EE577B Troy Processor Project Report

		TABLE OF CONTENTS	
1	Introduct	ion	4
	11 Arit	hmetic Logic Unit (ALU) Module	5
	111	Inputs and Outputs	5
	112	Functional Test Results	5
	113	Synthesis Results	5
	114	Post-Synthesis Functional Test Results	5
	1.2 Con	trol Module	5
	121	Inputs and Outputs	5
	122	Functional Test Results	6
	123	Synthesis Results	
	12.5	Post-Synthesis Functional Test Results	7
	13 Data	Memory Module (provided – see constraints above)	8
	131	Inputs and Outputs	8
	132	Functional Test Results	8
	133	Synthesis Results	8
	134	Post-Synthesis Functional Test Results	8
	1.5.1	uction Memory Module (provided – see constraints above)	8
	1 4 1	Inputs and Outputs	8
	142	Functional Test Results	9
	1 4 3	Synthesis Results	9
	1 4 4	Post-Synthesis Functional Test Results	9
	15 Pros	ram Counter Module	9
	151	Inputs and Outputs	9
	152	Functional Test Results	9
	153	Synthesis Results	9
	154	Post-Synthesis Functional Test Results	10
	16 Regi	ister File Module	10
	161	Inputs and Outputs	10
	1.6.2	Functional Test Results	10
	1.6.2	Synthesis Results	11
	1.6.4	Post-Synthesis Functional Test Results	12
	17 Mux	Module	12
	171	Inputs and Outputs	14
	1.7.1	Functional Test Results	14
2	CPI Pine	line	14
-	2 1 CPL	I Module	
	2.1 CIC	Inputs and Outputs	
	2.1.1	Functional Test Results	
	2.1.2	Synthesis Results	25
	2.1.5	Post-Synthesis Functional Test Results	23
3	Instructio	n set implementation	
Δ	Adder/Su	htset imperioritation	
7	41 Fund	vtional Test Results	
	4.1 Fully 4.2 Sum	hesis Results	
	4.3 Post	-Synthesis Functional Test Results	

Page 2 of 48

EE577b Troy Processor Project by Zhyang Ong <u>zhiyang@ieee.org</u> and Andrew Mattheisen <u>amattheisen@gmail.com</u>

5 SI	nifter Design	
5.1	Functional Test Results	
5.2	Synthesis Results	
5.3	Post-Synthesis Functional Test Results	
6 M	ultiplier Design	
6.1	Functional Test Results	
6.2	Synthesis Results	
6.3	Post-Synthesis Functional Test Results	
7 H	azard management and data forwarding	41
7.1	Functional Test Results	
8 O	ptimization	
9 Sy	nthesis Report	
10	Final specifications of your designs	
11	Possible Future Enhancements	
12	Conclusion	
13	References	

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

1 Introduction

This report documents the process of designing and testing a 128 bit microprocessor to implement a limited set of instructions. The goal of this project is to create a microprocessor and maximize its speed. The design process is outlined below:

- Develop a conceptual design
- Divide the design into modules
- Implement each module with Register Transfer Logic (RTL) style Verilog Hardware Description Language (HDL)
- Test each module functionally using NcVerilog
- Synthesize all modules developed using Synopsys Design Compiler
- Verify each module synthesizes correctly via post synthesis functional testing
- Integrate all modules and test functionality
- Synthesize the Central Processing Unit (CPU) module
- Verify the top level design synthesized correctly via post synthesis functional testing

Note: The process described above must be iterated to develop an optimal solution.

The instruction following instruction set has been implemented:

- Wide word addition (WADD)
- Wide word AND (WAND)
- Wide word load from data memory using immediate addressing (WLD)
- Wide word signed multiplication of the even bytes or double-bytes (a double-byte is 16 bits) (WMULES)
- Wide word signed multiplication of the odd bytes or double-bytes (WMULOS)
- Wide word unsigned multiplication of the even bytes or double-bytes (WMULEU)
- Wide word unsigned multiplication of the odd bytes or double-bytes (WMULOU)
- Wide word move (WMV)
- Wide word NOT (WNOT)
- Wide word OR (WOR)
- Wide word byte permute (WPRM)
- Wide word shift logical left (WSLL)
- Wide word shift logical left immediate (WSLLI)
- Wide word shift arithmetic right (WSRA)
- Wide word shift arithmetic right immediate (WSRAI)
- Wide word shift logical right (WSRL)
- Wide word shift logical right immediate (WSRLI)
- Wide word store to data memory using immediate addressing (WST)
- Wide word subtraction (WSUB)
- Wide word XOR (WXOR)

Note: branch and jump commands were not implemented - this greatly simplifies the design.

The design of the microprocessor will be constrained by the following:

• The design must interface to a predefined Instruction Memory RTL style Verilog HDL module (this module is included in section xx).

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

- The design must interface to a predefined Data Memory RTL style Verilog HDL module (this module is included in section xx).
- Top level functional testing must be accomplished using a predefined Verilog HDL test bench (this module is included in section xx).
- The multiplier will not use the "*" symbol available in RTL style Verilog HDL (this is because the synopsis design compiler available at USC uses a sub-optimal multiplier when it encounters the "*" symbol in RTL style Verilog HDL).

Benchmarking is a common practice of measuring speed in the industry. Benchmarking will be used to measure the speed of the processor developed. Several benchmarks will be provided to test our microprocessor. We will measure the execution time the processor requires to run each benchmark program and report the results.

The basic modules used to construct the microprocessor are described in the following subsections.

1.1 Arithmetic Logic Unit (ALU) Module

1.1.1 Inputs and Outputs

Inputs: [0:4] aluop, [0:1] ww, [0:127] reg_a, [0:127] reg_b Outputs: [0:127] result

1.1.2 Functional Test Results

The ALU is composed of addition (see section 3), shift (see section 4), and multiplication (see section 5) logic.

1.1.3 Synthesis Results

The ALU is composed of addition (see section 3), shift (see section 4), and multiplication (see section 5) logic.

1.1.4 Post-Synthesis Functional Test Results

The ALU is composed of addition (see section 3), shift (see section 4), and multiplication (see section 5) logic.

1.2 Control Module

1.2.1 Inputs and Outputs

Inputs: [0:31] instruction Outputs: [0:4] aluop, [0:4] rrdaddra, [0:4] rrdaddrb, [0:4] rwraddrd, [0:2] regop, [0:1] ww, [0:20] maddr, memEn, memWrEn, [0:15] wbyteen, [0:127] immediate, (reginmuxop, aluinmuxop will also be included to control multiplexers in the pipeline as described in the next section)

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

1.2.2 Functional Test Results

Functional testing of the control logic was performed to ensure that the control logic was generating the expected signals for a given input stream of instructions. The input stream of instructions tested *each possible instruction at the time of implementation.

*The multiplication instructions had not implemented before the original

Functional testing of the control logic, and therefore are not tested at the module level.

The following instructions were tested and the output of the control module was carefully scrutinized to the internal logic was correct. As can be seen in the waveforms below, the control logic is very important – it provides signals to all other modules to control their operation.

The following instructions were checked: 32'b0001000110101010010110000000000; // ADD 32'b000100011010101001011010000000; // ADD 32'b00010001101010101010111000000000; // ADD 32'b00010001101010101010111000100000; // ADD 32'b00010001101010101010111001000000; // ADD 32'b00010001101010100101110010001001; // AND 32'b000001011010000000000000000000001: // LD 32'b0010000110101010000001001001001; // VMV 32'b00010001101010100000010100001000; // NOT 32'b000100011010101001011101001010: // OR 32'b000100011010101010101110100001100; // PRM 32'b000100011010101001011101010000; // SLL 32'b00010001101010100111110110010001; // SLLI 32'b000100011010101010101110110010110; // SRA 32'b00010001101010100111110110010111; // SRAI 32'b00010001101010100101110110010100; // SRL 32'b00010001101010100111110110010101; // SRLI 32'b000010011010000000000000000000001; // ST 32'b00010001101010100101111001000001; // SUB 32'b000100011010101010101111101001011: // XOR 32'b00000001101010100101110110001011; // NOP

