Customized Embedded Processor Design

Final Presentation

Peter Maucher, Yussuf Khalil, Tim Scheurer | 26 July 2021

Outline

1. Speed Optimization

- Added Instructions
- Added Resources
- Problems
- Benchmarks

2. Area Optimization

- Removed Resources
- Pipeline Changes
- 16 Register Machine
- Problems
- Benchmarks

3. Conclusion

Performance-Optimized CPU: “brownie-PERF”

4 instructions added

- `CLAMP0`: clamp `val` between `0` and `max`
- `SHADD`: multiply with $\frac{5}{4}$
- `ADDPRED`: add or sub using predicate
- `EXSM`: extract bits from bitfield

CLAMP0

`CLAMP0 dest val max` – Clamp value to $[0, max]$

- `val`: Value to clamp
- `max`: Upper boundary

- Move register value into range $[a, b]$
- Fixed lower boundary to zero ⇒ only two inputs required
- Not too specialized: usable in two distinct places in the program

SHADD

`SHADD dest bits add` – Multiply with $\frac{5}{4}$

- `bit`: 3-bit bitfield
- `add`: value to be shifted and added
- Compresses part of the *adpcm* algorithm into a single instruction
- Requires additional ALU

ADDPRED

`ADDPRED dest base off pred` – Add or subtract value from another based on predicate

- `base`: Base value
- `off`: Value added or subtracted
- `pred`: Predicate deciding to add or sub
- Requires three read ports from register file
  - ASIPmeister only allows 1, 2, or 4
  - Another forwarding unit needed

EXSM

`EXSM sig mag val` – Extract sign and magnitude out of 4-bit number

- Requires two write ports into register file

⇒ Amount of forwarding units doubled

Added Resources

- No custom resources added
- Copied some already existing ones:
  - One `ALU` (needed in `SHADD` instruction)
  - Four `FWUs` (for full forwarding with instructions having either 3 inputs or 2 outputs)

Problem: FPGA

- Major problems working with the FPGA
- browstd32 worked with executable compiled with `-O0` – our machine did not
    - same executable as before
    - only change to browstd32: `CLAMP0` instruction added

⇒ relied entirely on ModelSim for testing

Problem: Correct Forwarding

- At some point we kept getting garbage output
- Reason: Increasing the number of read and write ports prevented full forwarding

Problem: Correct Forwarding

- At some point we kept getting garbage output
- Reason: Increasing the number of read and write ports prevented full forwarding
- Solution: Increase number of forwarding units to $num_{write\_ports} \cdot num_{read\_ports}$ and add + adjust macros

Problem: Correct Forwarding

- At some point we kept getting garbage output
- Reason: Increasing the number of read and write ports prevented full forwarding
- Solution: Increase number of forwarding units to $num_{write\_ports} \cdot num_{read\_ports}$ and add + adjust macros
- In `ADDPRED` instruction:
- In WB stage: forwarded wrong value, but correct writeback
- Error only materialized later when compiler reordered instructions and forwarding became important

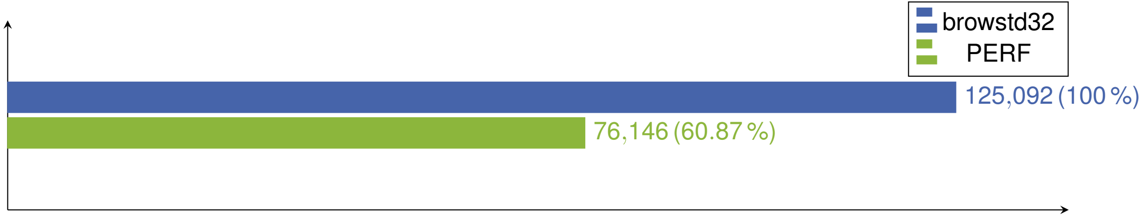
brownie-PERF vs browstd32

Benchmark options:

