mit einer Verzögerung von drei Takten taktsynchron geschrieben.

2.6 Pipelinestufe IE mit Stackpointer

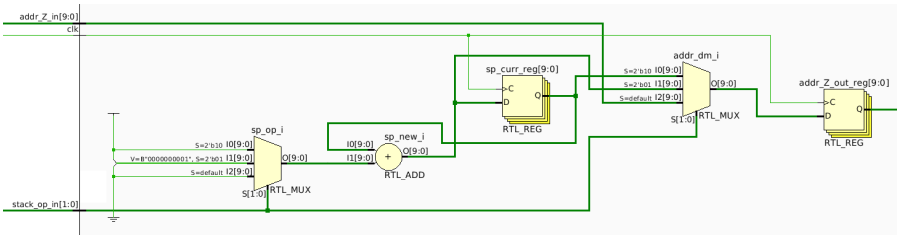

Abbildung 4: schematic IE (Stackpointer-Handler)

Die Komponente dient als Pipelinestufe zwischen der Instruktionsdekodierung (ID) und Instruktionsausführung (IE). Neben FlipFlops zum Speichern von Daten und Steuersignalen ist hier auch der Handler für den Stackpointer enthalten. Idealerweise würde der Handler für den Stackpointer in einer separaten Komponente implementiert werden. Da die De-/Inkrementierung des Stackpointers ein clock-Signal benötigt, wurde dieses direkt aus der Pipelinestufe verwendet. Des Weiteren wird die Auswahl des Pre- bzw. Postcounters und der Z-Adresse auf diese Weise mit einem einzigen Multiplexer realisiert.

2.7 ALU

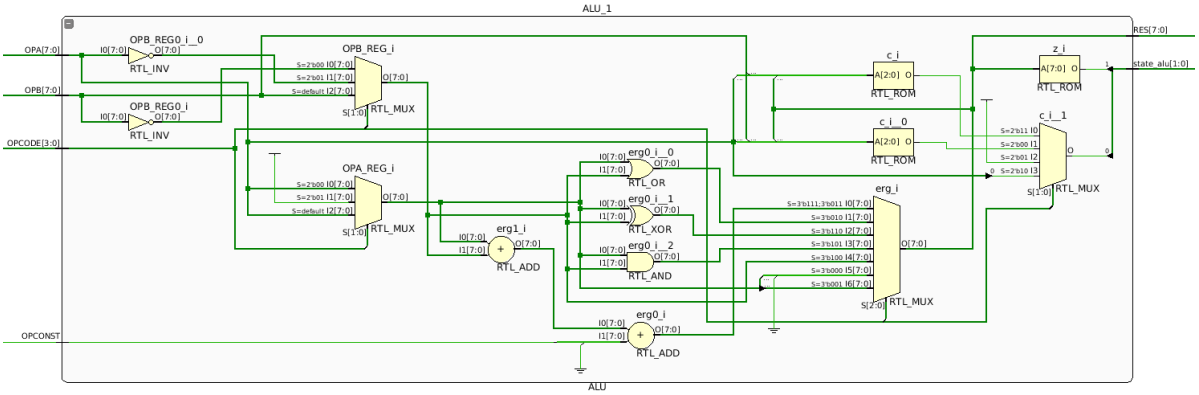

Abbildung 5: schematic ALU

Die arithmetisch-logische Einheit ALU führt Manipulationen des Operanten A durch und berechnet den aktuellen Zustand. Da die Instruktionsmenge nur bedingte Sprungbefehle umfasst, die auf das Carry- und Zeroflag zurückgreifen, wird nur das Carry- und Zerobit berechnet. Die Berechnung erfolgte ürsprünglich durch eine drei- bis vierstufige logische Verknüpfung, wurde aber zugunsten der Performance durch ROMs ersetzt.

Am Eingang der Alu werden zunächst die Operanten A und B aufbereitet. Hierfür muss lediglich zwischen drei Operationen (Subtraktion, Einer-Komplement, restliche Operationen) unterschieden werden. Für der Kodierung arithmetisch/logischer Operationen wurde besonderes Augenmerk darauf gelegt, das Kodewort so klein wie möglich zu halten, um Schaltzeit der beteiligten Multiplexer (siehe Abb. 5) zu reduzieren. Alle für die ALU relevanten Instruktionen werden auf acht arithmetische Operationen reduziert und können mit drei Bit eindeutig (de-)kodiert werden. Der Operant C (1, 0, Carry) wird bereits eine Taktstufe zuvor vom Dekoder aufbereitet und der ALU direkt zugeführt.