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

Ve a	ludra	.usc.ed	u:10025 (mattheis)		_																					
E												Nave	forn 1 -	SimVision												· 🗆
E	ile	Edit ⊻	fiew Exglore Forms	at Simulation	₩in	idows																				Help
6	ř	n ~	3 @a @& X	13 19 19 19 19 19 19 19 19 19 19 19 19 19	赩,	Ъ.															*	• 10 to 10	Send To: 🕻	D 🚟		V 📰 📾
Se	arch	Names	Signal 🕶	- j 🍂 #	1	Search Tim	es: Value	•		急众。																
	2 Tir	neA 🛒	= 215 • ns	J 🖧 • 🌾	×¢,			40 (4)	Simulation '	lime: 215ns	s+0 [1	ime Range	0 : 215ns		• •	999
×	۲	E	Baseline = 0																							
Ģ	8	Sursor-E	Baseline = 215ns		Das	eline = U																			1	imeA = 215ns
	¥ .		-1.1	Cursor -	°	10n	s 20r	15 30	ns 40r	s 50ns	5 60n:	s 70n	s 80r	15 90n	5 10	10ns 1	10ns 11	20ns 13	0ns 141	0ns 150	Ins 16	0ns 170	ins 180	ns 19	Ons 2	00ns 21
	~	- B->	aluinmuxop	U 71-10		00					Voo	10	000	00	0.3	loc	10	V11	114	V17	V1.4	115	¥10	101	YOR	Y10
		1-12-1	aiuop	110	XX	600	00000	10000		000000	100000	100000	103	100	VE O O	100	100000	11	10	17800	14	1.7200	10000	101	0000	100000
	18		initieulate	(h01a)	v t	1122	V1122	1122	¥11אס	V1122	1122	0530	21 3 3	1122	11127	N/112	λ»/11λ	xx/1122)1133	1122	11122	111221	10000	1122	1112	at (01 a at
		. IS.	maddr	1014	n vÞ	0258	0252		• 0a50	1	1025C)		0204	0205	0250	• (∩ ≥ 5)	D# (025)	 1▶/∩≥7⊓	• (125D	I 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	0250	■ 1111	0000	0255	► 0 Δ 5	EN DASDN
			memEn	0	Ë																					
	Ľ		memWrEn	о 0		i –																				
		∎	memop	'hx	x																					
Į		1- 端.	ppp	'h5	x	0	2	4				Y0	4	5									0	6	7	5
			reginmuxop	0			~						Î													
		∎-18+	regop	'h0	x							1			7				7		7			7		0
		∎-18+	rrdaddra	'h0A	хx	0A						00	0A										0D	0A		
		9-48+	rrdaddrb	'hOB	хх	0B						00			0B			OF	(0B	OF	0B	OF	0D	0B		
		- %+	rwraddrd	'hOD	ΧХ	0D																				
		1 喻+	wbyteen	'hOFOF	x۲	FFFF	FF00	AAAA	CCCC	FOFO		FFFF	FOFO	5555	3333	3 (555)	5 (333)	3 OFOF					FFFF	C000	000	3 OFOF
		±-i‱	ww	'h2	х	0	1	0	1	2		0	2	0	1	0	1	2					0	1		2
1																										
5																										
																										1
IJ		J		KII X	.Fb																					
1	0																								0	objects selected

1.2.3 Synthesis Results

The following results were found in the control.area report:

Number of ports:	226
Number of nets:	129
Number of cells:	98
Number of references:	9
Combinational area:	2548.000000
Noncombinational area:	0.000000
Total cell area:	2548.000000

The following timing information was extracted from the timing report: Max delay: 0.95 delay units.

The check_design report indicated that some input nets were not used and that some output nets were shorted together, or shorted to logic 0. We looked into these and found that all warnings were expected.

1.2.4 Post-Synthesis Functional Test Results

Using the same instruction set, the same results were obtained from the netlist generated by the synthesis tool (see below).

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

V2 al	udra.	usc.ed	u:10025 (mattheis)																							_loi×
E												Hav	efora 1 -	SinVision												
Ð	e <u>E</u>	dit ⊻	iew Explore Forma	t Simulation	₩inc	swot																				Help
đ	-	n 🗠	x @ @ × i	14 A A A A A A A A A A A A A A A A A A A	赩,	ы. С															*	• 🖏 S	Send To: 🕻	\$) E	e 📰 🤇	9 📰 🚟
Se	arch M	lames	Signal 🕶	<u>.</u>	s	earch Tirr	nes: Valu	.e •		- A A																
ľ.	Tim	eA 🚽	= 215 • ns	· ##:- 🌸	20-	1		(+) ()† (Simulatio	n Time: 215	ins + 0										Ti	me Range:	0:215ns		• 🗨 (9, 9, 9, 9,
×	D	E	Baseline = 0		Bure	line = 0																				
G	3	ursor-c	aseline = 215ns	Current -	0	1100	- 190) 	ine la	Doe let	ne leor		- leo	ne lon		0.000 11	0.000 1.0	0ne 12	on lui	ne 150	ne liso	ne 170	ne 190	ne line	TimeA	=215(0)ns
ß	١,	- 16	aluinmuxop	0	×		<u> </u>	<u> </u>	<u> </u>	<u>, 19</u>		<u>.</u>	13. PV	13 130	·····					13				13	10 200	
	C.	- šģ.	aluop	'h18	хx	00					09	18	09	08	0A	0C	10	11	16	17	14	15	18	01	0B	18
			clk	1																						
	e	- 4g.	immediate	′h000⊳	00	00000	0																			
	E	¥ĝ.	instr	′h01A)	х•	11AA•	11AZ	A• 11A4	↓ 11A	A• 11A4	↓ 11AA	• 05A0	• 21AA	• 11AA	• 11AA	• 11AZ	• 11AZ	↓ 11AA	 11AA 	11AA	11AA	11AA	09A0	11AA	11AA	01AA
]¥≩.	maddr	'h0A5•	X,	UA58	UA54	AP UA50	JUA5	C• UA50	DA5C	• 0000	• UAU4	• UAU5	UA5D	• UA51	≫ UA51	DA ID	• UA5D	UA /D	UA5D	UA (D)	10000	UA5E	UA5F	UA5D
			nemop	' h2	A X	2	72	72	ĬZ.	Z		2	2	2	2	12	2	2					2	2		Yz
			reginmuxop	0			<u>, с</u>							<u></u>	<u>, ч</u>	<u>, </u>							<u>,</u>	<u>, -</u>		<u>, e</u>
	e		regop	'hO	x	7						1	5	5	7			5	7	5	7	5	2	7		0
	e	Sg.	rrdaddra	'h0A	хx	0A						00	0A										0D	A0		<u> </u>
	e	¹ 8.	rrdaddrb	'h0B	хх	0B																				
	6	¥\$.	rwraddrd	'h0D	хx	0D																				
	e		wbyteen	'hOFOF	xx,	FFFF	FF00	aaai	A CCC	C FOF)	FFFF	FOFC	5555	3333	5555	3333	3 OFOF					FFFF	C000	0003	OFOF
		- %.	ww	'hZ	X	2																				
	L																									
	L																									
J	R	1	51		12																					
10																									0 ot	ojects selected

1.3 Data Memory Module (provided – see constraints above)

1.3.1 Inputs and Outputs

Inputs: clk, wren, memEn, [0:7] memAddr, [0:127] dataIn Outputs: [0:127] dataOut

1.3.2 Functional Test Results

There was no need to test the data memory module individually since it was provided to us and we were unable to alter it.

1.3.3 Synthesis Results

The data memory module was unsynthesizeable, as advertized when it was provided to us, therefore we did not attempt to synthesize it.

1.3.4 Post-Synthesis Functional Test Results

Since no synthesis was performed, no post synthesis functional testing was performed either.

1.4 Instruction Memory Module (provided – see constraints above)

1.4.1 Inputs and Outputs

Inputs: [0:7] addr Outputs: [0:31]instruction

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

1.4.2 Functional Test Results

There was no need to test the instruction memory module individually since it was provided to us and we were unable to alter it.

1.4.3 Synthesis Results

The instruction memory module was unsynthesizeable, as advertized when it was provided to us, therefore we did not attempt to synthesize it.

1.4.4 Post-Synthesis Functional Test Results

Since no synthesis was performed, no post synthesis functional testing was performed either.

1.5 Program Counter Module

1.5.1 Inputs and Outputs

Inputs: clk and reset Outputs: [0:20] program_counter

1.5.2 Functional Test Results

The following waveform was generated by the program counter. As you can see, the value of the program counter increments by 4 each positive clock edge – which will allow us to address the next 32 bit wide memory location in the instruction memory.



1.5.3 Synthesis Results

The following results were found in the control.area report: Number of ports: 34

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

Number of nets:	102
Number of cells:	66
Number of references:	4
Combinational area:	3576.000000
Noncombinational area:	3072.000000
Total cell area:	6648.000000

The following timing information was extracted from the timing report: Max delay: 5.0 delay units.

The check_design report indicated that one of the synthesized cells did not drive any nets. We kept this in mind while testing the functionality of the synthesized code and found that the warning could be ignored (see next section).

1.5.4 Post-Synthesis Functional Test Results

Using the same inputs, the same results were obtained from the netlist generated by the synthesis tool (see below).



1.6 Register File Module

1.6.1 Inputs and Outputs

Inputs: clk, [0:127] wrdata, wren, rd1en, rd2en, [0:4] wraddr, [0:4] rd1addr, [0:4] rd2addr, [0:15] wbyteen

Outputs: [0:127] rd1data, [0:127] rd2data

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

1.6.2 Functional Test Results

It is difficult to fit the results into a single screen shot, but below are a few screenshots showing that several registers are written to then later the correct values are read from the registers.



