!252-0286-00L
Project Oberon's primary goal...
>was to design and implement an entire system from scratch,
>and to structure it in such a way that it can be described,
>explained, and understood as a whole.
>
- Niklaus Wirth & Juerg Gutknecht (1992), "Project Oberon", Addison Wesley

ETH Vorlesung Systembau / Lecture System Construction
>Case Study: Custom designed Single-Processor System
>Paul Reed (paulreed@paddedcell.com)
>
- [First Lecture: Custom-designed Single-Processor: the RISC Architecture]
- Second Lecture: Project Oberon on RISC
>
>Review
- Building from scratch can be done, and can be beneficial
- configurable hardware allows co-design of hardware and software
- simple and powerful, open and extendable 32-bit RISC architecture
- exercises: simple access to hardware features, existing or new
- familiarity with FPGA tools: Verilog source to bitstream on-board

Motivation for Project Oberon
- existence proof for building from scratch
- "as simple as possible, not simpler" - Einstein
- strictly no "bells and whistles"
- clear, extensible design documented in one book
- teaching principles, "rather than only knowledge and skills" - Wirth

Oberon System Features
- fast file system, module loader and garbage-collector
- high-level, type-safe language for applications and implementation
- graphical, tiled window interface
- "Module.Command [params]" execution from any visible text
- self-hosted, yet small and efficient (~200K including compiler)
- built (and documented!) during 1986-1988 by two people part-time
- ideal for resource-constrained systems; reliability; security

Oberon User Interface (demo)
- heavily mouse-oriented, three buttons with "interclicking"
- command (middle button), e.g. execute command Edit.Open
- select (right button), e.g. select parameter (^)
- point (left button), e.g. set caret
- vertical user and system tracks for viewer placement
- non-overlapping viewers, temporary overlays
- "tool" viewers for organising commands

Oberon Core Structure
- [handout OberonCore.pdf]
- inner core: memory, files and module loader
- outer core: viewer and task management
- applications: compiler, graphics editor; networking

Oberon Inner Core
- Kernel.Mod: memory allocation and garbage collection, disk sectors
- FileDir.Mod: flat file directory and garbage collection
- Files.Mod: files as byte streams, "riders" for access
- Modules.Mod: (recursively) load object code, execute commands

Oberon Outer Core
- Input.Mod, Display.Mod: drivers for keyboard, mouse and screen
- Viewers.Mod: division of screen into "tracks" and "viewers"
- Fonts.Mod: font file and glyph management
- Texts.Mod: text file manipulation and scanning
- Oberon.Mod: task loop and command execution
- MenuViewers.Mod: viewers with title and menu of commands
- TextFrames.Mod: rendering and editing of text
- System.Mod, Edit.Mod: commands for viewers/system, text editing
- that's it! :)

Applications
- OR[SBGP].Mod: self-hosted Oberon compiler
- recursive-descent, single-pass direct code-generation in 2854 lines
- Graphics.Mod, GraphicFrames.Mod etc.: graphics editor
- RS232.Mod, SCC.Mod: extra device drivers (serial, network)
- Net.Mod, Print*.Mod: optional network and printing system
- (file xfer, mail, instant messaging, time synch, remote printing)

Simple Application (Stars)
- [handout Stars.pdf: "MODULE Stars"]
- bounce stars around a menu viewer
- module body: create background task with handler
- Open: create viewer and install into display hierarchy
- Handle: interpret viewer messages
- Step1: broadcast "Step" message to advance viewer display
- Run and Stop: install and remove background task

"Project Oberon" on RISC
- revival of the original system described in the book
- RISC5 processor replaces defunct NS32032
- complete system on 1MB RAM Spartan-3 board+daughterboard (SD-Card)
- simple 2.4GHz wireless module replaces twisted-pair network
- now on compact custom-designed Spartan-3AN board: OberonStation
- new 2013 edition of the Project Oberon book
- all freely available online (projectoberon.com)

RISC5 Processor, Oberon Memory Map and I/O
- RISC5 adds external static RAM and floating-point to RISC0
- added SPI for SD-card and network, PS/2, GPIO (kept RS232 and 1mS timer)
- extra features only ~600 more lines of Verilog (RISC0 is ~400)
- 256 bytes system area incl. module table [000000H - 0000FFH]
- 512K module block area and stack [000100H - 07FFFFH]
- 425K heap [080000H - 0E7EFFH]
- 96K video framebuffer [0E7F00H - 0FFEFFH]
- 64 bytes memory-mapped I/O [0FFFC0H - 0FFFFFH]

RISC Boot Loader
- loads bootfile (inner core) from SD-Card
- alternate boot via RS232 if jumper/slide-switch set
- written in Oberon and compiled standalone into FPGA bitfile as "ROM"
- bitfile loaded from flash at power-on
- RISC begins loader execution at StartAdr (0FE000H)
- Oberon system starts with jump to 000000H
- also handles ABORT button (warm boot) to interrupt a crashed task