Tabelle 1: Kodierung arithmetischer Operationen

Operation (Instr.)	set sreg			
	set operant			
		Operation (Arithm.)		
add, lsl, adc, rol	1	1	1	1
com, sec	0	1	1	1
sub, subi, cp, cpi, dec, inc	0	0	1	1
or, ori	1	0	1	0
eor	1	1	1	0
and, andi, brcs	1	1	0	1
mov, ldi	1	1	0	0
lsr, clc	1	0	0	0
asr	1	0	0	1

2.8 Sprungdekoder

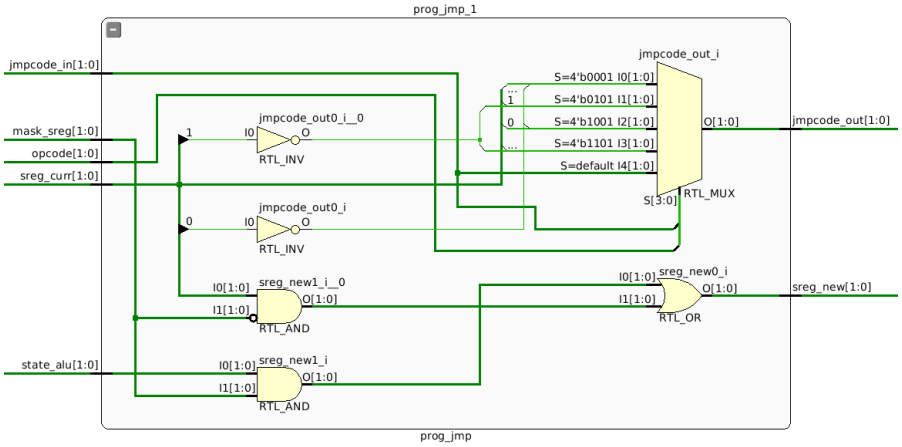

Abbildung 6: schematic Sprungdekoder

Die Komponente prog_jmp bereitet Sprungbefehle vor, indem das Eingangssignal "jmpcode" umkodiert wird. Für den Fall eines Sprungs (ausser inkrement) wird aus dem resultierenden "jmpcode" ein Steuersignal zum flushen der Pipelines abgeleitet. Dieses Steuersignal dient ebenfalls als Operant für den Programmzähler, um die Vorwärtszählung anzuhalten.

Das Statusregister hat für die Auswertung von bedingten Sprüngen zentrale Bedeutung. Es wird ebenfalls in dieser Komponente aktualisiert und in die vorgelagerte Pipelinestufe rückgeführt. Dies ermöglicht es FlipFlops für die Maskierung in der Writeback-Stufe einzusparen und das Carryflag als Operant C direkt in der IF/ID-Pipeline auszuwählen, sowie für den nächsten Takt (IE) der ALU zur Verfügung zu stellen.

Tabelle 2: Kodierung von Sprungoperationen

Sprungoperation (Instr.)	**Sprungtyp**	**Kodierung**			
		OPCODE[1...0]		jmpcode	
incr	relativ, unbedingt	d	d	1	1
brcc	relativ, bedingt	0	0	0	1
brne	relativ, bedingt	0	1	0	1
brcs	relativ, bedingt	1	0	0	1
breq	relativ, bedingt	1	1	0	1
rjmp	relativ,unbedingt	d	d	1	0
rcall	relativ,unbedingt	d	d	1	0
ret	absolut,unbedingt	d	d	0	0

