An 80-core GRVI Phalanx Overlay on PYNQ-Z1:
Pynq as a **High Productivity** Platform
For FPGA Design and Exploration

Jan Gray
jan@fpga.org
http://fpga.org/grvi-phalanx
My Story

- What I’m working on and how the new Pynq platform helped make it better, faster
GRVI Phalanx: FPGA Accelerator Framework

• A parallel computer overlay to make it easier to develop software-first accelerators

• **GRVI**: 320 LUT 32-bit RISC-V core **0.7 MIPS/LUT**

• **Phalanx**: processor/accelerator/IO fabric
  – **Clusters** of GRVI PEs, accelerators, and SRAM, composed with PCIe, NICs, and DRAM channels on an extreme bandwidth **Hoplite NOC**
  – PGAS with 32 byte/cluster/cycle message passing

• Program e.g. as OpenCL + HLS/RTL accelerators
(US+) Cluster: 8 GRVI PEs, 128 KB CRAM, Accelerators

HOPLITE ROUTER

300

IRAM 4-8 KB

PE

PE

PE

PE

IRAM 4-8 KB

PE

PE

PE

IRAM 4-8 KB

PE

PE

PE

IRAM 4-8 KB

PE

PE

PE

4:4

XBAR

4 x 32 KB UltraRAM = 128 KB

CRAM = CLUSTER DATA RAM

64

NOC ITF

ACCELERATOR(S)
F1 Instances
New Instance Family With Customizable Field Programmable Gate Arrays
Run Your Custom Logic On EC2
Preview Available Today
F1’s FPGA: UltraScale+ XCVU9P

- 900×169 slices = **1.2 M** 6-LUTs
- 180×12 = **2160** 36 Kb BRAMs
- 240×4 = **960** 288 Kb UltraRAMs
- 360×19 = **6840** DSPs
1680 RISC-Vs, 26 MB CRAM on XCVU9P

- 30×7 clusters { 8 GRVI, 128 KB CRAM, 300b router }
- First kilocore RISC-V, and the most 32b RISC cores on a chip in any technology
1, 32, 1680 RISC-Vs
# 1680 Core GRVI Phalanx Statistics

<table>
<thead>
<tr>
<th>Resource</th>
<th>Use</th>
<th>Util. %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical nets</td>
<td>3.2 M</td>
<td>-</td>
</tr>
<tr>
<td>Routable nets</td>
<td>1.8 M</td>
<td>-</td>
</tr>
<tr>
<td>CLB LUTs</td>
<td>795 K</td>
<td>67.2%</td>
</tr>
<tr>
<td>CLB registers</td>
<td>744 K</td>
<td>31.5%</td>
</tr>
<tr>
<td>BRAM</td>
<td>840</td>
<td>38.9%</td>
</tr>
<tr>
<td>URAM</td>
<td>840</td>
<td>87.5%</td>
</tr>
<tr>
<td>DSP</td>
<td>840</td>
<td>12.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frequency</th>
<th>250 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak MIPS</td>
<td>420 GIPS</td>
</tr>
<tr>
<td>CRAM Bandwidth</td>
<td>2.5 TB/s</td>
</tr>
<tr>
<td>NOC Bisection BW</td>
<td>900 Gb/s</td>
</tr>
<tr>
<td>Power (INA226)</td>
<td>31-40 W</td>
</tr>
<tr>
<td>Power/Core</td>
<td>18-24 mW/core</td>
</tr>
<tr>
<td>MAX VCU118 Temp</td>
<td>44C</td>
</tr>
</tbody>
</table>

**Vivado**  
2016.4 / ES1

**Max RAM use**  
~32 GB

**Flat build time**  
11 hours

**Tools bugs**  
0

>1000 BRAMs + 6000 DSPs available for accelerators
Cool. But:

• Not yet speaking AXI4 or AXI4-Stream
• Prerequisite for AWS F1 Shell Integration
  – 512b AXI4 host interfaces
  – 512b AXI4 DRAM interfaces
Recent Work (Better with Pynq)

• Bridge Phalanx to F1 Shell and its AXI4 DRAM itfs
  – Add AXI4/AXI-S master and slave interfaces
  – Prototype early host-side software stack

