18-643 Recitation 2: SoC as IPs and Lab 1

Shashank Obla
PhD Candidate, Department of ECE
Carnegie Mellon University
What’s today about?

• Your goal today: appreciate the power of IP-based design abstraction to go beyond the fabric

• Agenda:
  – Software as an IP in the Xilinx SDx
  – Vitis – what your hardware looks like
  – Starting to think about Optimizing Hardware
  – Bandwidth/Throughput and Latency
  – Discuss your questions about Lab 1

• This is not a lecture! You need to be interactive, lots of discussions, questions and answers
It’s still a stack!

A multi-level computer

<table>
<thead>
<tr>
<th>Level 0</th>
<th>Digital logic level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>Microarchitecture level</td>
</tr>
<tr>
<td></td>
<td>Hardware</td>
</tr>
<tr>
<td>Level 2</td>
<td>Instruction set architecture level</td>
</tr>
<tr>
<td></td>
<td>Interpretation (microprogram)</td>
</tr>
<tr>
<td>Level 3</td>
<td>Operating system machine level</td>
</tr>
<tr>
<td></td>
<td>Partial interpretation (operating system)</td>
</tr>
<tr>
<td>Level 4</td>
<td>Assembly language level</td>
</tr>
<tr>
<td></td>
<td>Translation (assembler)</td>
</tr>
<tr>
<td>Level 5</td>
<td>Problem-oriented language level</td>
</tr>
<tr>
<td></td>
<td>Translation (compiler)</td>
</tr>
</tbody>
</table>

FPGA Stack

<table>
<thead>
<tr>
<th>Level 0</th>
<th>RTL level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>IP composition level</td>
</tr>
<tr>
<td></td>
<td>IP Generation</td>
</tr>
<tr>
<td>Level 2</td>
<td>Overlay level (optional)</td>
</tr>
<tr>
<td></td>
<td>Overlay Design</td>
</tr>
<tr>
<td>Level 3</td>
<td>High-level synthesis level</td>
</tr>
<tr>
<td></td>
<td>Translation (HLS)</td>
</tr>
<tr>
<td>Level 4</td>
<td>Hardware-Software co-design level</td>
</tr>
<tr>
<td></td>
<td>Translation (API)</td>
</tr>
<tr>
<td>Level 5</td>
<td>Domain-specific language level</td>
</tr>
<tr>
<td></td>
<td>Translation (compiler)</td>
</tr>
</tbody>
</table>
IP-Based Design

• Complexity wall
  – designer productivity grows slower than Moore’s Law on logic capacity
  – diminishing return on scaling design team size

⇒ 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 Interfaces, Transactions and Connections
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: 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 need not be memory,
    • additional side-effects
    • not idempotent

0xffff0000

FIFO
Concept: Streaming Protocols

• Think about packets in a network or even FIFO interfaces/connections
  – No memory location to read/write from
  – Cannot “request” data, only accept incoming data
• Low-Latency High Throughput communication
  – Preferred protocol in on-chip networks
  – Variable length transfers marked by packet end
• Each channel in the AXI4 Protocol as a stream
Concept: Clock Crossings

• When do you need it?
  – Multiple devices/IO have their own clocks
  – Different modules running at different frequencies

• Clock Domain Crossing Logic for handshakes
  – Minimal extra resources needed
  – Adds latency to transactions

• Dual-Clock Async. FIFO
  – Built in hardware
  – High Performance at the cost of more resources
Smart/Qsys Interconnect

- Automating connecting multiple initiators/targets
- Handle domain crossings and optimizations
  - data width, clock/reset, pending tx, even protocols
- Which protocol do you think they use internally?

Xilinx  Intel

Figure 1: System Leveraging AXI SmartConnect IP
LET’S SEE HOW THEY ARE USED IN THE XILINX SOC ENVIRONMENT
Zynq SoC-FPGA Designer Mindset

Vivado IP Integrator Screenshot
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
Fabric Module as AXI Initiator (cont.)

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)
   – Aside: AXI ACE protocol specifically for coherence 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 \(<\text{base}, \text{size}>\)
  - 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 exploiting spatial locality
Xilinx AXI CDMA vs DMA

Figure 1-1: AXI CDMA Block Diagram

Figure 1-2: AXI DMA Block Diagram
STARTING TO THINK ABOUT OPTIMIZING HARDWARE
How would this look in hardware?

for (int row = 0; row < dim; row++)
    for (int k = 0; k < dim; k++)
        for (int col = 0; col < dim; col++)
            C[row][col] += A[row][k] * B[k][col];

• What if you had only one MAC unit?
• Innermost loop might look like:
  1. Read A[row][k]
  2. Read B[k][col] and C[row][col]
  3. Compute and Write C[row][col]
  4. Back to Step 2

Looks all fine, doesn’t it?
Accessing Memory: Hardware vs Software

- Is something missing in the picture?
- Why doesn’t this work in hardware?
- How would you change it?
Working with the HLS compiler

• Working with defaults you get a single port
  – For all the three arrays

• The HLS compiler usually defaults to lowest resource utilization if you don’t say anything
What if you gave it multiple ports?

Not always the right thing though (losing control)
Lab 1: What’s it about?

- After Lab 0 you know you can use Vitis
- Get used to using Vitis and designing on it
  - Understand different parts of the process
  - Learn to negotiate with it
- Learn basics of benchmarking designs
  - Ops/Second in MMM
- Get your hands dirty on an actual example
  - Do you know what the metrics mean?
  - Designing experiments to measure it
  - Validating experiments: measuring what you want
Lab 1 Pitfalls

• How many operations in MMM 4096x4096?
• What is Emulation?
  – Software Emulation
  – Hardware Emulation
• Measuring Time
  – What does OpenCL enqueue do?
• MMM Optimizations?
• Measuring Bandwidth and Latency
  – Need best possible, what is best possible in DRAM? – know some DRAM characteristics
Let’s Review: Bandwidth & Latency

- What is throughput and latency?
- Why is throughput ≠ latency?
- How can we measure these quantities? What the best/worst case for each?
Beware of Optimizations

```c
int val;
for (int row = 0; row < dim; row++)
    for (int col = 0; col < dim; col++)
        val = A[row][col];
```

• If we time the code above, what did we measure?
• Need to prevent optimizations
  – Think about dependencies
• Loop flattening – look in reports
• All optimizations are not bad – don’t want unoptimized logic adding unwanted latency
Measuring Latency

int val = *A;

• If we time the code above, what did we measure?
• Need to amortize overheads
  – But isn’t that bandwidth?
  – Need to force dependencies

for (int i = 0; i < count; i++) {
  val = *A;
  // Need to wait for value A to come in!
}

• All optimizations are not bad – don’t want unoptimized logic adding unwanted latency
Parting Thoughts

• IP-based design automates mundane composition
  – still need to know how to design hardware
  – IPs have plenty of customization options
  – performance-resource tradeoff is yours to decide

• Same applies to any IP even software as an IP

• Keep in mind the pitfalls when doing Lab 1
  – Know what you’re measuring
  – And are you measuring it right?

Questions?