Platform System Interface

Design and Evaluation of Computing as a Whole

Daniel Maslowski

Agenda

- Designing a Computer
- Discovering a Computer
- Platforms and Systems
- Layers and Interfaces
- Research and Development

Designing a Computer

Design and Research (from 1960s on)

Design and Research (from 1960s on)

Design helps find *solutions*.

Design and Research (from 1960s on)

Design helps find *solutions*.

Design deals with *complexity*.

Design and Research (from 1960s on)

Design helps find *solutions*.

Design deals with *complexity*.

Design is an *iterative* process.

Design and Research (from 1960s on)

Design helps find *solutions*.

Design deals with *complexity*.

Design is an *iterative* process.

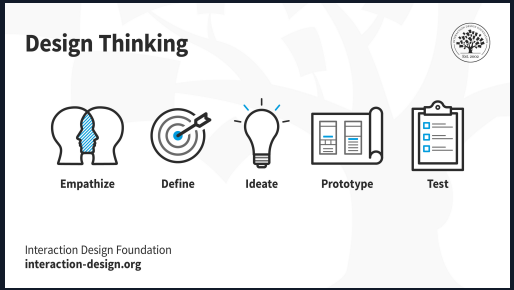


Image by Interaction Design Foundation, CC BY-SA 3.0

https://www.interaction-design.org/literature/topics/design-thinking

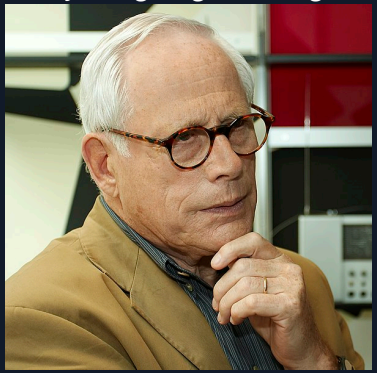
The Nature of Design Practice and Implications for Interaction Design Research

http://www.ijdesign.org/index.php/IJDesign/article/viewFile/240/139

Dieter Rams' Ten Principles of Good Design (late 1970s)

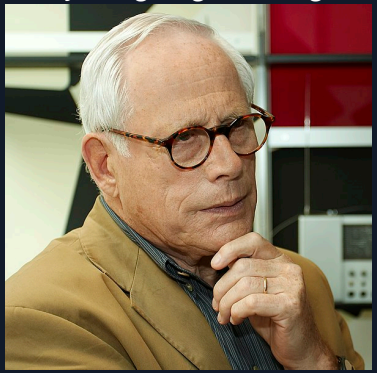
Dieter Rams' Ten Principles of Good Design (late 1970s)

"Is my design a good design?"



Dieter Rams' Ten Principles of Good Design (late 1970s)

"Is my design a good design?"



Good design…

1. is innovative.
2. makes a product useful.
3. is aesthetic.
4. makes a product understandable.
5. is honest.
6. is unobtrusive.
7. is long-lasting.
8. is thorough down to the last detail.
9. is environmentally friendly.
10. is as little design as possible.

https://en.wikipedia.org/wiki/Dieter_Rams

Holistic Architecture[1]

Holistic Architecture[1]

"Who *made* this..?!"

Holistic Architecture[1]

"Who *made* this..?!"

"Why didn't they *consider this*?
It's *so* obvious!"

Holistic Architecture[1]

"Who *made* this..?!"

"Why didn't they *consider this*?
It's *so* obvious!"

Holistic architecture means to design
for a *whole* system.

[1] https://www.interaction-design.org/literature/article/holistic-design-design-that-goes-beyond-the-problem

Holistic Architecture[1]

"Who *made* this..?!"

"Why didn't they *consider this*?
It's *so* obvious!"

Holistic architecture means to design for a *whole* system.

That is not easy and requires *knowledge* and *experience*.

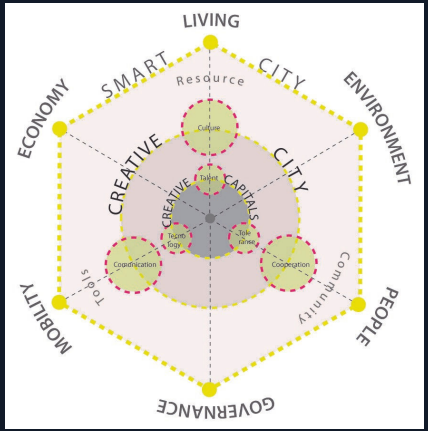


Image by Maurizio.Carta, CC BY 3.0

[1] https://www.interaction-design.org/literature/article/holistic-design-design-that-goes-beyond-the-problem

Explicit and Implicit Knowledge

Explicit and Implicit Knowledge

> *Explicit knowledge is the most basic form of knowledge and is easy to pass along because it's written down and accessible.*

https://bloomfire.com/blog/implicit-tacit-explicit-knowledge/

Explicit and Implicit Knowledge

> *Explicit knowledge is the most basic form of knowledge and is easy to pass along because it's written down and accessible.*

https://bloomfire.com/blog/implicit-tacit-explicit-knowledge/

> *Implicit Knowledge is knowledge that is gained through incidental activities, or without awareness that learning is occurring.*

https://trainingindustry.com/glossary/implicit-knowledge/

Tacit and Tribal Knowledge

Tacit and Tribal Knowledge

> *Tacit knowledge refers to the knowledge, skills, and abilities an individual gains through experience that is often difficult to put into words or otherwise communicate.*

https://helpjuice.com/blog/tacit-knowledge

Tacit and Tribal Knowledge

> *Tacit knowledge refers to the knowledge, skills, and abilities an individual gains through experience that is often difficult to put into words or otherwise communicate.*

https://helpjuice.com/blog/tacit-knowledge

> *Tribal knowledge refers to any unwritten knowledge within a company that is not widely known.*

https://www.lucidchart.com/blog/what-is-tribal-knowledge

Computer Knowledge

Computer Knowledge

A lot of knowledge about computers is hard to pass on and takes time to learn. Manuals can be very sparse and require experience to read.

Computer Knowledge

A lot of knowledge about computers is hard to pass on and takes time to learn. Manuals can be very sparse and require experience to read.

At the same time, it is a mystery to figure out what ideas are transferable, what is common between vendors and products, and what is specific.

Electromechanical Computers

Electromechanical Computers

Harvard Mark I
1944, general-purpose computer
First programmers: Richard Milton Bloch, Robert Campbell, Grace Hopper

Electromechanical Computers

Harvard Mark I
1944, general-purpose computer
First programmers: Richard Milton Bloch, Robert Campbell, Grace Hopper

Grace Hopper had the idea of a machine-independent programming language.

Electromechanical Computers

Harvard Mark I
1944, general-purpose computer
First programmers: Richard Milton Bloch, Robert Campbell, Grace Hopper

Grace Hopper had the idea of a machine-independent programming language. She created FLOW-MATIC, the basis for COBOL.