by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

Valalud	a.usc.ed	iu:10025 (mattheis)	_		×
. =				Reveforn 1 - SinVision	· 🗆
Eile	Edit }	v/iew Explore Forms	at Simulation	n Windows	Help
ê	n o	5 0n 🙉 🗙 🗄	10 10 10 10 10 10 10 10 10 10 10 10 10 1	🖄 🖏 - 🗞 Send To 🖏 🎆 🖿 🖁	
Searc	h Names	s: Signal 🕶	<u>_</u> %, %	🚯 Search Times Value 🕶 🔗 溴	
x2	imeA 💌	= 0 • ns	王 武• 金	🖺 🏡 📔 🔣 🔣 🚼 Simulation Time: 350ns+0 🛛 👘 👬 🛛 Time Range: 179 63ns: 223.39n 🖵 👷	
×⊙		Baseline = 0			
69	Cursor-I	Baseline = 0			
ŝ			Cursor 👻	180ns 190ns 200ns 210ns 2	20ns
P		clk	x		
	⊡ ∛	rd1addr	'hxx		J2
	⊡ -%+	rd1data	'hxxx	•xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	*******
		rd1en	х		
	■約	rd2addr	'hxx	07 06 0	15
	₽ ~%+	rd2data	′hxxx•	*xxxxxxxxx561C65AD1C65AD1C64D61C455 x	
	- \$\\\	rd2en	х		
	⊡ -%≩•	result	′hxxx	·(000000000000000000000000000000000000	
	由	temp	'hxxx	×1000000000000000000000000000000000000	
	⊞ #3	wbyteen	'hxxxx	· <u>/FFFF /</u> 7FFF <u>/</u>	OFF
	⊡ ¥3	wraddr	'hxx	•14	
	⊡ ¥3	wrdata	'hxxx	• <u>000000000000000000000000000000000000</u>	
	 4(E	wren	х		
	白	reg_file	('hxx►		
	.	- ≪ reg_file[0]	′hxxx	xxxxxxxxxxxxxxxxAE787897EAC22354	
	÷.	reg_file[1]	′hxxx►	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	
	÷.	reg_file[2]	'hxxx►	xxxxxxxxxxxxxxxxxx531264684565	
	÷.	- 🎭 reg_file[3]	'hxxx⊾	xxxxxxxxxxxxxxxxxxxx4987984614	
	B	-⊛ reg_file[4]	'hxxx,	xxxxxxx09C4AE54864C6AE464CA3544	
		👒 reg_file[5]	′hxxx	xxxxxxxx1AE151C53AE15C153AE1C4	
		- 🕸 reg_file[6]	′hxxx	xxxxxxxxxxD561C65AD1C6AD61C455	
	÷.	-% reg_file[7]	'hxxx⊦	xxxxxxxxxxxx6C5D4A564CD56CA552	
	÷-	- 🐝 reg_file[8]	'hxxx⊦	****************************	
	÷	-% reg_file[9]	'hxxx⊾	*****************************	
	B	-⊛ reg_file[10]	'hxxx	***************************************	
	œ-	👒 rea file[11]	'hxxx	***************************************	1
	N	·····	M3 ~		
0					U Objects selected

1.6.3 Synthesis Results

The following results were found in the control.area report:

24
421
405
10
9600.000000
6912.000000
16512.000000

The following timing information was extracted from the timing report: Max delay: 2.25 delay units.

The check_design report was empty, indicating no errors and no warnings.

1.6.4 Post-Synthesis Functional Test Results

Using the same inputs, the same results were obtained from the netlist generated by the synthesis tool (see below).

EE577b Troy Processor Project by Zhyang Ong <u>zhiyang@ieee.org</u> and Andrew Mattheisen<u>@gmail.com</u>

Veal	ıdra.	.usc.edu	:10025 (mattheis)																			2
		- 44 - 14	un Fusien Francis	. Cimulatian	105 m d = 110	_				Wa	veforn 1 - S	inVision						_				
En	e <u>t</u>	≣ant ⊻n	ew Explore Forma	it Simulation	Windows													48%		2000 - 2000	-	Help
e		0 (*	% ₽≙ # × €	18 18 K	19 de 19													· ● ●	Send I	° (2) 🚟		# 📰 🚟
Sea	irch l	Names:	Signal 👻	-1 M- #	Search	Times: Value	•		<u>\$</u>													
×2	Tim	neA ▼ =	350 • ns	1 記憶・ 🌰	🖄 🗗) 01 (Sin	nulation Time	350ns + 0									Time	Range: 0:3	50ns	I 🔍 🤆	199
× (2	B	aseline = 0		Bareline -	5																
Q		Jursor-Ba	aseline = 350ns		buschine -	<u>.</u>									leas					loaa	TimeA :	=350(0)ns
ŝ]	181	clock	Gursor 👻		hnnn	hnnn	hnnn	hnnn	TOURS	hooo	hnnn	hnnn	hnnr	2008	nhnn		nhnr	nhnn		nnnn	
		n55	count	'dx	x																	
		- Sp.	r	'hxxx	*****	xx																
	E	a-%.	r1_addr	'h07	xx	0A								00	01	02	03	04	05	06	07	
			r1_en	1																		
	G	⊡ ~%}•	r2_addr	'h00	хx	0B								07	06	05	04	03	02	01	00	
		16	r2_en	1																		
	B	•	rd1_d	'hxxx	XXXXXX	x• 0000	000000	000000	000000	000000	000			XXXX	x• xxx:	xx, xxx	(XX) XX	xx, xx	xxx, xx:	(XX) XX)	.xx, xx>	(XXXX)
	E	1 - Sig.	rd2_d	hxxx.	*****	x• 0000	000000	1000000	000000	1000000	100	Yor	102	XXXX	x• xxx:	xx , xx>	(XX) XXX	(XX) XX	xxx , xx:	(XX) XX3	.xx, xx>	(XXXX)
		11-585- 11-585-	w_addr	'n14 /h000	XX	78789	72348	102	18646	CAACE	UD CRAC/	DC46D		14				1000030	120			
	Ľ		wr_u wr en	0	*****		<u> 12340</u>	140343.	<u>140040</u> .	Jonnou-	JOBRO4.	DC40D.	JC05DA		000000	000000		,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,				
			wrbvtn	0 'h00FF	xxxx	OOFF	007F	003F	(001F	OFFF	07FF	03FF	01FF	FFFF	7FFF	OOFF						
	Ľ																					
	L																					
	L																					
	L																					
	L																					
	L																					
	L																					
	L																					
	L																					
	L																					
	L																					
																						1
	F	4		KI >	.F.b																	
0																					1 ot	oject selected

Valuar	.usc.euu:10025 (ii	lactricis)								
-					Waveform 1 - SimVision					- D
Eile	Edit ⊻iew Exp	lore Format Simula	ation Windows							Help
6	n ~ ∦ Ba		劉隆 1 5					∰• ∰•	Send To: 🔞 🚟 🗄	
Search	Names: Signal 🕶	· · ·	🖕 🦓 🛛 Search Times:	Value •						
1 x2 Ti	meA 💌 = 350	• ns • 心积•	坐 🕸 🖿 💷 🕻	【 🚺 🔂 [Simulation Time: 350ns + 0				Time Range	19.53ns : 63.29ns	
×⊙	Baseline =	0								
Q.	Cursor-Baseline =	350ns		h.,			1			
ŝ		Cursor	✓ 20ns	30ns		40ns	50n	· · ·	· · · · · · · · · · · · · · · · · · ·	60ns
	n S. count	′ dx	x							
	n‰. r	'hxx	** *****							
	eriad	dr 'h07	• 0A							
		1								
	ia sa r2_ad	dr 'h00	▶ <mark>0</mark> B							
	i≋i r2_en	1								
	⊞ 🥵 rd1_d	'hxx	** ********	xxxx• 000000000000000000000000000000000	00000000000					
	ie⊸‰ rd2_d	'hxx	** *******	xxxx• 000000000000000000000000000000000	000000000000					
	e ≪⊱ w_ade	dr 'h14	• 00			01				02
	ei⊸na⊳ wr_d	'h00	0 • 787897EA	12FEC60CAE787897EAC22354		7234897346546	5546546464566465	4666		48545618
	⊛ wr_en	0				Vaar				
	⊞⊸‱ wrbyti	1 'h00	FF • OOFF			(007F				003F
	-1					120			1300	350.00
10	~~		120							1 object selected

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

V2 alu	dra.usc	.edu:10025 (mattheis)			
E				Waveform 1 - Sim∀ision	· 🗆
Eile	Edit	View Explore Forms	at Simulation	Windows	Help
Ì	n	୍ ୪ 🖻 🎕 🗙	°∎ °≣ [®] ≣	• · · · · · · · · · · · · · · · · · · ·	
Sea	ch Narr	nes: Signal 🕶	- i i i i i i i i i i i i i i i i i i i	Search Times Value -	
X2	TimeA	• = 350 • ns	・ 小市・ 一個	📸 📔 🏧 🔣 🗄 📅 Simulation Time: 350ns + 0 🗍 Time Range: 179 6ns. 223.36ns -	<u>IQQQQ</u>
× (Baseline = 0			
6		si-baseline = 550hs	Curror *	190ns 190ns 200ns 200ns	22005
彝		≅• clock	0		2000
		⊪ count	'dx	x	
	.	°a⊷r	′hxxx	*****	
	0 \$	🕾 r1_addr	'h07	• (00)01	02
	4	≊ r1_en	1		Vac
	••••	⊛ r2_addr	'h00	• 07)06	105
	E S	™ 12_en ⊪rd1 d	'hyyy)	000000000000 xxxxxxxxxxxxxxxxxxxxxxxxxx	
	÷	ard1_d ⊪rd2_d	'hxxx	0000000000000 xxxxxxxxx565042564C552 xxxxxxx5561C65AD1C6AD61C455	
	0 5	⊛ w_addr	'h14	>14	
	• \$	‰ wr_d	′h000⊧	+ (000000000000000000000000000000000000	
		≊∙wr_en	0		
	÷\$	≌ wrbytn	'h00FF	•[FFFF	OOFF
	51	D			350ns 📿
1					1 object selected

1.7 Mux Module

1.7.1 Inputs and Outputs

Inputs: [0:127] dataIn1, [0:127] dataIn2, select Outputs: [0:127] dataOut

1.7.2 Functional Test Results

As can be seen below, the mux functions as expected.

