NTNU | Norwegian University of Science and Technology

Porting the xv6 OS to the Nezha D1 RISC-V Board

Michael Engel

Department of Computer Science

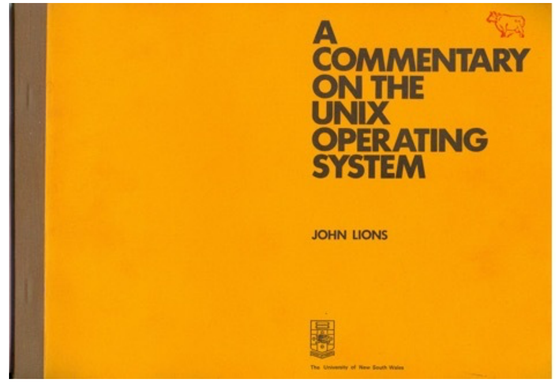
NTNU

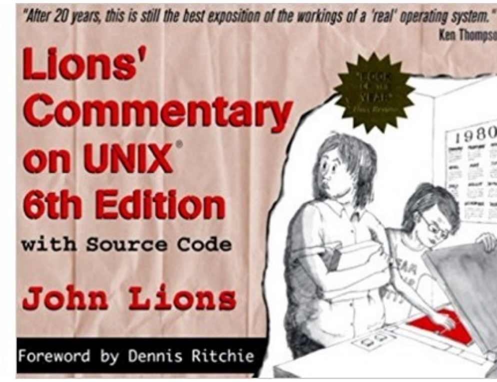
https://multicores.org

Motivation

- Provide a basis for OS teaching and research
  - Small and easy to understand for a single student
  - Useful in emulation (qemu) as well as on real hardware
  - Sufficiently complex to demonstrate important CPU features
    - Protection modes, virtual memory, interrupt handling, system calls, …
- Enable students to do quantitative analysis on real HW
- Show students the challenges of running bare-metal OS code on a real hardware system vs. an emulator
- Additional benefit:
  - Providing simple example code for other OS porting projects

The xv6 OS





- Small teaching OS
- Developed since summer 2006 for MIT's OS course
- Inspiration: 6th Edition PDP11 Unix (1975)
  - Used by Prof. John Lions at UNSW (Australia) to teach OS engineering
  - Lions' book ("commentary") on the 6th edition kernel source code [1]

- Problems of using 6th Edition/Lions' book:
  - In 1975: book violated AT&T's copyrights
    - only distributed to Unix licensees
    - Finally published in 1996
  - Today: (almost) nobody owns a PDP11…

xv6 Overview

- Kernel: written in C + some assembler
  - Monolithic kernel
  - ~5500 lines of C, 330 lines assembler
  - Multicore support
  - Subset of typical Unix system calls
  - No concept of users/permissions
- User land: a few typical Unix utilities
  - Support for ELF format binaries
  - init, sh, ls, grep, ln, rm, wc, cat, echo
  - Very minimal libc implementation

- xv6 is intentionally minimal to enable students to extend the system functionality

System call
int fork()
int exit(int status)
int wait(int *status)
int kill(int pid)
int getpid()
int sleep(int n)
int exec(char *file, char *argv[])
char *sbrk(int n)
int open(char *file, int flags)
int write(int fd, char *buf, int n)
int read(int fd, char *buf, int n)
int close(int fd)
int dup(int fd)
int pipe(int p[])
int chdir(char *dir)
int mkdir(char *dir)
int mknod(char *file, int, int)
int fstat(int fd, struct stat *st)
int stat(char *file, struct stat *st)
int link(char *file1, char *file2)
int unlink(char *file)

Status of xv6

- RV64 version stable and used in many courses
- x86 version working, but no longer maintained
- The xv6 companion book [2] gives many details on the structure and implementation as well as on RISC-V
  - Read it together with the RISC-V specs and RISC-V Reader

- Officially supported platforms:
  - x86 (32 bit) in qemu emulator and on real hardware
  - RISC-V (64 bit) in qemu
- Unofficial ports: [3]
  - Raspberry Pi 1/2 (32 bit ArmV7 BCM2835/2837 SoC)
  - RISC-V 32 bit platform in qemu
  - Kendryte K210 RISC-V SoC

