Experiences Using the RISC-V Ecosystem to Design an Accelerator-Centric SoC in TSMC 16nm

Tutu Ajayi², Khalid Al-Hawaj¹, Aporva Amarnath², Steve Dai¹, Scott Davidson⁴, Paul Gao⁴, Gai Liu¹, Anuj Rao⁴, Austin Rovinski², Ningxiao Sun⁴, Christopher Torng¹, Luis Vega⁴, Bandhav Veluri⁴, Shaolin Xie⁴, Chun Zhao⁴, Ritchie Zhao¹, Christopher Batten¹, Ronald G. Dreslinski², Rajesh K. Gupta³, Michael B. Taylor⁴, Zhiru Zhang¹

¹ Cornell University
² University of Michigan
³ University of California, San Diego
⁴ Bespoke Silicon Group, (U. Washington/ UC San Diego)

MICRO-50 October 14, 2017
Prototyping is important to complement the results of simulation-based research.

Many benefits to prototyping:

• Validating assumptions
• Validating design methodologies
• Measuring real system-level performance and energy efficiency
• Creating platforms for software research
• Building credibility with industry
• Building intuition for physical design
• Pedagogical benefits
• Building real things is fun!
The Continuing Need for Building Prototypes

The rise of the dark silicon era\(^1\), in which an increasing fraction of silicon must remain unpowered, is motivating an increasing trend towards accelerator-centric architectures.

Specialization research requires:

- New *simulation-based* evaluation methodologies based on accelerators\(^2\)
- New *prototyping* methodologies for rapidly building accelerator-centric prototypes

Unfortunately, building research prototypes can be tremendously challenging.

---


Prototyping with the RISC-V Software/Hardware Ecosystem

Software Toolchain
- A complete, off-the-shelf software stack (e.g., binutils, GCC, newlib/glibc, Linux kernel & distros) for both embedded and general-purpose

Architecture
- RISC-V ISA specification designed to be both modular and extensible, with a small base ISA and optional extensions

Microarchitecture
- On-chip network specifications and implementations (NASTI, TileLink)
- RISC-V processor implementations for both in-order (Berkeley Rocket) and out-of-order (Berkeley BOOM) cores

Physical Design
- Previous spins of chips for reference

Testing
- Standard core verification test suites + Turn-key FPGA gateware
The Celerity System-on-Chip

**Celerity**, an accelerator-centric SoC with a tiered accelerator fabric that targets highly performant and energy-efficient embedded systems

Funded by the DARPA CRAFT program, “Circuit Realization At Faster Timescales”

The goal was to develop new methodologies to design chips more quickly

We leveraged the RISC-V software/hardware ecosystem as we built Celerity, and we believe it was instrumental in enabling a team of **20 graduate students** to tape out a complex SoC in **only 9 months**
Celerity: Chip Overview

- TSMC 16nm FFC
- 25 mm² die area (5mm x 5mm)
- ~385 million transistors
- 511 RISC-V cores
  - 5 Linux-capable RV64G Berkeley Rocket cores
  - 496-core RV32IM mesh tiled array “manycore”
  - 10-core RV32IM mesh tiled array (low voltage)
- Binarized Neural Network Specialized Accelerator
- On-chip synthesizable PLLs and DC/DC LDO
  - Developed in-house
- 3 Clock domains
  - 400 MHz – DDR I/O
  - 625 MHz – Rocket core + Specialized accelerator
  - 1.05 GHz – Manycore array
- 672-pin flip chip BGA package
- 9-months from PDK access to tape-out
Agenda

- Introduction
- For each Tier:
  - What did we build?
  - How did we build it?
  - RISC-V Ecosystem Successes
  - RISC-V Ecosystem Challenges
- Conclusion

Celerity :: Introduction
Celerity: General-Purpose Tier

- How did we build it?
- Successes with RISC-V
- Challenges with RISC-V
General-Purpose Tier Overview