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

Maxeforn 1 - SinVision
Ele Edit View Explore Format Symulation Windows
🔐 n 🖓 k 10 & K 編 22 陽 15
Search Names Signal - 🚽 🍂 🛝 Search Times Value - 🎘 🏂
🔀 TimeA 🗹 = [30 🖉 [mo 🛨 j.t) 🕅 - 🏨 🖄 j 📔 🛄 🛄 🔄 📅 🗱 (Simulation Time: 30ms + 0] Time Range [0:30ms 🔤 ਉ, 👷 🕵
Section = 0
(1) 1000 1000 1000 1000 1000 1000 1000 1
na 'h000' xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
∲ ጭ out /h111 × xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
© Executing commands in console 0 objects sele

Due to the simplicity of the module (out<=sel?inb:ina;), synthesis and post synthesis testing were not required.

2 CPU Pipeline

Minimizing execution time is the goal of this project. Therefore a pipeline was implemented to increase the throughput. The benchmark instruction set is expected to be long, so the small additional latency should be overshadowed by the increase in throughput. Traditionally, processors are pipelined into the following 5 stages:

- Instruction Fetch (IF) An instruction is fetched from the instruction memory
- Instruction Fetch (IF) An instruction is fetched from the instruction memory
 Instruction Decede and Pagister Fatch (ID/PE) The instruction is deceded and
- Instruction Decode and Register Fetch (ID/RF) The instruction is decoded and the applicable registers are fetched from the register file
- Execution (EX) The ALU is used to perform the required operation on the data from the registers
- MEMory access (MEM) the data memory is written to or read from
- WriteBack the result of the data memory or ALU operation are written back to the register file.

For this project, we were able to combine the MEM and EX stages of the pipeline since each of the instructions in the instruction set EITHER access memory OR use the ALU (but not both, as non-immediate memory access would). Therefore, we implemented the following 4-stage pipeline:

- Stage 1 IF
- Stage 2 ID/RF
- Stage 3 EX/MEM
- Stage 4 WB

by Zhyang Ong <u>zhiyang@ieee.org</u> and Andrew Mattheisen<u>@gmail.com</u>

Flip Flops (FF) were used between stages 1 and 2, between stages 2 and 3, and between stages 3 and 4 to create the pipeline. The FFs between stages 1 and 2 are contained in a module called Pipe1. The FFs between stages 2 and 3 are contained in a module called Pipe2. The FFs between stages 3 and 4 are contained in a module called Pipe3. The entire pipeline is shown in the following figures:





EE577b Troy Processor Project by Zhyang Ong <u>zhiyang@ieee.org</u> and Andrew Mattheisen <u>amattheisen@gmail.com</u>











by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com



It should be noted that the register file in stage 2 is the same register file in stage 4. In stage 2, only the read capability of the register file is exercised. In stage 4, only the write capability is exercised. Therefore an instruction in stage 2 can use the read side of the register file while an instruction in stage 4 of the pipeline can simultaneously access the write side of the register file.

As with any pipeline, it can only be operated at the speed of the slowest stage. In the pipeline design shown above, the MEM/EX stage (stage 3) is the slowest stage. Since the module representing the data memory is provided, there is no chance to optimize it. However, the ALU module is not provided, and therefore can be carefully designed to minimize latency. This design will be described in the following sections (3-6).

When a processor is pipelined, the possibility for data hazards is created. Section 7 addresses these hazards and shows how data forwarding was used to eliminate all data hazards.

2.1 CPU Module

2.1.1 Inputs and Outputs

Inputs: clk, reset, [0:31] instruction, [0:31] pc, [0:127] dataIn, Outputs: [0:127] dataOut, memEn, memWrEn, memAddr

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

2.1.2 Functional Test Results

The following instruction set was used to test the cpu module: 07E00000 // wld wr31, 0; 04A00001 // wld wr5, 1 ; 121FF881 // wsubaw wr16, wr31, wr31; 1225FA4B // wxoruh wr17, wr5, wr31 ; 11F00008 // wnotab wr15, wr16 ; 1245FC89 // wandew wr18, wr5, wr31; 23100400 // wmveb wr24, wr16 ; 23050500 // wmvob wr24, wr5 ; 1265FD4A // woroh wr19, wr5, wr31 ; 129FFB40 // wadddh wr20, wr31, wr31 ; 102F3815 // wsrliab wr1, wr15, 7; 12BFF850 // wsllah wr21, wr31, wr31 ; 12DFC016 // wsraab wr22, wr31, wr24 ; 12FFC094 // wsrlaw wr23, wr31, wr24 ; 133F1011 // wslliab wr25, wr31, 2; 135F0857 // wsraiah wr26, wr31, 1; 0AA00002 // wst wr21, 0x02 ; 13FFF80A // worab wr31, wr31, wr31; 13FFF80A // worab wr31, wr31, wr31; 13FFF80A // worab wr31, wr31, wr31 ; 13FFF80A // worab wr31, wr31, wr31; 04E00003 // wld wr7, 3 05000004 // wld wr8, 4 13C74000 // addab wr30, wr7, wr8 ; 13A74000 // addab wr29, wr7, wr8 ; 13874000 // addab wr28, wr7, wr8 ; 13674000 // addab wr27, wr7, wr8 ; 13C74044 // mules wr30, wr7, wr8 ; 13A74045 // muleu wr29, wr7, wr8 ; 13874046 // mulos wr28, wr7, wr8 ; 13674047 // mulou wr27, wr7, wr8 ; 00000000 // NOP ;

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

F	Receform 1 - Silvision /						
Eile	Eile Edit View Exglore Formet Simulation Windows						
8	ŝ	~ L 🖻 🎕 🗙 (N. 55	🐝 - 👞 Send To: 🐯 🧱	e z 🛛 📰 📓	
Sean	ch Nan	nes: Signal 🕶	- 10 10	Search Times, Value -			
	TimeA 📲 💷 🖉 👘 🖓 👘 🕹 🖿 🖬 🕅 🕅 📅 🗑 Simulation Time 390s+1 [
× (i)							
-	Curs	or-Baseline = 390ns		Baseline = 0		TimeA = 390/01ns	
ŝ			Cursor -	0 100ns 200ns	300ns		
	œ 4	🚯 aluOut	′h000	xxx+ 0000000000000000000000000000000000	000000000000000	000• <mark>(000000•</mark> -3	
	÷	‰ aluOut3	'h000⊧	• <u>[000000000000000000000000000000000000</u>	<u>,000 1000000000 10000</u>	000000• <u>1000•</u>	
		aluinmuxop	0				
		aluinmuxop2	U /1 1 0			2 /1 0	
		a aluop	' n10			7 1 1 0 5 Y 0 7 Y 1 9	
		aluopz w olk	1				
		‰ cirk ≋ dataln	⊥ /Ъ000⊳				
i l		⇒ datain3	'h000	/ 000000000000000000000000000000000000	• Yo• Yo• Yooooooooooooo	0000000000	
	in -ii	ataOut	′h000▶	xxx* 0000000000000000000000000000000000	¥0000000000000000000000000000000000000	000× 1000000×	
	Γ.,	m hz1 a or d	0			^	
		⊷ hz1 b	0				
	•	⊸ ⊪ hz1data	′h000⊧	▶ 000000000000000000000000000000000000	• O• O• O000000000 O	000000000	
		⊯ hz3	z				
		≋ hz3_a_or_d	0				
	4	™ hz3_b	0				
	•	🐃 immediate	′hxX0∙	▶ 0000000000000000 [F800+]0+]F+ 0000+ [F800+]3+]F+ [C000+]1+]0+]0+]F800000000+ [0000+]40	000000000000000000000000000000000000000	•(0• xX0000•	
	0	immediate2	′hxX0⊧	> 00000000000000000000 F800> 0+ F+ 0000> F800+ 3+ F+ C000> 1+ 0+ 0+ F800000000+ 0000+	400000000000000000000000000000000000000	000•0• xX0•	
	•	instruction	′hxxx	• 07E00000 0• 1• 1• 1• 1• 1• 2• 2• 1• 1• 1• 1• 1• 1• 1• 1• 1• 1• 1• 1• 1•	1 1 1 1 1 1 1 1 1 1 0	******	
	•	instruction1	′hxxx	• 000000000 0• 0• 10• 11• 11• 11• 12• 12• 12• 11• 11• 11• 11	1. 1. 1. 1. 1. 1. 1.	0 xxxxxx	
	•	Instruction2	hxxx•				
		Se instruction3	' hxxx				
		se memAdar	'nxxx'				
	u .	> memEn	0			10. 100000	
		memEn pre	0			1	
J	V						
1	Executi	ing commands in console				1 object selected	

The outputs were carefully scrutinized to ensure that re result of each operation was stored in the correct location.