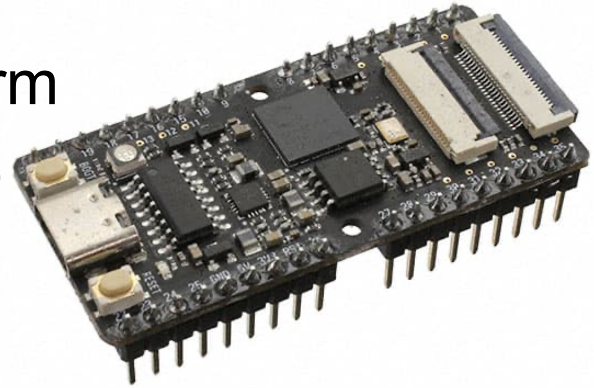
xv6 port to real hardware

Kendryte K210 port

- K210 SoC: Dual Core RV64GC
- 8 MB on-chip SRAM

Advantages

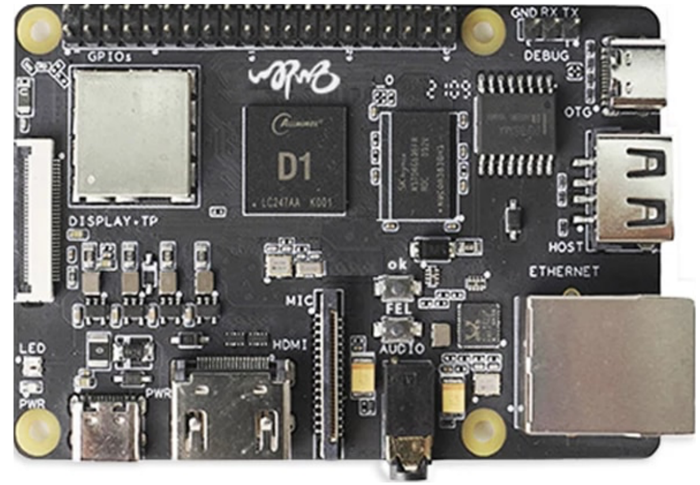
- Widely available, small embedded platform
- Many typical peripherals (i2c, spi, uart…)
- Inexpensive boards available (from $15)



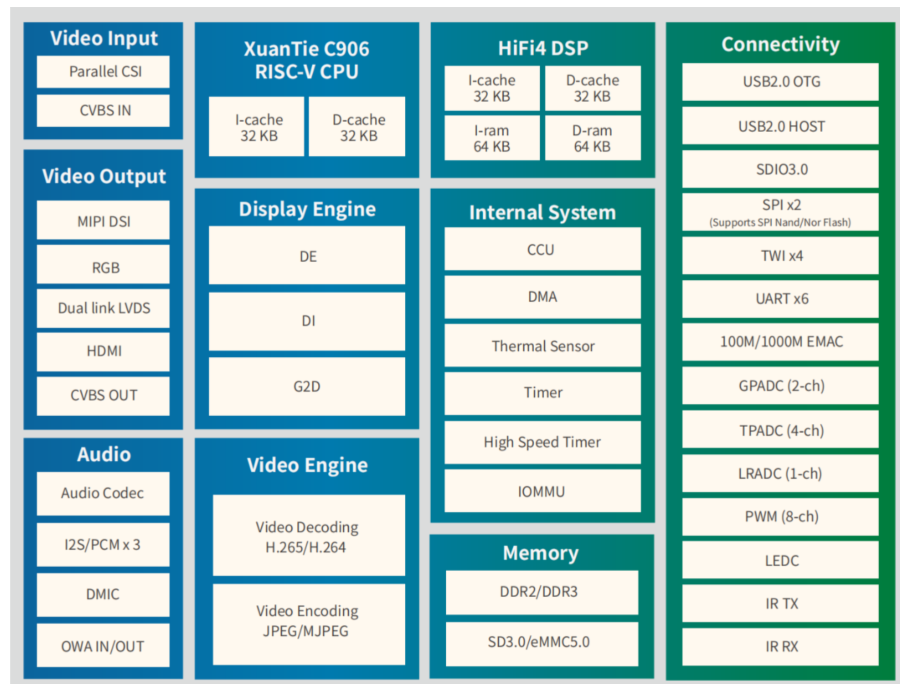
Problems and limitations

- Outdated hardware – privileged spec 1.9.1 (2016)
  - e.g. different MMU configuration
- K210 documentation is lacking many details
- xv6 port still has some stability problems

