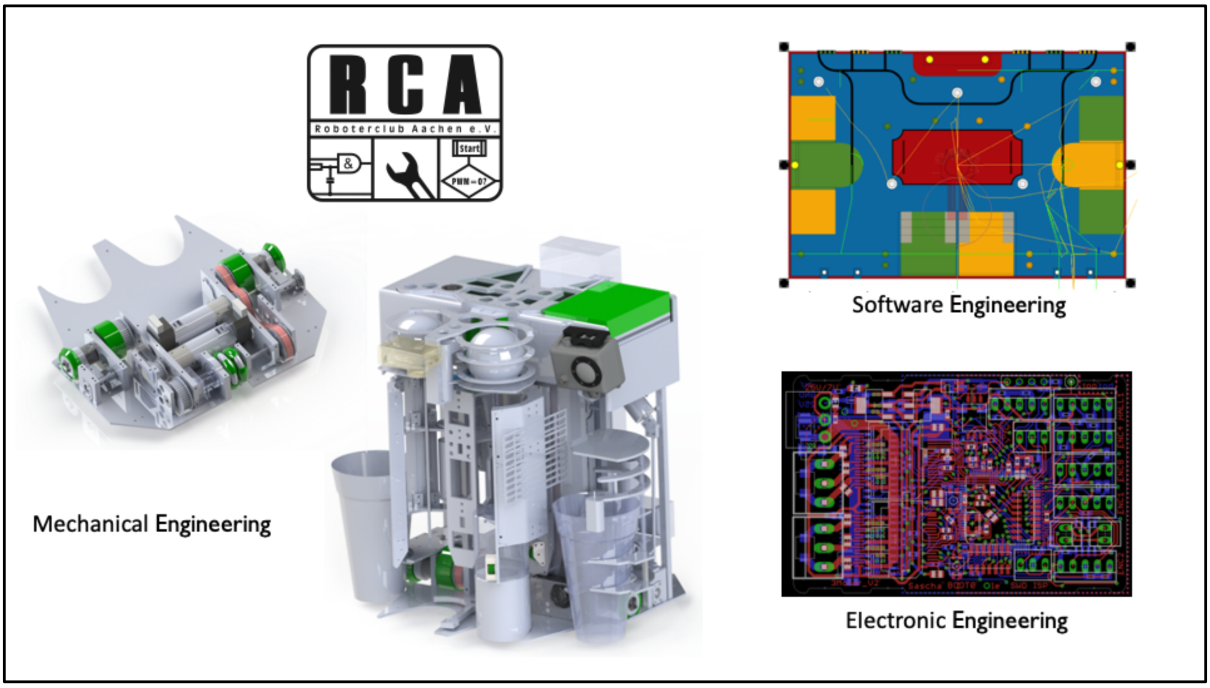
Improving Embedded Software with Data Science

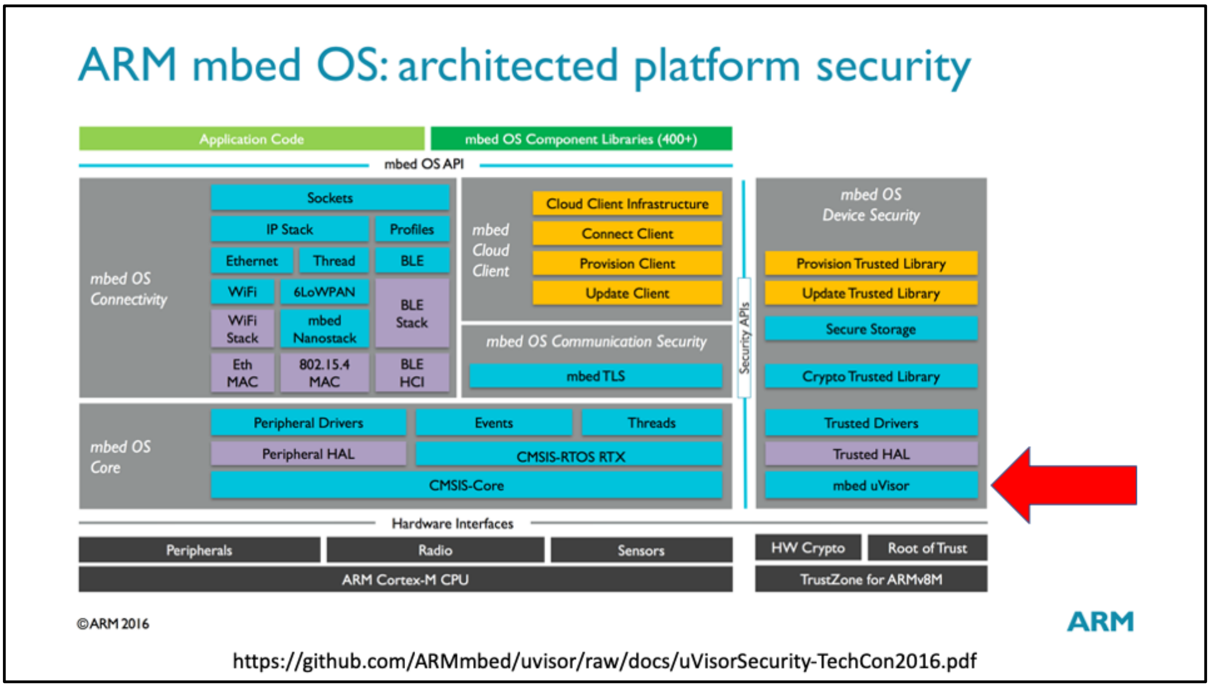
Niklas Hauser

DLR Erfahrungsaustausch 2018

According to my parents I study something with computers.



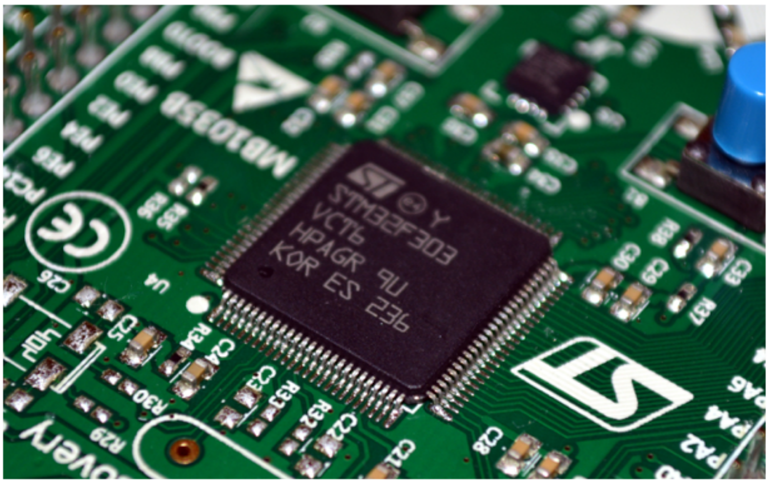
But I also build autonomous robots in my spare time.



2.5y ago, I got hired by ARM to work on mbed OS Security, specifically uVisor.
I spent almost 2y on the lowest levels of the ARMv7-M and ARMv8-M architectures.
I’ve seen ALL the dirt of Cortex-M and I still (mostly) love it.

Then I quit work to hack on the largest research model railway in Germany.
YOLO.

