ETH Vorlesung Systembau / Lecture System Construction
>252-0286-00L - Dr. Felix Friedrich, Paul Reed
>
>Case Study: Custom-designed Single-Processor System
>Paul Reed (paulreed@paddedcell.com)
>
- First Lecture: The RISC Architecture
- [Second Lecture: Project Oberon on RISC]

The belief that...
>...complex systems require armies of designers and programmers
>is wrong. A system that is not understood in its entirety, or at
>least to a significant degree of detail by a single individual,
>should probably not be built.
>
- Niklaus Wirth (Feb. 1995), "A Plea for Lean Software", IEEE Computer

Introduction
- RISC single-processor personal computer designed from scratch
- Hardware on field-programmable gate array (FPGA)
- (this lecture) Motivation and goals; RISC CPU
- (next lecture) Graphical workstation OS and compiler (Project Oberon)

Motivation
- "Project Oberon" (1992) by N. Wirth & J. Gutknecht, at ETH Zurich
- building a complete system from scratch is achievable and beneficial
- available commercial systems are far from perfect
- not just a "toy" system: complete and self-hosting
- personally: need good and reliable tools for commercial programming

Case Study Goals
- weigh pros and cons of designing from scratch
- overview of using FPGAs to design custom hardware
- competence in building complete custom system from the ground up
- understanding of "how it really works" from hardware to application
- courage to apply "lean systems" approach wherever appropriate

Discussion: Why Build from Scratch? (1)
- reduce complexity: no "baggage"
- clear design: easy to see where to extend or fix
- increase control, reduce the number of dependencies
- more choices of implementation
- design based solely on problem domain and experience

Discussion: Why Build from Scratch? (2)
- eliminate surprises: deliver on time and on budget
- highly flexible solution
- more of what the customer asked for
- source of competitive advantage
- accept less of what you don't like

Discussion: Why not Build from Scratch?
- duplication of effort: re-inventing the wheel
- more fundamental knowledge required
- may be more actual work (the first time)
- restricted component choices
- not for the short-term

Introduction to Configurable Hardware
- evolution of programmable logic (PALs/GALs, CPLDs)
- look-up tables (LUTs), registers and interconnect
- current field programmable gate array (FPGA) technology
- loadable configuration, not "set in stone" like VLSI / ASIC
- applications from telecommunications to automotive and industrial
- now big enough for entire system-on-chip

Introduction to HDLs
- hardware description language to define circuits formally
- used for both simulation and synthesis
- commercial examples: Verilog, VHDL
- developed at ETH: Lola, Active Cells
- VERY different from conventional programming languages

Hardware Flashing-LED Test
- [handout TestLEDs-Verilog.pdf: "TestLEDs.v"]
- hardware-only solution as a simple example of Verilog
- define module inputs and outputs, registers, and wires
- wiring up: "assign"
- register-transfer: "always @()"
- constraints file (Xilinx .ucf) for pin assignment

Introduction to Niklaus Wirth's RISC Processor
- originally a 32-bit virtual target for "Compiler Construction"
- RISC vs. CISC; registers vs. stack machine
- Harvard vs. Von Neumann memory architecture
- hardware floating-point option
- now defined in Verilog and implemented on FPGA

RISC Architecture Overview
- [handout RISC-Architecture.pdf: "The RISC Architecture"]
- program counter (PC) and instruction register (IR)
- instruction decode logic
- "register file" consisting of 16 general-purpose 32-bit registers
- arithmetic and logic unit; barrel shifter; flags NZCV
- memory interface

The RISC Instruction Set
- arithmetic and logic instructions (reg/reg and reg/immediate)
- load and store register to/from data memory (word and byte)
- conditional branch (-and-link)
- that's it!

RISC0 Implementation on a Xilinx FPGA
- [handout RISC0-Verilog.pdf: "module RISC0Top..."]
- Harvard RISC0 core in Verilog
- on-FPGA ROM for program, on-FPGA RAM for data
- memory-mapped I/O ports
- port examples: timer, LEDs, switches/jumpers, RS232
- Verilog "top" module: outside-world interface
- user constraints file (UCF)

Software Flashing-LED Test
- [handout TestLEDs-Oberon.pdf: "MODULE* TestLEDs"]
- "MODULE*" signifies a standalone module e.g. ROM
- initialisation of stack and global base
- main loop - output to LED port
- nested delay loops
- class exercise set-up / demonstration

Exercise 1: RISC on the OberonStation FPGA Board

Exercise 1a: Tools and Workflow
- [handout OberonStation.pdf: "OberonStation"]
- [handout XilinxSetupRISC0.pdf: "RISC0 Project Setup and Test Instructions"]
- [handout ORC-Compile.pdf: "ORC: The Oberon-07 Command-line Compiler"]
- install Xilinx ISE and Oberon cross-compiler ORC
- create RISC0 project, add Verilog source code (src directory)
- compile TestLEDs.Mod Oberon program, prom.mem to proj dir
- in ISE generate "programming file", ie hardware bitstream
- download to board using programming tool, e.g. iMPACT
- compile TestSwi.Mod example, update prom.mem and regenerate bitstream

Exercise 1b: Develop an Instruction Timer
- use TestLEDs.Mod as template, add variable t
- SYSTEM.GET(-64, t): 32-bit mS tick counter at port -64
- get tick in t at beginning, and into z at end, of outer loop
- run middle loop 100 times, inner loop 10000 times
- display (z - t) DIV 100 on LEDs at end of outer loop
- note mS, then compare after adding a DIV in inner loop
- (optional) calculate exact cycle time for DIV instruction

Exercise 1c (optional): Compare Hardware Implementations
- use above performance meter to measure multiply
- change hardware to use Multiply1.v employing MULT18X18
- (remove Multiplier.v, add Multiplier1.v, edit RISC0.v)
- measure performance of multiply again
- consider pros and cons of both designs

[end of first lecture and exercise]