ini vi

Keefan 1 - Siwiisan k						
Ele Edit View Explore Format Simulation Windows						
● ● ○ ★ Be @ × 編編館 (19)	Send 1	° 🖏 🚟 🖻 🛛	2 🖗 👪 📓			
Search Names Signal - Search Times Value - 🧐 🎘						
[] [] [] [] [] [] [] [] [] [] [] [] [] [Time Range: 0:3	90ns 🚽 🤄				
X0 Baseline = 0	,	M.	526 Jacob 10258 Acces			
Cursor-Baseline = 22.21ns Baseline = 0						
To Current ~ 0 100ns 200ns	300ns					
	xx x xx xx xx		> x x } x x x }			
	· · · · · · · · · · · · · · · · · · ·		• × • • • • •			
	xx*xx*xx*xx*	xx+xx+xx+xx	• xx• xxx•			
• red field () hxxx	XX•XX•XX•XX•	xx•xx•xx•xx	• xx• xxx•			
	XX XX XX XX	xx•xx•xx•xx	• xx• xxx•			
=-% red lie[14] , pxxx, xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xx•xx•xx•xx•	xx•xx•xx•xx	•xx•xxx•			
e −% req file[15] /hxxx ⁺ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	F F F F	F* F* F* F*	F* FFF*			
	0+ 0+ 0+ 0+	0+ (0+ (0+ (0+	0000			
e → reg_file[17] 'hxxx' xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	55•55•55•55•	55• 55• 55• 55	▶ 55+ 554+			
e ≪ reg_file[18] ′hxxx+ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	00+00+00+00+	00+00+00+00	▶ 00 ⊦ 001►			
🖶 🐟 reg_file[19] 'hxxx' xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	XX, XX, XX, XX,	xx•xx•xx•xx	×x× xxx			
e reg_file[20] 'hxxx+ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	XX, XX, XX, XX,	xx•xx•xx•xx	*xx*xx*			
• • reg_file[21] 'hxxx+ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx+xx+	0 0 0 0	0+ 0+ 0+ 0+	0 002			
• • reg_file[22] 'hxxx+ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	0 0 0 0	0+ 0+ 0+ 0+	0000			
•	(0+ (0+ (0+)0+)	0+ (0+ (0+ (0+	000			
φ ≪ reg_file[24] ′hxxx ⁺ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	0+ 0+ 0+ 0+ 0+	0+ 0+ 0+ 0+	0 005			
e-% reg_file[25] /hxxx* xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	0+ 0+ 0+ 0+ 0+	0• 0• 0• 0•	0• 004•			
e-% reg_file[26] /hxxx* xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	0+ 0+ 0+ 0+ 0+	0+ (0+ (0+ (0+	<u> (</u> 0⊁ <u>)</u> 000⊁			
e-æ reg_file[22] / μ×xx, xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xx, xx, xx, xx,	xx•0• 0• 0•	000			
e-æ red-lie[53] , µxxx, xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xx, xx, xx, xx,	0) (0) (0) (0)	(0) (000)			
∲ ⊛. reg_file[29] ′hxxx ⁺ ××××××××××××××××××××××××××××××××××××	xx•xx•xx•0•	0 0 0	0000			
θ−‰ reg_file[30] ′hxxx ⁺ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xx•xx•0• 0•	0 (0 (0	<u>(0) (000)</u>			
<u> </u>	· <u>(0) (0) (0) (0)</u>	0 (0 (0)0	0• 001•			
0			1 object selected			

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

as correct after the run:
0:0011332266775544ccddffeeaabb9988
1 : 55555555555555555555555555555555555
2:0022cc883b805440a0008000d8008800
3:0000000000000000000000000000000000000
4:0000000000000000000000000000000000000
5 : xxxxxxxxxxxxxxxxxxxxxxxxxxxx
63 : xxxxxxxxxxxxxxxxxxxxxxxxxxxx

At this point hazard detection had already been implemented as can be seen by the signals hz1_a_or_d, hz1_b, hz3_a_or_d, and hz3_b. Detailed disussion about the hazards in the above functional test will be discussed in section 7.

2.1.3 Synthesis Results

See section 9.

2.1.4 Post-Synthesis Functional Test Results

The post synthesis functional test results show that the cpu has the same functionality after synthesis.

As part of this project, the following 6 random instruction sets were generated to determine the speed of the pipelined processor:

<INCLUDE HERE>

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

3 Instruction set implementation

The instruction set introduced in the introduction is described in detail in this section. The rest of this section has been copied with permission from the USC document for EE577B Fall 2007 isa_ov.pdf.

Chapter 1 - Troy WideWord Processor Instruction Set Overview

Notation

This chapter gives an instruction set overview and the following chapter gives a detailed instruction description. All instructions are 32 bits. Big-Endian byte and bit labeling is used, meaning that bit/byte 0 is the most significant. Other conventions are listed in the table below.

	IABLE I	. Instruction	i Glossary
Symbol	Meaning	Symbol	Meaning
$A \leftarrow B$	Assignment	x ^ y	x bitwise exclusive ORed with y
${x,y}$	Bit string concatenation	~x	bitwise inversion of x
$\{y\{x\}\}$	x replicated y times	MEM[EA]	Memory contents at effective address EA
x[y:z]	Selection of bits y through z from x	0xvalue	Hexadecimal value
x & y	x bitwise ANDed with y	0bvalue	Binary value
x y	x bitwise ORed with y	(rX)	Contents of general-purpose register X

The following table gives the rules of precedence and associativity for the pseudocode operators. All operators on the same line have equal precedence, and all operators on a given line have higher precedence than those on the lines below them.

TA	TABLE 2. Precedence of Pseudocode Operators					
	Operator	Associativity				
	MEM[n]	left to right				
	x[y:z]	left to right				
	${y{x}}$	left to right				
	~	right to left				
	X,÷	left to right				
	+, -	left to right				
	{x,y}	left to right				
	=, !=, <, <=, >, >=	left to right				
	^, &	left to right				
		left to right				
	<i>←</i>	none				

Page 2 of 26

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

Instruction Formats

As shown in Figure 1, most Troy WideWord instructions use a three-operand format to specify two 128-bit source registers (wrA and wrB) and a 128-bit destination register (wrD). Load and store use a different instruction format shown in Figure 2. Note that these classifications are generalizations, and some instructions will vary somewhat from the format for which it is classified. For instance, the *shift immediate* instructions are classified as W-type instructions, although they specify a 5-bit immediate shift amount in the place of wrB. The end result is that not all W-type instructions can be decoded in exactly the same way; the wrB field may be a register specifier or it may be a 5-bit immediate.

 6 bits	5 bits	5 bits	5 bits	3 bits	2 bits	6 bits
opcode	wrD	wrA	wrB	PPP	WW	function

Figure 1 Format W for WideWord Arithmetic/Logical Operations

6 bits	5 bits	21 bits
opcode	rD	immediate address

Figure 2 Format M for Wide-Word Load/Store Operations

The control fields are defined as follows:

WW (width)

The 2-bit WW field sets the width of the WideWord operands to eight, sixteen, or thirty-two bits, which primarily affects the shift operations, configuration of the carry chain for additions and subtractions, and the multiplication operations. The encoding of these bits is listed in the following table:

WW Value	Operand Width	Assembler Mnemonic
00	8 bits	b
01	16 bits	h
10	32 bits	w
11	Reserved	NA

Page 3 of 26

PPP (participation)

The 3-bit *PPP*, or participation, field specifies what kind of selective execution, if any, governs the operation. Recall that under selective execution only certain subfields commit their results during the writeback stage. The subfields participating are specified by the decoding of *PPP*. The encoding of the *PPP* bits is listed in the following table:

PPP Value	Participation Definition	Assembler Mnemonic
000	All subfields participate	a
001	reserved	none
010	Upper 64-bits participate	u
011	Lower 64-bits participate	d
100	Subfields with even index participate	e
101	Subfields with odd index participate	0
110	Only most significant subfield (subfield 0) participates	m
111	Only least significant subfield participates	1

For those participation modes that refer to the even/odd indices or most/least significant subfields, the exact bits that participate depend also on the operand width specified by the *WW* field. The following table shows the possible subfield indices for the different values of *WW* (recall that Big-Endan labeling is used so that subfield to is always the most significant regardless of operand size):

Subfield Indices within a WideWord for						ord for I	Differing	Operan	d Width	s			-			
WW Value	most sig.															least sig.
00	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
01	01 0 1 2		3 4		4	5		6		7						
10			0		1			2			3					

This table may be useful for visualizing which subfield(s) participate based on the selective execution mode and operand width value. For instance, an "upper" participate mode using 32-bit operands means that only words 0 and 1 participate.