Automation, Embedded, Data Science



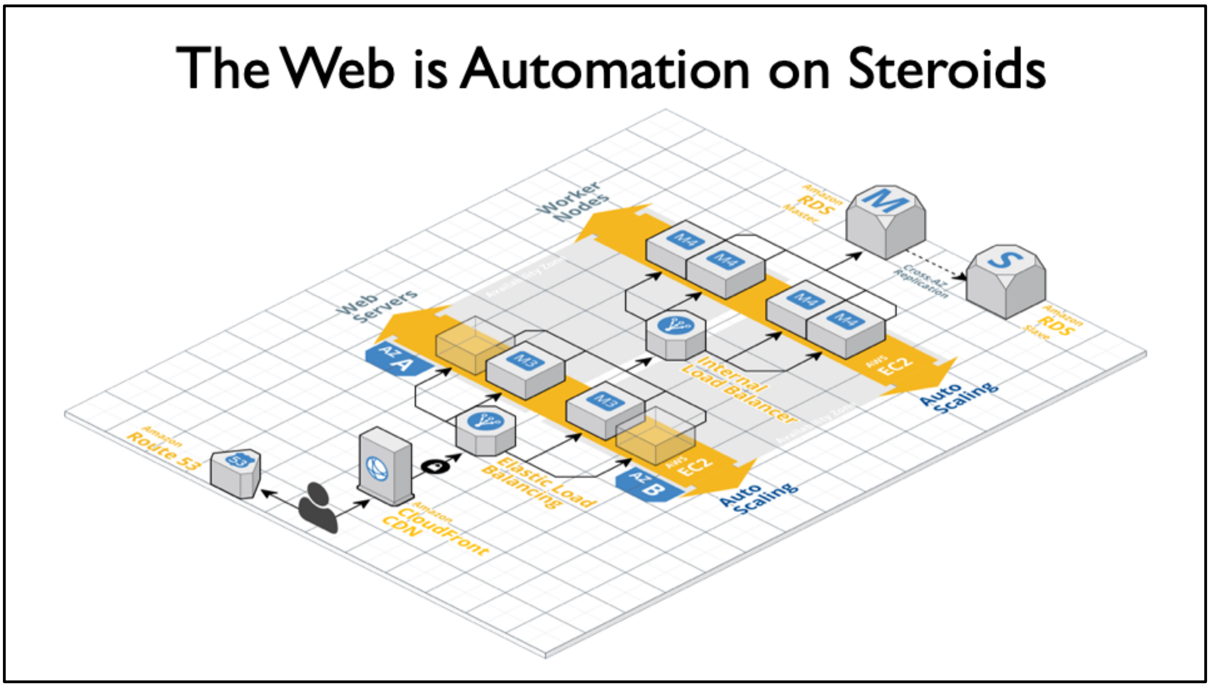
This is an introduction talk, so I'll keep the technical details light and mostly talk about concepts and ideas.

EMBEDDED: For the purpose of this talk I mean MICROCONTROLLERS, specifically AVR and ARM Cortex-M.

Automation is the Key to Productivity

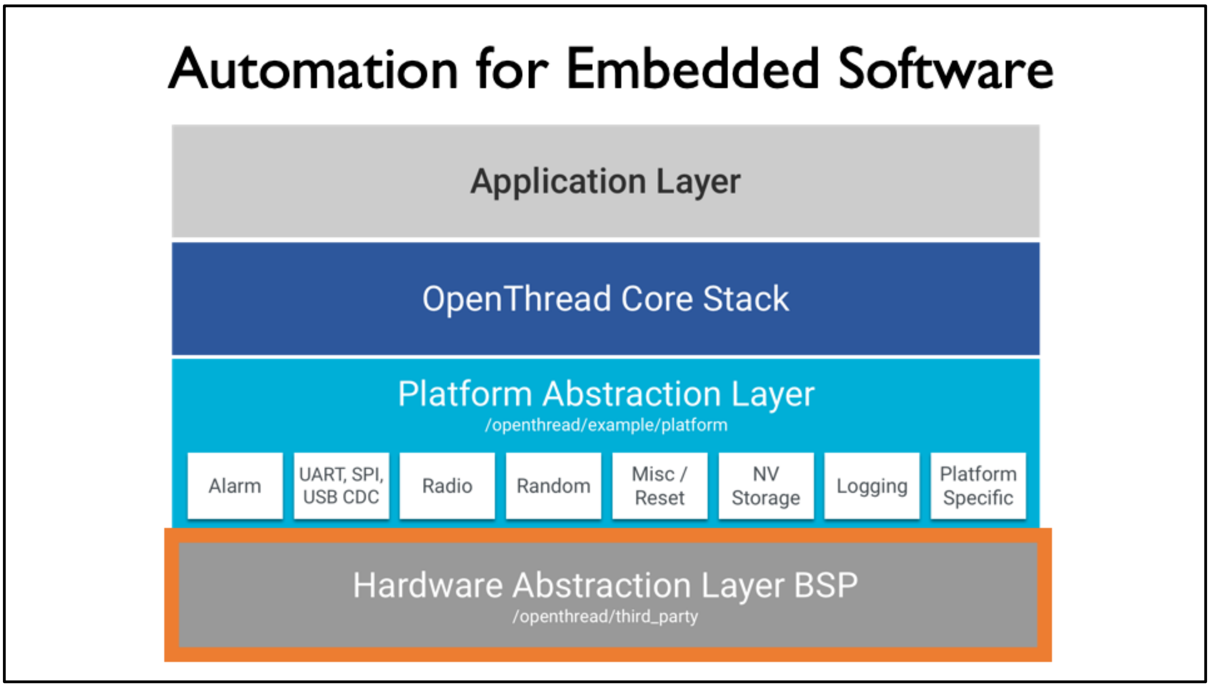
There many types of automation.

It evolved from a pure mechanization of labor to globe-spanning just-in-time industries.
This massive complexity only scaled with the introduction of computers.



The Web is the largest Automaton we've ever built!
Everything you know about industrial automation, the Web does at >10x scale.

The key here is an asymmetry of effort: The developer configures the automation once, and then it just works™.



What similar technology can I use for embedded software to get similar scaling benefits?

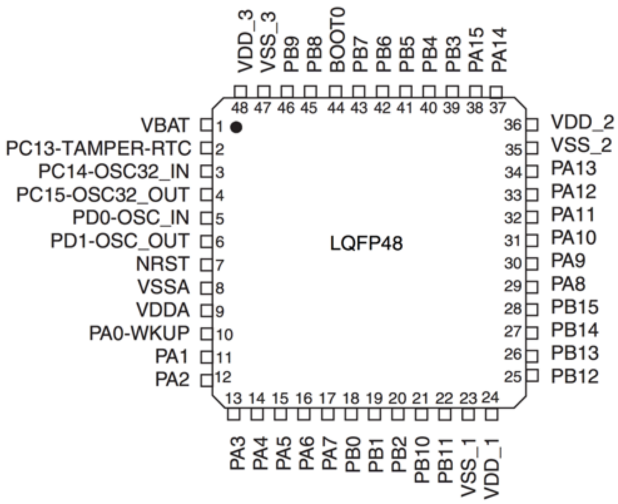
What needs to be scaled in embedded software? PORTING

HALs are usually provided by the vendor, to abstract hardware differences between devices.
USUALLY CODED BY HAND!

This is an insane BOTTLENECK.

PAL = Problem Anderer Leute

Device pinout definitions



A typical abstraction of the HAL is to provide functions to access PINS.
A single device can be packages in different ways, so the pins may differ.

