Beyond PCs: Accelerator Controls on Programmable Logic

Mark Plesko
The Three Lifetime Occupations of Cosylab

• See previous workshops and conferences:
  • A simple, versatile I/O controller
  • Java programming
  • Code generators

• Now, all in one article, all in one chip 😊
Part 1: The “non-plus-ultra” Versatile Controller

- A GSI paper* suggests to make a controller with FPGA instead of CPU
  - Same hardware board addresses different I/O needs
  - Implement hardware with software

- Motivation:
  - Reconfigurable hardware (FPGA) is much more flexible than general purpose CPUs
    - And even faster despite lower clock rates
    - But implementations and development effort is high
  - Vendors offer generic processor cores
    - VHDL code (e.g., Altera NIOS or Xilinx PicoBlaze),
    - Multiple powerful cores are fixed on the chip
      - E.g., Xilinx Virtex II includes PowerPC cores

- Mix CPU and logic functionality as needed

*M.Sayed, W.Panschow, Actual FPGAs – The Way Out Of Manifold Hardware Problems
Generic Controller Hardware Architecture Overview

- FPGA
- Microprocessor
- Cache
- I/O
- RAM (board)
- Flash
- FPGA cfg (copy 1)
- FPGA cfg (copy 2)
- ROM (copy 1)
- ROM (copy 2)
- Filesystem(s)
- Memory access logic
- RAM (FPGA)
- Module A
- Module B
- Module C
- Module D
- Wishbone
- FPGA
- RAM (FPGA)
The Bus *(Wishbone)*

- Open bus (spec maintained at OpenCores)
- Integration of IP cores on a SoC
- Efficient, flexible and easy to work with
- Many interconnection types (crossbar, data flow, point-to-point, shared bus, switched fabric, three-state)

**WISHBONE COMPATIBLE**
Peripherals

• Digital I/O: FPGA pins
  – Need only dedicated transition boards with connectors

• Serial
  – Open UART core is available
  – Wishbone compatible

• GPIB:
  – Commercially available extensions (e.g., National Instruments, via PCI bus)

• System-to-system buses with FPGA:
  – ISA (PC104)
  – VME
  – PCI
  – PCIExpress
Network Communication

• For 10/100Mbit ethernet MAC, an open core exists
  – by Igor Mohor
  – Wishbone compatible
• For Gbit+ ethernet, proprietary cores are available
Application Example 1:
Nanosecond Resolution Timing

- Delaying of signals relative to a trigger
- Sub-nanosecond resolution
- Achievable in FPGA with PLLs (0.5ns, 50ps jitter)
- Processor for control and communication
Application Example 2: Motion Control

- Control of servo, stepper and other kinds of actuators
- Typically implemented with DSPs
  - Difficult to program and configure
- Co-design:
  - SW: motion programs, control, communication
  - HW: PID loops (up to several MHz)
Application Example 3: Network Traffic Manipulation

- Filtering and routing of packets
- **Software:**
  - Control
  - TCP stack
  - Signaling
- **Hardware:**
  - Gigabit MAC
  - Filtering
  - Exotic buffers
Application Example 4: Versatile I/O Controller

- High density, high-performance I/Os
  - Serial, GPIB, USB, firewire, ...
  - Same board in different configurations
  - 10-100 I/Os per FPGA
- Software:
  - Control and configuration
  - Communication (with I/Os and/or SCADA)
- Hardware:
  - UARTs, USB controllers, ...
  - Multiple cores
  - Dedicated DSPs (e.g., fireware camera image processing)
  - Interlocks on digital inputs
Part II: Put Java on the FPGA!

- FPGAs and cores a nice, but I still want Java!