Electromechanical Computers

Harvard Mark I

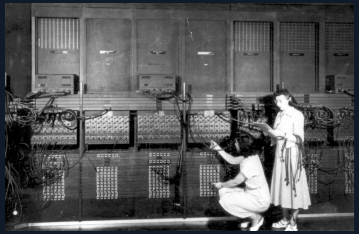
1944, general-purpose computer
First programmers: Richard Milton Bloch, Robert Campbell, Grace Hopper

Grace Hopper had the idea of a machine-independent programming language. She created FLOW-MATIC, the basis for COBOL.

https://en.wikipedia.org/wiki/Harvard_Mark_I

Electronic Computers

Electronic Computers

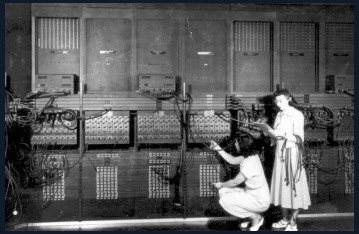


ENIAC

1945, first programmable, general-purpose digital computer

First ENIAC programmers: Jean Bartik, Betty Holberton, Kathleen Antonelli, Marlyn Meltzer, Ruth Teitelbaum, Frances Spence

Electronic Computers



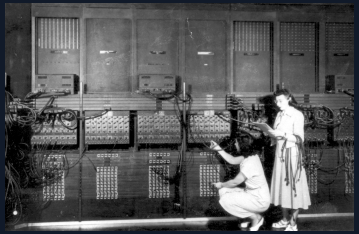
ENIAC

1945, first programmable, general-purpose digital computer

First ENIAC programmers: Jean Bartik, Betty Holberton, Kathleen Antonelli, Marlyn Meltzer, Ruth Teitelbaum, Frances Spence

Betty Holberton invented breakpoints.

Electronic Computers



ENIAC

1945, first programmable, general-purpose digital computer

First ENIAC programmers: Jean Bartik, Betty Holberton, Kathleen Antonelli, Marlyn Meltzer, Ruth Teitelbaum, Frances Spence

Betty Holberton invented breakpoints.

Kathleen Antonelli invented subroutines.

https://en.wikipedia.org/wiki/ENIAC

Transistor

Transistor

Yes, the tiny digital switch that makes our machines go vroom vroom.
It just turned 75 on December 23. :-)

Transistor

Yes, the tiny digital switch that makes our machines go vroom vroom.
It just turned 75 on December 23. :-)

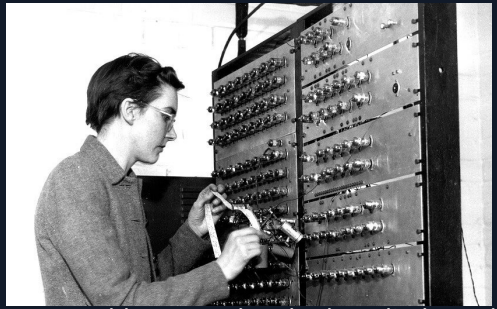
> *John Bardeen, Walter Brattain and William Shockley invented the first working transistors at Bell Labs, the point-contact transistor in 1947. Shockley introduced the improved bipolar junction transistor in 1948, which entered production in the early 1950s and led to the first widespread use of transistors.*

https://en.wikipedia.org/wiki/History_of_the_transistor

https://www.pbs.org/transistor/index.html

Assembly Language (1947)

Assembly Language (1947)



Kathleen Booth, who has died aged 100, co-designed of one of the world's first operational computers and wrote two of the earliest books on computer design and programming; she was also credited with the invention of the first "assembly language", a programming language designed to be readable by users

https://www.telegraph.co.uk/obituaries/2022/10/25/kathleen-booth-computer-pioneer-who-made-major-breakthrough/

Electronic Delay Storage Automatic Calculator (1949)[2]

Electronic Delay Storage Automatic Calculator (1949)[2]

PREFACE

This statement is the first part (Part A) of a description of the electronic calculating machine which has been built at Cambridge. The statement is so prepared that it can stand by itself; it gives a general idea of the way in which the machine works, and is divided into the following five sections:-

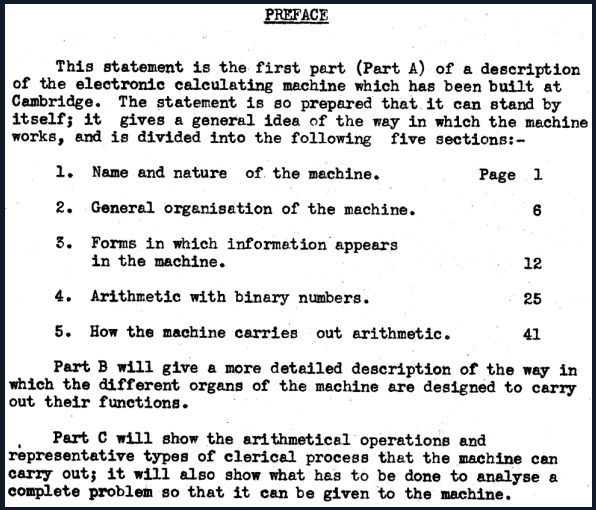
1. Name and nature of the machine. Page 1
2. General organisation of the machine. 6
3. Forms in which information appears in the machine. 12
4. Arithmetic with binary numbers. 25
5. How the machine carries out arithmetic. 41

Part B will give a more detailed description of the way in which the different organs of the machine are designed to carry out their functions.

Part C will show the arithmetical operations and representative types of clerical process that the machine can carry out; it will also show what has to be done to analyse a complete problem so that it can be given to the machine.

[2] https://www.leo-computers.org.uk/images/How%20EDSAC%20Works.pdf

Electronic Delay Storage Automatic Calculator (1949)[2]



PREFACE

This statement is the first part (Part A) of a description of the electronic calculating machine which has been built at Cambridge. The statement is so prepared that it can stand by itself; it gives a general idea of the way in which the machine works, and is divided into the following five sections:-

1. Name and nature of the machine. Page 1
2. General organisation of the machine. 6
3. Forms in which information appears in the machine. 12
4. Arithmetic with binary numbers. 25
5. How the machine carries out arithmetic. 41

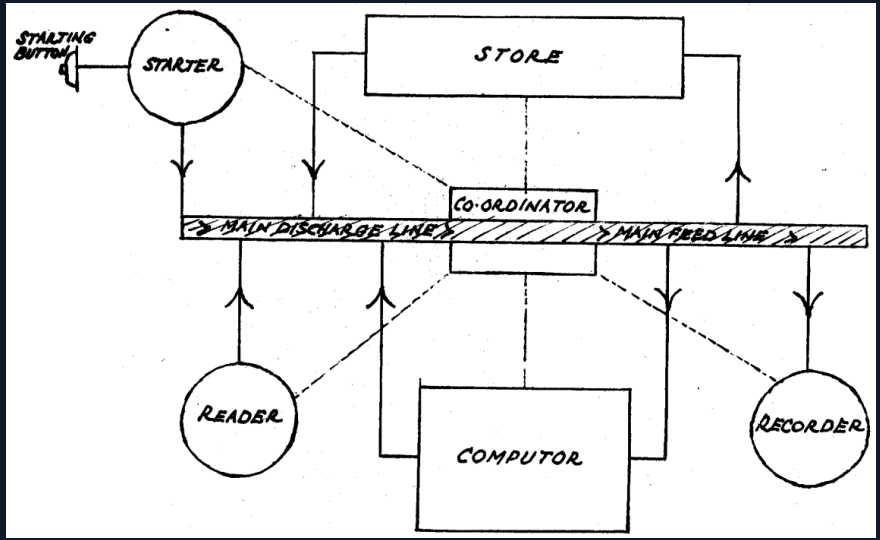
Part B will give a more detailed description of the way in which the different organs of the machine are designed to carry out their functions.

Part C will show the arithmetical operations and representative types of clerical process that the machine can carry out; it will also show what has to be done to analyse a complete problem so that it can be given to the machine.