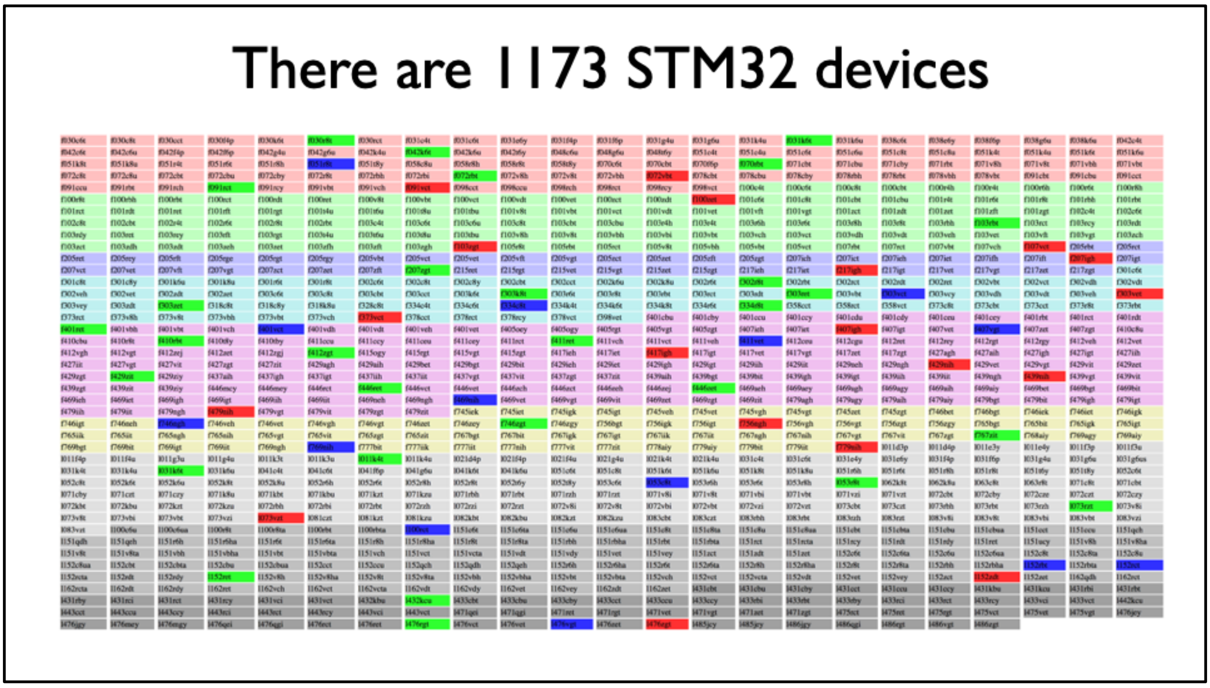
Mbed OS is ported manually (!)

```
// TIM4 cannot be used because already used by the us_ticker
MBED_WEAK const PinMap PinMap_PWM[] = {
    {PA_1,  PWM_2, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 2, 0)}, // TIM2_CH2 - Default
    {PA_2,  PWM_2, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 3, 0)}, // TIM2_CH3 - Default (warning: not connected
    {PA_3,  PWM_2, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 4, 0)}, // TIM2_CH4 - Default (warning: not connected
    {PA_6,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 1, 0)}, // TIM3_CH1 - Default
    {PA_7,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 2, 0)}, // TIM3_CH2 - Default
//  {PA_7,  PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 6, 1, 1)}, // TIM1_CH1N - GPIO_PartialRemap_TIM1
    {PA_8,  PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 1, 0)}, // TIM1_CH1 - Default
    {PA_9,  PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 2, 0)}, // TIM1_CH2 - Default
    {PA_10, PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 3, 0)}, // TIM1_CH3 - Default
    {PA_11, PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 4, 0)}, // TIM1_CH4 - Default
    {PA_15, PWM_2, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 8, 1, 0)}, // TIM2_CH1_ETR - GPIO_FullRemap_TIM2

    {PB_0,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 3, 0)}, // TIM3_CH3 - Default
//  {PB_0,  PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 6, 2, 1)}, // TIM1_CH2N - GPIO_PartialRemap_TIM1
    {PB_1,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 4, 0)}, // TIM3_CH4 - Default
//  {PB_1,  PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 6, 3, 1)}, // TIM1_CH3N - GPIO_PartialRemap_TIM1
    {PB_3,  PWM_2, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 8, 2, 0)}, // TIM2_CH2 - GPIO_FullRemap_TIM2
    {PB_4,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 7, 1, 0)}, // TIM3_CH1 - GPIO_PartialRemap_TIM3
    {PB_5,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 7, 2, 0)}, // TIM3_CH2 - GPIO_PartialRemap_TIM3
//  {PB_6,  PWM_4, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 1, 0)}, // TIM4_CH1 - Default (used by ticker)
//  {PB_7,  PWM_4, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 2, 0)}, // TIM4_CH2 - Default (used by ticker)
```

In mbed OS this table describes the pin signal connections to the internal peripherals.
You can see some manually commmented out lines.

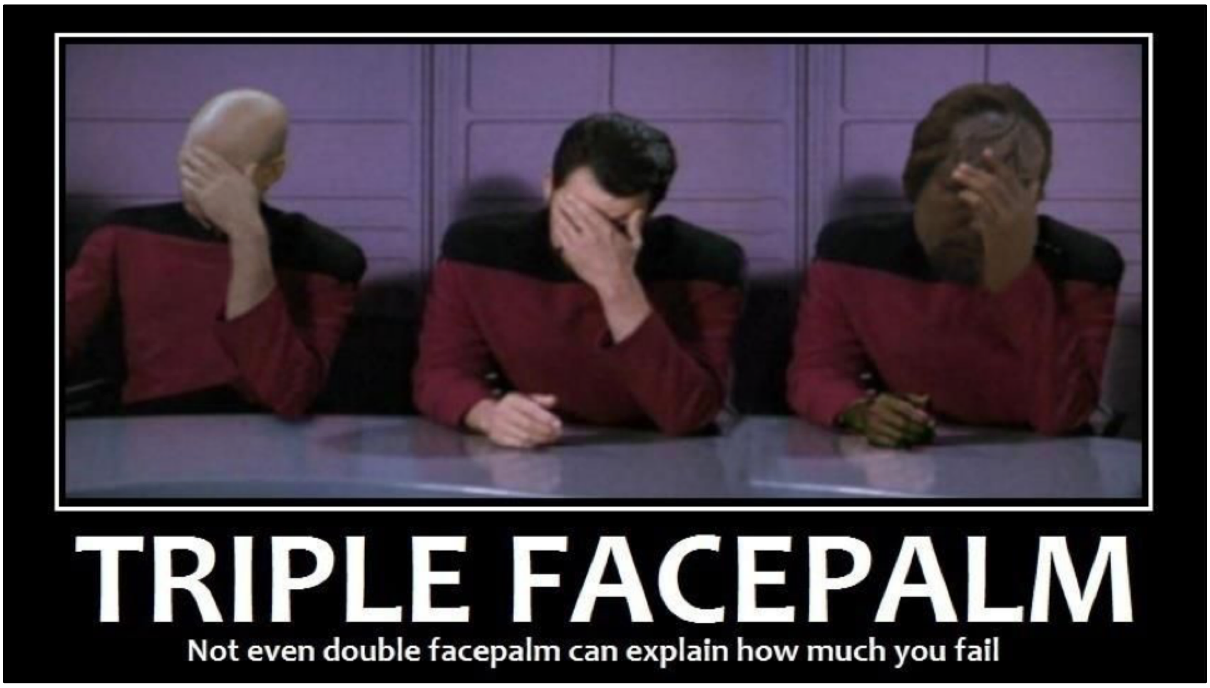
ST has 4-6 full-time engineers just for porting Mbed OS to STM32.
They do it completely by hand.



ST adds new devices every couple of months.
Most of them differ in memory size, package and peripherals.

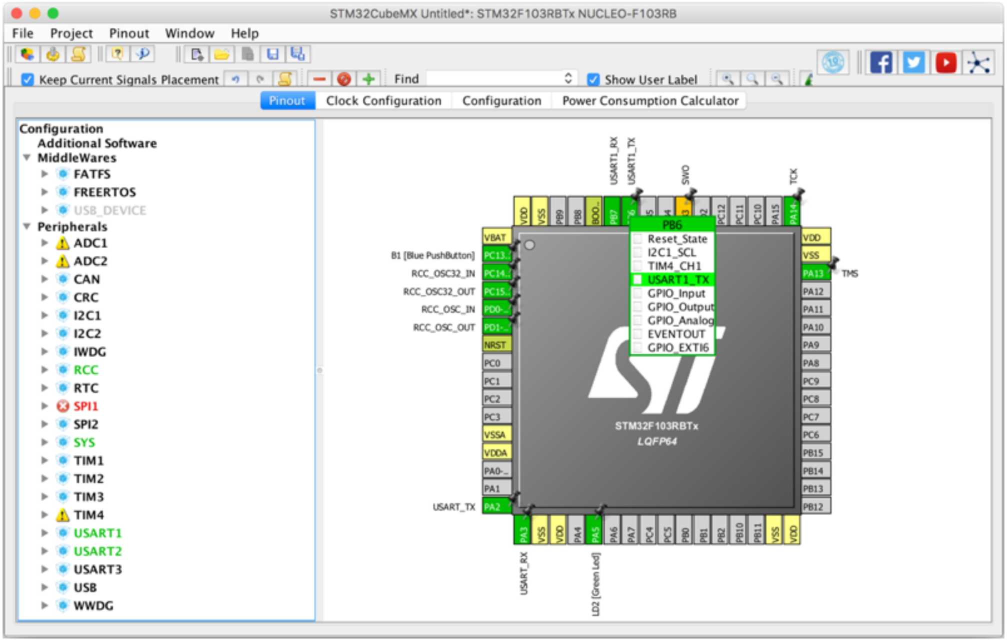
- 55 manually created versions of the pinout data
- 51 Linkerscripts (some manually patched)
- ~200 startup scripts (some manually patched)

