Plan 9 on 64-bit RISC-V †

Geoff Collyer
geoff@collyer.net

ABSTRACT

We have ported Plan 9 to several RISC-V 'Unix-capable' (RV64GCSU) implementations: the Microchip™ Polarfire Icicle™, *tinyemu* emulator, a pre-release Beagle V™, StarFive™ VisionFive™ 2, SiFive™ HiFive Unmatched™, HiFive Premier P550, and Milk-V Jupiter™. Other systems were considered or tried, but rejected, usually for inadequate documentation.

This paper describes the porting process and makes recommendations. Work continues.

1. Position Statement

The RISC-V architecture is elegant; I don't have any serious criticism of it, including at least the standard IMAFD extensions for Unix-capable systems. (These are base Integer instructions, Multiplication and division, Atomic memory operations, and single- and double-precision floating point. IMAFD is also written as 'G'. 'C' indicates ability to execute the compressed instructions.) *However*, proposed and approved extensions beyond IMAFD, and other additions, are often flawed or downright rubbish.

> *People admire complexity.*
>
> — Rob Pike

> *Beauty is more important in computing than anywhere else in technology because software is so complicated. Beauty is the ultimate defence against complexity.*
>
> — David Gelernter

Much of this is hardware and software adopted unthinkingly from PCs and ARM devices, regardless of technical merit, probably to re-use existing designs and IP and Linux code. Several large corporations (e.g., Intel, Alibaba) seem unable to comprehend the 'R' in 'RISC' (i.e., keep it simple, stupid), but visible complexity is not actually required for good performance, even if that's easier for the hardware designers. *Unnecessary visible complexity is a failure of design.* I don't really expect a company with an instruction set with over 800 instructions (over 2,000 by some reckoning),

† This is a modified version of this paper [Col2023]

hundreds of MSRs, and a 2,680-page Ethernet controller manual to understand this. We'll revisit this in *Recommendations and Observations*.

1.1. Demented Standards

> *How do you tell a bad standard? If it begins with ''I'', e.g, I2O, IPMI. If it ends with ''I'', e.g., ACPI, EFI, IPMI, etc. If it has the word ''intelligent'' in it, e.g., I2O, IPMI. Or, the best, if it has all three, e.g., IPMI.*
>
> — Ron Minnich

'Device trees', ACPI, (U)EFI and GUIDs are problems, not solutions, and we try to avoid them at all costs. (Why use meaningful names when you can use long, meaningless strings of hex digits?) The RISC-V Platform Specification subcommittee is just flat-out wrong to adopt virtually every mistake ever made on the IBM PC or ARM. The RISC-V Platform Specification is disappointing; RISC-V presented an opportunity to rethink and replace this string of disasters, most of which provide very little benefit.

1.2. Wretched Hardware

> *Throw out the hardware, let's do it right.*
>
> — Steely Dan, *Aja*

Life is too short to deal with SD and MMC cards, GPIOs, PHYs, I2C, SPI and other single-bit interfaces to guaranteed-model-specific hardware, and this crud shouldn't be necessary; hardware should be usable immediately after coming out of reset. If the BIOS or *U-boot* initialize devices sufficiently to use them, that's good enough.

2. Background

In July 2020, the Microchip Polarfire Icicle board [Mic] was due to be the first available RISC-V [Pat2017, Int] system that looked capable of running a Unix-like operating system, including paging hardware, a gigabyte of RAM (which turns out to be actually 2GB in 2 banks), and gigabit Ethernet, [Xil] with multiple CPUs (also called cores and RISC-V *harts*) capable of 64-bit and (in theory) 32-bit operation. It has one SiFive E51 RV64IMA core that lacks supervisor mode (a 'hobbled' hart) and starts the other four, which are SiFive U54 RV64GC cores at 600 MHz. The board contains no graphics hardware. We assume conformity to a minimum Privileged ISA Specification of 1.10.

Richard Miller had a 32-bit RISC-V Plan 9 [Pik1990] C compiler suite [Mil2020] already, and was willing to create a 64-bit compiler suite. Without this, I would not have started this porting effort.

I originally planned to port the Plan 9 *9k* kernel, which already ran on 64-bit `amd64` systems and could exploit a large address space, to RV32GC mode on the Icicle while Richard worked on the RV64 C compiler, then use the RV32GC port as a basis for an RV64GC port when the RV64 compiler was ready. *9k* implements the same system calls and a subset of the devices that *9* implements. The major device drivers are now identical in my *9* and *9k*.