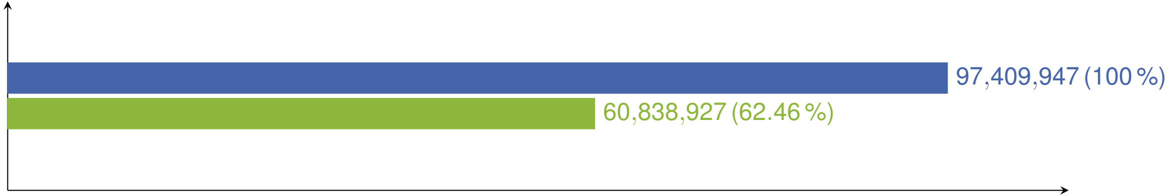
- Generated in ModelSim/ISE using the MINI and BRAM data
- Always `-O3`

Performance [Cycles]

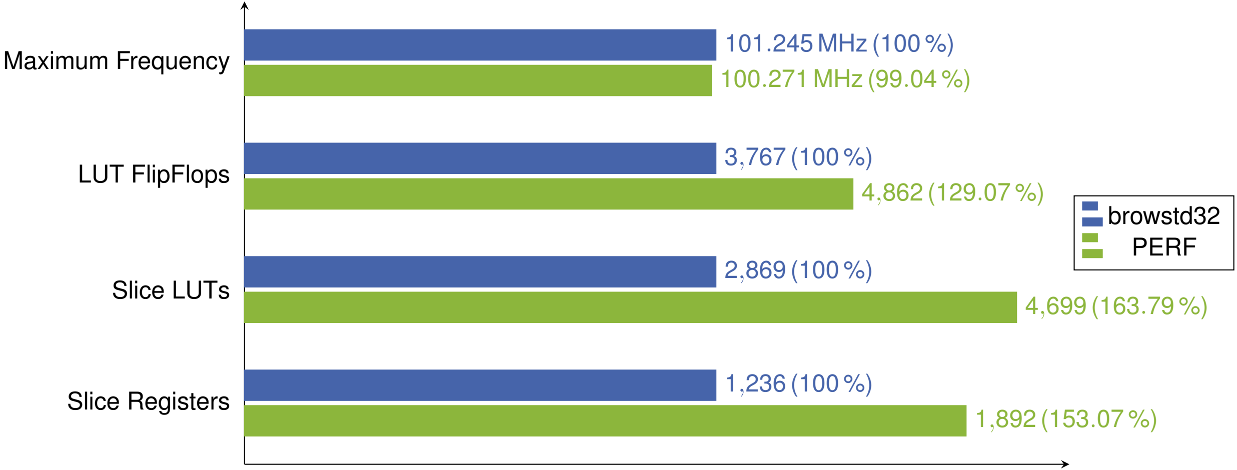
MINI.h without UART



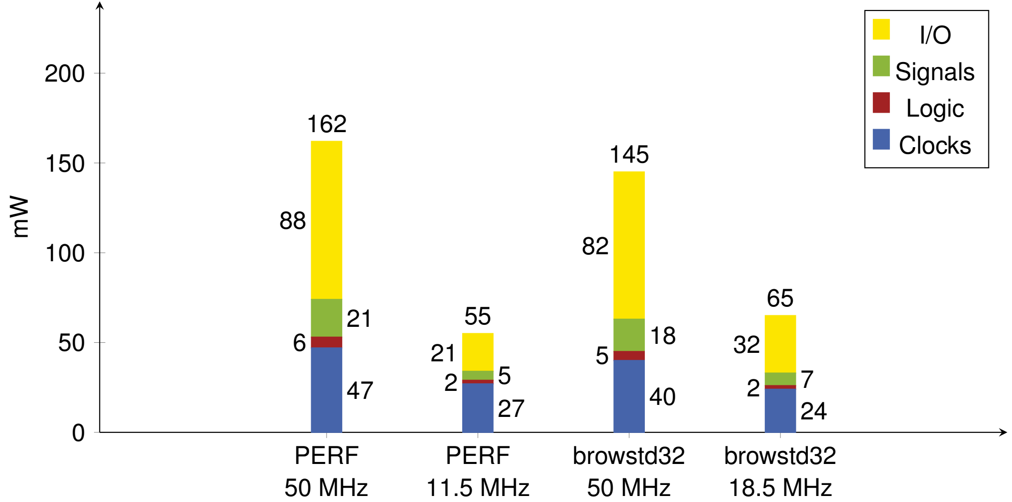
BRAM.h without UART



Area


Maximum Frequency
101.245 MHz (100 %)
100.271 MHz (99.04 %)
LUT FlipFlops
3,767 (100 %)
4,862 (129.07 %)
Slice LUTs
2,869 (100 %)
4,699 (163.79 %)
Slice Registers
1,236 (100 %)
1,892 (153.07 %)
browstd32
PERF

Dynamic Power



Area-Optimized CPU: “brownie-AREA”

Three major optimizations

- Removed unneeded resources
- 3-stage pipeline
- 16 register machine “brownie-AREA-16”
  - Also benchmarked 32 register machine

  ⇒ “brownie-AREA-32” and “brownie-AREA-16”

Remove unneeded resources

Execution Units

- `MUL`
- `DIV`

Instructions

- mul/div: `MUL, DIV, DIVU, MOD, MODU`
- alu: `NAND, NOR, XORI`
- shift: `LRS`
- load/store: `LH, SB, SH`
- special: `RETI, EXBW, EXHW`

Remove unneeded resources

Execution Units

- `MUL`
- `DIV`

Instructions

- mul/div: `MUL, DIV, DIVU, MOD, MODU`
- alu: `NAND, NOR, XORI`
- shift: `LRS`
- load/store: `LH, SB, SH`
- special: `RETI, EXBW, EXHW`

Kept `CLAMP0` instruction, though

- “brownie-AREA-32” still 20% faster than standard brownie
- Easily implemented, small area impact

4-Stage Pipeline ⇒ 3-Stage Pipeline

- High clock not as important (`-O3` is fast enough for everything)
- Removes a lot of hardware
- Forwarding units now only need 1 port
- No more branch delay slots (yay)

4-Stage Pipeline ⇒ 3-Stage Pipeline

- High clock not as important (`-O3` is fast enough for everything)
- Removes a lot of hardware
- Forwarding units now only need 1 port
- No more branch delay slots (yay)
- Several options for merging stages