- We needed to modify the linker- and startup script for uVisor



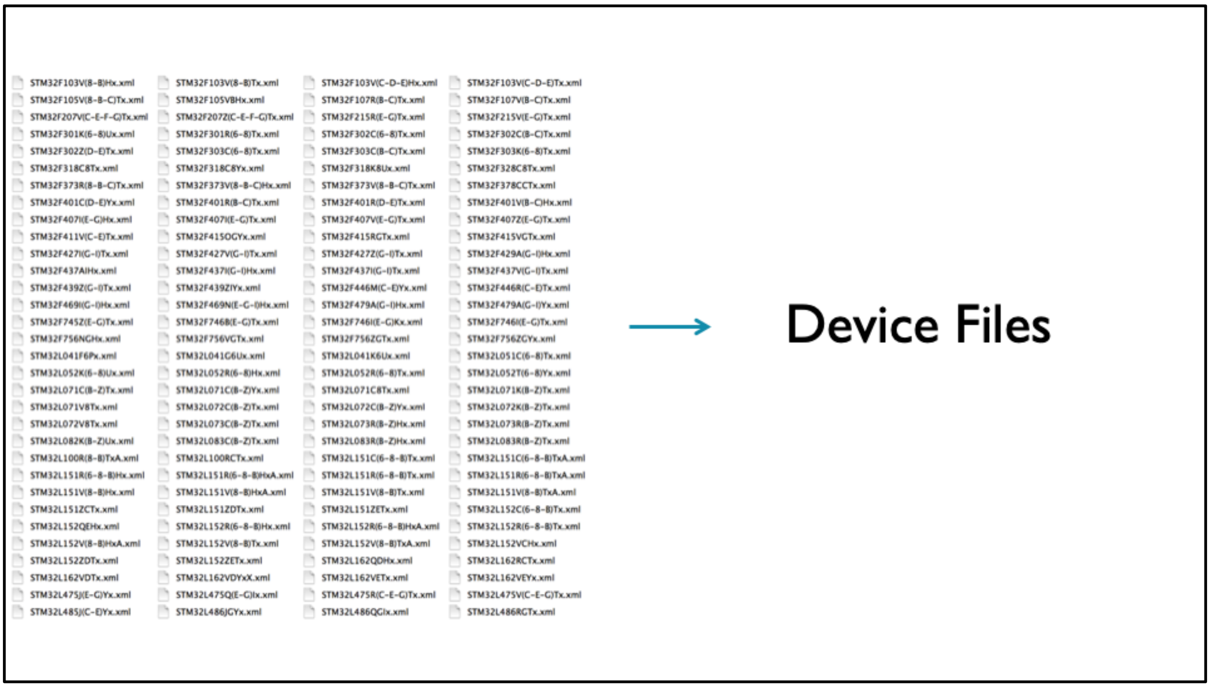
X Billion IoT devices by 2035?
HOW ARE YOU GOING TO MAINTAIN THIS?!?

Let‘s think about this problem in a calm and structured manner.
Tea?



STMicro publishes a GUI App to initialize your STM32.
It‘s pretty good, the usability of this is great, you can just visually configure all your pins.
Understand conflicts between peripheral signal connections etc.

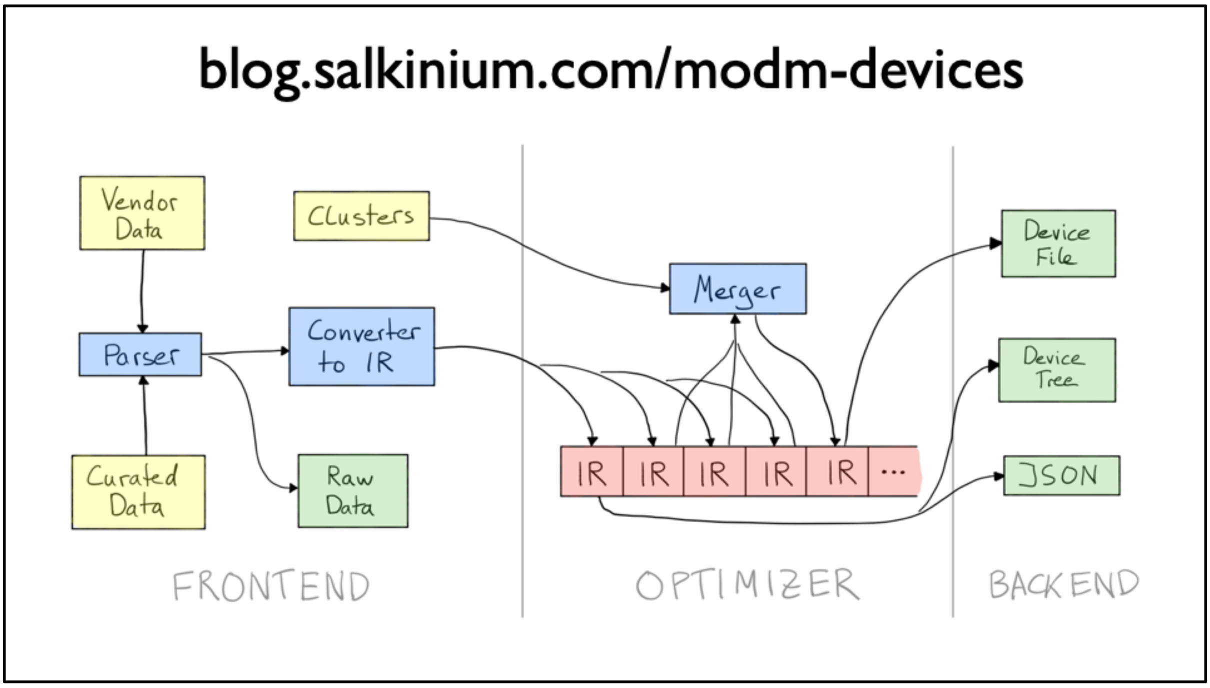
But I got curious, how does this work?
It‘s backed by a lot of DATA!



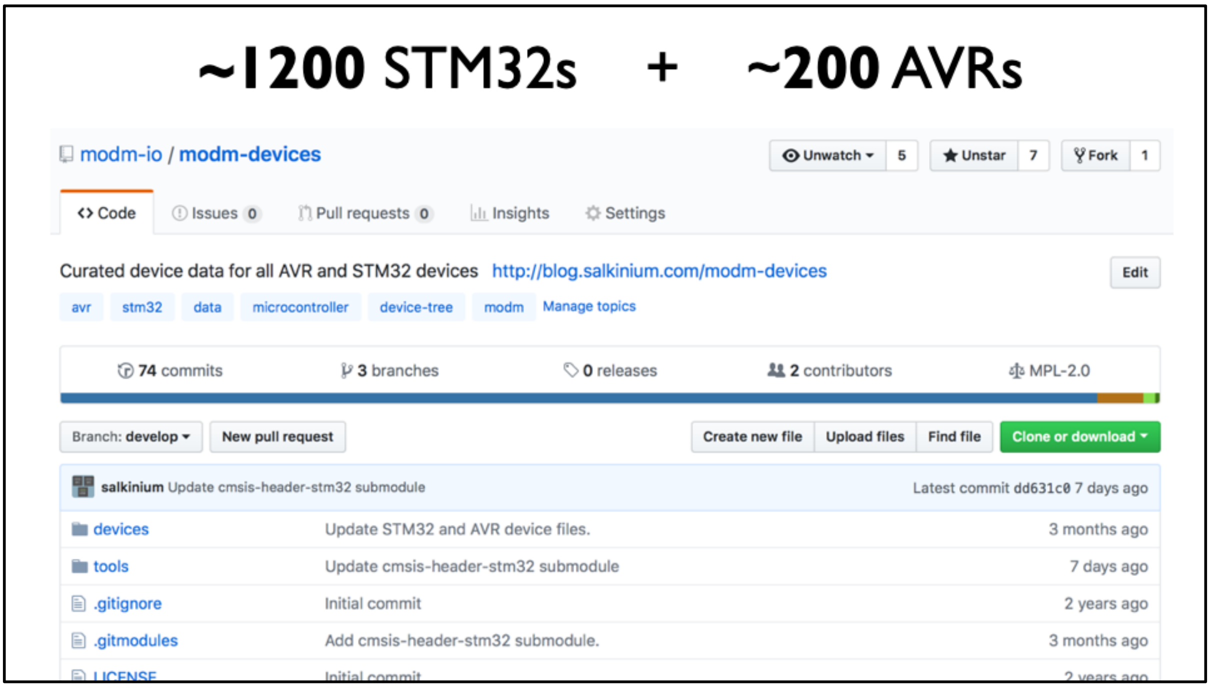
70MB of XML files, with a lot of device details.

