Evaluation of RISC-V for Pixel Visual Core

Matt Cockrell (mcockrell@google.com)
May 9th, 2018
Overview

- Use case
- Core selection
- Integration
Background: Pixel Visual Core

“Pixel Visual Core is the Google-designed Image Processing Unit (IPU)—a fully programmable, domain-specific processor designed from scratch to deliver maximum performance at low power.”[1]

Critical point: A dedicated A53 (top left) aggregates application layer IPU resource requests and configures appropriately.

Pixel Visual Core today

Main CPU

Dedicated A-Class CPU runs OS and supports IPU

I/O

IPU
Modern mobile SoC with multiple devices.

Main CPU no longer dedicated only to IPU ⇒ Local controller is desired

Add micro-controller as job scheduling and dispatch unit.
Core Selection Considerations

**Level of Effort**
- How difficult would it be to work with and integrate.

**Risk**
- Stability and reliability of support.

**License**
- Flexibility of use.
Candidate 1: Bottle Rocket  (https://github.com/google/bottlerocket)

- Internal Project to demonstrate ability to easily develop custom RISC-V implementation by leveraging Rocket Chip.
- Implements RV32IMC
- Represents evaluating Rocket Chip as an option.
Candidate 2: Merlin (https://github.com/origintfj/riscv)

- Core provided from a hobbyists developed compatible with “QFlow”
- Implements 32IC
- The hobbyist was a team member, use of this core would become a “build from scratch” candidate.
Candidate 3: RI5CY (RISK-EE) (https://github.com/pulp-platform/riscv)

- Provided from the PULP team
- Implements RV32IMC with added extensions
- This candidate comes from and is maintained by academia
## Open core comparison

<table>
<thead>
<tr>
<th>Core</th>
<th>Level of Effort</th>
<th>Risk</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottle Rocket</td>
<td>High</td>
<td>Med</td>
<td>✔</td>
</tr>
<tr>
<td>Merlin</td>
<td>Low</td>
<td>High</td>
<td>✔</td>
</tr>
<tr>
<td>RI5CY</td>
<td>Low</td>
<td>Med</td>
<td>✔</td>
</tr>
</tbody>
</table>
Recommended Candidate

Selected RI5CY from PULP:

- Had been taped-out
- Provided infrastructure
- Solderpad license
- It was implemented in SystemVerilog (instead of Chisel):
  - SystemVerilog builds on established physical design and verification flows
  - Chisel generated Verilog loses designer’s intent making it difficult to read and debug
  - Chisel generated code makes certain physical design items difficult such as sync/async clocks, power domains, clock domains, etc.
Integration of RI5CY

The Good:
- RTL provided in SystemVerilog
- ETH/PULP Team
- Debug capability
- Able to work with Valtrix to verify
- Documentation

The Bad:
- Numerous lint errors
- Ad hoc verified
- Bugs found:
  - PULPino Compiler
  - Documentation
  - Extensions
- Debug setup requires PULPino specific utilities.

The Ugly (Scary):
- Version control
- Bugs found in:
  - Multiplier
  - LSU
Recap and next steps

Where we have been:
- Describe possible PVC configuration mechanism
- Continued evaluation of RI5CY
- Shared experience of integrating open source IP

Where we are going:
- Add full compliance for privilege/debug specification.
- Evaluate performance impact after adding RI5CY to PVC
Questions?