Lecture 16

Fine-Grained Reconfigurable Computing: FPGAs

Joel Emer and Daniel Sanchez

6.888 Parallel and Heterogeneous Computer Architecture
Spring 2013
Field Programmable Gate Arrays (FPGA)
Evolution of FPGA applications

- Logic Replacement
  - Low design cost and effort
  - Low volume applications
  - Often replaced with ASIC as volume increases

- Algorithmic computation
  - Offloads a general purpose processor
  - Used for multiple algorithms
  - ASIC replacement not expected
Benefits of FPGA computation

- Custom operations/data types
- Flexible flow control
- Local state access
- Fine grain parallelism
- Custom communication
- Reduced memory references
- Better power efficiency
- Better area efficiency
QPI-attached FPGA platform

Four Socket QuickAssist Platform Topology

Intel QuickAssist QPI-based FPGA Accelerator Platform (QAP)

Accelerator Hardware Module (AHM)

- Intel® Xeon processor 7000 series
- Altera Stratix IV Module
- Xilinx Virtex 6 Module

FPGA

QuickAssist FPGA

QPI Links

PCIe attached FPGA

M

P

AHM

M

P

P

M

CS

CS

I/O

I/O

Intel® Xeon processor 7000 series
### Reed Solomon Results

WiMAX requirement is to support a throughput of 134Mbps

<table>
<thead>
<tr>
<th></th>
<th>Xilinx</th>
<th>Catapult-C</th>
<th>Bluespec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Equivalent Gate Count</strong></td>
<td>297,409</td>
<td>596,730</td>
<td>267,741</td>
</tr>
<tr>
<td><strong>Frequency (MHz)</strong></td>
<td>145.3</td>
<td>91.2</td>
<td>108.5</td>
</tr>
<tr>
<td><strong>Steady State (Cycles/Block)</strong></td>
<td>660</td>
<td>2073</td>
<td>276</td>
</tr>
<tr>
<td><strong>Data rate (Mbps)</strong></td>
<td>392.8</td>
<td>89.7</td>
<td>701.3</td>
</tr>
</tbody>
</table>

- Lower is better
- Higher is better

Source: MIT, Abhinav Agarwal, Alfred Ng - CSG
BORPH

- Berkeley Operating system for ReProgrammable Hardware
- OS for reconfigurable computers
  - Treats reconfigurable hardware as computational resources
- UNIX interface to HW designs
  - Familiar to both software and hardware engineers
  - Design language independent
- Goal:

  Make FPGA-based reconfigurable computers easy to use
Conventional View of FPGA Systems

6.888 Spring 2013 - Sanchez and Emer – L16
BORPH Layers

- **User Process (SW)**
- **User Process (SW)**
- **User Process (SW)**

**Hardware Platform (Network, UART, HD...)**

**Device Driver**

**User Library**

- **file**
- **pipe**
- **IPC**
- **ioreg virtual file**
- **socket**

**BORPH Kernel**

**Peer-to-Peer Relationship**

**Hardware User Library**

- **User Process (HW)**
- **User Process (HW)**
Overview of BORPH Concepts

- Hardware process
- Hardware syscall interface
- Interacting with an FPGA
  - ioreg virtual file interface
  - Hardware file I/O
Hardware Process (1)

- An executing instance of a hardware design
  - SW: An executing instance of a program

- Normal UNIX process
  - Has pid, check status with `ps`, `kill`, etc

- Unit of management

- Created when a BORPH Object File (BOF) file is `exec`-ed
  - Kernel selects and configure hardware region automatically
Benefits of UNIX Process Model

- Very easy for user to reason about
- Enable FPGA designs to become active component of the system
  - e.g. an FIR filter:
    - Conventional: a **passive** entity where software sends/receives data
    - BORPH: an **active** entity that pulls/pushes data as needed
- Enable multiple instances of the same FPGA design running in the system
  - No more fixed accelerator concept
  - Works well in true reconfigurable computing systems
HW Processes I/O

- I/O managed by kernel
  - Similar to SW
- Hide details from users
  - e.g. HW-SW, HW-HW UNIX file pipe
- Standard UNIX I/O mechanism
  - File I/O, pipe, signal
- HW specific service
  - ioreg virtual file system

Don’t ask “How do I … in HW”. Think: “What if it were SW?”
ioreg Virtual File System

- Maps user defined hardware constructs as virtual files under the process’s `/proc/<pid>/hw/ioreg/` directory
  - Single word register
  - Memory: On-chip + Off-chip
  - FIFO
- Example:
  - `/proc/123/hw/ioreg/COUNTERVAL`
- ioreg information embedded in the executing BOF file
- `read` and `write` system calls translated to message packet by the *kernel*
- Any UNIX program can communicate with hardware processes
  - Shell: `echo 1 > /proc/123/hw/ioreg/enable`
  - C:
    ```c
    MEM_FILE = fopen("/proc/123/hw/ioreg/MyMemory", "r");
    fread(swbuf, 1, MEM_SIZE, MEM_FILE);
    ```
  - Python, Java, etc...
Hardware File I/O

- Access to the general file system from hardware processes
- Debug by printing
  - printf
- Read test vectors, record output

- SW/HW processes chained by file pipe

```
bash$ receiver.bof < file.in > file.out
```

```
bash$ decode video.in | resize | edgdet.bof | encode > video.out
```

6.888 Spring 2013 - Sanchez and Emer – L16
Latency-Insensitive Design: A Higher Semantic

- Inter-module communication by latency insensitive channels
  - Changing the timing behavior of a module does not affect functional correctness of the program

- Many HW designs use this methodology
  - Improved modularity
  - Simplified design-space exploration

- Implemented with guarded FIFOs in current RTLs
Latency-Insensitive Design: A Higher Semantic

Behavior of LI channels does not affect functional correctness.
Latency-Insensitive Design: A Higher Semantic

- There are many FIFOs in the design
  - It may not be safe to modify some of them
- Compilers see only wires and registers
  - Reasoning about cycle accuracy is difficult

But the programmer knows about the LI property...
A Syntax for LI Design

- Programmer needs to differentiate LI channels from normal FIFOs
- Latency-Insensitive Send/Recv endpoints
  - Implementation chosen by compiler
  - FIFO order
  - Guaranteed delivery
- Explicit programmer contract
  - Unspecified buffering & unspecified latency
  - Programmer responsible for correct annotation

```module mkTimeP;
  Send#(Inst) send <- mkSend("Decode");
endmodule
```

```module mkFuncP;
 Recv#(Inst) recv <- mkRecv("Decode");
endmodule```

Easy to use – often a textual substitution!
Connected User Application

Platform_0
- Platform_0 Comms
  - mkA_Stub
  - mkB_Stub
- mkC

Platform_1
- Platform_1 Comms
- mkA
- mkB
- mkC_Stub

Key:
- User
- Library
- Generated