Group Name: iloveAVSD

NATIONAL CHENG KUNG UNIVERSITY
College of Electrical Engineering and Computer Science
DEPARTMENT OF ELECTRICAL ENGINEERING
Advanced VLSI System Design (Graduate Level)
Fall 2024

Summary of Final Project

Please don’t just write yes/no if there need more details, and use single-sided printing

<table>
<tr><td colspan="2">Simulate at SoC(yes/no)</td><td colspan="3">yes</td></tr>
<tr><td colspan="5">Basic</td></tr>
<tr><td rowspan="7">M
C
U</td><td rowspan="2">Pipeline</td><td>Stage</td><td>Max working Freq.</td><td>Data Width</td></tr>
<tr><td>5 (5 stage)</td><td>1GHz (>800 MHz)</td><td>32bits (>32 bits)</td></tr>
<tr><td>Number of Instructions</td><td colspan="3">53+22+49 (basic+FPU+VPU) (>65)</td></tr>
<tr><td>Realized Cache Specification</td><td colspan="3">Cache size : 1KB
Block size (cache line size) : 16Byte (4 words)
Entry : 32
2 way set associative
Cache replacement policies : LRU (Least Recently Used Cache)
Read Policy : Read miss - read allocate
Write Policy :
Write hit - write through
Write miss - write no allocate
(e.g. L1 size, associative, read/write policy)</td></tr>
<tr><td>Cache Hit Rate of each program</td><td colspan="3">
<table>
<tr><th>Hit Rate (%)</th><th>I-cache</th><th>D-cache (read)</th><th>D-cache (write)</th></tr>
<tr><td>Prog0</td><td>97.49</td><td>87.66</td><td>3.6</td></tr>
<tr><td>Prog1</td><td>99.8</td><td>99.61</td><td>64.09</td></tr>
<tr><td>Prog2</td><td>99.99</td><td>99.24</td><td>54.02</td></tr>
<tr><td>Prog3</td><td>99.93</td><td>99.96</td><td>89.78</td></tr>
<tr><td>Prog4</td><td>97.63</td><td>94.85</td><td>7.6</td></tr>
<tr><td>Prog5</td><td>97.65</td><td>92.35</td><td>3.07</td></tr>
<tr><td>Prog6</td><td>99.8</td><td>92.25</td><td>21.02</td></tr>
<tr><td>Prog7</td><td>99.8</td><td>92.25</td><td>21.02</td></tr>
<tr><td>Prog8</td><td>99.43</td><td>79.5</td><td>4.79</td></tr>
<tr><td>Prog9</td><td>99.98</td><td>87.97</td><td>96.29</td></tr>
<tr><td>Prog10</td><td>97.64</td><td>94.53</td><td>9.7</td></tr>
<tr><td>Prog11</td><td>99.79</td><td>95.38</td><td>20.99</td></tr>
<tr><td>Prog12</td><td>99.26</td><td>88.81</td><td>3.47</td></tr>
<tr><td>Prog13</td><td>97.22</td><td>99.99</td><td>98.44</td></tr>
<tr><td>Prog14</td><td>98.23</td><td>95.01</td><td>12.44</td></tr>
</table>
</td></tr>
<tr><td>List of Realized Forwarding in Types and Stages</td><td colspan="3">Two types of Data Hazard : ID <-> WB and EXE <-> MEM, WB
** USE_FP = { RS1 (2), RS2 (1), RD (0) } **
check 1. Is using RS : (|RS1 & ~USE_FP) & USE_FP -> also check x0
check 2. Is using RD : (|RD & ~USE_FP) & USE_FP -> may not need to check
check 2. Is using same type of RS/RD (FP or INT)
check 3. Is RS equals to RD
(e.g. which kind)</td></tr>
<tr><td>Realized Performance Counters (IPC) of each program</td><td colspan="3">
<table>
<tr><td>IPC</td><td>Prog0</td><td>Prog1</td><td>Prog2</td><td>Prog3</td><td>Prog4</td><td>Prog5</td><td>Prog6</td></tr>
<tr><td>per cycle</td><td>0.42</td><td>0.6</td><td>0.73</td><td>0.69</td><td>0.43</td><td>0.4</td><td>0.33</td></tr>
<tr><td>Prog7</td><td>Prog8</td><td>Prog9</td><td>Prog10</td><td>Prog11</td><td>Prog12</td><td>Prog13</td><td>Prog14</td></tr>
<tr><td>0.33</td><td>0.18</td><td>0.45</td><td>0.46</td><td>0.39</td><td>0.53</td><td>0.71</td><td>0.48</td></tr>
</table>
</td></tr>
</table>

