GOMACTech-2019

BlackParrot
An Open-Source, Industrial-Strength, RISC-V, Linux Capable, Multicore Processor

28 MAR 2019

Mark Wyse
University of Washington
We need the “Linux of RISC-V Cores”

Open source software has inspired an open source hardware movement

RISC-V is an open ISA, not an open source processor

Existing open source RISC-V processor implementations aren’t there:
– Implementations are closely held and do not focus on distributed collaboration
– Employ “freemium” models with inherent conflicts of interest
– Use languages that are unfamiliar to HW designers
– Lack Best-of-Class Implementations

“SiFive [chips] are based on rocket-chip, but have proprietary enhancements and, as the BSD license allows, are not open-source.”
BlackParrot: Four Success Metrics

(achieve these and BlackParrot will become the Linux of RISC-V)

Will People Trust Our Code?
- Is it easy to understand?
- Is it secure?
- Is it validated?
- Will you put it in Silicon?
BlackParrot: Four Success Metrics

(achieve these and BlackParrot will become the Linux of RISC-V)

Will People Trust Our Code?
Is it easy to understand?
Is it secure?
Is it validated?
Will you put it in Silicon?

BlackParrot is a “stone soup” designed to:

- convince the smartest people in the world to improve it.
- scale to many users.
- get companies to invest and become stewards of the code.

We want to gather a village...

BlackParrot: Four Success Metrics
(achieve these and BlackParrot will become the Linux of RISC-V)

Will People Trust Our Code?
Is it easy to understand?
Is it secure?
Is it validated?
Will you put it in Silicon?

BlackParrot is a “stone soup” designed to:

- convince the smartest people in the world to improve it.
- scale to many users.
- get companies to invest and become stewards of the code.

Functionality
Virality
Quality

Does the code have the features people need?
And leave out the ones they don’t?
BlackParrot: Four Success Metrics
(achieve these and BlackParrot will become the Linux of RISC-V)

Will People Trust Our Code?
Is it easy to understand?
Is it secure?
Is it validated?
Will you put it in Silicon?

BlackParrot is a “stone soup” designed to:
- convince the smartest people in the world to improve it.
- scale to many users.
- get companies to invest and become stewards of the code.

Does the code have the features people need?
And leave out the ones they don’t?

Is the code Pareto optimal in terms of Power, Performance, and Area?
The BlackParrot Manifesto

- **BE TINY**
  - Place a premium on a small, understandable, agile, secure code base.
  - Minimize unused features or configurations that increase complexity and verification.

- **BE MODULAR**
  - Use well-defined interfaces that enable scalable, global participation.
  - Enable modular testability & CI.

- **BE FRIENDLY**
  - Welcome contributions and distributed ownership.
  - Combat “Not Invented Here” Syndrome.
  - Ultimate goal: stewardship of code distributed across industry and the world.
  - Be easy to use.
The BlackParrot “Genesis Release” Team

Prof. Michael Taylor

Prof. Ajay Joshi

Prof. Mark Oskin

Dan Petrisko
Farzam Gilani
Mark Wyse
Tommy Jung

Leila Delshad
Boyou Zhou
Zahra Azad
Furkan Eris

Katie Lim
Yongqin Wang
Bandhav Veluri
Shaolin Xie
Tavio Guarino
BlackParrot Overview

- RISC-V RV64GC, Linux Capable
- 8 stage pipelined in-order, cache coherent multicore
- Modular Design, Clean Interfaces, Clean Code
  - Front End (FE) – instruction fetch (speculative)
  - Back End (BE) – instruction execution (non-speculative)
  - Memory End (ME) – memory system, cache coherence
- Target SoC: Accelerator chip or embedded system that needs a performant, Linux capable control system
  - Drop it in and get it working with high PPA and little to no physical design effort
  - Anti-target: a Xeon-competitive server core
    - Requires large physical design team (50-100 people)
BlackParrot FE Overview

- FE speculatively fetches instructions
- FE does not modify any arch. state
- FE-BE interface
  - FE queue FIFO (ordered stream of instructions and instruction-related events from FE to BE)
  - FE cmd FIFO (“correction commands” from BE to FE)
- FE-ME interface
  - Sends and receives coherence messages
  - LCE to/from CCE interfaces
BlackParrot BE Overview

- BE maintains “true” architectural state
  - PC, Regfiles, CSR
- 6 stage, single-issue, non-stalling BE pipeline
  - Design avoids global stall signal, single stall point at ISD (instruction stall / dispatch)
    - Avoids physical design with large fanout signal
  - Cache misses poisons subsequent instructions and instruction is redispached on cache line fill.
  - “Rollback” FIFO at BE/FE iface avoids refetch on cache miss
- Modular Data Cache with Clean Interface
  - 32 kB, 8-way set associative, 64-byte cache line
  - 3-cycle hit latency
  - LCE & BE-ME interface used to communicate with L2$
BlackParrot ME Overview

- Cache Coherence and Memory Consistency, L2 and DRAM Interface
  - TSO consistency (currently SC!)
- Cache Coherence Engines (CCE)
  - Microprogrammed coherence controller
  - Protocol changes via microcode update
- Local Cache Engines (LCE)
  - L1 entities (I$, D$, I/O$)
- Protocol
  - Latency Insensitive, Directory-Based
  - Strong Invariants that lead to simple correctness proofs
  - Currently Exclusive/Invalid
    - Microcode updates will provide MESI soon
BlackParrot – Example Layout

- Single Core Floorplan
  - Red/Pink = FE
  - Yellow = BE
  - Purple/Green = ME
- Area = 0.948 mm$^2$
  - 40nm process
- \(~1.5\) GHz at 40nm
  - \(~2.0\) GHz at 16nm
BlackParrot

• BlackParrot is the RISC-V processor for all
• BlackParrot v1.0 will be released June 2019
  – “pre-alpha” version available now
• Actively improving and adding features:
  – Linux capability
  – PPA optimizations
• See the (rapidly changing) code on github
  – https://github.com/black-parrot/pre-alpha-release/tree/dev
• First tapeout June 2019

• Thanks to Andreas Olofsson and the staff of the DARPA POSH program
Thank You!