modm-devices contains

- CPU: Type and Interrupt Vector Table
- Memories: Flash, RAM, Backup
- Peripherals: Type and Instances
- Gpio: Name and Signals

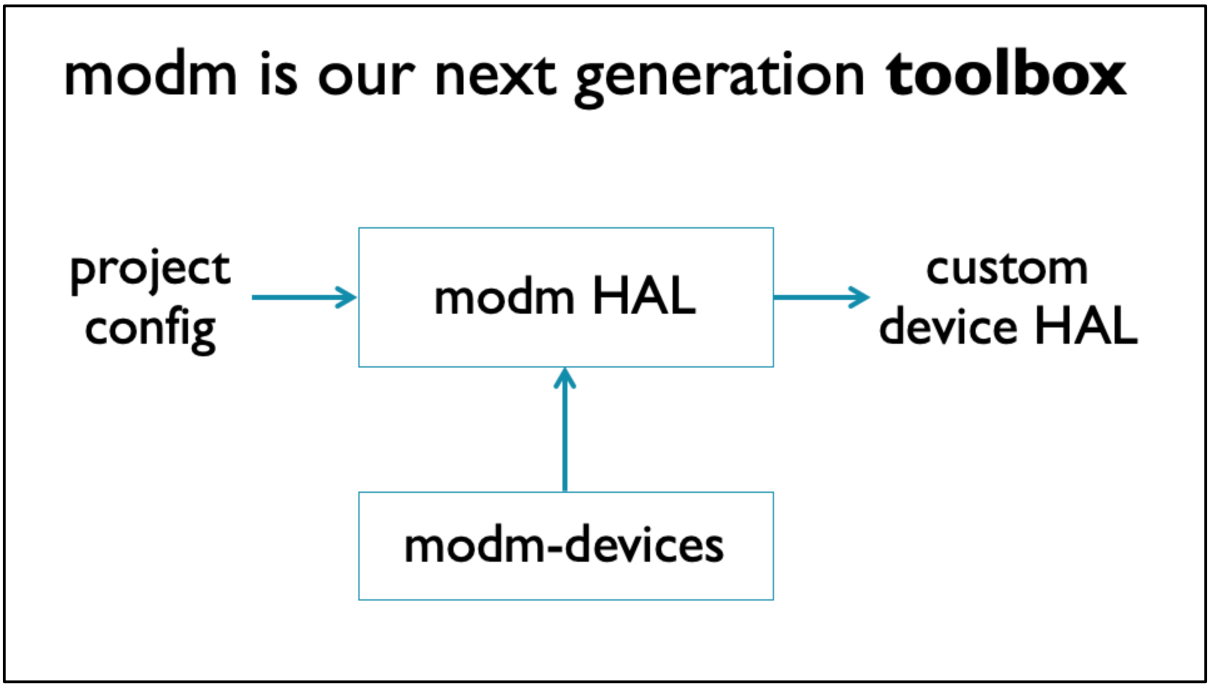


The reality is a bit more involved.

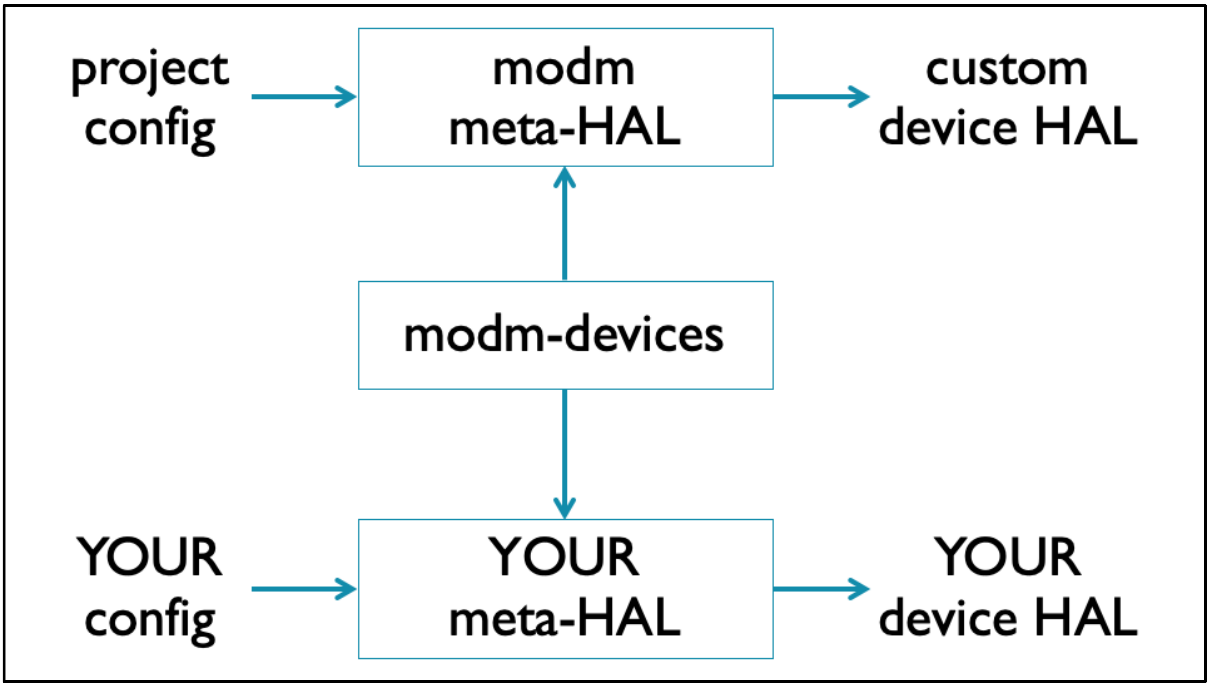


YOU can use this data too, it‘s on GitHub

Fabian and I worked on modm for the last two years.
It's a C++ HAL for STM32 and AVR.
We use it to build the robot software for the Roboterclub Aachen.



modm is our INTERPRETATION of modm-devices.
It‘s a library GENERATOR, we specify a target id and it generates the HAL for us.
We have ported ~80 AVRs and ~850 STM32 so far.



Share the data, not the HAL.
Easier to customize your HAL to your specific needs.

THIS IS LANGUAGE INDEPENDENT!
You generate C, Go, Rust, TEXT, you can also just generate documentation.

You can generate just ONE file, then gradually expand.
Particularly useful to replace pain-points in existing code bases.