Page 4 of 26

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

Alphabetical list of instructions

TABLE 3. Preliminary Encoding of DIVA Instruction Set

				Enco	Encoding			
Instruction	Format	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
WADD	W	000100	wrD	wrA	wrB	PPPWW	000000	
WAND	W	000100	wrD	wrA	wrB	PPPWW	001001	
WLD	M	000001	wrD		immedi	ate_address	•	
WMULES	W	000100	wrD	wrA	wrB	PPPWW	000100	
WMULEU	W	000100	wrD	wrA	wrB	PPPWW	000101	
WMULOS	W	000100	wrD	wrA	wrB	PPPWW	000110	
WMULOU	W	000100	wrD	wrA	wrB	PPPWW	000111	
WMV	W	001000	wrD	wrA	00000	PPPWW	000000	
WNOT	W	000100	wrD	wrA	00000	PPPWW	001000	
WOR	W	000100	wrD	wrA	wrB	PPPWW	001010	
WPRM	W	000100	wrD	wrA	wrB	PPP00	001100	
WSLL	W	000100	wrD	wrA	wrB	PPPWW	010000	
WSLLI	W	000100	wrD	wrA	shift_amount	PPPWW	010001	
WSRA	W	000100	wrD	wrA	wrB	PPPWW	010110	
WSRAI	W	000100	wrD	wrA	shift_amount	PPPWW	010111	
WSRL	W	000100	wrD	wrA	wrB	PPPWW	010100	
WSRLI	W	000100	wrD	wrA	shift_amount	PPPWW	010101	
WST	М	000010	wrD		immedi	ate_address		
WSUB	W	000100	wrD	wrA	wrB	PPPWW	000001	
WXOR	W	000100	wrD	wrA	wrB	PPPWW	001011	

Page 5 of 26

waddx - WideWord Add





Variable values in the following equations are as follows:

WW Value	size
00	8
01	16
10	32

for i = 0 to (128 - size) by size

if PPP bits enable writeback for this subfield

 $wrD[i:(i+(size-1))] \leftarrow (wrA)[i:(i+(size-1))] + (wrB)[i:(i+(size-1))]$

The WW field determines if the 128-bit contents of wrA and wrB are treated as 32 bytes, 16 half-words, or 8 words. The aggregate sums of the aligned data fields of wrA and wrB are placed into wrD, subject to participation.

waddx - WideWord Add

Page 7 of 26

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com



The 128-bit contents of wrA are ANDed with the 128-bit contents of wrB, and the result is placed into wrD, subject to participation. The WW field simply effects how participation applies for this operation.

wandx - WideWord AND

Page 8 of 26

wld - Load WideWord Register

wld wrD, immediate_address

000001		wrD	immediate_address
0	5	6 10	11 31

 $\textit{EA} \gets \{\texttt{immediate_address}, 4\{0\}\}$

 $wrD \leftarrow MEM[EA]$

The immediate address is assumed to be in units of 128-bit wide words. Thus, to obtain an effective address, EA, in units of bytes, 4 zeros are appended to the immediate_address. The 128-bit value at the memory location specified by EA is then loaded into wrD.

wld - Load WideWord Register

Page 9 of 26

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

wmules - WideWord Multiply Even Signed

wmulespw wrD, wrA, wrB



Each even-numbered signed-integer byte or half-word of wrA is multiplied by the corresponding signed-integer byte or half-word of wrB, where the WW field determines if the 128-bit contents of wrA and wrB are treated as bytes or half-words. The resulting signed halfword or word products are placed, in the same order, into wrD, subject to participation.

wmules - WideWord Multiply Even Signed

Page 10 of 26

wmuleu - WideWord Multiply Even Unsigned

wmuleupw wrD, wrA, wrB



WW Value	input size	output size
01	8	16
10	16	32

In the equation below, the variable size refers to the input size given in the table above.

for i = 0 to (128 - $2 \times \text{size}$) by $2 \times \text{size}$

if PPP bits enable writeback for this subfield $wrD[t:(i + (2 \times size - 1))] \leftarrow (wrA)[t:(i + (size - 1))] \times (wrB)[t:(i + (size - 1))]$

Each even-numbered unsigned-integer byte or half-word of wrA is multiplied by the corresponding unsigned-integer byte or half-word of wrB, where the WW field determines if the 128-bit contents of wrA and wrB are treated as bytes or half-words. The resulting unsigned half-

wrB, where the WW field determines if the 128-bit contents of wrA and wrB are treated as bytes or half-words. The resu word or word products are placed, in the same order, into wrD, subject to participation.

wmuleu - WideWord Multiply Even Unsigned

Page 11 of 26

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com



wmulospw wrD, wrA, wrB



Each odd-numbered signed-integer byte or half-word of wrA is multiplied by the corresponding signed-integer byte or half-word of wrB, where the WW field determines if the 128-bit contents of wrA and wrB are treated as bytes or half-words. The resulting signed halfword or word products are placed, in the same order, into wrD, subject to participation.

wmulos - WideWord Multiply Odd Signed

Page 12 of 26

wmulou - WideWord Multiply Odd Unsigned

wmuloupw wrD, wrA, wrB



WW Value	input size	output size
01	8	16
10	16	32

In the equation below, the variable size refers to the input size given in the table above.

for i = 0 to (128 - $2 \times size$) by $2 \times size$

if PPP bits enable writeback for this subfield

 $wrD[i:(i + (2 \times size - 1))] \leftarrow (wrA)[(i + size):(i + (2 \times size - 1))] \times (wrB)[(i + size):(i + (2 \times size - 1))]$

Each odd-numbered unsigned-integer byte or half-word of wrA is multiplied by the corresponding unsigned-integer byte or half-word of wrB, where the WW field determines if the 128-bit contents of wrA and wrB are treated as bytes or half-words. The resulting unsigned half-word or word products are placed, in the same order, into wrD, subject to participation.

wmulou - WideWord Multiply Odd Unsigned

Page 13 of 26

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com



wmvx - Move from WideWord to WideWord Page 14 of 26 wnotx - WideWord NOT wnotpw wrD, wrA 000100 wrD 00000 PPPWW 001000 wrA 0 56 10 11 15 16 20 21 25 26 31 Variable values in the following equations are as follows: WW Value size 01 16 10 for i = 0 to (128 - size) by size if PPP bits enable writeback for this subfield $wrD[i:(i + (size - 1))] \leftarrow \sim (wrA)[i:(i + (size - 1))]$ The 128-bit contents of wrA are bitwise inverted, and the result is placed into wrD, subject to participation. The WW field simply effects how participation applies for this operation.

wnotx - WideWord NOT

Page 15 of 26

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com



worx - WideWord OR Page 16 of 26 wprmx - WideWord Permute wprmp wrD, wrA, wrB 000100 wrD wrB 001100 wrA PPP00 0 56 10 11 15 16 20 21 25 26 31 for i = 0 to 120 by 8 $\,$ $s \leftarrow (wrB)[(i+4){:}(i+7)]$ if PPP bits enable writeback for this byte subfield $wrD[i{:}(i+7)] \leftarrow (wrA)[s \times \${:}((s \times \$)+7)]$ The contents of wrA are the source vector for this permutation operation. Bits 4 to 7 of each byte element of the contents of wrB are used to select a byte element from the source vector for each byte element of the result. The result is placed into wrD, subject to participation.

wprmx - WideWord Permute

Page 17 of 26

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com



Variable values in the following equations are as follows:

WW Value	size	bits
00	8	3
01	16	4
10	32	5

 $s \leftarrow \text{shift}_amount[(5 - bits):4]$

for i = 0 to (128 - size) by size

if PPP bits enable writeback for this subfield

 $wrD[i:(i+(size-1))] \leftarrow \{(wrA)[(i+s):(i+size-1)], s\{0\}\}$

The WW field determines if the 128-bit contents of wrA are treated as 32 bytes, 16 half-words, or 8 words. The contents of each data field of wrA are shifted left by the number of bits specified by the appropriate bits of the shift_amount, inserting zeros into the low order bits of each data field of the result. The result is placed into wrD, subject to participation.

wsllix - WideWord Shift Left Logical Immediate

Page 19 of 26

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com





wsraix - WideWord Shift Right Arithmetic Immediate

Page 21 of 26

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com





Variable values in the following equations are as follows:

WW Value	size	bits
00	8	3
01	16	4
10	32	5

 $s \leftarrow \text{shift}_amount[(5 - bits):4]$

for i = 0 to (128 - size) by size

if PPP bits enable writeback for this subfield

 $vrD[i:(i+(size-1))] \leftarrow \{s\{0\}, (wrA)[i:(i+size-s-1)]\}$

The WW field determines if the 128-bit contents of wrA are treated as 32 bytes, 16 half-words, or 8 words. The contents of each data field of wrA are shifted right by the number of bits specified by the appropriate bits of the shift_amount, inserting zeros into the high-order bits of each data field of the result. The result is placed into wrD, subject to participation.

wsrlix - WideWord Shift Right Logical Immediate

Page 23 of 26

EE577b Troy Processor Project by Zhyang Ong <u>zhiyang@ieee.org</u> and Andrew Mattheisen <u>amattheisen@gmail.com</u>