  - Fetch + Decode: more complicated
  - Execute + Writeback: can easily be merged
- ASIPmeister automatically merges stages (yay)

4-Stage Pipeline ⇒ 3-Stage Pipeline

- High clock not as important (`-O3` is fast enough for everything)
- Removes a lot of hardware
- Forwarding units now only need 1 port
- No more branch delay slots (yay)

- Several options for merging stages
  - Fetch + Decode: more complicated
  - Execute + Writeback: can easily be merged
- ASIPmeister automatically merges stages (yay)

- Small fixes afterwards:
- Wires with same name in old EXE and WB stages are now duplicated ⇐ rename or remove

16 Register Machine

- Optimize area: use as few registers as possible
- Still use compiler

16 Register Machine

- Optimize area: use as few registers as possible
- Still use compiler
- Generated code (`-O0`): uses `r0, r1` and `r3 - r10`
- Generated code (`-O3`): uses `r0, r1` and `r3 - r25`

16 Register Machine

- Optimize area: use as few registers as possible
- Still use compiler

- Generated code (`-O0`): uses `r0, r1` and `r3 - r10`
- Generated code (`-O3`): uses `r0, r1` and `r3 - r25`

- `r0-r15`: special registers that compiler always uses with `-O3`
- In ASIPMeister: drop-down menu for number of registers: 4, 8, 16, 32.
- ⇒ reduce to 16 registers

16 Register Problems: Compiler

- Compiler not aware of #Registers
- Always assumes 32

16 Register Problems: Compiler

- Compiler not aware of #Registers
- Always assumes 32
- Solution: use compiler option `-ffixed-rXX` for XX $\in \{16 \dots 31\}$
- Tells compiler that register can not be used except for calling convention reasons

16 Register Problems: Assembly

- In helper code (`handler.s` and `startup.s`), `r16` is used as hard-coded register