Nezha/D1 hardware



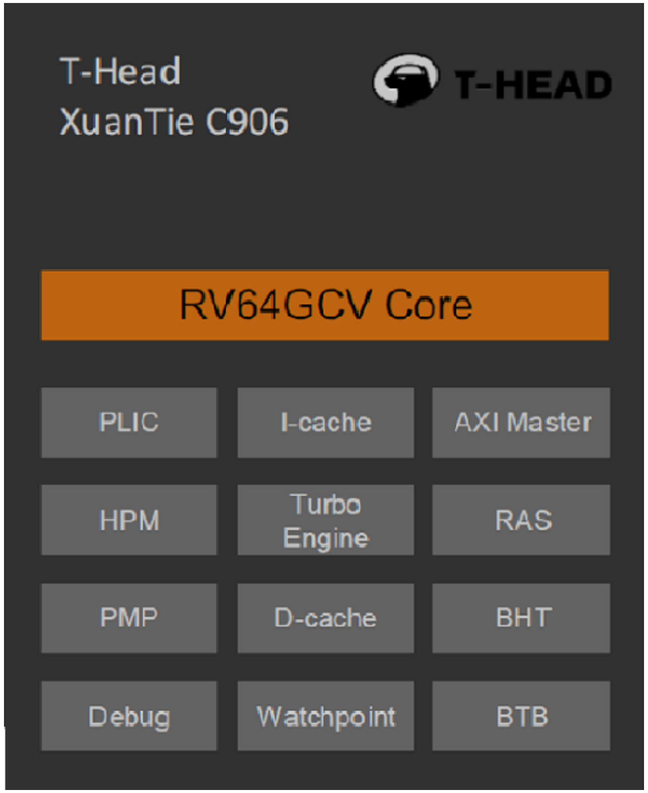
- Hardware
  - Raspberry Pi form factor
  - Allwinner D1 SoC @ 1 GHz
  - 0.5 GB, 1 GB or 2 GB DDR3 RAM
  - 256 MB NAND Flash
  - Separate Wifi+Bluetooth chip – XRadioTech XR829
- D1 SoC [4]
  - Single Core RV64GCV
  - HiFi4 DSP
  - Display/video engine
  - Numerous peripherals
    (many similar to Allwinner ARM SoCs)



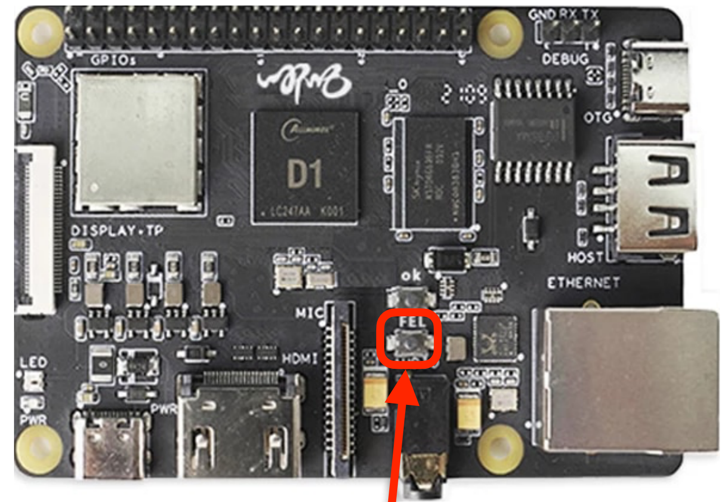
Nezha/D1 CPU

- T-Head XuanTie C906 CPU core [4]

Feature	Description
Architecture	RV64GCV
Pipeline	5-stages
Xuantie extension	Xuantie Instruction Extension(XIE) Xuantie Memory Attribute Extension(XMAE)
Float-point unit	Support RISCV Half、F、D instruction extension Support IEEE 754-2008 standard
Vecotr unit	Support RISCV V instruction extension(configurable) vectory register width 128bit element size support 8/16/32/64bit support INT8/INT16/INT64/BFP16/FP16/FP32/FP64
Bus interface	AXI4-128 master
Instruction cache	up to 64KB(configurable)
Data cache	up to 64KB (configurable)
Interrupt controller	Flexibly configurable Platform Level Interrupt Controller(PLIC)
Memory management unit	Sv39 virtual memory translation
PMP	Up to 16 regions
Debug	RISCV debug



Nezha/D1 software



- Standard system boot
  - OpenSBI firmware in M-mode
  - U-Boot in S-mode
    - Tip: stop boot with "S" key!
  - Linux
    - from NAND flash (TinaLinux) or SD card
- Alternative: bare metal boot via USB-C
  - Press FEL button at power-on and use xfel tool
  - Bare metal code examples help getting started [5]