1. Use of standard Java constructs
   - Leverage existing development tools
   - Verifiable byte-code

2. Static checking
   - Code generation (e.g., drivers for custom modules)
   - Register address overlap check/auto-assignment
Additional Requirements

4. Comply to RealTime Java (RTSJ)
   • Released first in January 2002,
   • second release, 1.0.1(b), on May 9th 2006.
   • specification lead: Peter Dibble of TimeSys Corporation
     • a provider of a real-time Linux and a reference real-time Java implementation

5. Support for debugging
   - Serial console
   - Java remote debugging
   - JTAG

6. Field upgrade of hardware and software
FPGA Java Processor Core Exists: JOP - *Java Optimized Processor*

- A processor that executes Java byte-codes
  - Complex byte-codes implemented in micro-code.
  - Deterministic behavior
- Occupies 1.0-1.8k LC (NIOS: 1.8k-2.9k); about 30% of Altera Cyclone EP1C6.
- *Hello World* application:
  - Boot time: < 10 ms (simulation at 100MHz)
  - Footprint: 50kB (includes required runtime)

- BAD
  - A research project
    - Requires validation before using in production
  - Not wishbone compliant (requires a wrapper)
  - Not RTSJ (Realtime Specification for Java) compliant
Processor Core
(Java Optimized Processor)
Hello World on FPGA
Extending the Processor: Hyper-threading

- JOP core component instantiated more than once
- Advantages:
  - Improved processing power
  - Decreased latencies
  - Dedicated core for real-time tasks
- Issues:
  - Multiple masters for memory and I/O buses (arbiter needed)
  - Hardware implementation of Java’s synchronization locks
  - Task scheduler needs to be aware of multiple cores
Hardware Architecture Overview

- Java Optimized Processor
  - RAM (stack)
  - Microcode ROM
  - Cache
  - SimpCon I/O

- RAM (FPGA)
- ROM (copy 1)
- ROM (copy 2)
- Filesystem(s)

- Flash
- FPGA cfg (copy 1)
- FPGA cfg (copy 2)
- ROM (copy 1)
- ROM (copy 2)
- Filesystem(s)

- Memory access logic
- Wishbone
- Module A
- Module B
- Module C
- Module D
- RAM (board)
- FPGA

- Wishbone
- Module A
- Module B
- Module C
- Module D
- RAM (board)
Part III: Generators for Everything (also for new buzzwords?)

- Development on a Java-like platform is **more efficient** than using other tools
  - “Nearly everybody” knows Eclipse
- But need to define hardware in VHDL
- A **hardware-software co-design** approach would ameliorate the disadvantage
Proposed Development Workflow

- Define platform
  - Generate hardware (VHDL)
  - Generate JVM (Java)
- Design board (pin layout)
- Implement software
- Synthesize and route hardware
- Automated tests on workstation
- Prepare and upload flash image

Types:
- Manual
- Automated
Software Architecture Overview

Driver A
Driver B
Driver C

interrupts

Application

Java Virtual Machine (JVM)

Network

File
Sample Java Code

class MySystemOnChip extends SystemOnChip {
    public void defineHardware() {
        this.setDescription("Digital and serial I/O controller");
        wb = new WishboneBus("wb");
        this.addBus(wb);
        ...
        jop0 = new JavaProcessor("jop0");
        jop0.has FloatingPoint(true);
        wb.addMaster(jop0);
        ...
        flash0 = new CompactFlash("flash0");
        flash0.addPartition("fpgacfg0", 512*kB);
        flash0.addPartition("hda0");
        ...
        eth0 = new OpenCoresEthernet("eth0");
        wb.addSlave(eth0);
        ...
    } // defineHardware method
public void init() {
    initDrivers();
    initMemoryManager();
    initFileSystem();
    initNetwork();
    initScheduler();
    initServices();
    run();
}

public void initNetwork() {
    // set the MAC address of the Ethernet adapter
    eth0.setMACVendor(0x00CAFE);
    eth0.setMACSerial((this.getSerial() & 0xFFFF) << 8);

    // initialize the IP protocol (version 4)
    ip = new InternetProtocolV4();
    // tell the Ethernet driver to dispatch events to IP protocol
    eth0.addProtocol(ip);
    ...
}
What Needs to be Developed To Make it Work

- Java remote debugging
- GPIB, UART, PID, memory access logic, field update
- Ethernet, TCP/IP stack
- JOP JVM refactoring, garbage collector
- Platform generator, documentation, testing, developer education
Conclusion

- HW/SW co-design has already been shown to be effective with FPGA cores
  - Development productivity greatly improves (WRT plain VHDL)

- With Eclipse and Java, productivity increase could be even more significant

- But the proposed architecture requires still a significant effort to realize
  - Nice research project