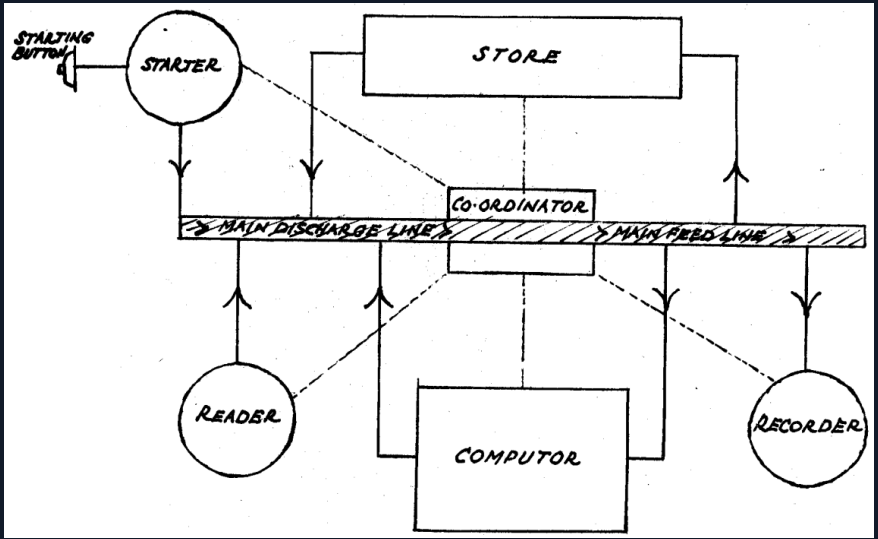
Assembly language: “initial orders”

[2] https://www.leo-computers.org.uk/images/How%20EDSAC%20Works.pdf

EDSAC Diagram



EDSAC Diagram



design picked up by J. Lyons & Co. Ltd. for *business* purposes - LEO I

Integrated Circuit

1958, Jack Kilby at Texas Instruments

1959, Robert Noyce at the Fairchild Semiconductor

https://www.pbs.org/transistor/background1/events/icinv.html

Home Computers

Home Computers

https://en.wikipedia.org/wiki/Home_computer

“1977 Trinity”: Commodore PET 2001-8, Apple II, TRS-80 Model I

Photo by Tim Colegrove, CC BY-SA 4.0,
https://commons.wikimedia.org/wiki/File:Trinity77.jpg

Personal Computers

Personal Computers

Kenbak-I

1971, considered the first *personal computer*

https://en.wikipedia.org/wiki/History_of_personal_computers

Personal Computers

Kenbak-I

1971, considered the first *personal computer*
https://en.wikipedia.org/wiki/History_of_personal_computers

Xerox Alto

1973, Xerox PARC: windows based GUI, *desktop*, mouse, ethernet

Personal Computers

Kenbak-I

1971, considered the first *personal computer*
https://en.wikipedia.org/wiki/History_of_personal_computers

Xerox Alto

1973, Xerox PARC: windows based GUI, *desktop*, mouse, ethernet

Paradigm shift

The computer now has a *consumer* rather than only a *programmer*.
It is run by an *operating system* instead of an *operator*.

Microprocessors

Microprocessors

1971, Intel 4004, first CPU, 4-bit

Microprocessors

1971, Intel 4004, first CPU, 4-bit

IBM PC (model 5150)

1981, based on Intel 8088

> *The design process was kept under a policy of strict secrecy, with all other IBM divisions kept in the dark about the project.*

https://en.wikipedia.org/wiki/IBM_Personal_Computer

Microprocessors

1971, Intel 4004, first CPU, 4-bit

IBM PC (model 5150)

1981, based on Intel 8088

> *The design process was kept under a policy of strict secrecy, with all other IBM divisions kept in the dark about the project.*

https://en.wikipedia.org/wiki/IBM_Personal_Computer

First Laptop: Gavilan SC

1983, based on Intel 8088

> *Jack Hall, an award-winning industrial designer, was chosen to work out the ergonomics, mechanics and overall appearance of the Gavilan.*

https://en.wikipedia.org/wiki/Gavilan_SC

Discovering a Computer

Computer = Processor + Memory + Peripherals

… almost *everything* is or contains a computer today.

Everyday Electronics

Everyday Electronics

Shopping Center and Supermarket

- parking lots: sensors and capacity displays
- elevators, escalators, automatic doors
- price tags (e-ink displays)
- barcode scanners and electronic payment

Everyday Electronics

Shopping Center and Supermarket

- parking lots: sensors and capacity displays
- elevators, escalators, automatic doors
- price tags (e-ink displays)
- barcode scanners and electronic payment

IoT and Friends

- fridges, coffee machines, dishwashers, laundry machines…
- gadgets, wearables…
- routers, IP cameras, network storage…
- industrial control systems, appliances…

Everyday Electronics

Shopping Center and Supermarket

- parking lots: sensors and capacity displays
- elevators, escalators, automatic doors
- price tags (e-ink displays)
- barcode scanners and electronic payment

IoT and Friends

- fridges, coffee machines, dishwashers, laundry machines…
- gadgets, wearables…
- routers, IP cameras, network storage…
- industrial control systems, appliances…

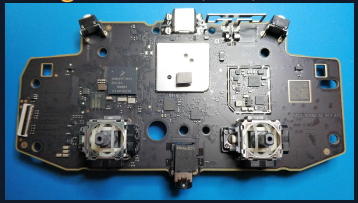
Smart Home/Building/City

Idea: Automation, energy saving, data collection

Example: lights that turn on when approaching and off after leaving

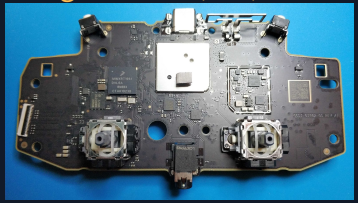
Entertainment or Education

Google Stadia (RIP)



Entertainment or Education

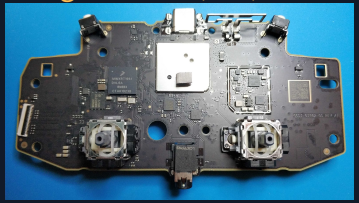
Google Stadia (RIP)



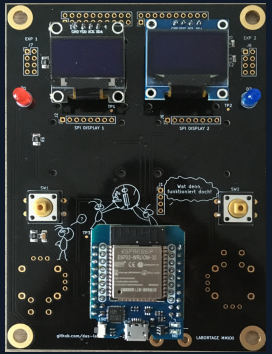
EOL: January 18, 2023

Entertainment or Education

Google Stadia (RIP)



EOL: January 18, 2023

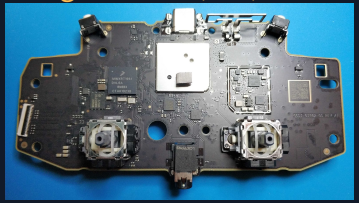


Labor Badge

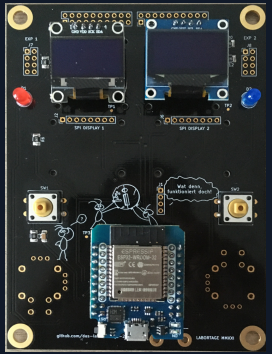
- modular
- reusable
- discoverable
- programmable

Entertainment or Education

Google Stadia (RIP)



EOL: January 18, 2023



Labor Badge

- modular
- reusable
- discoverable
- programmable

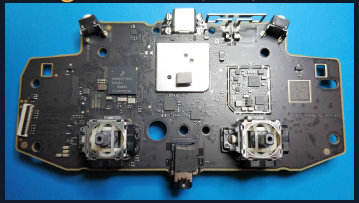
Help wanted:
Design other SoM carrier boards

https://blog.google/products/stadia/message-on-stadia-streaming-strategy/

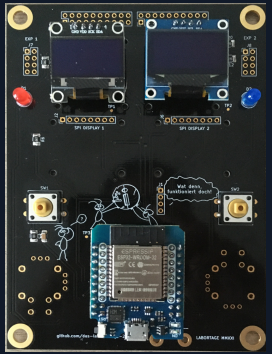
https://github.com/das-labor/badge-2021

Entertainment or Education

Google Stadia (RIP)



EOL: January 18, 2023



Labor Badge

- modular
- reusable
- discoverable
- programmable

Help wanted:
Design other SoM carrier boards

https://blog.google/products/stadia/message-on-stadia-streaming-strategy/

https://github.com/das-labor/badge-2021

Console hacking is still a thing; see Nintendo and PlayStation. :-)

Community Computers

Community Computers

Anachro

https://anachro.computer/

What is Anachro? Anachro is two things: A Network Protocol, and a PC architecture for a microcontroller-based system.