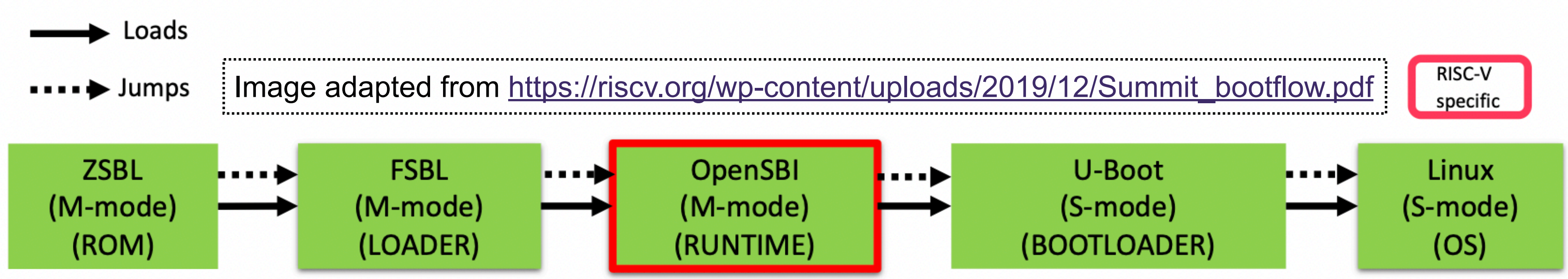


Image adapted from https://riscv.org/wp-content/uploads/2019/12/Summit_bootflow.pdf

xv6 port to Nezha/D1 [6]

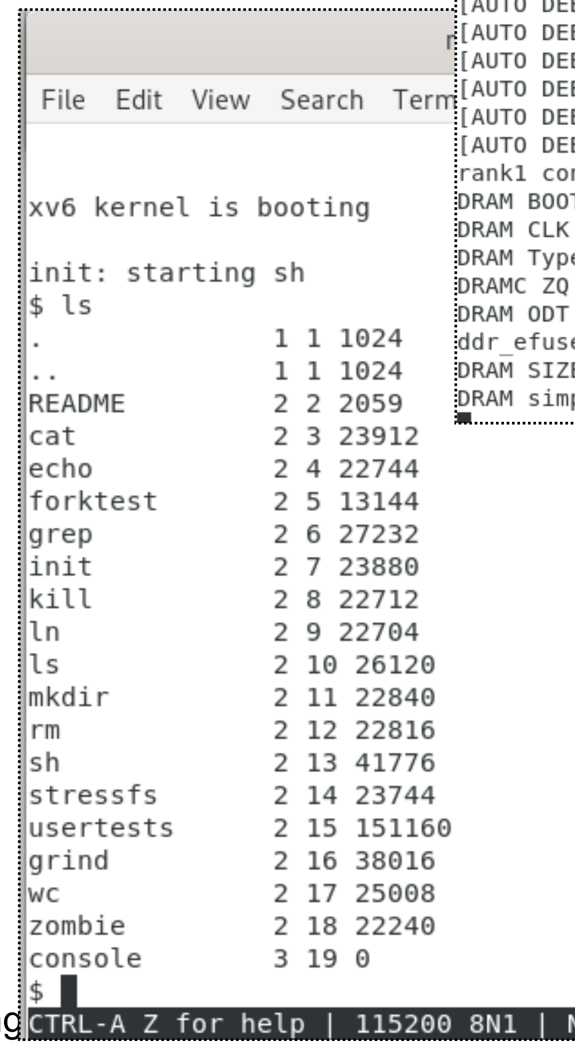
- Booting via OpenSBI + U-Boot
  - OS kernel is started in Supervisor mode
  - xv6 was built to start in Machine mode
    - e.g. timer interrupt handling relies on this
  - xv6 port to K210 uses OpenSBI + U-Boot
    - could also be adapted for the Nezha
    - problem: OpenSBI for Nezha not well documented
- Booting bare metal – this is what is implemented
  - Use FEL boot loader and xfel tool
  - Disadvantage: kernel has to *initialize all hardware*
    - DDR RAM timing calibration
    - Clocks and PLLs

