Towards a Rocket Chip Based Implementation of the RISC-V GPC Architecture

Paderborn University / Heinz Nixdorf Institute

Lars Luchterhandt\textsuperscript{a}, Tom Nellius\textsuperscript{a}, Robert Beck\textsuperscript{a}, Rainer Dömer\textsuperscript{b}, Pascal Kneuper\textsuperscript{a}, Wolfgang Müller\textsuperscript{a}, and Babak Sadiye\textsuperscript{a}

\textsuperscript{a}Paderborn University / Heinz Nixdorf Institute, Paderborn, Germany
\textsuperscript{b}University of California, Irvine, USA
Towards a Rocket Chip Based Implementation of the RISC-V GPC Architecture

- Grid of Processing Cells (GPC) Platform [1]
  - Prototype implementation at RTL

- Design a RISC-V SoC of the GPC Platform

- RTL simulation of SoC

- Functional test using a demo program

GPC Platform – Grid of Processing Cells (GPC)

- Introduced by R. Dömer at UC Irvine
- Addresses bottleneck to main memory
- Replace cache hierarchy with on-chip memories
Chipyard Framework

- Open-source **framework** for RISC-V **SoC development** by UC Berkeley
- Powered by **Chisel** (Hardware Construction Language)

Chipyard Features…
- SoC generators
  - Rocket Chip SoC generator
- RTL simulation
  - Verilator simulator
- SoC software development
- Support for VLSI flow & FPGA prototyping flow
Chisel HCL

- Open-source Hardware Construction Language by UC Berkeley
- “Constructing Hardware In a Scala Embedded Language”
- Embedded in programming language Scala

Benefits from Scala
- Object-orientated
- Functional-programming
  - Higher level of abstraction in hardware design

- Allows implementation of parameterizable generators
- Generators generate specific circuits
- Verilog HDL can be emitted
Rocket Chip Generator

- RISC-V SoC generator written in Chisel
- **Generates** scalable **Rocket Chip** system
- Basis of Chipyard framework
- Highly customizable

Rocket Chip generator features…

- **Rocket Core**
  - In-order RISC-V core with five-stage pipeline
- Memory system
- On-chip network (TileLink)
- Debug Unit, Interrupt Controller, …
Rocket Chip System

RocketTile
- Rocket Core
- L1 Data Cache
- L1 Instruction Cache
- Page-Table Walker
- SystemBus connection

Starting point for GPC prototype implementation
Rocket Chip Based GPC Prototype

- RocketTile \rightarrow Processing Cell
- Modifications
  - Scratchpad memory instead of L1-Data cache
**Rocket Chip Based GPC Prototype**

- **RocketTile → Processing Cell**
- ** Modifications**
  - Scratchpad memory instead of L1-Data cache
  - No L2 cache

---

**Diagram**

- RocketTile
  - L1I
  - L1D
  - TileBus
- PTW
- AXI Master
- AXI to TileLink
- FrontBus
- SystemBus
- ControlBus
  - BootROM
  - PLIC
  - CLIC
  - Debug Unit
- PeripheryBus
  - AXI Mem
  - TileLink to AXI
  - Other Device
  - JTAG
- AXI Slave
- TileLink to AXI
- L2 Bank
- L2 Bank
- Memory Bus
- No L2 cache
Rocket Chip Based GPC Prototype

- RocketTile → Processing Cell
- Modifications
  - Scratchpad memory instead of L1-Data cache
  - No L2 cache
  - No AXI interfaces

Diagram:
- RocketTile
- PTW
- L1I
- L1D
- TileBus
- SystemBus
- AXI Master
- AXI to TileLink
- FrontBus
- ControlBus
- BootROM
- PLIC
- CLIC
- Debug Unit
- PeripheryBus
- JTAG
- TileLink to AXI
- Other Device
- AXI Slave
Rocket Chip Based GPC Prototype

- RocketTile \rightarrow Processing Cell

- Modifications
  - Scratchpad memory instead of L1-Data cache
  - No L2 cache
  - No AXI interfaces
  - Replicate RocketTiles
Rocket Chip Based GPC Prototype

- RocketTiles
- Each with own scratchpad memory
- Access to neighboring scratchpads via SystemBus

GPC Prototype

- N RocketTiles
- Each with own scratchpad memory
- Access to neighboring scratchpads via SystemBus
Each core has its own scratchpad memory

Scratchpads have non-overlapping memory addresses ranges

<table>
<thead>
<tr>
<th>Base</th>
<th>Top</th>
<th>Device</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>0x 8000 0000</td>
<td>0x 8000 8000</td>
<td>Scratchpad 0</td>
<td>32 KiB</td>
</tr>
<tr>
<td>0x 8000 8000</td>
<td>0x 8001 0000</td>
<td>Scratchpad 1</td>
<td>32 KiB</td>
</tr>
<tr>
<td>0x 8001 0000</td>
<td>0x 8001 8000</td>
<td>Scratchpad 2</td>
<td>32 KiB</td>
</tr>
<tr>
<td>0x 8001 8000</td>
<td>0x 8002 0000</td>
<td>Scratchpad 3</td>
<td>32 KiB</td>
</tr>
</tbody>
</table>
Bare-Metal Software Compilation

- Cross-compile C programs for GPC prototype
- Configure RISC-V cross compiler toolchain

Simple Hello-World demo C program

```c
void thread_entry(int cid, int nc) {
    printf("Hello from core %d of %d!\n", cid, nc);
    exit(0);
}
```

Binary File
Further Modifications and Enhancements

- Bootloader modification
  - Rocket cores should boot from own scratchpad memory

- RISC-V Frontend Server (FESVR) modification
  - Load **multiple** binaries to the scratchpad memories
  - Used in RTL simulation
Simulation of the GPC Prototype

- Open-source Verilog and SystemVerilog simulator
- Generates C++ Code to simulate given Verilog RTL model

```plaintext
$ ./simulator-QuadGPCTinyRocketConfig
  → multiload=4,0,0x8000,.\hello
Listening on port 33501
[UART] UART0 is here (stdin/stdout).
Loading hello with offset 0x0 done
Loading hello with offset 0x8000 done
Loading hello with offset 0x10000 done
Loading hello with offset 0x18000 done
Hello from core 1 of 4!
Hello from core 2 of 4!
Hello from core 3 of 4!
Hello from core 0 of 4!
```
Conclusion

- Early Rocket Chip Based GPC prototype at RTL
- Functional test by RTL simulation of GPC prototype
  - Promising basis for further implementation
Future Work

- Implement entire **communication architecture** of GPC
  - Avoid SystemBus access
  - Communication between processing cells
  - Virtual address space per processing cells

- FPGA synthesis of GPC Prototype