OpenRISC Instruction Set Architecture Extensions

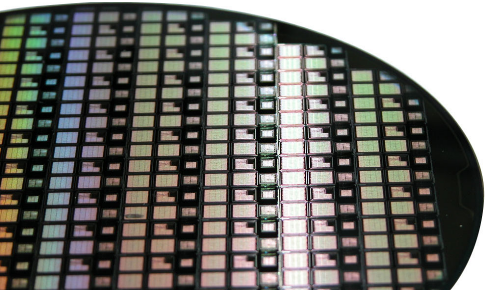
Master Thesis

Zurich, May 2015

Andreas Traber

Advisors
Michael Gautschi
Antonio Pullini
Prof. Dr. Luca Benini

ETH zürich
Integrated Systems Laboratory

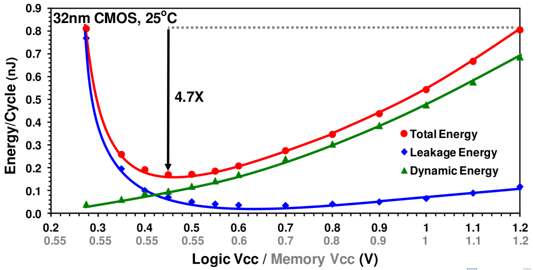


Outline

Motivation

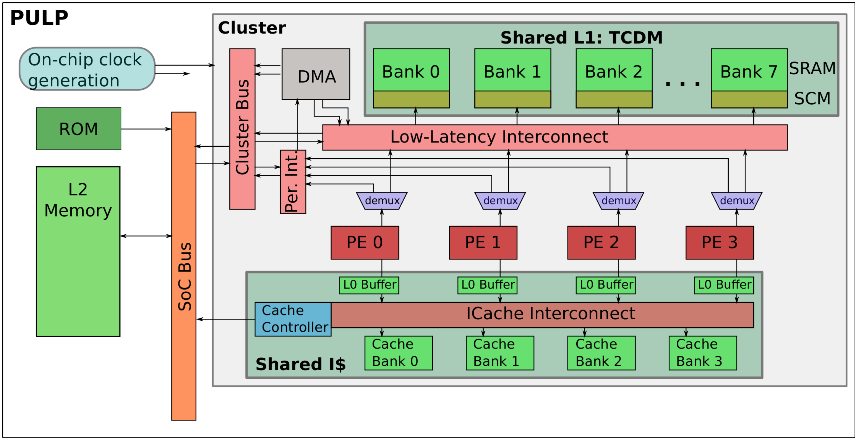
Ultra-Low-Power Computing

- Todays embedded devices need a vast amount of computing power
  Internet of things, wearable devices, sensors, smartphones, . . .
- Need to be energy efficient
  Operate digital circuits near the threshold voltage
- Use parallelization to recover performance
  Turn off unused processing elements