xv6 development flow

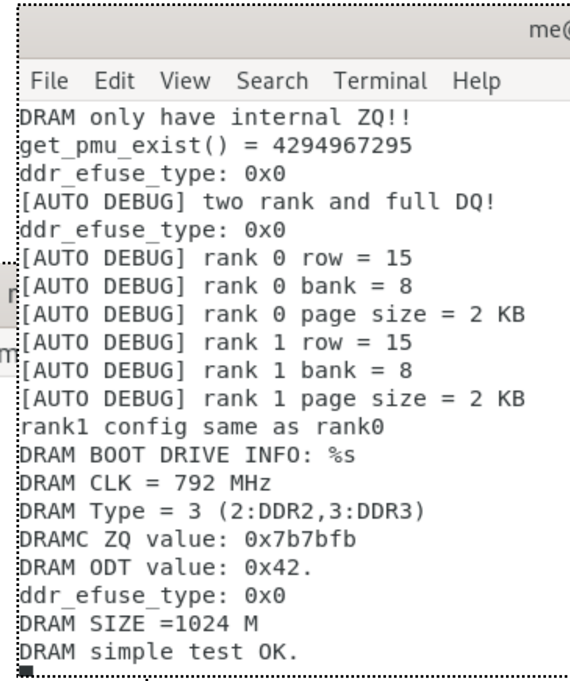
Use xfel to

- init DDR3 timing
  `xfel ddr ddr3`
- load the xv6 kernel to RAM
  `xfel write 0x40000000 \`
  `kernel.bin`
- start the xv6 kernel
  `xfel exec 0x40000000`
- Connect to serial console
  - screen, minicom, …
- …find bugs, fix them, compile new kernel, start again… 😀

xv6 is running!



DRAM init output



Challenges of porting to the D1

- Clock/PLL init – adapted from bare metal examples for now
- DRAM init – currently using the DDR3 init code from xfel
- PMP – xv6 patch was required to configure physical memory protection
  - This was not emulated in qemu until very recently!
  - Effect: kernel hangs when switching to S-mode – hard to debug
- MMU configuration
  - C906 MMU requires A (access) and D (dirty) bits set for PT entries
  - Otherwise, system will hang after enabling VM by writing to satp CSR
- Interrupt handling: very different from the interrupt model qemu emulates
- Peripheral/Device driver complexity much higher than qemu's virtio emulation
- Debugging – printf for now, JTAG would be nice!
- CPU data sheet
  - SoC data book: very comprehensive, English
  - C906 core data book only in Chinese…

Current status of the xv6 port

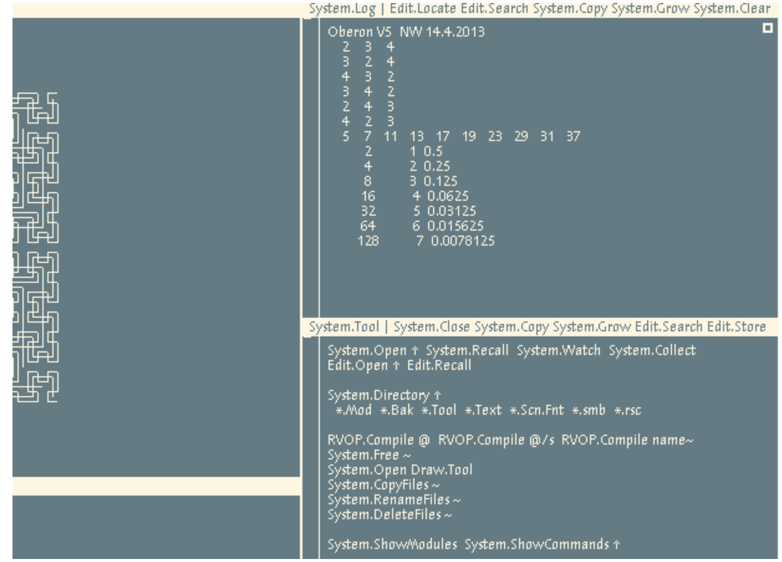
- xv6 compiles (cross-compilation possible on Linux, macOS, Windows 10 WSL), boots to a shell and runs stable
  - uptime tested > 24 hours
- Supported features:
  - Clock/PLL init (from bare metal examples)
  - SV39 MMU
  - CLINT/PLIC interrupt controllers
  - 16550-compatible UART console
  - RAM disk containing the root file system
- Unsupported:
  - Everything else 🙂 – i.e., most of the peripherals
  - xv6 uses neither the Xuantie instruction extensions nor memory attribute extensions