wst	wrD	, immed	liate_address	;		
_						
	000010	wrD	10.11	immedia	ate_address	
Ed (imme		20	10 11			
$MEM[EA] \leftarrow$	(wrD)	1				
The immedia are appended	te_address is assu to the immediate	med to be in un address. The 1	nits of 128-bit wide wor 28-bit contents of wrD	ds. Thus, to obtai are stored at the n	n an effective add nemory location s	lress, EA, in unit pecified by EA.
	-	-			-	
		wst -	Store WideWord Re	gister		
		wst -	Store WideWord Re	gister		
		wst -	Store WideWord Re	gister		
wsubx	- WideWo	wst-	Store WideWord Re	gister		
wsubx	- WideWe	wst-	Store WideWord Re	gister.		
wsubx wsubp	- WideWo v wrD	wst- ord Subt , wrA, w	-Store WideWord Re tract vrB	gister		
wsub <i>x</i> wsub <i>p</i>	- WideWo v wrD	wst- ord Subt , wrA, w	Store WideWord Re tract vrB	gister		
wsubx wsubp	- WideWo v wrD	wst- ord Subi	Store WideWord Re tract vrB	gister. 	Papirity	000001
wsubx wsubpr	- WideWo v wrD	wst- ord Subt , wrA, w	Store WideWord Re tract vrB 10 11 15			000001
wsubx wsubpr	- WideWo v wrD 000100 0	wst- ord Sub , wrA, w wrD 5 6	Store WideWord Re tract vrB 0 11 15	•gister 	PPPWW 21 25	000001
wsubx wsubpr [Variable va	- WideWe	wst - prd Subi , wrA, w wrD 5 6 owing equati	Store WideWord Re tract vrB wrA 10 11 15 ions are as follows:		PPPWW 21 25	000001
wsubx wsubpr [Variable va www.value 0	- WideWe	wst - ord Subt , wrA, w wrD 5 6 owing equati	Store WideWord Re tract vrB wrA 10 11 15 ions are as follows:	gister 	PPPWW 21 25	000001
wsubx wsubpn Variable v. WWVatee 00 01	- WideWe	wst - ord Subt , wrA, w wrD 5 6 owing equati	Store WideWord Re tract vrB wrA 10 11 15 ions are as follows:	gister 	PPPWW 21 25	000001
Wsubx wsubpt Variable va WWValue 00 01 10	- WideWo v wrD 000100 0 alues in the foll size 8 16 32	wst- ord Subt , wrA, w wrD 5 6 owing equati	Store WideWord Re tract vrB wrA 10 11 15 ions are as follows:		PPPWW 21 25	000001
wsubx wsubpr Variable va Variable va WW Value 00 01 10 for i = 0 to (1	- WideWa v wrD 000100 0 alues in the foll size 8 16 32 28 - size) by size	wst- ord Subt , wrA, w wrD 5 6 owing equati	Store WideWord Re tract vrB wrA 10 11 15 ions are as follows:	<u>wrB</u> 16 20	PPPWW 21 25	000001
WSUDX WSUDPT Variable va Variable va Variable va Variable va Variable va va variable va variable va va variable va variable va variable va va variable va va variable va va va va va va va va va va va va va v	- WideWe v wrD 000100 0 alues in the foll size 8 16 32 28 - size) by size ts enable writebac	wst - ord Subt , wrA, w wrD 5 6 owing equati k for this subfit	Store WideWord Re tract vrB wrA 10 11 15 ions are as follows: eld	<u>wrB</u> 16 20	PPPWW 21 25	000001
WSUDX WSUDPT Variable va Variable va varia	- WideWo v wrD 000100 0 alues in the foll size 8 16 32 28 - size) by size ts enable writebac [f:(i+(size-1))]	wst - Drd Sub , wrA, w , wrA, w 5 - 6 owing equations with the subfit of the subfit o	eld		PPPWW 21 25	000001
WSUDX WSUDDT Variable v3 Variable v3 Variable v3 for i = 0 to (1 if PPP bi wrE	- WideWo v wrD 000100 0 alues in the foll size 8 16 32 28 - size) by size ts enable writebac [1:(1+(stre-1))] d datasynings if th	wst - Drd Sub , wrA, w , wrA, w 5 6 owing equations 5 6 wing equations 6 wing equations 6 wrD 5 6 wrD 5 7 6 100 6 100 7 1000 7 1000 7 1000 7 1000 7 1000 7 10	eld	wrB 16 20 i + (zize - 1))] + 1	PPPWW 21 25	000001 5 26
Wsubx wsubpt	- WideWo v wrD 000100 0 alues in the foll size 8 16 32 28 - size) by size ts enable writebac [i:(i+(size-1))] d determines if the ligned data fields	wst - ord Sub , wrA, w , wrA, w 5 6 owing equations f (wrA)[i:(i+ e (128-bit context)]	eld (zize - 1))] + ~(wrB)[i:(wrB 16 20 i + (size - 1))] + 1 treated as 32 byt ubject to particip	PPPWW 21 25 es, 16 half-words ation.	000001 5 26
wsubx wsubpt [] Variable value 00 01 10 for i = 0 to (1 if PPP bing wrE The WW field ences of the state	- WideWo v wrD 000100 0 alues in the foll size 8 16 32 28 - size) by size ts enable writebac [i:(i+(zize - 1))] d determines if the ligned data fields	wst - ord Sub , wrA, w , wrA, w 5 6 owing equations 6 (wrA)[i:(i+1)] 6 (wrA)[i:(i+1)]	eld (zize - 1))] + ~(wrB)[i:(muther and wrB are	wrB 16 20 i + (size - 1))] + 1 treated as 32 byt ubject to particip	PPPWW 21 25 es, 16 half-words ation.	000001 5 26
Wsubx wsubpt	- WideWo v wrD 000100 0 alues in the foll size 8 16 32 28 - size) by size is enable writebac [i:(i + (zize - 1))] d determines if the ligned data fields	wst - ord Subi , wrA, w \downarrow wrD 5 6 owing equati \downarrow k for this subfit \leftarrow (wrA)[<i>i</i> :(<i>i</i> + e 128-bit context)	eld (zize - 1))] + ~(wrB)[i:(ents of wrA and wrB are B are placed into wrD, i	wrB 16 20 i + (zize - 1))] + 1 treated as 32 byt ubject to particip	PPPWW 21 25 es, 16 half-words ation.	000001 5 26

wsubx - WideWord Subtract

Page 25 of 26

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

	000100	W	rrD	wrA	w	тB	PPPWW	001011
0		5 6	10 11	15	16	20	21 2:	5 26
Variable val	ues in the follo	owing eq	uations are	e as follows:				
WW Value	size	1 .	-					
00	8							
01	16	1						
10	32							
for $i = 0$ to (12)	R . size) by size							
if PPP bits	enable writebacl	k for this s	ubfield					
un Dia	(i+(size 1))]		(i + (size 1))] ∧ (umR)[d)	i + (eie e	104		
wrD[1	.(1 - (3128 - 1))] (= (wr.a)[1	.(1 * (5128 = 1	/J (W/D)[E(1 . (2126 -	- • <i>1</i> /1		
TI 10013			0.0.1.34	1 100 1 3		D 1.		1. D. 1.
The 128-bit con	atents of wrA are	exclusive w particir	-Ored with pation applies	the 128-bit con tor this operat	ents of w ion.	TB, and t	ne resuit is place	a into wrD, subje
The WW field	STUDIY CHECK IN							

wxorx - WideWord Exclusive-OR

Page 26 of 26

4 Adder/Subtractor design

The Synopsys DesignWare intellectual property (IP) library consists of high performance IP blocks for system development and integration to reduce development time, and timeto-market (Cohen et al, 1996). Given the short design time of this processor project and the high performance requirement, we decided to utilize the virtual microarchitecture library from DesignWare IP to implement the adder design (Synopsys, 2001). This would save us time from implementing advanced adder designs, using Verilog, in structural RTL. It would also save us numerous man-hours from iterating the design process to verify that the design functions correctly, and optimize the design for high performance within our aggressive schedule.

(Koren, 1993)

Some adder designs http://www.ece.iit.edu/~jstine/book/

http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=835116

Simulators for various adder designs, along with other arithmetic circuits, can be obtained at <u>http://www.ecs.umass.edu/ece/koren/arith/simulator/</u>.

4.1 Functional Test Results

See section 8.

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

4.2 Synthesis Results

The following results were found in the control.area report:

Number of ports:	391
Number of nets:	4270
Number of cells:	3623
Number of references:	43
Combinational area:	17115.000000
Noncombinational area:	0.000000
Total cell area:	17115.000000

The following timing information was extracted from the timing report: Max delay: 7.11 delay units.

The check_design report indicated that six of the synthesized cells did not drive any nets. We kept this in mind while testing the functionality of the synthesized code and found that the warnings could be ignored.

4.3 Post-Synthesis Functional Test Results

See section 10.

5 Shifter Design

5.1 Functional Test Results

See section 8.

5.2 Synthesis Results

TBD – redesigned and removed latches

5.3 Post-Synthesis Functional Test Results

See section 10.

6 Multiplier Design

6.1 Functional Test Results

See section 8.

6.2 Synthesis Results

The following results were found in the control.area report:

Number of ports:	391
Number of nets:	32234
Number of cells:	27146
Number of references:	222
Combinational area:	1410694.000000
Noncombinational area:	0.000000

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

Total cell area: 1410694.000000

The following timing information was extracted from the timing report:
Max delay:Max delay:37.5 delay units.This is by far the worst delay of any of the ALU functions.Therefore this is on the
critical path in the pipelined CPU.See Section 11 for more about this.

The check_design report indicated that 777 of the synthesized cells did not drive any nets. We kept this in mind while testing the functionality of the synthesized code and found that the warnings could be ignored.

6.3 Post-Synthesis Functional Test Results

See section 10.