- 5 Berkeley Rocket Cores (RV64G)
- Workload
  - General-purpose compute
  - Operating system (e.g. Linux & TCP/IP Stack)
  - Interrupt and Exception handling
  - Program dispatch and control flow
- Interface
  - Interface to off-chip I/O and other peripherals
  - 4 Cores connect to the manycore array
  - 1 Core interfaces with the BNN
- Memory
  - Each core executes independently within its own address space
  - Memory management for all tiers
Berkeley Rocket Cores

• 5 Berkeley Rocket Cores
  (https://github.com/freechipsproject/rocket-chip)
  • Generated from Chisel
  • RV64G ISA
  • 5-stage, in-order, scalar processor
  • Double-precision floating point
  • I-Cache: 16KB 4-way assoc.
  • D-Cache: 16KB 4-way assoc.

• Physical Implementation
  • 625 MHz (Critical path in FSB)
  • 0.19 mm² per core

http://www.lowrisc.org/docs/tagged-memory-v0.1/rocket-core/
Design Iterations

1. Loopback
*Baseline design to validate FSB and Northbridge*

2. Alpaca
*Implemented NASTI bridge and connected rocket core*

3. Bison
*Implemented accelerator connected through Blackboxed RoCC*

4. Coyote
*Modularized RoCC interface to accelerator*

Celerity :: General-Purpose Tier :: What is it? • How did we build it? • Successes with RISC-V • Challenges with RISC-V
Off-Chip Interface and Northbridge

- Open-source BaseJump IP Library
  - [http://bjump.org](http://bjump.org)
- Front Side bus
  - BaseJump Communication Link
  - High Speed (DDR) Source-Synchronous Communication Interface
- Packaging
  - Modified BaseJump BGA Package and I/O Ring
- Validation
  - BaseJump Super Trouble PCB (Daughter Card)
  - BaseJump Motherboard (ZedBoard)
RISC-V Successes

- Berkeley Rocket Cores
  - Very quickly generated validated designs
  - Vibrant ecosystem to provide feedback and support
  - Test and Validation infrastructure
  - Software and Toolchain support
- Flexible memory system and peripheral I/O support
  - Easy integration with BaseJump IP Library
- Balances extensibility and software support
RISC-V Lessons Learned

- Component stability, compatibility and versioning
- Chisel adoption
- RTL simulation issues
  - Deciphering Chisel generated RTL
  - Register initialization and X-Pessimism
Celerity: Massively Parallel Tier

http://bjump.org/manycore

Developed by Taylor’s Bespoke Silicon Group @ UW
The tiled architecture

496 RISC-V Cores

The Vanilla core: Simple but efficient to run C code without any toolchain modification

- ISA: RV32IM
- Pipeline: 5-stage, fully forwarded, in-order, single issue
- Scratchpad memory: 4KB for I Mem, 4KB for D Mem
- Second Tape-out of this tiled architecture (10-core)
Mesh Network

- **Link Protocol**: Forward/Reverse paths, parameterizable address/data bits
- **Credit-Based**: Each packet will be acknowledged with response
- **Flow control**: Endpoint controls the number of the outstanding packet.
- **Router**: Simple XY-dimension routing, buffered
Manycore Links to General-Purpose and Specialized Tier

Cross Clock Domain interface
- **To General-Purpose Tier:** Convert RoCC to link protocol, support configuring DMA, write and reset manycore etc.
- **To Specialized Tier:** Aggregate link interface to increase the bandwidth and throughput
Producer-consumer programming model: extended instructions for efficient inter-tile synchronization

- **Load Reserved (lr.w):** load value and set the reservation address
- **Load-on-broken-reservation (lr.lbr):** stall if the reserved address didn’t written by other cores
- **Consumer:** wait on <address, value>
- **Benefits:** No polling, no interrupt, fast response, stalled pipeline can save power
Thread Density Comparison

- **Timing**: 1.05 GHz @ 16 nm
- **Area**: 0.024 mm² @ 16 nm
- **Si Utilization Ratio**: 90%