Group Name: iloveAVSD

	Interrupt mechanism	WDT_interrupt, DMA_interrupt		
Memory	On-chip memory	IM	DM	
		64KB	128KB	
	Off-chip memory	SDRAM	ROM	
		2MB	8KB	
BUS	Specification	Operating Frequency	Bit-width	
		1GHz (>400 MHz)	32bits (>32 bits)	
	Specify Memory and I/O mapping	Slave	Start address	End address
		ROM	0x0000_0000	0x0000_1FFF
		IM	0x0001_0000	0x0001_FFFF
		DM	0x0002_0000	0x0003_FFFF
		DRAM	0x2000_0000	0x201F_FFFF
		DMA	0x1002_0000	0x1002_0400
		WDT	0x1001_0000	0x1001_03FF
	Implemented Features of AXI Bus, Level of Realization, Outstanding number	The AXI in our design is a cross-bar bridge. The outstanding number is 1, but the read/write channel can work separately. We use a round-robin arbiter in our AXI.		
System	Specify Cooperation between CPU, Bus, Memory, EPU and others	As the instruction reach execute stage in CPU, if it’s a vector instruction, it’ll be assigned to VPU through the frontend in EX stage. VPU will access L1-Cache for load/store operation. (e.g. How your system works)		
	Specify Hardware interrupt & Interrupt service routines	The interrupt flow first checks the mie CSR register. If the local interrupt enable bit is set to 0, the interrupt is ignored. If it is set to 1, the next step is to check the global interrupt enable bit in the mstatus CSR register. If the global interrupt enable bit is also set to 1, the interrupt is considered valid, and the CPU will jump to the trap handler to process the interrupt. It will finally go back to normal execution once mret inst. is executed. When the CPU executes the WFI instruction, it transitions to the WFI state, which stalls the entire CPU pipeline. If an interrupt is not taken (i.e., an interrupt is received but the local interrupt is disabled), the CPU remains in the WFI state. Only when the interrupt is considered valid (after checking both local and global interrupt enables) does the CPU jump to the trap handler and then return to normal execution. (>2 kind, and how they work)		
	Specify Mechanism for Booting from an external ROM	CPU will fetch from ROM for booting instruction through Cache, as executing the booting instruction, CPU will set source address, destination address, data length into DMA, which is made for moving data from the external DRAM to on chip IM/DM. CPU will in the wfi state, waiting for DMA sent its interrupt signal to CPU addressing that it has complete data transfer.		

	Specify Realized DMA(Direct Memory Access) and Usage	wfi instruction will occur after the enable of DMA is written to 1 by CPU, then the DMA will start the process of moving data from source address to destination address which are both written into DMA by CPU. Each round, DMA will read 4 sets of data from source and write 4 sets of data to destination, until moving DMALEN sets of data.	
Code analysis (Superlint)		1 - (137/13993) = 99.02% (should >99% error free)	
System w/ EPU (yes/no)		yes	
		Synthesis	APR
clock period		1ns	1.25ns
Power		52.3867mW	8.43e+07 nW
Area		384536.033485	3180397.85164800
Chip cost		4,068,000 (NTD)	

EPU			
EPU	Max working Freq.		1GHz
	Processing speed (throughput or…)		VPU can handle 8 operation per cycle since it has 8 lanes. Which results in 8*1GHz = 8GHz-op (8G op per sec)
	Realized Specification of Functionalities in details		Out EPU is a vector processing unit which supports SIMD.
	Comparison with other works if any		Not that we know of
Verification	MCU	prog0 pass ratio	100%
	EPU	# and types of Direct test or constrained random test	There's a program for VPU instruction functionality test, which check all the instructions supported.
		Specify types, length, operation conditions of benchmarks	Prog5: vector instruction direct test Prog6: vector addition Prog7: vector multiplication Prog8: AXPY Prog9: matrix multiplication Prog10: convolution Prog11: ReLU Prog12: 64-bit addition Prog13: Bicubic Prog14: Deutsch-Jozsa algorithm (at least 7 benchmark)
	SYSTEM	prog0 PR simulation time	
		prog1 PR simulation time	
		Specify types, length, operation conditions of benchmarks	Prog0: provided by TA Prog1: quick sort Prog2: image gray scaling Prog3: WDT prog4: FPU direct test

Group Name: iloveAVSD

Advanced	
10 more instructions	FPU: 22 more instructions VPU: 49 more instructions
64-bit add/sub, store/load	With vadd64/vsub64, vle64/vse64 operation See prog12
Synthesize AXI bus with burst and fully work with IPs	Not yet…
More cache (L2 or L3)	No…
stack or other mechanisms to facilitate function calls	Return address stack With stack depth = 8
dynamic branch prediction	A two-level saturating branch prediction BHT, BTB size are both 32
Verify with FPGAs, specify FPGA board, what module has been put on the board and how you confirm results	No…
CRT for more than two IPs	No…
I/O PADs	Not yet…
floating-point co-processor	FPU with 22 new instructions See prog4
Bootable by an operating system	No…
Other Properties, please specify	Formal verification (ISA formal), AXI (starvation freedom, mutual exclusion, req->taken)
References	