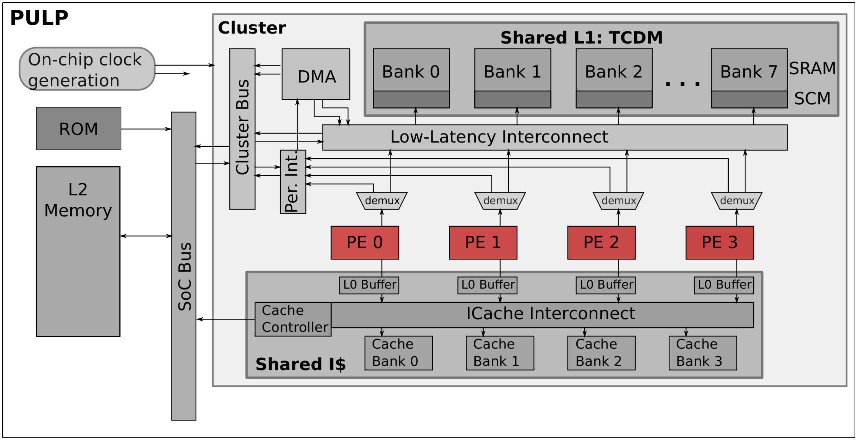
PULP

Parallel Ultra-Low power Processor



PULP

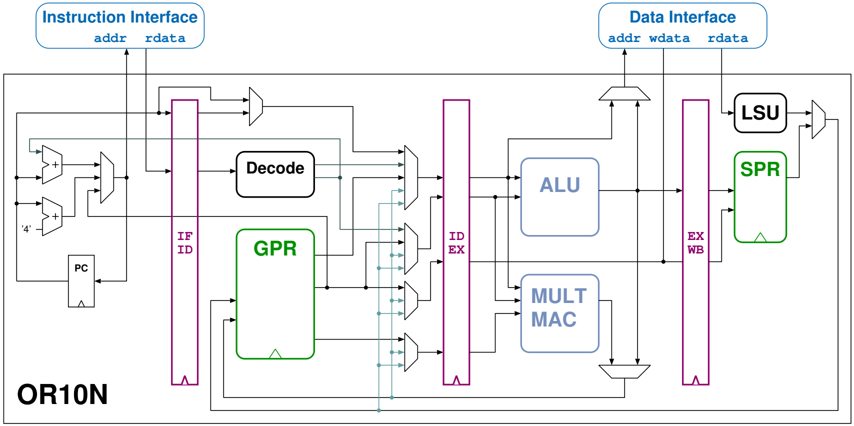
Parallel Ultra-Low power Processor



- My work is focused on the PE

PE: OR10N

In-order 4-stage OpenRISC CPU



Goals

- Higher energy efficiency

Goals

- Higher energy efficiency
- Doing more computations per cycle ⇒ ISA extensions
  - Hardware loops
  - Auto-incrementing load/stores
  - Vectorial instructions
  - Improved MAC
  - Bit counting instructions

Goals

- Higher energy efficiency
- Doing more computations per cycle ⇒ ISA extensions
  - Hardware loops
  - Auto-incrementing load/stores
  - Vectorial instructions
  - Improved MAC
  - Bit counting instructions
- No increase in clock cycle time

Goals

- Higher energy efficiency
- Doing more computations per cycle ⇒ ISA extensions
  - Hardware loops
  - Auto-incrementing load/stores
  - Vectorial instructions
  - Improved MAC
  - Bit counting instructions
- No increase in clock cycle time
- Extensions supported by compiler

  New instructions generated automatically

Vectorial ALU

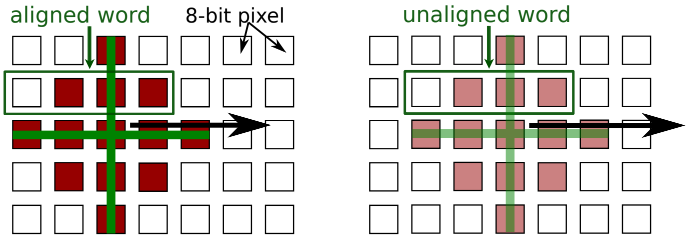
Use subword parallelism

- Re-use existing ALU and data path
- Segment the data path dynamically into 2/4 parts
- No separate register file needed

Vectorial ALU

Use subword parallelism

- Re-use existing ALU and data path
- Segment the data path dynamically into 2/4 parts
- No separate register file needed
- Unaligned memory access



Vectorial ALU

Encoding

31							0		
001010	DDDDDD	AAAAA	BBBBB	---	00	sub-opcode	0	**lv.inst.h**	**rD, rA, rB**
001010	DDDDDD	AAAAA	BBBBB	---	01	sub-opcode	0	**lv.inst.h.sc**	**rD, rA, rB**
001010	DDDDDD	AAAAA	IIIII	III	10	sub-opcode	0	**lv.inst.h.sci**	**rD, rA, I**

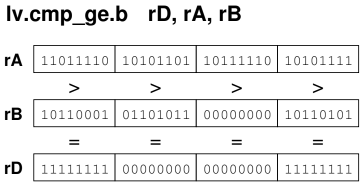
001010	DDDDDD	AAAAA	BBBBB	---	00	sub-opcode	1	**lv.inst.b**	**rD, rA, rB**
001010	DDDDDD	AAAAA	BBBBB	---	01	sub-opcode	1	**lv.inst.b.sc**	**rD, rA, rB**
001010	DDDDDD	AAAAA	IIIII	III	10	sub-opcode	1	**lv.inst.b.sci**	**rD, rA, I**