EE577b Troy Processor Project by Zhyang Ong <u>zhiyang@ieee.org</u> and Andrew Mattheisen <u>amattheisen@gmail.com</u>

7 Hazard management and data forwarding

Hazard detection occurs due to the pipelining process when multipleinstructions in the pipeline are accessing the same storage location (register file memory location). The hazard detection module inspects theinstruction signals in all stages of the pipeline at the same time as shown below:



There are two types of data hazards in the 4 stage pipeline described in section 2. These hazards are described below.



by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

Data forwarding is used to overcome HZ1. The data paths used to forward data and avert HZ1 are highlighted in the figure below:





Data forwarding is used to overcome HZ3. The data paths used to forward data and avert HZ3 are highlighted in the figure below:



EE577b Troy Processor Project by Zhyang Ong <u>zhiyang@ieee.org</u> and Andrew Mattheisen <u>amattheisen@gmail.com</u>

7.1 Functional Test Results

From section 2, the following hazards exist:

HEY INSTR		TVPE	REGD	REGA	REGB	HAZARD DETECTION SIGNALS
07F00000		wld	wr31	REOA	0	UGINALO
04A00001	000001 00101 00000 00000 00000000000	wld	wr5.		1	
121FF881	000100 10000 11111 11111 0001000001	wsubaw	wr16.	wr31.	wr31	HZ1 a or d. HZ1 b
1225FA4B	000100 10001 00101 11111 01001001011	wxoruh	wr17.	wr5.	wr31	HZ1 a or d
11F00008	000100 01111 10000 00000 00000001000	wnotab	wr15.	wr16		HZ1 a or d
1245FC89	000100 10010 00101 11111 10010001001	wandew	wr18.	wr5.	wr31	
23100400	001000_11000_10000_00000_1000000000	wmveb	wr24,	wr16		
23050500	001000_11000_00101_00000_10100000000	wmvob	wr24,	wr5		
1265FD4A	000100_10011_00101_11111_10101001010	woroh	wr19,	wr5,	wr31	
129FFB40	000100_10100_11111_11111_01101000000	wadddh	wr20,	wr31,	wr31	
102F3815	000100_00001_01111_00111_00000010101	wsrliab	wr1,	wr15,	7	
12BFF850	000100_10101_11111_11111_00001010000	wsllah	wr21,	wr31,	wr31	
12DFC016	000100_10110_11111_11000_00000010110	wsraab	wr22,	wr31,	wr24	
12FFC094	000100_10111_11111_11000_00010010100	wsrlaw	wr23,	wr31,	wr24	
133F1011	000100_11001_11111_00010_00000010001	wslliab	wr25,	wr31,	2	
135F0857	000100_11010_11111_00001_00001010111	wsraiah	wr26,	wr31,	1	
0AA00002	000010_10101_00000_00000_00000000010	wst	wr21,		2	
13FFF80A	000100_11111_11111_11111_00000001010	worab	wr31,	wr31,	wr31	
13FFF80A	000100_111111_11111_11111_00000001010	worab	wr31,	wr31,	wr31	
						HZ1_a_or_d, HZ1_b,
13FFF80A	000100_11111_11111_11111_00000001010	worab	wr31,	wr31,	wr31	HZ3_a_or_d, HZ3_b
13555804	000100 11111 11111 11111 0000001010	worah	wr31	wr31	wr21	HZI_a_or_d, HZI_b, HZ3 a or d HZ3 b
04E00003		wid	wr31,	wron,	2	$HZ3_a_or_d, HZ3_b$
05000004		wld	wr8		3 4	1120_a_01_0, 1120_0
13074000		addab	wr30	wr7	- wr8	H71 a or d
13A74000		addab	wr29	wr7	wr8	HZ1_b_HZ3_b
13874000		addab	wr28	wr7	wr8	1121_0, 1120_0
13674000	000100 11011 00111 01000 0000000000	addab	wr27	wr7	wr8	
13C74044	000100 11110 00111 01000 00001000100	mules	wr30	wr7	wr8	
13A74045	000100 11101 00111 01000 00001000101	muleu	wr29	wr7	wr8	
13874046	000100 11100 00111 01000 00001000110	mulos	wr28	wr7	wr8	
13674047	000100 11011 00111 01000 00001000111	mulou	wr27	wr7	wr8	
0000000	000000_00000_00000_00000000000000000000					

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

From section 2, the following waveform was generated depicting the hazard detection signals hz1_a_or_d, hz1_b, hz3_a_or_d, and hz3_b. It can be seen that the waveform matches the expected result above.

V2 alud	ra.usc.ed	u:10025 (mattheis)								<u>_D×</u>
F						Nav	eforn 1 - Si k ision			
Eile	Edit }	{iew Exglore For <u>m</u> a	t Simulation	₩indows						Help
Ŧ	$\mathbf{n} \sim$	🌡 🖻 🎕 🗙 🛛)ية هو ال	1. S					\$ ₩ • • ₩	Send To: 🚳 🚟 🖻 🏗 🗐 📰
Searc	h Names	Signal 🕶	- M. M	🖌 Search Times: Value 🕶	1	à.				
1	imeA •	= 390 • ns	-7.◎魚・ -余い	🕸, 🚺 🗱 🔣 📆	📆 🛛 Simulation Time: 4	390ns+1			Time Range	0:390ns - 🕂 🕀 🕞 🕄
×⊙		Baseline = 0								
തി	Cursor-I	Baseline = 390ns		Baseline = 0						Time A = 290/01pr
			Cursor 🕶	0		100ns		200ns	300ns	
1 100	H 🗞	aluOut	′h000⊧	xxx• 000000000	00000000000	▶ (F▶ (0▶ (0▶ (5▶	<u> 5+ 0+ 0+ 0+ 0+</u>	· 0• 0• 0• 0• 0• 0011	1332266• 0000• 00000000	00•)000000000•)00000•) ³
	⊡ %}	aluOut3	′h000⊳	• (000000000000	00000000000000	0•(5•(F•(0•)0•)5• (5• (0• (0•)0•	· (0+ (0+ (0+ (0+ (0+ (0	0011332266+ 00000+ 00000	0000• (000000000• (000•)
	(BE)	aluinmuxop	0							
	(8)	aluinmuxop2	0]						
	⊞ -%¥	aluop	'h18	• (18	01 0B 0	A0) 00 00 0A	00 15 10 16 14	4 11 17 18 OA	18 00	04 05 06 07 18
	田一 ⁵ 論•	aluop2	'h18	• (00 (:	18 01 0	B 08 09 00	0A 00 15 10 16	5 14 11 17 18 OA	18 00	04 05 06 07 18
		clk	1							
	田 称	dataIn	′h000⊧	xxx• 000000000	<u>0000+ (00+)(5+)(0</u>	000000000000000000000000000000000000000	000000000000000000000000000000000000000	00000	000000000 \0+ \0+ \0+ \0 00000	000000000000000000000000000000000000000
	⊡ ~%;	dataIn3	′h000⊧	▶ {0000000000000	0000000•]0•]5	• <u>{</u> 0000000000000000	000000000000000000000000000000000000000	0000000 (0	<u>» {00000000000 }0• {0• {0• }00</u>	000000000000000000000000000000000000000
i.	田 -16+	dataOut	′h000⊧	xxx• (000000000	00000000 0011	• <u>(</u> 0• <u>(</u> 0• <u>)</u> 0000•	0011• (3• (0•)00	0011 <u>00 00 00011</u>	1332266• 10000• 100000000	000000000000000 <u>1</u> 000000•]
		hz1_a_or_d	0							
		hz1_b	0							
	i∎-≪≩.	hz1data	′h000⊳	• {0000000000000	0000000•]0•]5	• (0• (5• (F• (0•	<u>}0+ }5+ }5+ }0+ }0+</u>	· <u> </u>	• (0011332266• (0• (0• (00	0000000• (000000000•)
		hz3	z							
	(1 X)	hz3_a_or_d	0							
	(85)	hz3_b	0							
	⊡ ~%}•	immediate	′hxX0⊧	• 0000000000000	0000• [F800•]0	F• 0000• F8	00• 3• F• C000•	• 1• 0• 0• F80000	J000• 0000• 40000000000	000000000000000000000000000000000000000
	由	immediate2	′hxX0►	• 000000000000	0000000• F800	• (0• (F• 0000•	F800• 3• F• C	000• (1• 0• 0• 0• F800	1000000 0000 4000000	00000000000000000000000000000000000000
1	由お	instruction	′hxxx	• 07E00000 (· 2• 2• 1• 1•		· 1. OF ISFFF80A		1 1 1 1 0 xxxxxxx
E.	i∎≦γ̂r.	instruction1	′hxxx	• (00000000) (• (1• (2• (2• (1•		· 1• 1• 0• 13FFF80		IF IF IF IF OF XXXXXF
1	⊡ ~%}.	instruction2	′hxxx•		10. 10. 11. 11	1 1 2 2		1. 1. 1. 1. 1. 1.	10, 10, 11, 11, 11,	1 1 1 1 1 1 1 1 XXX
5	⊞ %}	instruction3	'hxxx•			1 1 1 2	2 1 1 1 1		SPPPSUA UNUN (IN IN	1 1 1 1 1 1 1 1 1 1 1 1
	■-16+	memAddr	'hxxx•		10+ 11+ 10			1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	NA 10 10 10 74000	
	œ%¢.	<pre>*emAddr_pre</pre>	′hxxx►	• (000000		• <u>(0• (1• (0•)(</u> 0•	<u>) 10 (00 (10 (10) 10</u>	IFF80A	0 10 10 74000	
	Ķ→	memEn	0							
	- 161 	memen pre		5						
10	xecuting	commands in console		-J We						1 object selected

8 Optimization

ME

9 Synthesis Report

ME

(nicely compiled not the output of the tool)

10 Final specifications of your designs

ME (claims)

11 Possible Future Enhancements

If more time was allowed to complete this project, we would have investigated other multiplication algorithms and even pipelined the multiplier because it is the slowest function of the ALU. If the multiply operation took multiple clocks to accomplish and all most other instructions only took a single clock in the MEM/EX stage, then overall the program's execute time for the benchmarks would decrease because the clock frequency that controls the pipeline could be increased.

by Zhyang Ong zhiyang@ieee.org and Andrew Mattheisen amattheisen@gmail.com

12 Conclusion

Whatever you do, reduce the workload for the next class you teach – this semester required more hours per week for school than previous semesters when I took 2 classes, and was miserable - seriously. I am so glad it is finally over.

13 References