Community Computers

Anachro

https://anachro.computer/

What is Anachro? Anachro is two things: A Network Protocol, and a PC architecture for a microcontroller-based system.

Neotron

https://neotron-compute.github.io/Neotron-Book/

https://github.com/Neotron-Compute/Neotron-Pico

A Neotron system powered by the Raspberry Pi Pico, in a micro-ATX form-factor.

Bringing Up the Neotron PICO - A retro-style mATX PC; Jonathan Pallant, Ben Jordan and Bil Herd

https://www.youtube.com/watch?v=X1-mt4mrZ9E

More Computers

More Computers

moss

https://github.com/mosscomp/moss

> *moss is a vertically-integrated computer with the following design goals:*
>
> - *Exceedingly understandable by users.*
> - *Competitive in performance.*

More Computers

moss

https://github.com/mosscomp/moss

> *moss is a vertically-integrated computer with the following design goals:*
>
> - *Exceedingly understandable by users.*
> - *Competitive in performance.*

Build an 8-bit computer from scratch

https://eater.net/8bit/

More Computers

moss

https://github.com/mosscomp/moss

> *moss is a vertically-integrated computer with the following design goals:*
>
> - *Exceedingly understandable by users.*
> - *Competitive in performance.*

Build an 8-bit computer from scratch

https://eater.net/8bit/

Start to an 80286 System

https://www.rehsdonline.com/post/start-to-an-80286-system
https://www.youtube.com/playlist?list=PL7sb-_3xk_CAMDL_dj9l-plqSrEzcqx1G

Mobile Devices

Mobile Devices

MNT Reform Laptop

The Much More Personal Computer

https://mntre.com/

Mobile Devices

MNT Reform Laptop

The Much More Personal Computer

https://mntre.com/

PinePhone

An Open Source Smartphone Supported by All Major Linux Phone Projects

https://www.pine64.org/pinephone/

Big Computers

Big Computers

OCP (Open Compute Project)

https://www.opencompute.org/about

> *The Open Compute Project (OCP) is a collaborative community focused on redesigning hardware technology to efficiently support the growing demands on compute infrastructure.*

Big Computers

OCP (Open Compute Project)

https://www.opencompute.org/about

> *The Open Compute Project (OCP) is a collaborative community focused on redesigning hardware technology to efficiently support the growing demands on compute infrastructure.*

Oxide Computer

> *A rack-scale server with tightly integrated hardware and software.*

https://oxide.computer/

Big Computers

OCP (Open Compute Project)

https://www.opencompute.org/about

> *The Open Compute Project (OCP) is a collaborative community focused on redesigning hardware technology to efficiently support the growing demands on compute infrastructure.*

Oxide Computer

> *A rack-scale server with tightly integrated hardware and software.*

https://oxide.computer/

Racklet

https://racklet.io/

> *Racklet is a fully-integrated, miniature server rack.*

Single Board Computers

Single Board Computers

Many are *marketed* as open source. Are they though?

Single Board Computers

Many are *marketed* as open source. Are they though?

Documentation

- *schematics* and board design
- manuals and instructions
- open *license*

Single Board Computers

Many are *marketed* as open source. Are they though?

Documentation

- *schematics* and board design
- manuals and instructions
- open *license*

Source Code

- *open* tools for flashing, debugging and image composition
- firmware, *from the start*, documented (U-Boot, oreboot, …)
- Linux or other OS, *mainline friendly* (git fork, *not* source dump)
- all code usable with upstream toolchains, or provide toolchains in a *reproducible* form (not only binaries for a specific architecture/OS)

Single Board Computers

Many are *marketed* as open source. Are they though?

Documentation

- *schematics* and board design
- manuals and instructions
- open *license*

Source Code

- *open* tools for flashing, debugging and image composition
- firmware, *from the start*, documented (U-Boot, oreboot, …)
- Linux or other OS, *mainline friendly* (git fork, *not* source dump)
- all code usable with upstream toolchains, or provide toolchains in a *reproducible* form (not only binaries for a specific architecture/OS)

OSHWA Certification: https://certification.oshwa.org/

Firmware

Firmware

Predicament

Firmware is a loosely defined term, sometimes called “the BIOS”.

Firmware

Predicament

Firmware is a loosely defined term, sometimes called “the BIOS”.

It has different meanings in marketing, colloquial speech, across products and vendors, and often wants to include many and more things than necessary.

Firmware

Predicament

Firmware is a loosely defined term, sometimes called “the BIOS”.

It has different meanings in marketing, colloquial speech, across products and vendors, and often wants to include many and more things than necessary.

At the very least, firmware is the software part of a computing platform that initializes hardware so that an operating system may run, or any *payload*, e.g., a bare metal application or hypervisor.

Firmware

Predicament

Firmware is a loosely defined term, sometimes called “the BIOS”.

It has different meanings in marketing, colloquial speech, across products and vendors, and often wants to include many and more things than necessary.

At the very least, firmware is the software part of a computing platform that initializes hardware so that an operating system may run, or any *payload*, e.g., a bare metal application or hypervisor.

Why would that ever be necessary, even?

Open Source Firmware Foundation (OSFF)

https://osfw.foundation/

Open Source Firmware Foundation (OSFF)

https://osfw.foundation/

*The OSFF is meant to be an **umbrella organization** for all parties interested in open-source firmware and acts as the first point of contact in the open-source firmware ecosystem.*

Fiedka the Firmware Editor

https://fiedka.app/

Fiedka the Firmware Editor

https://fiedka.app/

Features

- analyze firmware images
- visualize flash usage
- explore file systems
  - UEFI
  - PSP (AMD)
  - CBFS (coreboot)
- remove UEFI files
- embed LinuxBoot
- meta data export

Fiedka the Firmware Editor

https://fiedka.app/

Features

- analyze firmware images
- visualize flash usage
- explore file systems
  - UEFI
  - PSP (AMD)
  - CBFS (coreboot)
- remove UEFI files
- embed LinuxBoot
- meta data export

Work in progress

- SBoM (Software Bill of Materials)

Platforms and Systems

What are Platforms?

What are Platforms?

A computing platform or digital platform is an environment in which a piece of software is executed.

It may be the hardware or the operating system (OS), even a web browser and associated application programming interfaces, or other underlying software, as long as the program code is executed with it.

Computing platforms have different abstraction levels, including a computer architecture, an OS, or runtime libraries.

A computing platform is the stage on which computer programs can run.

https://en.wikipedia.org/wiki/Computing_platform

Platform System Interface (PSI)

Platform System Interface (PSI)

https://github.com/platform-system-interface/psi-spec

Goal: Derive a specification, summarizing firmware projects, their boot flows, how they interact as a platform with the actual operating system.

Platform System Interface (PSI)

https://github.com/platform-system-interface/psi-spec

Goal: Derive a specification, summarizing firmware projects, their boot flows, how they interact as a platform with the actual operating system.

How: Extract features, compare approaches, reevaluate, improve.