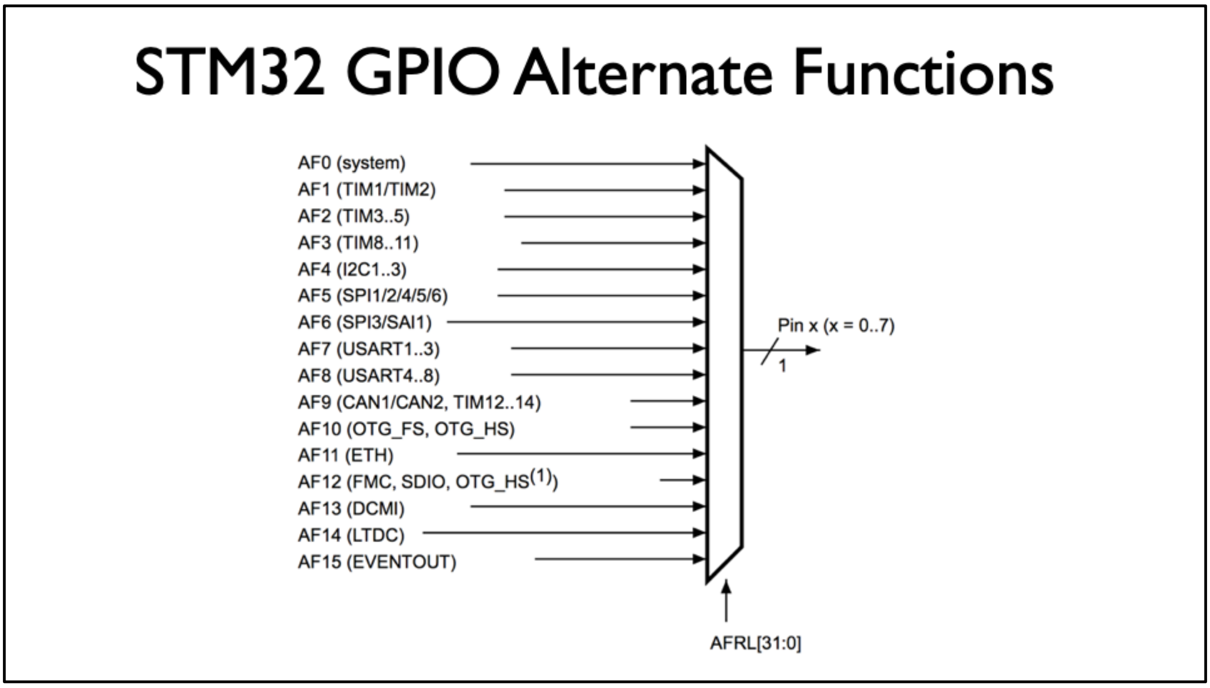
Vertical vs. Horizontal Porting

Feature	STM32F0	STM32F1	STM32F3	STM32F4	STM32F7
Core	✓	✓	✓	✓	✓
GPIO	✓	✓	✓	✓	✓
Clock	✓	✓	✓	✓	✓
UART	✓	✓	✓	✓	✓
SPI	✓	✓	✓	✓	✓

There is a change in how you port your targets now:
Before: For each target implement the feature.
Now: for each feature port to all targets

I glossed over the details of the actual code generation.
You can read up on this on our website with examples.
Installation guide, getting started guide, explanations how it works.

I‘m not here to sell you on our HAL, I want to show you specific problems and how we used our toolbox to solve them.



I‘ve already touched on signal connections.
Each pin has a number of peripherals that it can be connected to.

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11	I2C1/2/3	SPI1/2/3/4/5/6	SPI2/3/SAI1	SPI2/3/USART1/2/3	USART6/UART4/5/7/8	CAN1/2/TIM12/13/14/QUADSPI/LCD	QUADSPI/OTG2_HS/OTG1_FS	ETH	FMC/SDIO/OTG2_FS	DCMI/DSI HOST	LCD	SYS
Port A	PA0	-	TIM2_CH1/TIM2_ETR	TIM5_CH1	TIM8_ETR	-	-	-	USART2_CTS	UART4_TX	-	-	ETH_MII_CRS	-	-	-	EVENTOUT
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_RTS	UART4_RX	QUADSPI_BK1_IO3	-	ETH_MII_RX_CLK/ETH_RMII_REF_CLK	-	-	LCD_R2	EVENTOUT
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_TX	-	-	-	ETH_MDIO	-	-	LCD_R1	EVENTOUT
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-	-	-	USART2_RX	-	LCD_B2	OTG_HS_ULPI_D0	ETH_MII_COL	-	-	LCD_B5	EVENTOUT
	PA4	-	-	-	-	-	SPI1_NSS	SPI3_NSS/I2S3_WS	USART2_CK	-	-	-	-	OTG_HS_SOF	DCMI_HSYNC	LCD_VSYNC	EVENTOUT
	PA5	-	TIM2_CH1/TIM2_ETR	-	TIM8_CH1N	-	SPI1_SCK	-	-	-	-	OTG_HS_ULPI_CK	-	-	-	LCD_R4	EVENTOUT
	PA6	-	TIM1_BKIN	TIM3_CH1	TIM8_BKIN	-	SPI1_MISO	-	-	-	TIM13_CH1	-	-	-	DCMI_PIXCLK	LCD_G2	EVENTOUT
	PA7	-	TIM1_CH1N	TIM3_CH2	TIM8_CH1N	-	SPI1_MOSI	-	-	-	TIM14_CH1	QUADSPI_CLK	ETH_MII_RX_DV/ETH_RMII_CRS_DV	FMC_SDNWE	-	-	EVENTOUT
	PA8	MCO1	TIM1_CH1	-	-	I2C3_SCL	-	-	USART1_CK	-	-	OTG_FS_SOF	-	-	-	LCD_R6	EVENTOUT
	PA9	-	TIM1_CH2	-	-	I2C3_SMBA	SPI2_SCK/I2S2_CK	-	USART1_TX	-	-	-	-	-	DCMI_D0	-	EVENTOUT
	PA10	-	TIM1_CH3	-	-	-	-	-	USART1_RX	-	-	OTG_FS_ID	-	-	DCMI_D1	-	EVENTOUT
	PA11	-	TIM1_CH4	-	-	-	-	-	USART1_CTS	-	CAN1_RX	OTG_FS_DM	-	-	-	LCD_R4	EVENTOUT
	PA12	-	TIM1_ETR	-	-	-	-	-	USART1_RTS	-	CAN1_TX	OTG_FS_DP	-	-	-	LCD_R5	EVENTOUT
	PA13	JTMS-SWDIO	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENTOUT
	PA14	JTCK-SWCLK	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENTOUT
	PA15	JTDI	TIM2_CH1/TIM2_ETR	-	-	-	SPI1_NSS	SPI3_NSS/I2S3_WS	-	-	-	-	-	-	-	-	EVENTOUT

These connections are hardcoded, and the possibilities are described in a very long table in the datasheet.
Several pages of tables.
You do NOT want to read through that.

Gpio Signal Connection API

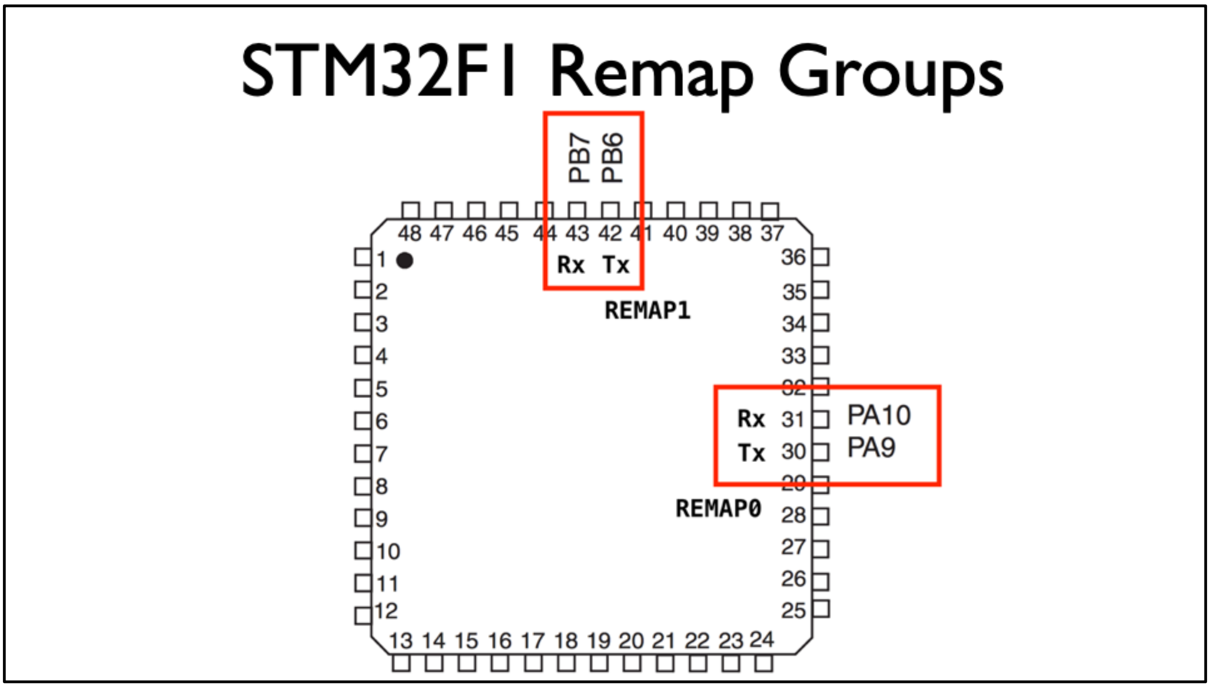
```
GpioB7::connect(Uart1);
GpioB6::connect(Uart1);
```