• Old method: test new bridges on VCU118, exercise with C/C++ tests

• Better way: Pynq/Zynq first!
  – Small device, hard DRAM controllers, fast PAR
  – Pynq interactive testing, rapid iteration, prototyping
  – Same AXI4 signaling and infrastructure IP as F1 VU9P
GRVI Phalanx on Zynq (at the moment)
Enabling AXI4 Stream, MM

• AXI4-Stream interfaces ⇔ Phalanx messages on Hoplite
  – TDATA[255:0] TUSER[31:0] TID etc.
• MsgBufs bridge (AXI4-L) IP core
  – Python PS ⇔ AXI4L ⇔ MsgBufs ⇔ AXI-S ⇔ Phalanx
  – nKx32 BRAM + 2 MMIO CSRs
• Pynq experience
  – Overlay .download() (SAMBA) (faster-than-JTAG download)
  – MMIO .write() and .read() to write message bytes / send message / receive message / read message back
  – Interactivity → play → shorter time to insight/idea
Phalanx ↔ AXI4-MM Bridge ↔ DRAM

- Phalanx-AXI4-MM Bridge is an AXI4 DMA controller
  - 32 B write or $32n$ B read request messages from Phalanx
  - Burst writes/reads on AXI4-MM ↔ PS HP[0:3] ↔ DRAM
  - Send $n$ 32 B read response msgs to Phalanx dest(s)

- **Pynq experience**
  - XInk().cma_alloc().read().write -- easy CMA DRAM
  - Example IPython test: format, send request messages via MsgBufs to MM bridge → see (.read() or via buffers) CMA DRAM writes; watch it all with Vivado AXI4 ILA
Leveraging Linux and the Python Ecosystem

• “Can I host a RISC-V GCC toolchain on Pynq?”
  $ git clone http://github.com/riscv/riscv-tools; ...; make  # !
  $ riscv32-unknown-elf-g++ ...  # yes!

• “How to load *app.elf* sections, send into Phalanx?”
  – Load .elf: “There’s a library for that. Several actually.”
    
    ```python
    from elftools.elf.elffile import ELFFile
    def load_sections(file):
        with open(file, 'rb') as f:
            elf = ELFFile(f)
            iram0_bytes = elf.get_section_by_name('.iram0').data()
            iram0 = [i for i, in Struct('<L').iter_unpack(iram0_bytes)]
            data_bytes = elf.get_section_by_name('.data').data()
            data = [i for i, in Struct('<L').iter_unpack(data_bytes)]
            return (iram0, data)
    ```

  – Send: 10 lines of Python + MMIO .write()s

• 1 hour to prototype – saved days of reinvention and bugs

• Edit; make; load; run parallel app, all on Zynq; < 1 second
FCCM 2017 PYNQ-Z1 Demo: Parallel Streaming DRAM Readback Test: $80 \times 2^{28} \times 256$ B
Summary of Some Pynq/Zynq Aha!s

- git clone http://github.com/riscv/riscv-tools; make!
- git version control of IP and tests *on the FPGA*!
- Python ecosystem – pyelftools!
- Samba!
- MMIO, Xlnk, and interactive Python test scripting!
  - Try more things, explore more corners, more quickly
  - Composition of Python tests and Vivado ILA
  - Will use for “interactive” hardware documentation too!
- IP Integrator ecology! Digilent rgb2dvi
  = faster design iterations / insights = faster progress
Pynq “1.0” Impressions

- Nice OOB: well documented, getting started guide, plenty of examples, “it just works”, questions do get answered on the email support list
- (I’ve only scratched the surface, haven’t used the MicroBlaze channels etc.)
- Clever adoption of massive Python community ways and assets, and Zynq Linux strengths
- Enables fast/easy dissemination of new FPGA work
- Live, interactive hardware documentation is mind blowing
- A promising teaching/maker platform, but also for rapid FPGA design/prototyping/training/appnotes in industry
- Makes exploring new FPGA ideas lightweight, fresh, fast, easy, *fun again*
- So I am trying new (to me) Xilinx technologies and features