Computer Architecture: Buses

Computer Architecture: Buses

https://en.wikipedia.org/wiki/Bus_(computing)

Computer Architecture: Buses

https://en.wikipedia.org/wiki/Bus_(computing)

Wires

Example I2C: VCC, GND, SCL (clock), SCA (data)
Many I2C buses are in your laptop, even within HDMI and VGA ports.
They are often used for connecting sensors, e.g., for temperature.
Linux: `i2c-tools`

Computer Architecture: Buses

https://en.wikipedia.org/wiki/Bus_(computing)

Wires

Example I2C: VCC, GND, SCL (clock), SCA (data)
Many I2C buses are in your laptop, even within HDMI and VGA ports.
They are often used for connecting sensors, e.g., for temperature.
Linux: `i2c-tools`

Protocols

Example SPI flash: Commands for reading and writing data
We load from SPI flash in oreboot on the Allwinner D1 SoC.

Computer Architecture: Buses

https://en.wikipedia.org/wiki/Bus_(computing)

Wires

Example I2C: VCC, GND, SCL (clock), SCA (data)
Many I2C buses are in your laptop, even within HDMI and VGA ports.
They are often used for connecting sensors, e.g., for temperature.
Linux: `i2c-tools`

Protocols

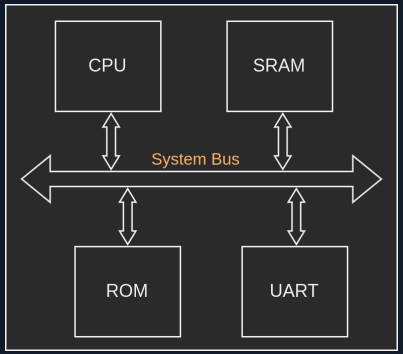
Example SPI flash: Commands for reading and writing data
We load from SPI flash in oreboot on the Allwinner D1 SoC.

Conventions and Standards

Example: USB
https://www.electronics-notes.com/articles/connectivity/usb-universal-serial-bus/basics-tutorial.php
Those make up *interfaces*, enabling a *market* through compatibility.

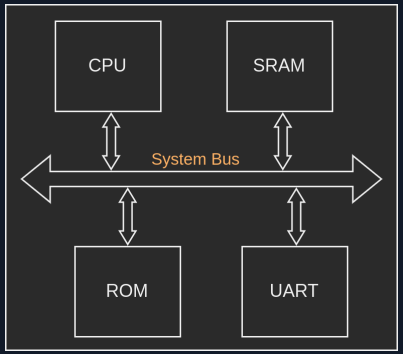
Internal Buses

Internal Buses



There are even buses *within* chips.

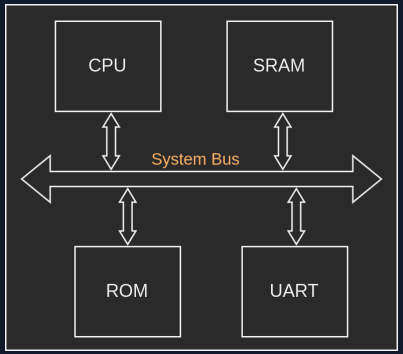
Internal Buses



There are even buses *within* chips.

Those buses connect the components of a chip, also called *blocks* or *cores*.

Internal Buses



There are even buses *within* chips.

Those buses connect the components of a chip, also called *blocks* or *cores*.

Example:
Advanced High-Performance Bus (AHB)

AHB is part of AMBA (Advanced Microcontroller Bus Architecture).

https://developer.arm.com/Architectures/AMBA

Complexity in Computers

Complexity in Computers

x86 (CISC, very complex)

https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

Complexity in Computers

x86 (CISC, very complex)

https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

ARM (yet somewhat complex)

https://www.arm.com/glossary/risc

> *A Reduced Instruction Set Computer is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions*

Complexity in Computers

x86 (CISC, very complex)

https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

ARM (yet somewhat complex)

https://www.arm.com/glossary/risc

> *A Reduced Instruction Set Computer is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions*

RISC-V (open specifications)

https://riscv.org/announcements/2022/12/risc-v-sees-significant-growth-and-technical-progress-in-2022-with-billions-of-risc-v-cores-in-market/

> *RISC-V combines a modular technical approach with an open, royalty-free license model*

Call for Simplicity: Reduction

Call for Simplicity: Reduction

> *Developers are drawn to complexity like moths to a flame often with the same result.*

https://nealford.com/books/productiveprogrammer

Call for Simplicity: Reduction

> *Developers are drawn to complexity like moths to a flame often with the same result.*

https://nealford.com/books/productiveprogrammer

Complexity can be inherent or *given*, e.g., in Physics.

Call for Simplicity: Reduction

> *Developers are drawn to complexity like moths to a flame often with the same result.*

https://nealford.com/books/productiveprogrammer

Complexity can be inherent or *given*, e.g., in Physics.

Note: We *discover* physics, we do not invent it!

Call for Simplicity: Reduction

> *Developers are drawn to complexity like moths to a flame often with the same result.*

https://nealford.com/books/productiveprogrammer

Complexity can be inherent or *given*, e.g., in Physics.

Note: We *discover* physics, we do not invent it!

When it gets too much, the *measure* is *reduction*.

Call for Simplicity: Reduction

> *Developers are drawn to complexity like moths to a flame often with the same result.*

https://nealford.com/books/productiveprogrammer

Complexity can be inherent or *given*, e.g., in Physics.

Note: We *discover* physics, we do not invent it!

When it gets too much, the *measure* is *reduction*.

The *opposite* is **Simplicity**.

Software Architecture

Software Architecture

Software architecture is for developers to live inside.

Kevlin Henney, Refactoring Is Not Just Clickbait, NDC Oslo 2022
https://www.youtube.com/watch?v=piUesxuZkIQ&t=546

Software Architecture

> *Software architecture is for developers to live inside.*

Kevlin Henney, Refactoring Is Not Just Clickbait, NDC Oslo 2022
https://www.youtube.com/watch?v=piUesxuZkIQ&t=546

> *Most architects and developers pursue the Latest and Greatest with great fervor, yet the history of engineering, including software projects, contains rich lessons that we risk repeating ad nauseam.*

https://joyofcoding.org/2017/speaker/neal-ford/

UEFI vs NERF and FASR

https://uefi.org/about

> *These extensible, globally-recognized specifications bring new functionality and enhanced security to the evolution of devices, firmware and operating systems, as well as facilitate interoperability between platforms and systems that comply with next-generation technologies.*

UEFI vs NERF and FASR

https://uefi.org/about

These extensible, globally-recognized specifications bring new functionality and enhanced security to the evolution of devices, firmware and operating systems, as well as facilitate interoperability between platforms and systems that comply with next-generation technologies.

https://trmm.net/NERF/

The LinuxBoot project (formerly NERF) is a collaboration between Google, Facebook, Horizon Computing Solutions, and Two Sigma that aims to build an open, customizable, and slightly more secure firmware for server machines based on Linux.

UEFI vs NERF and FASR

https://uefi.org/about

These extensible, globally-recognized specifications bring new functionality and enhanced security to the evolution of devices, firmware and operating systems, as well as facilitate interoperability between platforms and systems that comply with next-generation technologies.

https://trmm.net/NERF/

The LinuxBoot project (formerly NERF) is a collaboration between Google, Facebook, Horizon Computing Solutions, and Two Sigma that aims to build an open, customizable, and slightly more secure firmware for server machines based on Linux.