2.9 Daten- und IO-Speicher

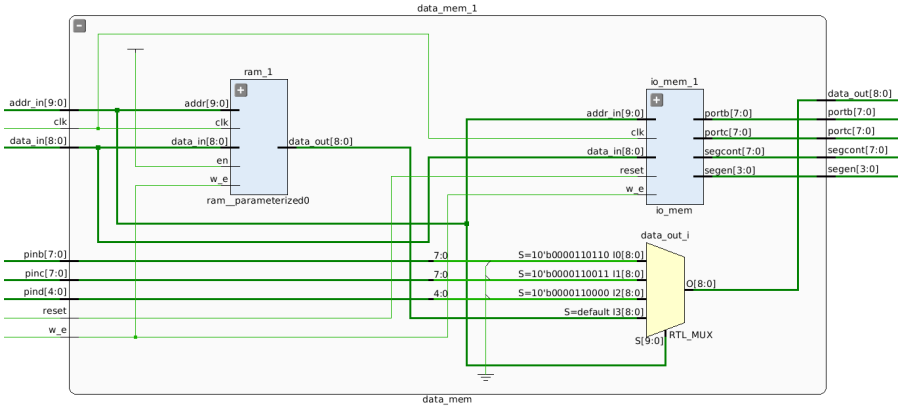

Abbildung 7: schematic Daten- und IO-Speicher

Der Datenspeicher umfasst 1024 Speicherplätze zu je 9 Bit und ist als Unterkomponente implementiert. Die Registerbreite wurde auf 9 Bit erweitert, um rcall-Operationen in einem Takt realisieren zu können.

Der Datenspeicher wird als distributed-RAM deklariert. Ein angepasstes Design mit BlockRAM und entsprechenden Änderungen der writeback-Pipelinestufe haben Platzierungen im Floorplan zur Folge, die ungünstigere Laufzeitergebnisse liefern.

Im Falle einer Port- oder Segmentadressierung werden Daten nicht im Datenspeicher, sondern in einem separaten IO-Speicher abgelegt. Die IO-Unterkomponente verfügt über einen Treiber, der zyklisch die Ausgabe auf der 7-Segment-Anzeige aktualisiert.

Die Signale der Hardwarepins werden im Entwurf formal nicht getaktet. Bei der Synthese werden aber automatisch FlipFlops erzeugt, damit die Signale taktsynchron abgefragt werden können.

2.10 Pipelinestufe WB

Die Komponente writeback dient als reine Taktstufe, in der lediglich Daten und Steuersignale gespeichert bzw. abgefragt werden.

3 Leistungsbeschreibung

Tabelle 3: Instructionset

Kodierung	Kommando	Operanten	Takte
0000000000000000	nop		1
000011rddddd rrrr	lsl	r	1
000011rddddd rrrr	add	r,r	1
000110rddddd rrrr	sub	r,r	1
000101rddddd rrrr	cp	r,r	1
000111rddddd rrrr	rol	r	1
000111rddddd rrrr	adc	r,r	1
001000rddddd rrrr	and	r,r	1
001001rddddd rrrr	eor	r,r	1
001010rddddd rrrr	or	r,r	1
001011rddddd rrrr	mov	r,r	1
1000000ddddd 0000	ld	r,e	1
1000001rrrrr 0000	st	e,r	1
111101lllllll 000	brcc	l	1,3
111101lllllll 001	brne	l	1,3
111100lllllll 001	breq	l	1,3
111100lllllll 000	brcs	l	1,3
1001010rrrrr 0000	com	r	1
1001010rrrrr 0101	asr	r	1
1001010rrrrr 1010	dec	r	1
1001010rrrrr 0011	inc	r	1
1001010rrrrr 0110	lsr	r	1
1001010100001000	ret		1,3
1001010000001000	sec		1
1001010011001000	cls		1
1001000rrrrr 1111	pop	r	1
1001001rrrrr 1111	push	r	1
1110KKKKdddd KKKK	ldi	d,M	1
0011KKKKdddd KKKK	cpi	d,M	1
0101KKKKdddd KKKK	subi	d,M	1
0110KKKKdddd KKKK	ori	d,M	1
0111KKKKdddd KKKK	andi	d,M	1
1100LLLLLLLLLLL	rjmp	L	1,3
1101LLLLLLLLLLL	rcall	L	1,3
10110PPddddd PPPP	in	r,P	1
10111PPrrrrr PPPP	out	P,r	1

Legend:

r	any register	P	Port address value [0 to 63] (in, out)
d	'ldi' register (r16-r31)	K	immediate value [0 to 63]
e	pointer register (Z)	l	signed pc relative offset [-64 to 63]
M	immediate value [0 to 255]	L	signed pc relative offset [-1024 to 1023]

3.1 IPC

Die Taktrate des Prozessors beträgt $173MHz$ mit einer Sklackreserve (WNS) von $0.004ns$.

Ermittlung der Instruction per Cycle (IPC) für einen definierten Instruktionsmix:

Tabelle 4: Instruktionsmix

Instruktionstyp	Anteil	Zyklen
Load	15%	$15 \cdot 1$
Store	5%	$5 \cdot 1$
Branch(taken)	16%	$16 \cdot 3$
Branch(not taken)	4%	$4 \cdot 1$
Arithm./Logisch	60%	$60 \cdot 1$
Σ	100%	132

Der IPC-Wert beträgt $\frac{100}{132} = 0.75$. Durchschnittlich werden 131 MIPS verarbeitet.