2.1. Timeline

The hardware was due to arrive in mid-September 2020, which eventually slipped to mid-October. In the meantime, we developed using the *tinyemu* emulator, which emulates both RV32 and RV64 architectures on any processor architecture, though only a single emulated processor. *Tinyemu* supplies no SBI and starts the kernel in machine mode; all the other implementations have an SBI and start the kernel in supervisor mode (except the RVB-ICE, which appears to start it in machine mode). Richard ported *tinyemu* to Plan 9 and added a serial port and *virtio* Ethernet. I made small changes to it to improve debugging capabilities and it has proven helpful in finding bugs where the hardware's response has been to just sit there dumbly. Perhaps the SBI could accept requests on a UART to dump a hart's registers.

When the hardware arrived, we had a *9k* kernel running on *tinyemu*, but we then discovered some things that would require changes. *Tinyemu* starts our kernel in RISC-V 'machine' mode, in which paging is disabled, but all machine facilities may be configured. The lack of paging encourages starting RAM at `0x80000000` or higher, to match Unix kernel conventions. By early December, we had a 64-bit kernel running on one CPU of the Icicle board using 39-bit virtual addresses ('Sv39'). By late December, all U54 CPUs were scheduling processes. A few C compiler fixes arrived through mid-January. After that, various mysterious misbehaviour disappeared. By early February, graceful reboot was working and by mid-February, paging with 48-bit virtual addresses ('Sv48') was working. The system seems to be complete and solid enough to use as a CPU server, and should be relatively easy to adapt to future RV64G systems.

Since then, we have made minor fixes, code and performance improvements, and adapted to various RISC-V systems, including improving SoC (system-on-chip) configurability. In particular, self-checking code has been added to verify sanity in various conditions, and to attempt to tolerate the unexpected.

2.2. RISC-V Peculiarities

Memory alignment requirements are stricter than most people are used to: natural alignment for scalars up to and including `vlong`†. Otherwise, we get alignment exceptions. The 64-bit compiler promotes most scalars to `long` when pushing them as function arguments, only `vlong`s, `double`s, pointers, and some `struct`s are wider. However, there can be gaps on the stack, e.g., when pushing an `int` then a pointer.

On most systems (except really cheap or broken ones), all CPUs, memory caches and DMA accesses are coherent, which is a delight. The RISC-V specifications encourage this, but it is nevertheless unusual, surprising and noteworthy for RISC designs.

---

† On Plan 9, `vlong` is `long long`, which is always 64 bits.

2.3. Hardware and Firmware

A note on terminology: the *CLINT* is the per-CPU simple interrupt controller; the *PLIC* is the system-wide more-complex interrupt controller. The PLIC feeds into the CLINTs as the external interrupt signal. The *SBI*[Int2022] is a sort of BIOS, but unlike a PC BIOS, it cannot be circumvented.

On the hardware, the boot ROM/flash starts (typically) OpenSBI which then starts *U-boot*, which starts our kernel in 'supervisor' mode, from which there is no escape, with additional undocumented restrictions:

- read-only (or no) access to the CLINT's timer registers;
- have to use SBI calls to set the CLINT timer (and maybe send and clear IPIs);
- SBI v0.2 HSM (hart state management) calls are not implemented in the provided Icicle OpenSBI;
- *U-boot* on the Icicle only starts all CPUs (*hart*s in RISC-V terminology) if one uses the `bootm` command with an FDT to run a *uImage* claiming to be a Linux kernel.

A result is that we can't switch the Icicle into RV32GC mode with the stock boot 'ROM', though it is possible in machine mode. So we abandoned the 32-bit port since it can't run on the available hardware, though it still ran in *tinyemu*, as the current wave of Unix-capable systems are all RV64GC, as will be any Unix-capable systems with more than 2GB of RAM.

There are conflicting accounts of the details of how RISC-V harts are started, particularly at what PC. *U-boot* on Icicle starts them all at once at the entry point in the *uImage* file, while the other systems' *U-boot* starts only hart 1.

2.3.1. Polarfire Icicle

There are other bits of ill-documented hardware:

- there's an L2 cache which adheres to RISC-V cache coherence principles, so can be largely ignored;
- the PLIC context ids apparently have consecutive values starting at 0: E51 hart 0 M (machine) mode, U54 hart 1 M, hart 1 S (supervisor) mode, hart 2 M, hart 2 S, hart 3 M, hart 3 S, hart 4 M, and hart 4 S. These should be predictable or discoverable without consulting a 'device tree'.