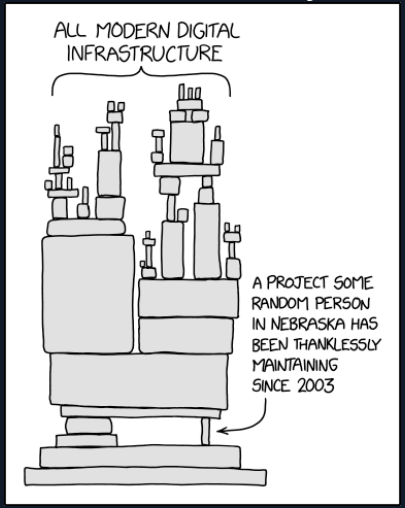
https://learn.microsoft.com/en-us/windows-hardware/drivers/bringup/firmware-attack-surface-reduction

Microsoft has started working with partners to overcome the compatibility issues

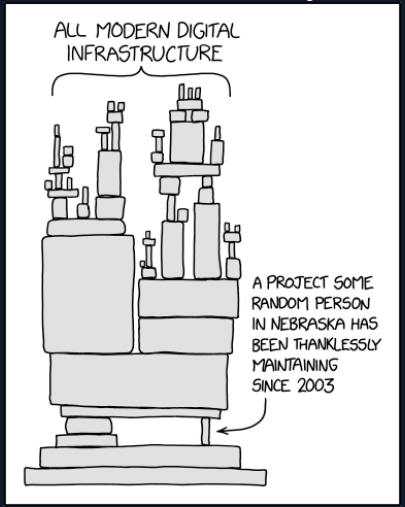
Layers and Interfaces

The Problem with Layers

The Problem with Layers



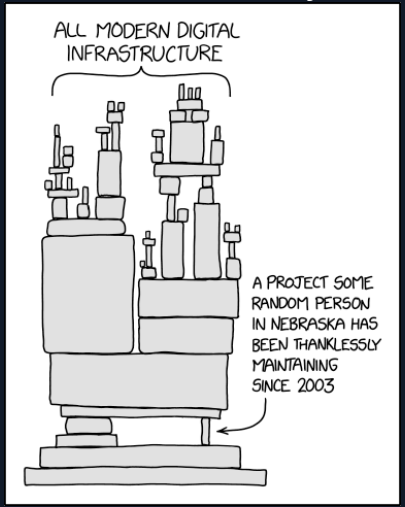
The Problem with Layers



Comic by Randall Munroe, CC BY-NC 2.5
https://xkcd.com/2347/

Layering implies interfaces.

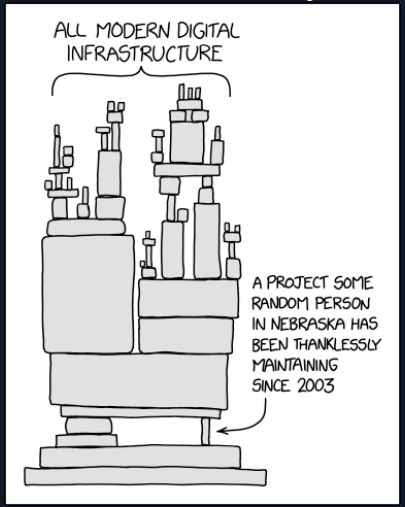
The Problem with Layers



Comic by Randall Munroe, CC BY-NC 2.5
https://xkcd.com/2347/

Layering implies interfaces.
Interfaces are hard to design.

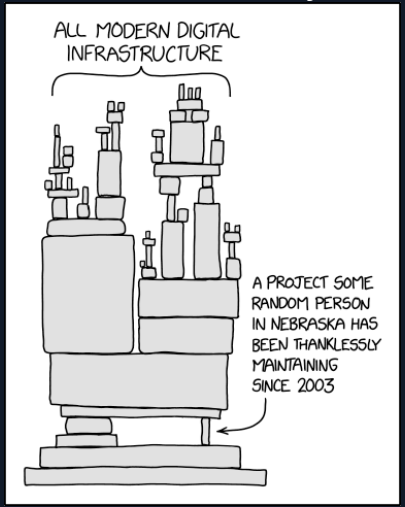
The Problem with Layers



Comic by Randall Munroe, CC BY-NC 2.5
https://xkcd.com/2347/

Layering implies interfaces.
Interfaces are hard to design.
Sometimes unnecessary, but…

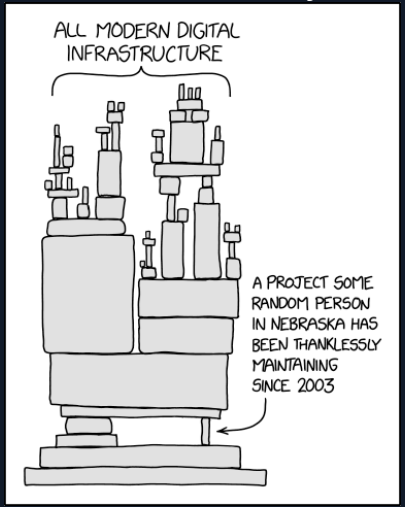
The Problem with Layers



Comic by Randall Munroe, CC BY-NC 2.5
https://xkcd.com/2347/

Layering implies interfaces.
Interfaces are hard to design.
Sometimes unnecessary, but…
… spark creativity in other places.

The Problem with Layers

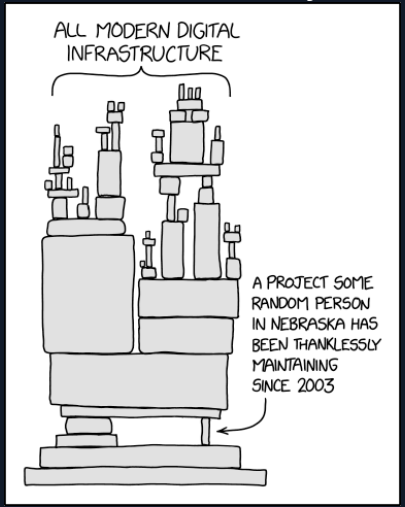


Comic by Randall Munroe, CC BY-NC 2.5
https://xkcd.com/2347/

Layering implies interfaces.
Interfaces are hard to design.
Sometimes unnecessary, but…
… spark creativity in other places.

Example LinuxBIOS: Put Linux in flash because the vendor BIOS did not work.
Note: LinuxBIOS has evolved into coreboot.

The Problem with Layers



Layering implies interfaces.
Interfaces are hard to design.
Sometimes unnecessary, but…
… spark creativity in other places.

Example LinuxBIOS: Put Linux in flash because the vendor BIOS did not work.
Note: LinuxBIOS has evolved into coreboot.

Example LinuxBoot: Put Linux in SPI flash to replace part of vendor UEFI PI (platform init).

Firmware and Ownership

Firmware and Ownership

Pico Host Boot Loader

phbl is the program run from the x86 reset vector that loads and invokes the phase1 host operating system package

https://github.com/oxidecomputer/phbl

Firmware and Ownership

Pico Host Boot Loader

phbl is the program run from the x86 reset vector that loads and invokes the phase1 host operating system package

https://github.com/oxidecomputer/phbl

OSF (Open System Firmware)

https://www.opencompute.org/projects/open-system-firmware

Open system firmware is an open development project, the goal of which is to allow OCP owners to "own their firmware" – to move the point of control of firmware to the system owner.

Firmware and Ownership

Pico Host Boot Loader

phbl is the program run from the x86 reset vector that loads and invokes the phase1 host operating system package

https://github.com/oxidecomputer/phbl

OSF (Open System Firmware)

https://www.opencompute.org/projects/open-system-firmware

Open system firmware is an open development project, the goal of which is to allow OCP owners to "own their firmware" – to move the point of control of firmware to the system owner.

Composition and Layering

Firmware and Ownership

Pico Host Boot Loader

phbl is the program run from the x86 reset vector that loads and invokes the phase1 host operating system package

https://github.com/oxidecomputer/phbl

OSF (Open System Firmware)

https://www.opencompute.org/projects/open-system-firmware

Open system firmware is an open development project, the goal of which is to allow OCP owners to "own their firmware" – to move the point of control of firmware to the system owner.

Composition and Layering

Layers grow *vertically*.