- Sub-opcodes
  - ADD
  - SUB
  - **AVG**
  - **MIN**
  - **MAX**
  - **ABS**
  - SRL
  - SRA
  - SLL
  - OR
  - AND
  - XOR
  - **INS**
  - **EXT**

Vectorial ALU

Comparisons

- `lv.cmp_*`
  Vectorial comparison
- `lv.all_*`
  Set flag if all comparisons evaluate to true
- `lv.any_*`
  Set flag if any comparison evaluates to true


lv.cmp_ge.b rD, rA, rB
rA 11011110 10101101 10111110 10101111
> > > >
rB 10110001 01101011 00000000 10110101
= = = =
rD 11111111 00000000 00000000 11111111

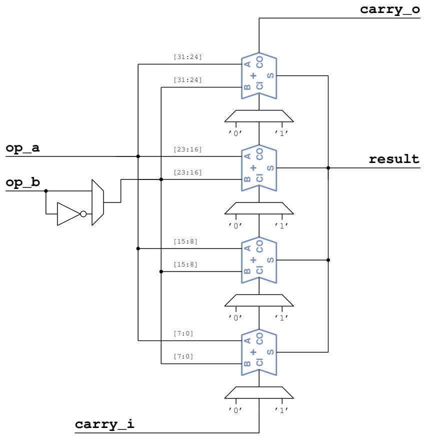
Vectorial ALU

Simple Example: Vectorial ADD/SUB

- For SUB: Invert operand b
- Four slices chained together
- Suppress carries where needed

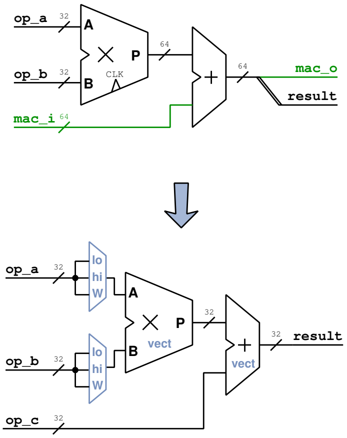
  0 for ADD

  1 for SUB



Improved MAC Unit

- Use only 32-bit multiplication result
- Put accumulator in GPR
  Multiple accumulerators in GPR
  Instead of one in SPR
- Vectorial MAC
  Offers same parallelism as ALU
- 16-bit subword selection
  Accelerates 64-bit multiplication
- Single-cycle execution
  No pipeline stalls
  Two-cycles with old MAC

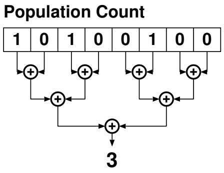


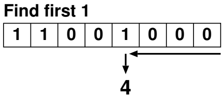
Bit Counting

- Population Count: Number of bits set to 1
- Count leading bits
- Find first/last 1 in a word
- Not automatically generated by compiler
  Need to use builtins