2.3.2. SiFive U740

On SiFive-U740-based systems, use of the `WFI` instruction, instead of `PAUSE`, to save power when idling originally produced strange and varied behaviour: console serial output got stuck, or time gradually stopped advancing, or the system became very busy, possibly servicing interrupts. Changes to the system since the first publication of this paper appear to have cured whatever the problem was.

2.3.2.1. Beagle V

The now-cancelled Beagle V has 8 GB of RAM, and an L2 cache that is *not* coherent with DMA, thus requiring manual cache flushing, unlike proper RISC-V systems. (This was claimed to be a bug that would be fixed in production hardware, the JH7110 SoC.) It also has a newer OpenSBI implementation that provides the HSM operations.

2.3.2.2. HiFive Unmatched

OpenSBI's *sbi_get_hart_status* appears to often report the wrong hart as the sole started hart. This was true with 2021 firmware and still with 2025 firmware.

2.3.2.3. StarFive VisionFive 2

The successor to the Beagle V, incorporating the JH7110 SoC, is running. The Synopsys™ DWMAC Ethernet controller is version 5.20, which is newer than the Beagle V's version 3.7, and incompatible. Many clock signals had to be enabled and components taken out of reset via CRGs (system control registers) in order to communicate with the DWMAC at all.

2.3.3. HiFive Premier P550

My HiFive Premier P550 has 16 GB of RAM and 4 1.4 GHz Sifive U84 cores. It is based on the Eswin EIC7700X SoC. The U84 is an out-of-order CPU capable of issuing 3 instructions per cycle. Like the Beagle V, the P550's DMA is incoherent with its caches, which is a shame in an otherwise nicely made board. The caches are nominally all coherent with each other. I've seen some behaviour that suggests that that may not be strictly true, at least without adding fences. There are private L1 and L2 caches and a shared L3 cache. It has the fastest RISC-V CPUs I've seen yet, but elapsed time for kernel builds, for example, is higher than I think it should be.

2.3.4. Milk-V Jupiter

My Jupiter has 8 GB of RAM and 8 1.6 GHz Spacemit X60 cores. The caches are nominally all coherent with each other. Like the Beagle V, the Jupiter's DMA is incoherent with its caches, which is a shame. To allow manually compensating, it includes the *Zicbom* and *Svpbmt* extensions.

The Ethernet controller is Spacemit's own design, the K1-X. Configuring its clock signals took more effort than it should have: Spacemit's documentation and the Linux driver weren't entirely consistent.

2.3.5. XuanTie™/T-Head RVB-ICE

This uses the XuanTie C910 CPU, which was claimed to be quite fast. After enabling paging, something goes off the rails. Linux runs on this hardware, so presumably there's some extremely obscure magic needed, despite T-Head's claim of RISC-V compatibility. English documentation not generated by Google Translate is now available, but there seems to be no hope of getting this machine to run Plan 9. Given the bugs in the C910, notably those found by RISCVuzz [Tho2024] and others with configuring PMP, this is no great loss.

3. Plan 9 Changes

These are largely confined to the architecture-dependent source directories.

3.1. Removed Assumption of Memory at Address Zero

The original *9k* assumed that RAM started at physical address 0, and it took some trial-and-error to find and repair the myriad dependencies, notably in initial memory discovery and allocation.

3.2. No Virtual Page Table

The technique of the 'virtual page table' [MIT] (VPT), which injects the page table into itself as a top-level PTE, is used in the `386` and `amd64` ports, but appears to be inapplicable to RISC-V. Lifting a level 1 PTE into the root (level 2) PTE would vastly increase the address space that it covers, since size is implied by level. So some page table updates had to be made explicit and do their own allocations, which is clearer anyway (the existing VPT code is obscure).

3.3. Variable Page Sizes and Page Table Levels

The system implements *Sv39*, *Sv48*, *Sv57*, and *Sv64* paging, where available. Of the supported systems, so far all support *Sv39* but *tinyemu* and the SiFive Premier P550 implement *Sv48* too. *Sv57* and *Sv64* are untested to date, but are straight-forward extensions from *Sv48*.