Firmware and Ownership

Pico Host Boot Loader

phbl is the program run from the x86 reset vector that loads and invokes the phase1 host operating system package

https://github.com/oxidecomputer/phbl

OSF (Open System Firmware)

https://www.opencompute.org/projects/open-system-firmware

Open system firmware is an open development project, the goal of which is to allow OCP owners to "own their firmware" – to move the point of control of firmware to the system owner.

Composition and Layering

Layers grow *vertically*.

Components can live *on the same layer*.

Firmware and Ownership

Pico Host Boot Loader

phbl is the program run from the x86 reset vector that loads and invokes the phase1 host operating system package

https://github.com/oxidecomputer/phbl

OSF (Open System Firmware)

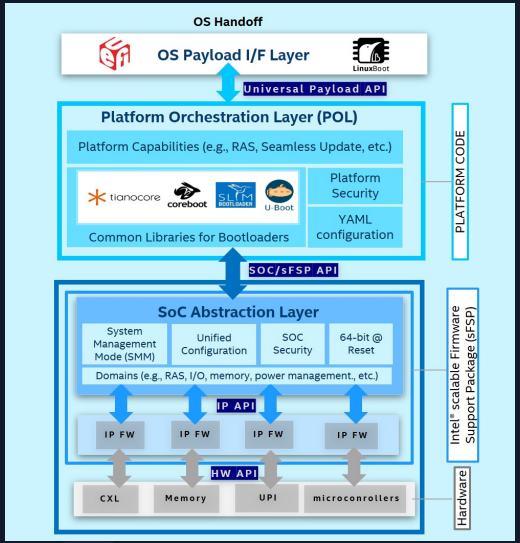
https://www.opencompute.org/projects/open-system-firmware

Open system firmware is an open development project, the goal of which is to allow OCP owners to "own their firmware" – to move the point of control of firmware to the system owner.

Composition and Layering

Layers grow *vertically*.
Components can live *on the same layer*.
Not having ownership results in being stuck with layers.

Intel's Universal Scalable Firmware[3]



Note: (s)FSP components are distributed in binary form, hard to audit or fix.

They make up a large portion of the code and bury the understanding of the platform.
Their APIs carry potential for error and vulnerabilities.

[3] https://universalscalablefirmware.github.io/documentation/1_terminology.html

Silicon Interface Design

https://osfw.foundation/workstreams/silicon-interface-design/

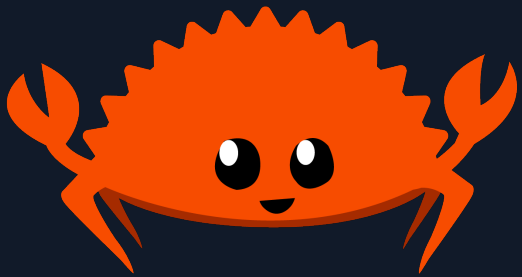
Silicon Interface Design

https://osfw.foundation/workstreams/silicon-interface-design/

> *Integrating binary blobs that handle parts of the silicon initialization is a common technique within the open-source firmware ecosystem to retain control over parts of the code, from a SoC vendor perspective.*

oreboot

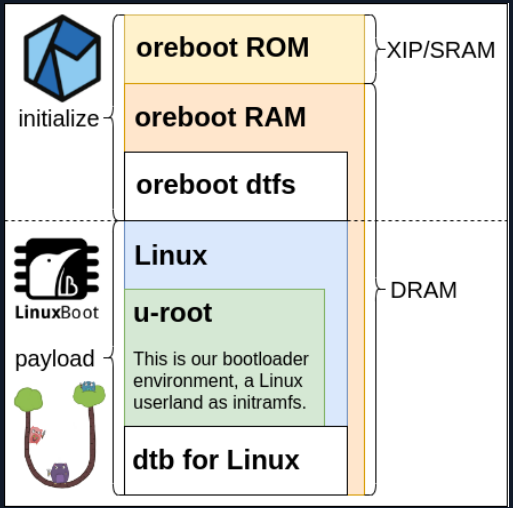
oreboot is firmware written in Rust.



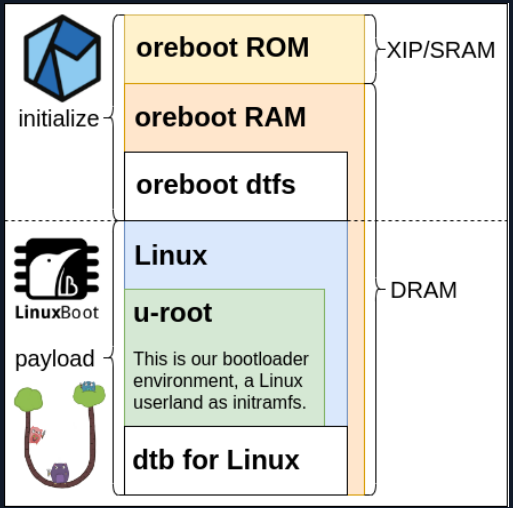
https://github.com/oreboot

Ferris the crab from https://rustacean.net/

oreboot Stages



oreboot Stages



XIP/SRAM

- early init, MMIO
- PLLs, clocks, GPIOs
- UART, say hello
- DRAM controller
- storage setup
  - SPI flash, SD card, eMMC…

DRAM

- what didn't fit in SRAM
  - extract payload
  - set up handlers
- run payload (done)

https://github.com/oreboot/oreboot/blob/main/Documentation/boot-flow.md

Firmware Runtime Services

Firmware Runtime Services

Idea: Define interfaces in a software part of a platform.

Firmware Runtime Services

Idea: Define interfaces in a software part of a platform.

Vulnerability REsearch

Vulnerabillity Category	Count	Average Impact
PEI Memory Corruption	3	CVSS: 8.0 (High)
SMM Memory Corruption	57	CVSS: 8.0 (High)
DXE Memory Corruption	10	CVSS: 7.7 (High)
Mitigation Failures	2	CVSS: 6.0 (HighMedium)

Whoops! They also present an attack surface.

RISC-V Runtime Services

RISC-V Runtime Services

Runtime Services are listed in **platform specs**, referencing the **SBI spec**.
https://github.com/riscv/riscv-platform-specs

RISC-V Runtime Services

Runtime Services are listed in **platform specs**, referencing the **SBI spec**.
https://github.com/riscv/riscv-platform-specs

The SBI (*Supervisor Binary Interface*) spec is a living document:
https://github.com/riscv-non-isa/riscv-sbi-doc

RISC-V Runtime Services

Runtime Services are listed in **platform specs**, referencing the **SBI spec**.
https://github.com/riscv/riscv-platform-specs

The SBI (*Supervisor Binary Interface*) spec is a living document:
https://github.com/riscv-non-isa/riscv-sbi-doc

It defines extensions and functions similar to system calls.

RISC-V Runtime Services

Runtime Services are listed in **platform specs**, referencing the **SBI spec**.
https://github.com/riscv/riscv-platform-specs

The SBI (*Supervisor Binary Interface*) spec is a living document:
https://github.com/riscv-non-isa/riscv-sbi-doc

It defines extensions and functions similar to system calls.

Ports need to be written per platform (core/SoC/board).

RISC-V Runtime Services

Runtime Services are listed in **platform specs**, referencing the **SBI spec**.
https://github.com/riscv/riscv-platform-specs

The SBI (*Supervisor Binary Interface*) spec is a living document:
https://github.com/riscv-non-isa/riscv-sbi-doc

It defines extensions and functions similar to system calls.

Ports need to be written per platform (core/SoC/board).

RISC-V PRS TG (Platform Runtime Services Task Group) is concerned with specs around ACPI, UEFI, SBI, and possible other interfaces.

https://lists.riscv.org/g/tech-prs

https://github.com/riscv-admin/prs

RISC-V Environment Calls

RISC-V Environment Calls

Arguments and extension/function are passed through *A* (argument) registers.