16 Register Problems: Assembly

- In helper code (`handler.s` and `startup.s`), `r16` is used as hard-coded register
- Solution: copy over files (and rest of build folder)
- Change `r16` to `r7` (double return register, unneeded)
- Would have allowed for 8 registers with new calling convention

16 Register Problems: ASIPmeister

- *Everything* changes to 4 bit register width
- Need to change a lot of macros and wires
- Need to change most instruction formats
- Always change MSB to `dont_care`, so compiler still works

16 Register Problems: ASIPmeister

- *Everything* changes to 4 bit register width
- Need to change a lot of macros and wires
- Need to change most instruction formats
- Always change MSB to `dont_care`, so compiler still works
- Assembler generation in ASIPmeister fails
- Solution: apparently still works anyway

Other Problem: Removed Too Many Instructions

- Wrote small script that parses all instructions in generated assembly
- Assumption: contains all code

Other Problem: Removed Too Many Instructions

- Wrote small script that parses all instructions in generated assembly
- Assumption: contains all code

- Apparently not: `startup.s`, `handler.s` excluded
- Disabled `ORI` instruction, which enables interrupts
- Traps for end of process never enabled
- Infinite loop due to program falling through from `_startup` (at address `0x0`) to `_main`, which then returned to `_startup...`

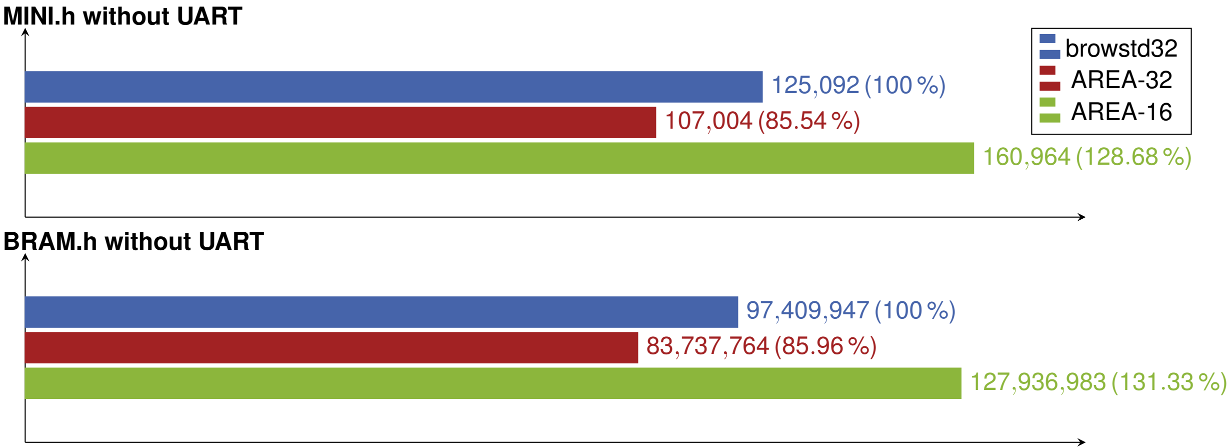
- Solution: check these files too, re-enable instructions