RISC Tools and Emulators
- emulator straightforward: simplicity of instruction set and I/O
- Peter de Wachter's portable RISC emulator (see projectoberon.com)
- (RS232 emulation for file transfer in and out of virtual machine)
- cross-compiler straightforward: simplicity of language and compiler
- projectoberon.com resources built using scripts and cross-compiler

Building RISC Oberon
- [handout OberonStationQR13.pdf: "OberonStation Quick Reference"]
- (S3BOARD only) add custom daughterboard for SD-Card and PS/2 mouse
- download Verilog from projectoberon.com
- compile bootloader to prom.mem
- build RISC5Top bitfile and program board flash
- raw copy RISC.img to microSD card
- add PS/2 keyboard+mouse and VGA monitor, and boot!

Nordic nRF24L01+ 2.4GHz SPI Wireless Network Transceiver
- 1Mb/s wireless to replace low-cost 230kb/s twisted-pair wiring
- nRF24L01 replaces original serial comms controller (SCC)
- simple packet frame format with up to 32 bytes of payload
- straightforward SPI command interface, FIFOs for rx and tx packets
- optional hardware acknowledge and re-transmit of dropped frames
- exercise 2c & 2d: provide network traffic monitor and time updates

Exercise 2: Project Oberon on the OberonStation Board

Exercise 2a: Tools and Workflow (build optional)
- build OberonStation system from projectoberon.com (see README)
- build step 1: build RISC5 bitfile from OStationVerilog.zip
- build step 2: program FPGA and flash to make permanent
- build step 3: raw write RISC.img to SD-card with dd/win32diskimager
- boot system - then Edit.Open Stars.Mod, change interval (e.g. 50mS)
- select MODULE, ORP.Compile @/s and run: Stars.Open, Stars.Run
- (if error: mark source viewer [ctrl-z] and Edit.Locate selected pos)
- System.Free Stars~ to unload when finished or before new version
- when making changes SAVE YOUR WORK frequently!

Exercise 2b: Develop a Screensaver
- Step 1: modify Blink.Mod to IMPORT Input, Display, Viewers
- new global INTEGERs lastX, lastY to detect mouse move in Tick()
- re-employ z to count up on LEDs if no movement (and test it)
- (note MUST run Blink.Stop before unloading module - why?)
- Step 2: when z gets to (say) 10, blank screen (ReplConst)
- (and when z is beyond 10, stop counting at some point!)
- if z is 10 or beyond and movement, restore screen - but how?
- Step 3: use an M: Viewers.ViewerMsg to suspend/restore all viewers

Exercise 2c: Develop a Wireless Network Traffic Monitor
- (requires nRF24L01 SPI module, see "Quick Reference" handout from lecture)
- Step 1: modify Blink.Mod to add the following consts and procedures:
- CONSTs spiData = -48; spiCtrl = -44; CHANNEL = 5;
- SPI(n: INTEGER) writes n to spiData and waits for spiCtrl bit0=1
- (hint: ... REPEAT UNTIL SYSTEM.BIT(spiCtrl, 0))
- BeginCmd(cmd: INTEGER) sets spiCtrl to 02H (select network) then SPI(cmd)
- EndCmd() sets spiCtrl to 00H; EnableRadio() sets spiCtrl to 08H
- WriteReg(reg, val: INTEGER) BeginCmd(20H + reg MOD 32); SPI(val); EndCmd
- Setup() writes reg 0=7FH, 1&4=0, 5=CHANNEL, 6=07H, EnableRadio
- Step 2: modify Tick() to add local INTEGERs status and carrier
- BeginCmd(9); get status; SPI(0); get carrier; EndCmd
- i.e. SYSTEM.GET(spiData, status); SPI(0); SYSTEM.GET(spiData, carrier)
- display carrier * 80H + status DIV 2 * 2 + z on LEDs
- in Run, and at end of Tick, call Setup to reset receiver for next tick

Exercise 2d: (optional) Develop a Network Time Client
- Step 1: add proc. WordMode() to set ctrl to 06H;
- in Setup() also write reg 7=70H, 11H=20H, and
- BeginCmd(0E2H); EndCmd; all before enabling radio
- Step 2: in Tick after displaying status but before Setup():
- if ODD(status DIV 40H) then BeginCmd 61H, set WordMode
- SPI(0), get spiData into new variable hdr;
- SPI(0), get spiData into new variable len;
- SPI(0), get spiData into new variable time; EndCmd
- Step 3: if hdr DIV 1000000H = 47H and len=4, then Oberon.SetClock(time)
- import Display and ReplConst(1, 900, 10, 100, 100, 2) when time updated
- (or ReplConst(1, 1013, 744, 4, 4, 2) is more subtle)
- Step 4: run and check time is updated with System.Date

[end of second lecture and exercise]

Present.Show
Present.ShowPage 5
Present.ShowPage 18