3.4. SoC Configuration

Configuration for a new SoC requires editing the `conf` sections of kernel configuration files, which now include descriptions, in C, of the SoC's devices, and fundamental addresses needed early or in `mkfile` are specified in the `/sys/src/9k/rv` directory, in the file *arch*`/defs`, where *arch* is a short name for the subarchitecture (e.g., `te` for *tinyemu*). The appendix contains an example of the 64-bit *tinyemu* configuration. See `tecpu` and `pfcpu` for complete examples.

3.5. Starting CPUs During Bootstrap

On x86 systems, a single CPU starts at bootstrap, and it then starts the others. RISC-V systems may start CPUs (*hart*s) at any time. The Icicle starts them all at once when *U-boot*'s `bootm` command starts the kernel, which is necessary because its SBI lacks the HSM commands that would otherwise be needed. The other systems start a single CPU (or at least try to), which uses the SBI HSM calls to start the others. The start-up code now copes with those possibilities, and the situation of having just been restarted via `/dev/reboot`. `bootm` works better than `go` in general. With `go`, interrupts on some systems seem to not work.

3.6. A C Idiom

In a C expression such as in the following, using Plan 9 types:

```
uvlong uvl, va;
uvl &= ~((1<<5) - 1);       /* zero low 5 bits */
uvl = va & ~((1U<<12) - 1);/* get pure page number */
```

the result will probably not be what was intended. The ~ operator will have an `int` or `uint` operand, yielding a result of the same type, 32 bits wide. This result will be widened for the & or &= operator, but it may be zero-extended, thus ensuring that the result in `uvl` will have zeroes in its upper 32 bits. In particular, 64-bit physical addresses of RAM on RISC-V were being truncated. *6c* and now *jc* detect this inadvertent zero-extension in the `uint` case.

Ensuring that the operand (and thus result type) of ~ is `vlong` or `uvlong` avoids this problem. We have made this change throughout *9k*.

4. Performance

These are times to build the Plan 9 `rv` kernel from scratch mostly on RV64GC systems with 1Gb/s Ethernet using the same 10Gb/s Ethernet file server, except as noted. These were all effectively diskless, as is normal for Plan 9 systems. To load caches before measuring, these commands were executed:

```
mk clean; mk; mk clean; time mk >/dev/null
```

and yielded the results in this table:

user	sys	real	C	Iss.	GHz	description
1.07u	1.73s	2.04r	4	7?	3.8	amd64 Xeon, 10GbE
1.71u	1.04s	2.07r	4	6.2?	3.1	386 nuc5i7 †
2.81u	2.22s	7.40r	4	3	1.4	hifive premier p550, ipis ‡
6.00u	3.11s	10.31r	8	2	1.6	milk-v jupiter, no ipis ‡
6.47u	3.29s	8.93r	4	2	1.2	hifive unmatched, no ipis
7.36u	3.65s	10.12r	4	2	1.25	visionfive 2, ipis 0 ns.
6.65u	3.75s	7.66r	4	2	1.25	visionfive 2, no ipis
3.43u	4.15s	7.24r	4	~6.8	1.5	arm raspberry pi 4 HZ=200 †
6.90u	4.21s	7.72r	4	2	1.2	hifive unmatched, ipis 0 ns.
7.43u	4.30s	6.84r	4	2	1.25	visionfive 2, ipis >2?s.
6.74u	5.64s	11.37r	8	2	1.6	milk-v jupiter, ipis ‡
9.44u	5.81s	7.79r	4	2	1.4	amd64 amd apu2
14.60u	7.09s	14.47r	4	1	0.6	Icicle, no ipis
14.99u	9.11s	50.91r	2	2	1	pre-release beagle v ‡ *
10.14u	13.63s	19.10r	2	2	1	arm cortex-a7 trimslice † ‡
38.52u	20.44s	64.98r	1	1.5	0.68	mips 24k routerboard, no fp †
144.87u	47.99s	242.52r	1	7?	3.5	tinyemu on 386 Xeon HZ=200 †

C is the number of cores, Iss. is the number of instructions issued per cycle per core, GHz is the CPU speed in gigahertz.

See this earlier paper [Col2010] for comparison with older Plan 9 systems of various architectures.

---

* using a different, 1Gb/s file server

‡ handicapped by incoherent DMA

† 32-bit Plan 9

5. Recommendations and Observations

Microchip's documentation seems to be unclear if it's intended for someone repackaging the hardware or for the ultimate end user. It often specifies that some value is programmable but doesn't provide the choice of value used in the Icicle. It would be helpful to have end user documentation.

