Improving Ibex Performance

lowRISC

Greg Chadwick
RISC-V Devroom FOSDEM 1st February 2020
Ibex

- Microcontroller class CPU with two stage pipeline
- 32-bit RISC-V IMC/EMC with M-Mode, U-Mode and PMP
- Written in SystemVerilog
- Initially developed as Zero-riscy as part of the PULP platform by ETH Zurich
- Now developed by lowRISC, a not for profit company building open source silicon through collaborative engineering
- Used by the recently announced OpenTitan, an open source silicon root of trust
Improving Performance

● Aim to reduce total cycles to execute Coremark and Embench
● Need to be careful about optimising for the benchmark only
● Analysis of execution provides a useful guide for what to improve
● Must consider how applicable improvements will be to code that isn’t benchmarks
● Planned improvements will be configurable options
  ○ Choose a smaller/simpler Ibex or a faster one
Trial System

- Simulate Ibex with Verilator
- Dual ported memory containing code and data
- Single cycle memory access latency
- Reasonable analogue of a best case ‘real’ system
Analysis Techniques (1)

- Run the benchmark
- Trace the simulation
- Examine trace in GTKWave
  - Look at signals indicating top-level stall
  - Choose a few points to examine why stall is occurring
- No quantitative analysis but quick and easy way to survey what kinds of things are slowing down execution
Trace in GTKWave, Branch Stall

```
bne t2, s5, 100404
```

ALU checks branch condition
Trace in GTKWave, Branch Stall

```
116664 ns  116668 ns  116672 ns
0010048C  00100410  00100404
001FF938  0000030A  00100410
00000002  0000004F  FFFFFFF4
00000000  00000000  00000000
00000000  00000000  00100404
```

ALU checks branch condition
ALU calculates branch target

```
bne t2,s5,100404
```
Trace in GTKWave, Branch Stall

```
branch_taken bne t2,s5,100404
```

ALU checks branch condition
ALU calculates branch target
Branch Taken
Trace in GTKWave, Load Stall

1st February 2020

Iw t3,12(sp)
Trace in GTKWave, Load Stall

\texttt{lw \ t3,12(sp)}
Analysis Techniques (2)

- Log performance counters after benchmark run
- Use previous survey to decide on interesting things to count
- Examine with spreadsheet to produce quantitative data on effect stall conditions from informal survey have on performance
Branch Stall %

% of total cycles spent calculating branch target

1st February 2020
Memory Stall %

% of total cycles spent waiting for memory response
Branch target ALU

- Add second ALU to calculate branch targets
- Compute branch target and branch condition in parallel
- Minor area increase for ~4% performance gain
Implementation Trials

- Need to check impact of change on frequency and area
- Built experimental synthesis flow using Yosys with Timing Analysis via OpenSTA
- Using the nangate 45nm library available from the OpenROAD repository
- Better numbers likely achievable with commercial tools and library
  - Flow used to see relative changes and areas of timing pressure
Branch Target ALU Implementation Results

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>Branch Target ALU</th>
<th>% change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coremark/MHz</td>
<td>2.40</td>
<td>2.51</td>
<td>+4.5 %</td>
</tr>
<tr>
<td>Area</td>
<td>27,345 μm²</td>
<td>27,666 μm²</td>
<td>+1.2 %</td>
</tr>
<tr>
<td>Fmax</td>
<td>269 MHz</td>
<td>234 MHz</td>
<td>-13.0 %</td>
</tr>
<tr>
<td>Coremark</td>
<td>645.6</td>
<td>587.3</td>
<td>-9.0 %</td>
</tr>
</tbody>
</table>

- Adding in branch target ALU reduced maximum frequency
- Overall worse performance at Fmax (but better per MHz)
- What can we do about it?
Can you spot the problem (1)?
Can you spot the problem (2)?

- Previously the branch decision was stored in a flop after being computed by the main ALU.
- Now it's being fed straight in the PC Mux select.
- Main ALU result used to feed into PC selection mux (as it computed the target), which was the worst path.
- It now goes via extra logic into the select.
- So worst path has got longer.
How Do We Fix It?

- Need main ALU result earlier
- Key issue is selects for ALU operand mux, provided by the decoder
- Decoder complex blob of logic, so outputs not as early as we like
- Make the ALU operand mux select outputs earlier from the decoder and we can solve the problem
Instruction Flop Fan-Out

- Instruction flop in ID/EX has a large fan-out
  - Meaning it feeds its data to many different gates
- Requires buffering to ensure it can drive everything it connects to
- Reduce required buffering by duplicating it
- Split decode to decide ALU operand select and operation from duplicated register
- Decode all other control from other register
## Improved Branch Target ALU Implementation

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>Branch Target ALU</th>
<th>% change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coremark/MHz</td>
<td>2.40</td>
<td>2.51</td>
<td>+4.5 %</td>
</tr>
<tr>
<td>Area (μm²)</td>
<td>27,345</td>
<td>27,579</td>
<td>+0.9 %</td>
</tr>
<tr>
<td>Fmax (MHz)</td>
<td>269</td>
<td>250</td>
<td>-7.6 %</td>
</tr>
<tr>
<td>Coremark</td>
<td>645.6</td>
<td>627.5</td>
<td>-2.8 %</td>
</tr>
</tbody>
</table>

- Slightly better area due to reduced buffering
- Still haven’t restored Fmax
  - Yosys/ABC doesn’t take IO timing constraints into account
  - So doesn’t optimise worst path properly
  - May not want to run at Fmax anyway
Writeback Stage

- Add a third pipeline stage, writeback which holds the value to be written to the register file
- Load data from memory writes direct to the register file
- Drops a stall cycle for loads & stores as response only needed the cycle after ID/EX
- Greatly Simplified Diagram!
  - Significant new stalling and hazard logic needed
Writeback Implementation

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>Writeback + BT ALU</th>
<th>% change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coremark/MHz</td>
<td>2.40</td>
<td>2.88</td>
<td>+20.0 %</td>
</tr>
<tr>
<td>Area</td>
<td>27345 μm²</td>
<td>29212 μm²</td>
<td>+6.8 %</td>
</tr>
<tr>
<td>Fmax</td>
<td>269 MHz</td>
<td>253 MHz</td>
<td>-6.3 %</td>
</tr>
<tr>
<td>Coremark</td>
<td>645.60</td>
<td>728.64</td>
<td>+12.9 %</td>
</tr>
</tbody>
</table>

- Notable area cost
  - Outweighed by performance gains
- Little change in Fmax from BT ALU implementation
  - Worst case path from BT ALU change still dominates
Overall Speedup

<table>
<thead>
<tr>
<th></th>
<th>Coremark/MHz</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>2.40</td>
<td>-</td>
</tr>
<tr>
<td>BT ALU</td>
<td>2.51</td>
<td>4.5%</td>
</tr>
<tr>
<td>Writeback + BT ALU</td>
<td>2.88</td>
<td>20%</td>
</tr>
</tbody>
</table>

Geomean Speedup

<table>
<thead>
<tr>
<th></th>
<th>Geomean Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT ALU</td>
<td>4.42%</td>
</tr>
<tr>
<td>Writeback + BT ALU</td>
<td>21.3%</td>
</tr>
</tbody>
</table>

1st February 2020
Find Out More

- Check out the Ibex repository
  www.github.com/lowRISC(ibex)
- Third pipeline stage + benchmarking infrastructure not yet in main repository
  - See my ‘ibex_fosdem’ branch at
    www.github.com/GregAC(ibex) to take a look
- See the lowRISC website at www.lowrisc.org
  - Now recruiting!
- My email: gac@lowrisc.org