Housekeeping

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

• Notices
  – Handout #3: course project, status rpt “due” Friday
  – Handout #4: Lab 1, due noon, 9/27, noon
  – Ultra96 ready for pick up
  – Recitation starts this week, Wed 4:40~6:00

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

Efficiency ("good" per "cost")

committed:
- data type
- operations
- exploitable parallelism

ASIC

CGRA/GPU

FPGA

Ease

Versatility

18-643-F21-L04-S3, James C. Hoe, CMU/ECE/CALCM, ©2021
What is FPGA?

- Spatial data and compute
- Highly concurrent
- Finely controllable
- Wire-cycle granularity actions
- Reprogrammable

not CPU
not multicore
not GPU
no software of any kind
not ASIC
2010: Xilinx Zynq SoC FPGA
Xilinx Zynq SoC FPGA

Zynq SoC-FPGA Designer Mindset
IP-Based Design

• Complexity wall
  – designer productivity grows slower than Moore’s Law on logic capacity
  – diminishing return on scaling design team size
    \[\Rightarrow \text{must stop designing individual gates}\]
• Decompose design as a connection of IPs
  – each IP fits in a manageable design complexity
    \[\text{Bonus, IPs can be reused across projects}\]
    \[\text{--- abstraction boundary ---}\]
  – IP integration fits in a manageable design complexity
Systematic Interconnect

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

• On-chip interconnect standards (e.g. AMBA) with address-mapped abstraction
  – each target IPs assigned an address range
  – initiator IPs issue read (or 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
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
AXI Abstraction Unmasked

programmable logic (PL)
processing system (PS)

AXI system “bus”

to off-chip DRAM

[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]
HW-SW Application Co-Design

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

Vivado IP Integrator

Xilinx Software Development Kit (SDK)
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
Role-and-Shell

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

<table>
<thead>
<tr>
<th>Role</th>
<th>ALMs</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>40G MAC/PHY (TOR)</td>
<td>9785</td>
<td>313</td>
</tr>
<tr>
<td>40G MAC/PHY (NIC)</td>
<td>13122</td>
<td>313</td>
</tr>
<tr>
<td>Network Bridge / Bypass</td>
<td>4685</td>
<td>313</td>
</tr>
<tr>
<td>DDR3 Memory Controller</td>
<td>13225</td>
<td>200</td>
</tr>
<tr>
<td>Elastic Router</td>
<td>3449</td>
<td>156</td>
</tr>
<tr>
<td>LTL Protocol Engine</td>
<td>11839</td>
<td>156</td>
</tr>
<tr>
<td>LTL Packet Switch</td>
<td>4815</td>
<td>-</td>
</tr>
<tr>
<td>PCIe DMA Engine</td>
<td>6817</td>
<td>250</td>
</tr>
<tr>
<td>Other</td>
<td>8273</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total Area Used</strong></td>
<td>131350</td>
<td>(76%)</td>
</tr>
<tr>
<td><strong>Total Area Available</strong></td>
<td>172600</td>
<td>-</td>
</tr>
</tbody>
</table>

1st-gen Stratix V Catapult
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

N instructions
T iterations
RxC-element tile
E replicas

sequential control

spatial SIMD datapath

Pay doubly interpretation overhead, okay?
2020: Diverging FPGA Architectures
What is FPGA architecture?

• If you asked 5 years ago

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

[Xilinx Zynq]

[Intel Agilex]

[Achronix Speedster MLP]

[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 base
  - push reconvergence of abstraction up the stack
Usage as AXI remains abstracted and automated
Xilinx Versal AI Engines

If not RTL then what?

HotChips 2018, “HW/SW Programmable Engine”
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”
Stratix-10 NX with AI Tensor Block

AI Tensor Block High-Level Diagram

[Intel Stratix-10 NX FPGA, Technical Brief]
From Humble Beginnings . . . .

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
  - 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
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