- Applications
  - Runtimes
  - Normalization

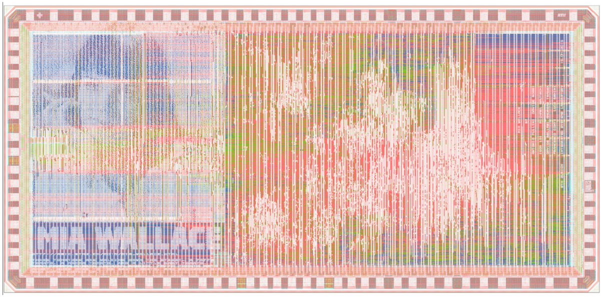
- Core area increase by 0.8%





Mia Wallace

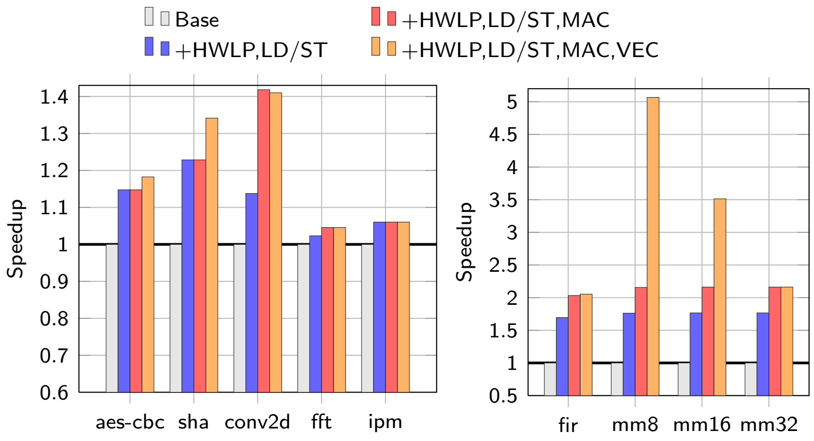
PULP chip with 4 cores in 65 nm



- Max. Frequency: 333 MHz @ 1.08V
- L2 Memory: 256 kB
- TCDM: 64 + 8 kB

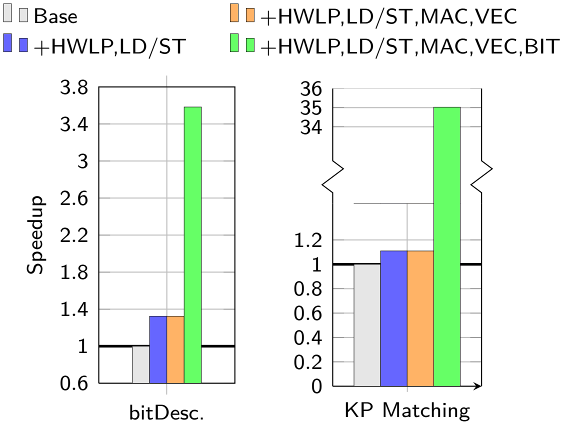
Performance: MAC and Vectorial

Increased up to 5x



Performance: Bit Count

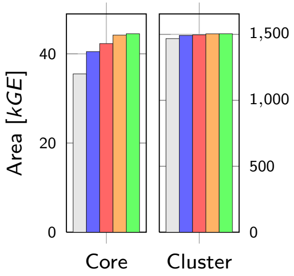
Increased up to 35x



Area & Timing

Area increase by 25%

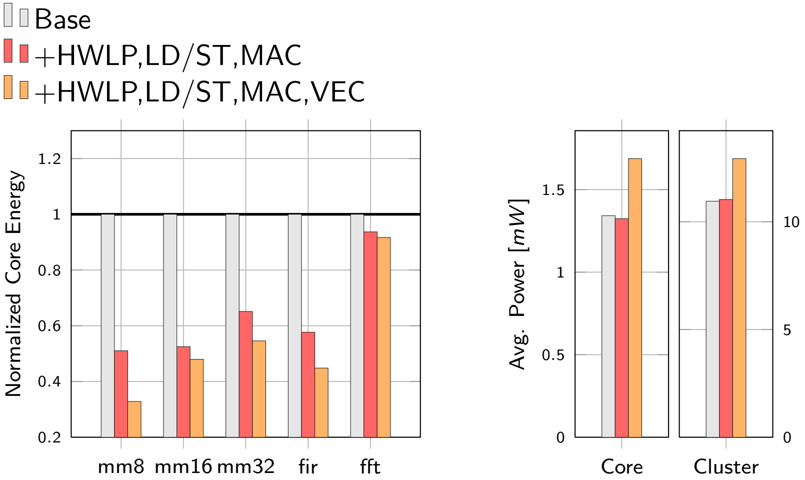
Feature	Area	
⬜ Baseline	35.5 kGE	
🟦 HWLP	3.0 kGE	+8.5%
🟦 Reg. File Add.	3.7 kGE	+10.4%
🟥 New MAC	1.2 kGE	+3.3%
🟧 Vectorial ALU	1.9 kGE	+5.3%
🟩 Bit Count	0.2 kGE	+0.8%
Total	*44.5 kGE*	



