18-643 Lecture 4: FPGAs with Purpose

James C. Hoe
Department of ECE
Carnegie Mellon University
Housekeeping

• Your goal today: appreciate modern “FPGAs” as heterogenous and purposefully architected

• Notices
  – Handout #4: Lab 1, due noon, 9/25, noon
  – Ultra96 pick up in HH-1301 btw 10~12 and 2~4.
  – Recitation starts this week, W 6:00~7:00

• Readings (see lecture schedule online)
  – Skim [Chromczak20] and [Ahmed16]
  – Skim [Caulfield16]
Differing Tradeoffs and Sweetspots

**Versatility**

- Efficiency
  - “good” per “cost”

**Ease**

- ASIC
- CGRA/GPU
- FPGA

**Versatility**

- committed:
  - data type
  - operations
  - exploitable parallelism

18-643-F23-L04-S3, James C. Hoe, CMU/ECE/CALCM, ©2023
All Systems, All Heterogenous

Wrong to think ASIC “most” efficient!!
FPGA’s Differentiated Sweetspot

- Spatial data and compute

- Highly concurrent

- Finely controllable

- Wire-cycle granularity actions

- Reprogrammable
2010: Xilinx Zynq SoC FPGA
Die Area “Return on Investment”

Soft-logic logic dominates die area, but compute/storage concentrated in DSP and BRAM—consider what if 100% soft or 100% hard
Xilinx Zynq SoC FPGA

Zynq SoC-FPGA Designer Mindset

Vivado IP Integrator Screenshot
HW/SW Co-Design

• An application is partitioned for mapping to
  – HW: everything SW is not good enough for
  – SW: everything else
• SW is the heart and soul
  – in control of HW
  – enables product differentiation
• SW can be harder than HW (Is this surprising?)
  – embodying most of the complexity
  – often dominate actual development time/effort
IP-Based Design

• Complexity wall
  – designer productivity grows slower than Moore’s Law on logic capacity
  – diminishing return on scaling design team size
  \(\Rightarrow\) must stop designing individual gates

• Decompose design as a connection of IPs
  – each IP fits in a manageable design complexity
    
    Bonus, IPs can be reused across projects
    
    ———— — abstraction boundary ————
  
  – IP integration fits in a manageable design complexity
Systematic Interconnect

• More IPs, more elaborate IPs ⇒ intractable to design wires at bit- and cycle-granularity

• On-chip interconnect standards (e.g. AXI) with \textit{address-mapped} abstraction
  – each \textbf{target} IPs assigned an \textbf{address} range
  – \textbf{initiator} IPs issue \textbf{read} (or \textbf{write}) transactions to pull (or push) data from (or to) addressed target IP
  – physical realization abstracted from IPs

• Plug-and-play integration of interface-compatible IPs

• Network-on-chip ("route data not wires")
AXI Abstraction Unmasked

[Fig 3-2, Zynq-7000 All Programmable SoC Technical Reference Manual]
PS/PL Data Crossing Options

programmable logic (PL)  
processing system (PS)

When to do what?  
See Appendix . . .

[Fig 3-2, Zynq-7000 All Programmable SoC Technical Reference Manual]
Explicit HW-SW Application Co-Design

Two-step process
• design SoC datapath
• program SoC behavior

Vivado IP Integrator

Xilinx Software Development Kit (SDK)
Vitis Software-Defined SoC

```c
int main(int argc, char* argv[]) {
    ...
    cl::Program program(context, devices, bins);
    ...
    cl::Buffer buffer_a(context, CL_MEM_READ_ONLY, size_in_bytes);
    ...
    q.enqueueMigrateMemObjects({buffer_a, buffer_b}, 0);
    ...
    q.enqueueTask(krnl_matrix_mult);
    ...
    q.enqueueMigrateMemObjects({buffer_result}, CL_MIGRATE_MEM_OBJECT_HOST);
    ...
}
```

The result will be correct, but will it be good?
2015: FPGAs in Datacenters
MSR Catapult Bing Experiment

[Putnam et al., 2014]

• “Small” scale test (1632 servers) to accelerate Bing ranking using FPGAs
  – fit in 10% server cost and power budget
  – algorithm updates in interval of weeks
  – datacenter Reliability/Availability/Serviceability

Key Result: 2x throughput at 95th percentile latency

• Takeaway
  – existential proof of datacenter application
  – modern FPGAs large/capable enough
  – Microsoft desperate enough to pivot from SW-only
In every Microsoft datacenter server  
[Caulfield, et al., 2016]

- Individually as SmartNIC (en/decrypt, virtualization)
- Individually as CPU off-load accelerator
- Collectively as a FPGA super-accelerator
  - operate separately from host
  - microseconds any FPGA to any FPGA