### Normalized Physical Threads (ALUops) per Area

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Normalized Area (32nm)</th>
<th>Area Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Celerity Tile @16nm</td>
<td>D-MEM = 4KB I-MEM = 4KB</td>
<td>0.024 * (32/16)² = 0.096 mm²</td>
</tr>
<tr>
<td>OpenPiton Tile @32nm</td>
<td>L1 D-Cache = 8KB L1 I-Cache = 16KB L1.5/L2 Cache = 72KB</td>
<td>1.17 mm² [1]</td>
</tr>
<tr>
<td>Raw Tile @180nm</td>
<td>L1 D-Cache = 32KB L1 I-SRAM = 96KB</td>
<td>16.0 * (32/180)² = 0.506 mm²</td>
</tr>
<tr>
<td>MIAOW GPU Compute Unit Lane @32nm</td>
<td>VRF = 256KB SRF = 2KB</td>
<td>15.0 / 16 = 0.938 mm² [2]</td>
</tr>
</tbody>
</table>

How did we build the massively parallel tier?

**In house**
- Design
  - Basejump STL library
  - Data flow
  - NoC
  - Arithmetic
  - ...
- Testing
  - RISC-V tool chain
  - Assembly Test Suite
  - Modified Runtime
  - C Compiler
  - ...

**Hierarchical Flow**
- NOC Router
- XBAR
- RISC-V Vanilla-5 Core
- I Mem
- D Mem

**One tile**
- Hard-macro
- I Mem | RF | D Mem

**Floorplan**

Celerity :: Massively Parallel Tier :: What is it? • How did we build it? • Successes with RISC-V • Challenges with RISC-V
RISC-V Ecosystem Successes

• Modular ISA
  • Flexible for both complex cores (i.e. Rocket) and simple cores (i.e. Vanilla)

• Extensible RoCC interface
  • 4 customizable instructions: we used one

• Comprehensive assembly test suite (434 test cases)

• Off-the-shelf toolchain
Building up the RISC-V Ecosystem

We provide an efficient RV-32IM implementation in System Verilog.

We consolidated Information about RoCC that was scattered across the internet.

With Celerity
- Efficient open source core
- Based on Systemverilog
- Silicon proven
- Public RoCC document V.2
  [bjump.org/rocc_doc]
- Exported RoCC interface on top level
Celerity: Specialization Tier

Celerity :: Specialization Tier :: What is it? • How did we build it? • Successes with RISC-V • Challenges with RISC-V
Case Study: Mapping Flexible Image Recognition to a Tiered Accelerator Fabric

Three steps to map applications to tiered accelerator fabric:

Step 1. Implement the algorithm using the general-purpose tier

Step 2. Accelerate the algorithm using either the massively parallel tier OR the specialization tier

Step 3. Improve performance by cooperatively using both the specialization AND the massively parallel tier
Step 1: Algorithm to Application

Binarized Neural Networks

- Training usually uses floating point, while inference usually uses lower precision weights and activations (often 8-bit or lower) to reduce implementation complexity
- Rastorgui et al. [3] and Courbariaux et al. [4] have recently shown single-bit precision weights and activations can achieve an accuracy of 89.8% on CIFAR-10
- Performance target requires ultra-low latency (batch size of one) and high throughput (60 classifications/second)

**Step 1: Algorithm to Application**

**Characterizing BNN Execution**

- Using just the general-purpose tier would be 200x slower than the performance target (60 classifications/sec).
- Binarized convolutional layers consume over 97% of dynamic instruction count.
- Perfect acceleration of just the binarized convolutional layers is still 5x slower than performance target.
- Perfect acceleration of all layers using the massively parallel tier could meet performance target but with significant energy consumption.

<table>
<thead>
<tr>
<th>Execution Time</th>
<th>2.15%</th>
<th>23.11%</th>
<th>12.02%</th>
<th>23.98%</th>
<th>12.85%</th>
<th>25.68%</th>
<th>0.2%</th>
<th>0.02%</th>
<th>&lt;0.01%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights' Size (Mbits)</td>
<td>0.003</td>
<td>0.14</td>
<td>0.28</td>
<td>0.56</td>
<td>1.13</td>
<td>2.25</td>
<td>8</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>Input Size (Kbits)</td>
<td>60</td>
<td>128</td>
<td>32</td>
<td>64</td>
<td>16</td>
<td>32</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Step 2: Application to Accelerator
BNN Specialized Accelerator