The RISC-V architecture tries to leave some things unspecified to allow implementations some leeway, requiring that platform documentation provide the actual values implemented, but the platform makers don't always do so. Concern for RISC-V implementors should be balanced with concern for users; vagueness is rarely useful to system programmers. It would be more helpful to be able to query such values programmatically without consulting a 'device tree'.

All the timers provided require *a priori* knowledge of their frequencies. To let software determine the actual frequencies, it would be very helpful to have a real-time clock that ticks at a known, fixed rate (e.g., 100 times per second) or a register containing the (fixed) CLINT timer frequency. As it stands, the frequencies have to be supplied to software.

Detecting and reporting infinitely-recursive traps (perhaps in SBI) would be quite helpful during development, for example, if the `STVEC` CSR (Control and Status Register) contains a no-longer valid virtual address. We have modified *tinyemu* to do this.

Requiring all RISC-V systems (or at least Unix-capable ones) to have an 8250-compatible console UART at a common, fixed physical address and a common frequency would help with porting. SiFive has its own non-8250-compatible UART.

Micro-USB connectors need to be braced very firmly; a slight tug on an attached cable should not yank the connector off the board. The Unmatched board is quite flimsy. Its SD card slot isn't much better.

The Icicle's power cable is fragile and prone to interrupting power when flexed.

Suppliers need to implement both PXE booting and the `saveenv` command in their *U-boot* variants from the very start. These are important capabilities for kernel developers and must not be pushed off into the future. The Icicle at least has working PXE booting on one Ethernet, but no `saveenv` command, so automatic booting of Plan 9 kernels at reset won't work. The otherwise-promising Sipeed Nezha board's *U-boot* lacks PXE booting entirely, which makes it too much of a hassle to be worthwhile.

5.1. Assessment of RISC-V

In general, RISC-V seems to be a pleasant architecture with a few minor infelicities. (Implementing graceful reboot on the Icicle was a challenge.) Some additions and extensions add the sort of unnecessary and clumsy complexity that has made X86 the dog's breakfast that it is (e.g., the XuanTie C910). The XuanTie processors seem to have reintroduced all the mistakes that ARM made and that RISC-V carefully omitted. SBI is another story altogether.

The `CSRR*` instructions hard-code the CSR number; they would be easier to invoke from C if the CSR number were held in *rs2* instead, thus allowing use of less assembly language while avoiding executing code generated on-the-fly.

If the kernel's stack pointer contains an invalid address (e.g., a change in page tables makes it invalid), the trap to report the invalid address will trap endlessly due to an invalid stack pointer. SBI could perhaps note and report this.

A register that returns the PLIC context for machine mode on the current CPU would ease PLIC use without requiring external assistance.

It would be useful in a few cases to be able to determine the nominal privilege mode, even if it's virtualized. Being able to probe for a given CSR without causing a trap would help too.

Machine mode seems dubious. Supervisor mode should be able to control the (possibly virtual) machine, and a mode without the possibility of paging is not helpful. Running Plan 9 or a UNIX kernel in machine mode with reasonable efficiency is infeasible; the kernel needs to use virtual memory. When running on *tinyemu*, we initially configure some M-mode-only facilities, delegate any possible M-mode traps and interrupts to S-mode, and switch to S-mode. Thereafter, we catch and forward M-mode traps to S-mode.

The focus on undetectable virtualization seems excessive. Being able to programmatically at least confirm various attributes of the environment in machine and supervisor modes would be helpful.

PMP (Physical Memory Protection) is probably unnecessary on systems with MMUs, and is a bit of a pain to configure.

5.1.1. SBI

SBI seems largely unnecessary yet it insists on disabling some hardware features that a kernel could use directly and requiring use of SBI instead. I don't want or need another layer of software between the hardware and my kernel. The SBI specification is imprecise. For example, what are the units of the timer functions? Which timer do they set? What is that timer's frequency? Is the timer global or per-hart? Under what conditions can *sbi_send_ipi* (FID 0, EID 4) fail? It has been seen to fail with valid hart ids on OpenSBI. Which supervisor-mode facilities has it disabled?

There is some evidence of bugs in OpenSBI calls, e.g., *sbi_get_hart_status*.

6. Future Work

Bootstrapping can be clumsy; a future upgrade to the Icicle's *U-boot* should yield an automatic way to PXE boot at power-on or reset. (Until then, *fshalt*(8) provides graceful reboots.)