- No increase in critical path

Energy

45% more energy efficient on average



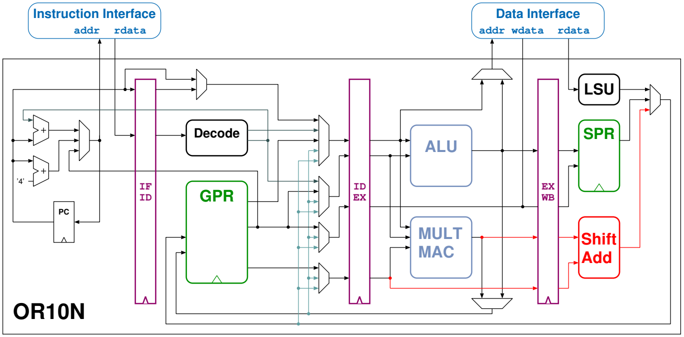
Conclusion

- Vectorial unit and new MAC cost only ~9% core area
  Less than 1% cluster area increase
- Up to 5x speedup compared to base OpenRISC on vectorial code
  Instructions can be generated automatically by the compiler
- Up to 35x speedup with bit counting instructions
- No increase in critical path
- 45% more energy efficient

Backup Slides

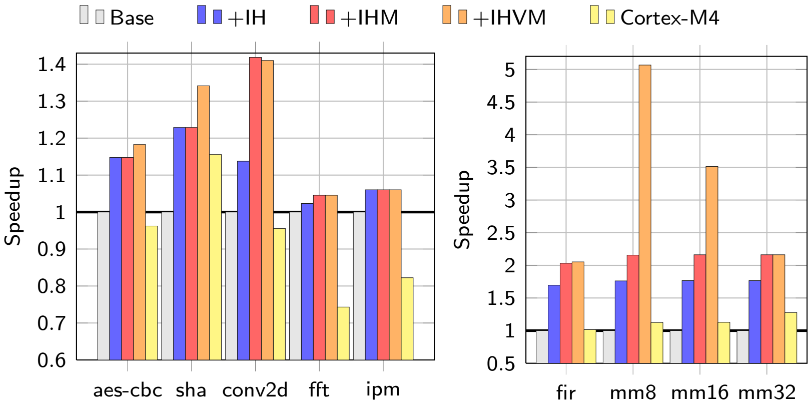
Outlook

- Look into fractional support
  $rD = (rD \gg 16) + (rA[31:16] \cdot rB[31:16]) \ll 1$
- Redesign multiplier



Performance: MAC and Vectorial

Increased up to 5x

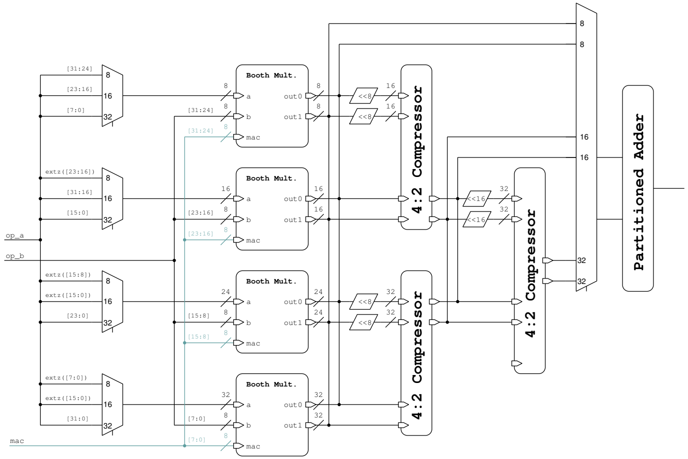


Area Multiplier

in 28nm

Design	Features	Area OR10N	Area Mult.	Timing
Baseline		20125	5950	7.57 ns
Vec. Booth Mult.	MAC	18925	5230	7.83 ns
Vec. Booth Mult.	MAC, subword	18850	5451	7.83 ns
Shared Segmentation	MAC	19518	6687	7.77 ns
Shared Segmentation	MAC, subword	18854	5125	7.84 ns
Behavioral	MAC	18900	4202	7.58 ns
Behavioral	MAC, subword	18884	4287	7.64 ns

Vectorial Booth Multiplier



op_a
op_b
mac
[31:24]
[23:16]
[7:0]
extz([23:16])
[31:16]
[15:0]
extz([15:8])
extz([15:0])
[23:0]
extz([7:0])
extz([15:0])
[31:0]
8
16
32
Booth Mult.
a
b
mac
out0
out1
[31:24]
[23:16]
[15:8]
[7:0]
<<8
<<16
4:2 Compressor
Partitioned Adder