3.2 Ressourcenbedarf

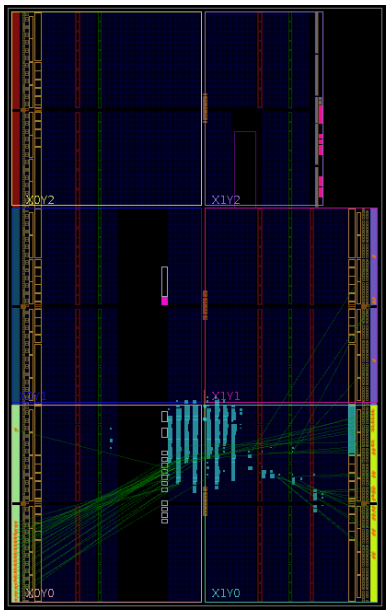

Abbildung 8: Floorplan

Tabelle 5: post-synthesis

Resource	Estimation	Available	Utilization
FF	477	41600	1.15%
LUT	958	20800	4.61%
Memory LUT	272	9600	2.83%
I/O	49	106	46.23%

Tabelle 6: post-implementation

Resource	Estimation	Available	Utilization
FF	477	41600	1.15%
LUT	918	20800	4.41%
Memory LUT	272	9600	2.83%
I/O	50	106	47.17%
BUFG	2	32	6.25%
MMCM	1	5	20.00%

Leistungsaufnahme = 225mW
Global Vertical Routing Utilization = 0.69%
Global Horizontal Routing Utilization = 0.84%

4 Maßnahmen zur Leistungssteigerung

statische Sprungvorhersage Die Anzahl benötigter Takte bei getätigten bedingten Sprüngen kann reduziert werden. Hierfür müsste der Programmzähler bzw. Speicher als Dual-Port-BlockMemory erweitert werden. Die Sprungadresse am Ausgang der Komponente "Programmzähler" wird dann direkt rückgeführt und gibt im nächsten Takt den Instruktionscode des Sprungsziels an einen zusätzlichen Decoder. Ebenfalls müsste das Registerfile für einen dualen Zugriff auf die Register erweitert werden. Der Sprungdekoder würde dann die Eingangssignale der Pipelinestufe IE zwischen getätigtem und nicht getätigtem Sprung schalten.

Mit dieser Methode könnte die Anzahl benötigter Takte von drei auf zwei reduziert und die MIPS bzw. IPC gesteigert werden.

Verbesserter Sprungdekoder Im aktuellen Design wird nach der Dekodierung bedingter als auch unbedingter Sprünge ein zusätzlicher Takt benötigt, um den Programmzähler anzuhalten und im nächsten Takt die Sprungadresse an den Programmzähler zu übergeben. Denkbar wäre, dem Programmzähler die Sprungadresse unmittelbar mit der Dekodierung zu übergeben. Die relative Sprungadresse steht ohnehin in jeder Pipelinestufe zur Verfügung. Für absolute Sprungadressen könnte ein separater Stack in der IE-Pipline implementiert werden, aus dem ohne Taktverzögerung die Adresse an den Programmzähler übergeben werden kann.

In Kombination mit der statischen Sprungvorhersage könnten alle Sprünge auf einen Takt reduziert werden. Die MIPS würden dann 1:1 vom Systemtakt abhängen.

Erhöhung des Systemtaktes Der kritische Pfad kann durch Änderungen im Design weiter reduziert werden. Bspw. könnte der Operant des Stackpointers (siehe Abschnitt 2.6) direkt verdrahten werden, statt hierfür einen Multiplexer zu verwenden. Jegliche Einsparung oder Erweiterung an logischen Elementen beeinflusst die Entscheidungsfreiheit der Optimierungsalgorithmen bei der Synthese. Insofern kann die Auswirkung auf die Slacktime nicht zuverlässig abgeschätzt werden.

Der logische Aufbau auf RT-Ebene ist nicht der einzige Einflussfaktor auf den kritischen Pfad. Die Namensgebung von Signalen und Komponenten kann das Syntheseergebnis ebenfalls signifikant beeinflussen.

weitere Möglichkeiten

- Superskalare Architektur (parallele Verarbeitung v. Instruktionen)
- dynamische Sprungvorhersage (Branch History Table oder Branch Target Buffer)
- Multiplizierer (keine Kombination v. add/branch Befehlen -> MIPS-Steigerung)