brownie-AREA vs browstd32

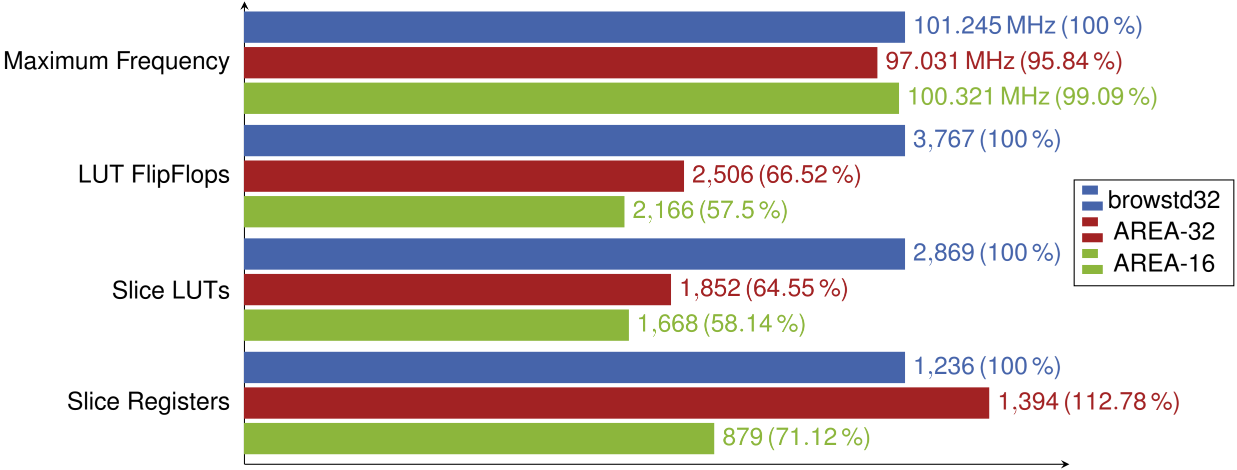
Benchmark options:

- Generated in ModelSim/ISE using the MINI and BRAM data
- Always `-O3`
- Custom `CLAMP0` instruction enabled for brownie-AREA-16/32
- Additional compiler option `-ffixed-r16 ...-ffixed-r31` for brownie-AREA-16

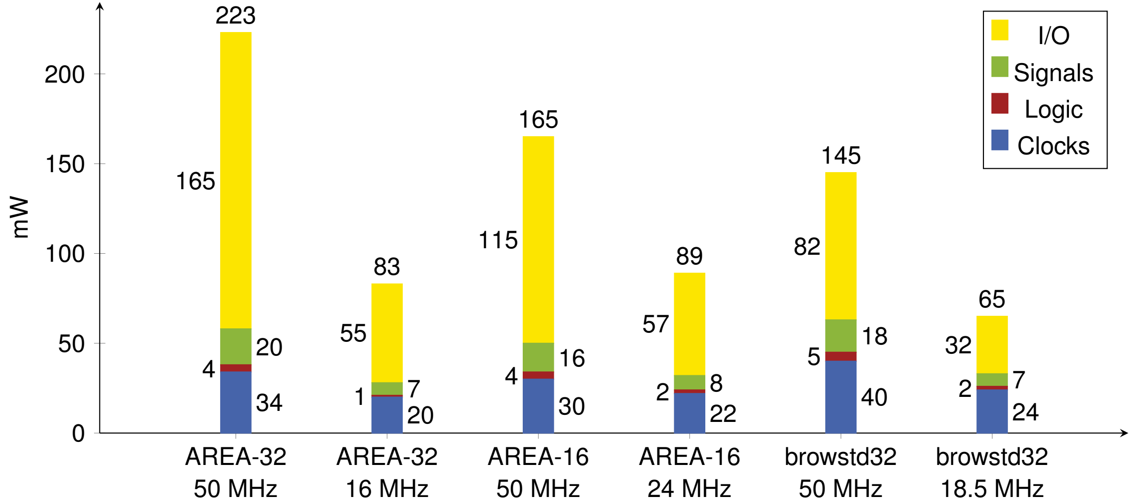
Performance [Cycles]



Area


Maximum Frequency
101.245 MHz (100 %)
97.031 MHz (95.84 %)
100.321 MHz (99.09 %)
LUT FlipFlops
3,767 (100 %)
2,506 (66.52 %)
2,166 (57.5 %)
Slice LUTs
2,869 (100 %)
1,852 (64.55 %)
1,668 (58.14 %)
Slice Registers
1,236 (100 %)
1,394 (112.78 %)
879 (71.12 %)
browstd32
AREA-32
AREA-16

Dynamic Power



Conclusion

Built two processors specialized towards different metrics.

Performance

- Speedup: 38% less cycles
- Cost (area): 30% more FlipFlops, 60% more LUTs
- Power consumption: 15% less power at min frequency

Area

- Area: 40% less FlipFlops and LUTs, 30% less slice registers
- Cost (performance): 30% more cycles
- Cost (power consumption): 37% more power at min frequency