But this code breaks RX on PB7!

```
GpioB7::connect(Uart1);
GpioA9::connect(Uart1);
```

We came up with this API.
Just connect the pin to the peripheral.

Worked fine for a long time, until we ported to STM32F1!
Things suddenly broke.



On the STM32F1 the GPIOs can only remap in groups.
NO INDIVIDUAL REMAP POSSIBLE!

Our API has side-effects and the last call to set the group wins.
This is an implicit assumption of our HAL API:
Signal connections are independent of each other!

It's FALSE.

Implicit Assumptions

1. Signal connection is **independent** of Pin
2. Map is **unique**: Pin + Peripheral → Signal

Violation: LTDC signals B2 and B5 on GpioA3

There is another assumption:
The combination of Pin NAME and Peripheral NAME is enough to **uniquely** identify the Signal connections.

So we wrote these assumptions as a test and ran it over all devices.
Shows about 400 devices, the more red, the more violations.

An example violation: LCD display peripheral signals map two Blue signal lines onto the same pin.

Improved GPIO Connection API

1. Signal connection is **independent** of Pin
2. Map is **unique**: Pin + Peripheral → Signal

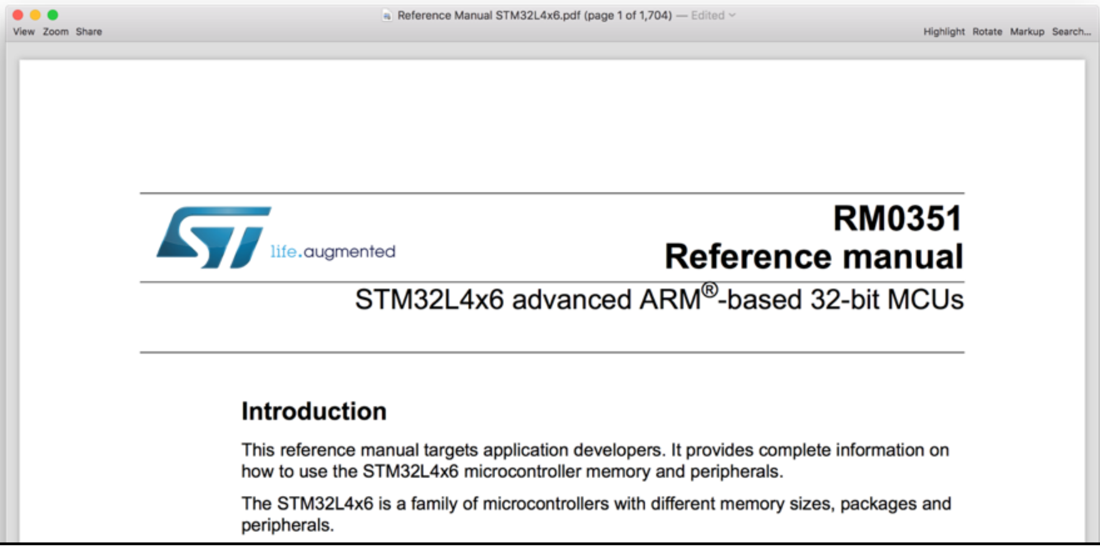
```
Uart1::connect(GpioB7::Rx,
               GpioB6::Tx);
```

~~Uart1::connect(GpioB7::Rx,~~
ERROR! ~~Gpio**A9**::Tx);~~

So we changed our API. Broke the entire code.
And this API now remaps both in a group.

Compiler checks group remap validity.

We need more data!



RM0351
Reference manual
STM32L4x6 advanced ARM®-based 32-bit MCUs

Introduction

This reference manual targets application developers. It provides complete information on how to use the STM32L4x6 microcontroller memory and peripherals.

The STM32L4x6 is a family of microcontrollers with different memory sizes, packages and peripherals.

The CubeMX data doesn‘t contain everything we want.
And it also doesn‘t scale to other vendors.
So let‘s have a look at a very verbose data source: THE REFERENCE MANUALS.

Let’s parse some PDFs

Table 39. Summary of the DMA1 requests for each channel

Request. number	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
0	ADC1	ADC2	ADC3	DFSDM1_ FLT0	DFSDM1_ FLT1	DFSDM1_ FLT2	DFSDM1_ FLT3
1	-	SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX	SAI2_A	SAI2_B
2	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
3	-	I2C3_TX	I2C3_RX	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX

This is a DMA channel to peripheral mapping.
This is data that‘s not available in the CubeMX dataset.
We need it anyways to provide a channel connection API.

PDF X-Ray Vision

Table 39. Summary of the DMA1 requests for each channel

Request number	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
0	ADC1	ADC2	ADC3	DFSDM1_ FLT0	DFSDM1_ FLT1	DFSDM1_ FLT2	DFSDM1_ FLT3
1	-	SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX	SAI2_A	SAI2_B
2	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
3	-	I2C3_TX	I2C3_RX	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX

I wrote a program to give me X-Ray vision of the PDF.
Read the Adobe PDF specification, it‘s quite fun.

PDF is a print format, it does not contain any semantical intofmraiton.
This is not a table. It is a bunch of lines and a bunch of text overlaid.

You can recognize tables fairly easily, and then translate their contents.

Table extraction is doable

Re-quest. num-ber	Chan nel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
0	ADC1	ADC2	ADC3	DFSD-M1_FLT0	DFSD-M1_FLT1	DFSD-M1_FLT2	DFSD-M1_FLT3
1	-	SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX	SAI2_A	SAI2_B
2	-	USAR-T3_TX	USAR-T3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
3	-	I2C3_TX	I2C3_RX	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX

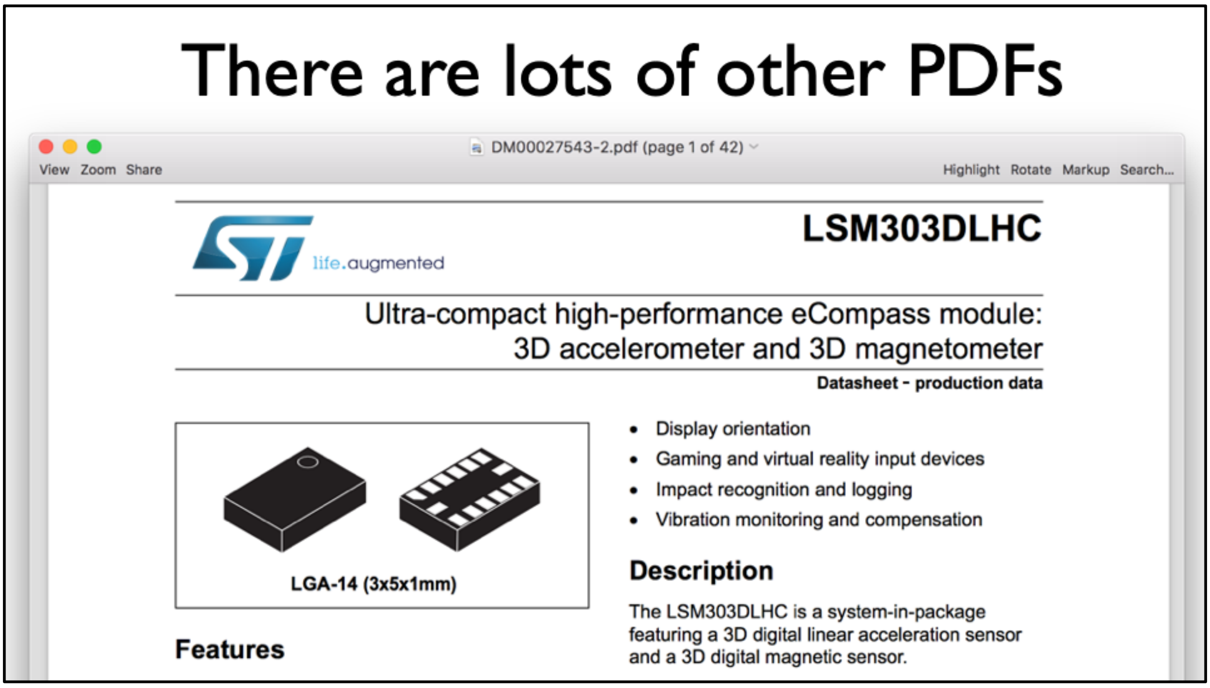
Understanding descriptions is very hard

This is a prototype extraction.
Tables are structured information by definition.
So it‘s somewhat easy to extract and use this data.

However, the textual descriptions of periphrals are very hard to turn into some kind of structured information.

The difficulty is not the table extraction it‘s the data cleanup and simplification.
You need to condense this information into something that easily usable for the developer.
There are hundreds of these datasheets + reference manuals.
This isn‘t easy.

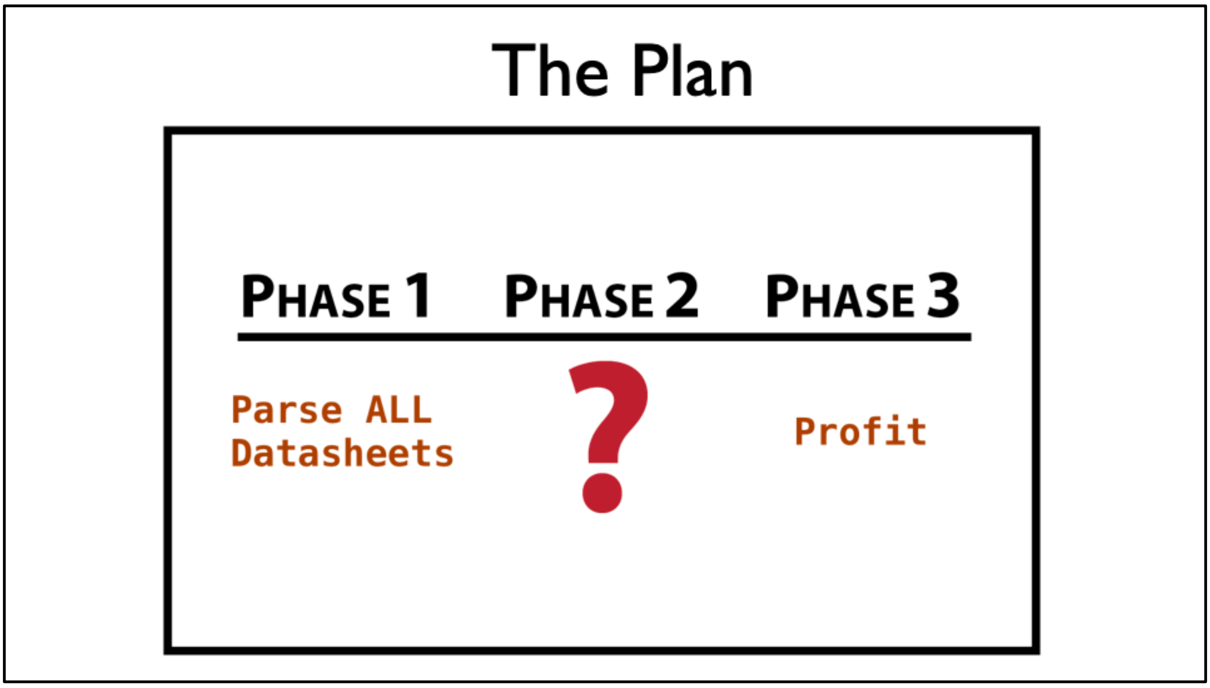
There are bugs in them, sometimes types in register names, sometimes worse.
It‘s going to be difficult to get some ground truth out of this.



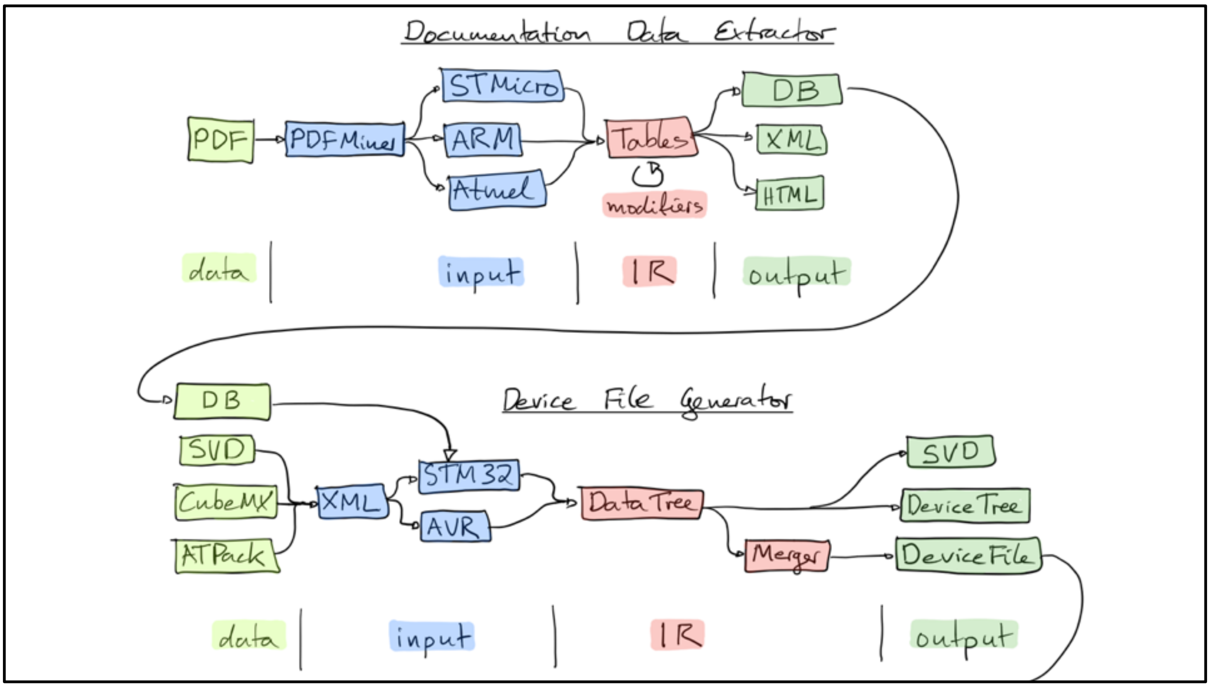
ST also produces SENSORS, which also have Datasheets.
They have the same formatting as the reference manuals, so you can just extract tables in there too.

And so we can build a database of sensors as well.
They are always the same: Memory Mapped IO via SPI/I2C.

They are by definition already platform-independent.
So why aren‘t there COMMON drivers for all platforms?
Abstract description of the protocol



This is really self-explanatory.



This is how I envision the future of Porting Embedded Software to hundreds of devices.

We parse ALL of the datasheets, THEN MAGIC, THEN PROFIT.

Thank you for listening!

Niklas Hauser: @salkinium (GitHub/Twitter)
(blog.)salkinium.com

modm.io
github.com/modm-io

EY MANN, WO IS' MEIN TSCHUNK?