“bump-in-the-wire”
Role-and-Shell

- Fixed “shell”: base NIC fxn & infrastructure wrapper
- Reloadable “roles”: network acceleration, local and remote CPU offload, FPGA accelerator plane

1st-gen Stratix V Catapult

overhead? 24% unused??
Overlay Programming (think μcode)

- ML programmers
  - don’t have time to design hardware
  - won’t wait 24-hrs to try a new algo
- HW designers bad at ML

Pay doubly interpretation overhead, okay?

sequential control

N instructions
T iterations
RxC-element tile
E replicas

spatial SIMD datapath
2020: Diverging FPGA Architectures
What is FPGA architecture?

• If you asked in 2015

One is Xilinx, the other Intel. Which is which?
Today’s FPGAs not RTL targets

[Xilinx Zynq]

[Intel Agilex]

[Achronix Speedster]

[Xilinx Versal]
Architecture follows Purpose

• FPGA vendors doing what markets want
  – future “FPGA” not sea-of-gates for RTL netlist
  – FPGAs wanted not because can’t afford ASICs

• Purposeful architectures for targeted use/app
  – make select things easier/cheaper to do
  – be very good at what it is intended to do

• Coping with architectural divergence
  – soft-logic adds malleability to “architecture”
  – 2.5/3D integration allows specialization off a
    common denominator
  – push reconvergence of abstraction up the stack
Xilinx Versal Hardened NoC

Usage as AXI remains abstracted and automated

ISFPGA 2019: “Network-on-Chip Programmable Platform in Versal™ ACAP Architecture”
Xilinx Versal AI Engines

If not RTL then what?
Why CGRAs now?
What is being traded off?

von Neuman

program/ sequencing mem

ALU

multicore, manycore, GPU, etc

CPU-like:
- coarse operator
- programmed sequencing

FPGA-like:
- fine operators
- logic netlist
  (no sequencing)

Spatial

LUT
LUT
LUT
Domain Specialized Programming Support

Deep Learning Frameworks
- mxnet
- TensorFlow
- Caffe

Xilinx ML Compiler

- Xilinx 16nm UltraScale+
- Xilinx Everest w/ Software PE

HotChips 2018, “HW/SW Programmable Engine”
The Achronix Integrated 2D NoC Enables High Bandwidth Designs

[achronix.com]

Figure 1 – Placement of Instances Using the Achronix 2D NoC in an AC7t1500

Figure 2 – Placement of Instances Using Soft 2D NoC
Stratix-10 NX with AI Tensor Block

AI Tensor Block High-Level Diagram

[Intel Stratix-10 NX FPGA, Technical Brief]
up to 143 INT8 TOPS at ~1 TOPS/W
From Humble Beginnings . . . .

**FIGURE 4:** The world’s first FPGA, the XC2064, was implemented on Seiko’s 2.5-μm CMOS process. It featured 85,000 transistors forming 64 CLBs and 58 I/O blocks. This 1,000-ASIC-gate equivalent initially ran at a whopping 18 MHz.

40 Years of Moore and More than Moore

<table>
<thead>
<tr>
<th></th>
<th>VP2802</th>
<th>VP1902</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Adaptable Engines</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Logic Cells (K)</td>
<td>7,326</td>
<td>18,507</td>
</tr>
<tr>
<td>LUTs</td>
<td>3,349,120</td>
<td>8,460,288</td>
</tr>
<tr>
<td>NoC Master / NoC Slave Ports</td>
<td>100</td>
<td>192</td>
</tr>
<tr>
<td>Super Logic Regions (SLRs)</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Distributed RAM (Mb)</td>
<td>102</td>
<td>258</td>
</tr>
<tr>
<td>Block RAM (Mb)</td>
<td>174</td>
<td>239</td>
</tr>
<tr>
<td>UltraRAM (Mb)</td>
<td>717</td>
<td>619</td>
</tr>
<tr>
<td>Multiport RAM (Mb)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Total PL Memory (Mb)</td>
<td>994</td>
<td>1116</td>
</tr>
<tr>
<td>DDR Memory Controllers</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>DDR Bus Width</td>
<td>256</td>
<td>896</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSP Engines</td>
<td>14,304</td>
<td>6,864</td>
</tr>
<tr>
<td>AI Engines Tiles</td>
<td>472</td>
<td>-</td>
</tr>
<tr>
<td>AI Engine Data Memory (Mb)</td>
<td>118</td>
<td>-</td>
</tr>
<tr>
<td>APU</td>
<td>dual ARM Cortex A72</td>
<td></td>
</tr>
<tr>
<td>RPU</td>
<td>dual ARM Cortex R5F</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>256KB ECC</td>
<td></td>
</tr>
<tr>
<td>Connectivity</td>
<td>Ethernet/CAN/USB/...</td>
<td></td>
</tr>
<tr>
<td><strong>Serial Transceivers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTY Transceivers (32.75Gb/s)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GTYP Transceivers (32.75Gb/s)</td>
<td>28(1)</td>
<td>128</td>
</tr>
<tr>
<td>GTM Transceivers (58G (112G))</td>
<td>140 (70)</td>
<td>32 (16)</td>
</tr>
<tr>
<td>PCIe® w/DMA &amp; CCIX (CPM4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PCIe w/DMA &amp; CCIX (CPM5)</td>
<td>2 x Gen5x8</td>
<td>-</td>
</tr>
<tr>
<td>GTCP Express</td>
<td>2 x Gen5x4</td>
<td>16 x Gen5x4</td>
</tr>
<tr>
<td>100G Multirate Ethernet MAC</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>600G Ethernet MAC</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>600G Interlaken</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>400G High-Speed Crypto Engine</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