1. Accelerator is configured to process a layer through RoCC command messages
2. Memory Unit starts streaming the weights into the accelerator and unpacking the binarized weights into appropriate buffers
3. Binary convolution compute unit processes input activations and weights to produce output activations
Step 2: Application to Accelerator
General-Purpose Tier for Weight Storage

• The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer’s weights
Step 3: Assisting Accelerators
Massively Parallel Tier for Weight Storage

- The BNN specialized accelerator can use one of the Rocket cores’ caches to load every layer’s weights.
- Each core in the massively parallel tier executes a remote-load-store program to orchestrate sending weights to the specialization tier via a hardware FIFO.

Celerity :: Specialization Tier :: What is it? • How did we build it? • Successes with RISC-V • Challenges with RISC-V
Performance Benefits of Cooperatively Using the Massively Parallel and the Specialization Tiers

<table>
<thead>
<tr>
<th></th>
<th>General-Purpose Tier</th>
<th>Specialization Tier</th>
<th>Specialization + Massively Parallel Tiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime per Image (ms)</td>
<td>4,024</td>
<td>20</td>
<td>3.3</td>
</tr>
<tr>
<td>Power (Watts)</td>
<td>0.2 – 0.5</td>
<td>0.2 – 0.5</td>
<td>0.5 – 2.0</td>
</tr>
<tr>
<td>Improvement in Perf / Power</td>
<td>1x</td>
<td>~200x</td>
<td>~400x</td>
</tr>
</tbody>
</table>

**General-Purpose Tier**
Software implementation assuming ideal performance estimated with an optimistic one instruction per cycle

**Specialization Tier**
Full-system RTL simulation of the BNN specialized accelerator running with a frequency of 625 MHz

**Specialization + Massively Parallel Tiers**
Full-system RTL simulation of the BNN specialized accelerator with the weights being streamed from the manycore
void bnn::dma_req() {
    while (1) {
        DmaMsg msg = dma_req.get();

        for (int i = 0; i < msg.len; i++) {
            HLS_PIPELINELOOP(HARD_STALL, 1);

            int req_type = 0;
            word_t data = 0;
            addr_t addr = msg.base + i*8;

            if (type == DMA_TYPE_WRITE) {
                data = msg.data;
                req_type = MemReqMsg::WRITE;
            } else {
                req_type = MemReqMsg::READ;
            }

            memreq.put(MemReqMsg(req_type, addr, data));
        }

        dma_resp.put(DMA_REQ_DONE);
    }
}
Design Methodology

- **SystemC**
- **StratusHLS**
- **RTL**
- **PyMTL**
- **Wrappers & Adapters**
- **Constraints**
- **Final RTL**
- **ASIC Flow**
- **Hard Macro**

**Including RoCC Interfaces**

**Celerity :: Specialization Tier ::**  
- What is it?  
- How did we build it?  
- Successes with RISC-V  
- Challenges with RISC-V
RISC-V Ecosystem Successes and Challenges

Successes

- The RoCC command and memory interface were both significant successes. We connected the accelerator with no changes to RV64G core, just as we did for the manycore array in the massively parallel tier.

Challenges

- Small challenge in the RoCC accelerator interface at the specific commit we chose to use
  - Memory management unit in RV64G used only physical addresses
  - We did a small workaround to give us virtual addresses as well
  - This challenge has already been fixed upstream
**The Celerity System-on-Chip**

*Celerity*, an accelerator-centric SoC with a tiered accelerator fabric that targets highly performant and energy-efficient embedded systems.

*Celerity’s goal* was to develop new methodologies to design chips more quickly.

We believe the RISC-V software/hardware ecosystem was instrumental in enabling a team of 20 graduate students to tape out a complex SoC in only 9 months.

**Acknowledgements:** DARPA, under the CRAFT program

Special thanks to Dr. Linton Salmon for program support and coordination.

We thank the many contributors to the open-source RISC-V software and hardware ecosystem with special thanks to U.C. Berkeley for forming the RISC-V ecosystem.

http://www.opencelerity.org