Use in education

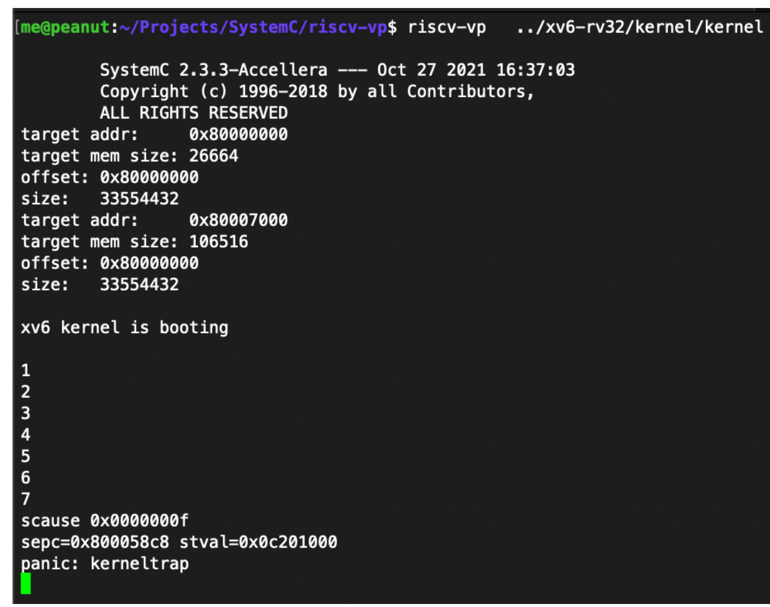
- Course in *OS engineering:*
  - Learn about the internals of an OS kernel
  - Interaction of hardware and software
    - Interrupts, Virtual memory management, DMA, …
  - Implement new OS features for xv6
    - Recreate some defining features of Unix evolution as well as some new ideas from research papers
- *Low-level programming:*
  - Assembler and C bare-metal programming for RISC-V
  - Linux device driver development
  - Nezha was not available in time for the course
    - uses Raspberry Pi 4's for now
- *Computer architecture:*
  - Explore performance evaluation, cache effects, power/energy…

Work in progress

- More drivers
  - SD card
  - future work: Ethernet, video, USB
- Better debugging facilities
  - JTAG/openocd
- Porting additional resource-aware operating systems
  - Project Oberon [7], Plan 9, Inferno
- Explore hardware/software co-design
  - Open source Verilog code for C906/C910 cores available [8]
  - SystemC models for generic RISC-V CPUs [9]
  - useful e.g. to explore new approaches for virtual memory management



Project Oberon on RV32



xv6 on SystemC RISC-V model

References

1. John Lions, *Lions' Commentary on UNIX 6th Edition*,
   Peer to Peer Communications, ISBN: 1-57398-013-7; 1st edition (June 14, 2000).
   Online version of Lions' Commentary: http://www.lemis.com/grog/Documentation/Lions/
   Online version of the 6th Edition Unix source code: http://v6.cuzuco.com/
2. Russ Cox, Frans Kaashoek, Robert Morris, *xv6: a simple, Unix-like teaching operating system*
   First RISC-V version: https://github.com/mit-pdos/xv6-riscv-book
   Book LaTeX source code: https://github.com/mit-pdos/xv6-riscv-book
3. *xv6 ports*:
   Raspberry Pi 1: https://github.com/zhiyihuang/xv6_rpi_port
   Raspberry Pi 2: https://github.com/zhiyihuang/xv6_rpi_port
   Kendryte K210: https://github.com/HUST-OS/xv6-k210
   RISC-V RV32: https://github.com/michaelengel/xv6-rv32
4. *D1 documentation*: https://linux-sunxi.org/D1
5. Nezha D1 *bare metal examples*: https://github.com/bigmagic123/d1-nezha-baremeta
6. *xv6 port to the Nezha/D1*: https://github.com/michaelengel/xv6-d1
7. *Project Oberon port to RISC-V*: https://github.com/solbjorg/oberon-riscv
8. T-Head processor *core source code* (C906, C910, E902, E906): https://github.com/T-head-Semi
   C910 processor core paper: https://ieeexplore.ieee.org/document/9138983
9. *SystemC RV32/RV64 models*:
   http://www.informatik.uni-bremen.de/agra/systemc-verification/riscv-vp.html
   Paper: http://www.informatik.uni-bremen.de/agra/doc/konf/2018FDL_RISCV_VP.pdf