There is hardware that we do not (yet) drive: an open PCI-E slot, an FPGA on the Icicle, and USB controller(s). Icicle documentation for USB is not obviously locatable. Icicle PCI-E requires newer HSS (hart software services) firmware.

7. Availability

A reasonably-stable distribution of the RISC-V kernel and the compiler used to build it, along with support files, is maintained in

`https://9p.io/sources/contrib/geoff/riscv/dist.9k-rv.tgz`.

8. Acknowledgements

Richard Miller developed the 32-bit and 64-bit RISC-V C compiler suites for Plan 9. He has been very helpful, fixing (minor) bugs, helping to find my obscure bugs, contributing *sdio/mmc* drivers, and extending the assemblers. The late Jim McKie created the 64-bit *9k* kernel and Charles Forsyth created the `amd64` compiler suite for the first architecture. We are building, of course, on years of work at Bell Labs creating and developing Plan 9.

References

Col2010. Geoff Collyer, ''Recent Plan 9 Work at Bell Labs,'' *Fifth International Workshop on Plan 9*, Seattle, invited talk, `http://www.collyer.net/who/-geoff/ports.pdf` (October 2010).

Col2023. Geoff Collyer, ''Plan 9 on 64-bit RISC-V,'' *Ninth International Workshop on Plan 9*, Waterloo (21 April 2023).

Int2022. RISC-V International, *RISC-V Supervisor Binary Interface Specification*, `https://github.com/riscv/riscv-sbi-doc/blob/master/-riscv-sbi.adoc`, 2022.

Int. RISC-V International, *RISC-V home*, `http://riscv.org`.

Mic. Microsemi, *Icicle*, `https://www.microsemi.com/products/-fpga-soc`.

Mil2020. Richard Miller, *A Plan 9 C Compiler for RISC-V RV32GC and RV64GC*, `https://ossg.bcs.org/wp-content/uploads/criscv64.pdf`, 19 Oct 2020.

MIT. MIT, *Address translation and sharing using page tables*, `https://pdos.csail.mit.edu/6.828/2007/lec/l5.html`.

Pat2017. David Patterson, Andrew Waterman, *The RISC-V Reader*, Strawberry Canyon (7 November 2017). `http://riscvbook.com`

Pik1990. Rob Pike, Dave Presotto, Ken Thompson, Howard Trickey, ''Plan 9 from Bell Labs,'' *Proc. of the Summer 1990 UKUUG Conf., London*, pp. 1-9 (July, 1990).

Tho2024. Fabian Thomas, Lorenz Hetterich et al., *RISCVuzz: Discovering Architectural CPU Vulnerabilities via Differential Hardware Fuzzing*, `https://ghostwriteattack.com/riscvuzz.pdf`, 7 August 2024.

Xil. Xilinx, *Zynq 7000 SoC Technical Reference Manual (UG585)*, `https://www.xilinx.com/support/documentation/user_guides/-ug585-Zynq-7000-TRM.pdf`.

Appendix

```
/* 64-bit tinyemu configuration from tecpu kernel config */
#include "riscv64.h"

int cpuserver = 1;
int idlepause = 1;
uvlong cpuhz = 156*1000*1000;  /* from timesync, emulated on 3ghz nuc */
uvlong timebase = 10*1000*1000;  /* clint ticks per second */
Membank membanks[] = {      /* (address, size) pairs */
      PHYSMEM, BANK0SIZE,
      0
};
char defnvram[] = "/boot/nvram";

uintptr uartregs[] = { PAUart0 };
int nuart = nelem(uartregs);
vlong uartfreq = 384000;

uchar ether0mac[] = { 2, 0, 0, 0, 0, 1 };

/* the emulated plic doesn't seem to follow the spec; we ignore it. */
Soc soc = {
      .clint     = (char *)PAClint,
      .uart = (char *)PAUart0,
      .plic = (char *)0x40100000,
      .ether[0] = (char *)0x40011000,
      .hobbled= 0,                /* only 1 hart */
};
Ioconf socconf[] = { /* devices without drivers that vmap their regs */
      { "clint", 64*KB, &soc.clint, },
      { "uart",  PGSZ, &soc.uart, 1, },
      { "plic",  4*MB, &soc.plic, },         /* common but smaller */
      0
};
Ioconf ioconfs[] = {  /* devices whose drivers vmap their regs */
      { "ether", 2*PGSZ, &soc.ether[0], 2, },
      0
};
```