RISC-V Environment Calls

Arguments and extension/function are passed through *A* (argument) registers.

Then the call is performed via the `ECALL` instruction.

RISC-V Environment Calls

Arguments and extension/function are passed through *A* (argument) registers.

Then the call is performed via the `ECALL` instruction.

Example, writing a **B** character to the serial console:

```
li a0, 'B'   # argument
li a7, 0x01  # extension "console putchar"
ecall
```

From Registers to Memory Access

From Registers to Memory Access

A single value read from a register and written to a UART seems comprehensible.

From Registers to Memory Access

A single value read from a register and written to a UART seems comprehensible.

What if that value is a memory pointer for *privileged* access?

From Registers to Memory Access

A single value read from a register and written to a UART seems comprehensible.

What if that value is a memory pointer for *privileged* access?

This enables memory safety issues we have had for decades.

From Registers to Memory Access

A single value read from a register and written to a UART seems comprehensible.

What if that value is a memory pointer for *privileged* access?

This enables memory safety issues we have had for decades.

Best solution: Remove the idea from your design.

From Registers to Memory Access

A single value read from a register and written to a UART seems comprehensible.

What if that value is a memory pointer for *privileged* access?

This enables memory safety issues we have had for decades.

Best solution: Remove the idea from your design.

What else can we do?

Research & Development

Capabilities

Capabilities

*One of the key words that describes capabilities is **unforgeable**. A pointer in C is forgeable, because untrusted code could cast an integer to a pointer, thus forging access to whatever that pointer value points to.*

Capabilities

> *One of the key words that describes capabilities is **unforgeable**. A pointer in C is forgeable, because untrusted code could cast an integer to a pointer, thus forging access to whatever that pointer value points to.*

https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-capabilities.md

Capabilities

> *One of the key words that describes capabilities is **unforgeable**. A pointer in C is forgeable, because untrusted code could cast an integer to a pointer, thus forging access to whatever that pointer value points to.*

https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-capabilities.md

CHERI (Capability Hardware Enhanced RISC Instructions)

> *is a joint research project of SRI International and the University of Cambridge to revisit fundamental design choices in hardware and software to dramatically improve system security*

https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/
https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/creating-the-morello-technology-demonstrator

Defending Software

Defending Software

What can be done in software, what can go in hardware?

Defending Software

What can be done in software, what can go in hardware?

Answering this question needs evaluation and experience.

Defending Software

What can be done in software, what can go in hardware?

Answering this question needs evaluation and experience.

Generic Tagging for RISC-V Binaries

> *COGENT removes the burden of compiler development from RISC-V hardware defenses that rely on embedding instruction meta-data into binaries*

https://arxiv.org/pdf/2212.05614.pdf

Hardware Vulnerabilities

Hardware Vulnerabilities

Meltdown and Spectre

exploit critical vulnerabilities in modern processors.

https://meltdownattack.com/

Hardware Vulnerabilities

Meltdown and Spectre

exploit critical vulnerabilities in modern processors.

https://meltdownattack.com/

Microarchitectural fault attacks

exploit the physical imperfections of modern computer systems.

Software-based Microarchitectural Attacks, Daniel Gruss, PhD Thesis
https://arxiv.org/pdf/1706.05973.pdf

Hardware Vulnerabilities

Meltdown and Spectre

exploit critical vulnerabilities in modern processors.

https://meltdownattack.com/

Microarchitectural fault attacks

exploit the physical imperfections of modern computer systems.

Software-based Microarchitectural Attacks, Daniel Gruss, PhD Thesis
https://arxiv.org/pdf/1706.05973.pdf

Micro-architectural side-channel attacks refer to a side-channel attack that exploit information leakage from the hardware infrastructure itself.

https://orenlab.sise.bgu.ac.il/AttacksonImplementationsCourseBook/06_Cache_Attacks_Guest_Lecture

Hardware Security

Hardware Security

Trusted Execution Environment (TEE)

> *The TEE is a secure area of the main processor of a connected device that ensures sensitive data is stored, processed and protected in an isolated and trusted environment. As such, it offers protection against software attacks generated in the Rich Operating System (Rich OS).*

https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Environment-15May2018.pdf

Hardware Security

Trusted Execution Environment (TEE)

> *The TEE is a secure area of the main processor of a connected device that ensures sensitive data is stored, processed and protected in an isolated and trusted environment. As such, it offers protection against software attacks generated in the Rich Operating System (Rich OS).*

https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Environment-15May2018.pdf

Confidential Computing

Idea: Process data on remote infrastructure without exposing it to the provider or other parties involved.
https://confidentialcomputing.io/

Getting Started With Hardware Design

Getting Started With Hardware Design

Talks

- Combat complexity - build your own open OS and hardware; Michael Engel, foss-north 2021
  https://conf.tube/w/p/b9a072ab-1c4d-4912-905c-3f68096582ca?playlistPosition=14
- The Genius of RISC-V Microprocessors; Erik Engheim, ACCU 2022
  https://www.youtube.com/watch?v=L9jvLsvkmdM
- Linux on Open Source Hardware with Open Source chip design; Drew Fustini, 36C3
  https://www.youtube.com/watch?v=mnOBTD9dgsg

Getting Started With Hardware Design

Talks

- Combat complexity - build your own open OS and hardware; Michael Engel, foss-north 2021
  https://conf.tube/w/p/b9a072ab-1c4d-4912-905c-3f68096582ca?playlistPosition=14
- The Genius of RISC-V Microprocessors; Erik Engheim, ACCU 2022
  https://www.youtube.com/watch?v=L9jvLsvkmdM
- Linux on Open Source Hardware with Open Source chip design; Drew Fustini, 36C3
  https://www.youtube.com/watch?v=mnOBTD9dgsg

Literature

- https://opencircuitsbook.com
- Patterson and Hennessy - Computer Organization and Design RISC-V Edition: The Hardware Software Interface

Design Your Own Computer

Design Your Own Computer

FPGA Boards

- OrangeCrab https://1bitsquared.de/products/orangecrab
- ULX3S https://radiona.org/ulx3s/

Design Your Own Computer

FPGA Boards

- OrangeCrab https://1bitsquared.de/products/orangecrab
- ULX3S https://radiona.org/ulx3s/

Chip Design

- LiteX (SoC framework) https://github.com/enjoy-digital/litex
- FuseSoC (package manager) http://fusesoc.net/
- https://github.com/T-head-Semi/openc906 (e.g., in D1 and BL808)
- Libre SoC https://libre-soc.org/
- FOSSi Foundation https://www.fossi-foundation.org/
- Zero to ASIC Course https://www.zerotoasiccourse.com/

Design Your Own Computer

FPGA Boards

- OrangeCrab https://1bitsquared.de/products/orangecrab
- ULX3S https://radiona.org/ulx3s/

Chip Design

- LiteX (SoC framework) https://github.com/enjoy-digital/litex
- FuseSoC (package manager) http://fusesoc.net/
- https://github.com/T-head-Semi/openc906 (e.g., in D1 and BL808)
- Libre SoC https://libre-soc.org/
- FOSSi Foundation https://www.fossi-foundation.org/
- Zero to ASIC Course https://www.zerotoasiccourse.com/

Fabbing

https://developers.google.com/silicon

Will Your Design be a Good Design?

Follow Me

https://github.com/orangecms
https://twitter.com/orangecms
https://twitch.tv/cyrevolt
https://youtube.com/@cyrevolt

Daniel Maslowski

https://github.com/platform-system-interface/psi-spec

https://metaspora.org/platform-system-interface-computing-as-whole.pdf