18-643-F23-L04-S35, James C. Hoe, CMU/ECE/CALCM, ©2023
Parting Thoughts

• SoC’ness complements FPGA’ness
  – hardware performance that is flexible
  – fast design turnaround (time-to-market)
  – low NRE investments
  – in-the-field update/upgrades

• FPGA “architecture” evolving rapidly
  – heterogeneity+cheap transistors --> perf/Watt
  – high-valued application leads to specialization
  – different high-valued applications lead to “speciation”

Don’t let what you see today limit your imagination
Looking Ahead

• Lab 1 (wk3/4): first design with Vitis and DFX
  – most important: know what is there
• Lab 2 (wk5/6): try out HLS
  – most important: decide if you like it
• Lab 3 (wk7/8): hands-on with acceleration
  – most important: have confidence it can work
• Project: we already started . . .
Appendix
(Ask TA in recitation)
Concept: Bus and Transactions

• All devices in system connected by a “bus”
  – initiators: devices who initiate transactions
  – targets: devices who respond to transactions
• Transaction based on a memory-like paradigm
  – “address”, “data”, “reading vs. writing”
  – initiator issues read/write transaction to an address
  – each target is assigned an address range to respond in a “memory-like” way, i.e., returning read-data or accepting write-data

AXI is the standard interface in Zynq
Concept: Split-Phase Bus Transactions

• Asynchronous request/response queues
  – multiple outstanding transactions in flight
  – in-order or out-of-order (need tags)
• No centralized arbitration; push request when not full
• No broadcast; only addressed target sees transaction
Concept: Memory Mapped I/O

- Think of normal ld/st as how processor “communicates” with memory
  - ld/st address identifies a specific memory location
  - ld/st data conveys information
- Can communicate with devices the same way
  - assign an address to register of external device
  - ld/st from the “mmap” address means reading/writing the register
  - BUT remember, it is not memory,
    - additional side-effects
    - not idempotent
Fabric Module as AXI target

- ARM core issues ld/st instructions to addresses corresponding to "mmapped" AXI device registers aka programmed I/O or PIO

- Nothing is simpler
- Very slow (latency and bandwidth)
- Very high overhead
  - ARM core blocks until ld response returns
  - many 10s of cycles

best for infrequent, simple manipulation of control/status registers
Fabric Module as AXI Initiator

1. Fabric can also issue mmap read/write as initiator

2. AXI HP
   - dedicated 64-bit DRAM read/write interfaces
     fastest paths to DRAM (latency and bandwidth)
   - no cache coherence
     • if data shared, ARM core must flush cache before handing off
     • major performance hiccup from (1) flush operation and (2) cold-cache restart

best for fabric-only data, DRAM-only data, or very coarse-grained sharing of large data blocks
3. “Accelerator Coherence Port”

- fabric issues memory read/write requests through ARM cores’ cache coherence domain
- shortest latency on cache hits
  - ARM core could even help by prefetching
  - if not careful, ARM cores and fabric could also interfere through cache pollution
- not necessarily best bandwidth (only one port)

best for fine-grained data sharing between ARM cores and fabric
DMA Controller

- AXI-target programming interface
  - programmable from ARM core and fabric
  - source and dest regions given as <base, size>
  - source and dest could be memory (cache coherent) or mmapped regions (e.g., ARM core scratch-pad or mmapped accelerator interface)
- Need to move large blocks to “amortize” DMA setup costs (PIO writes)
- Corollary: need to start moving well ahead of use

best for predictable, large block exchanges