

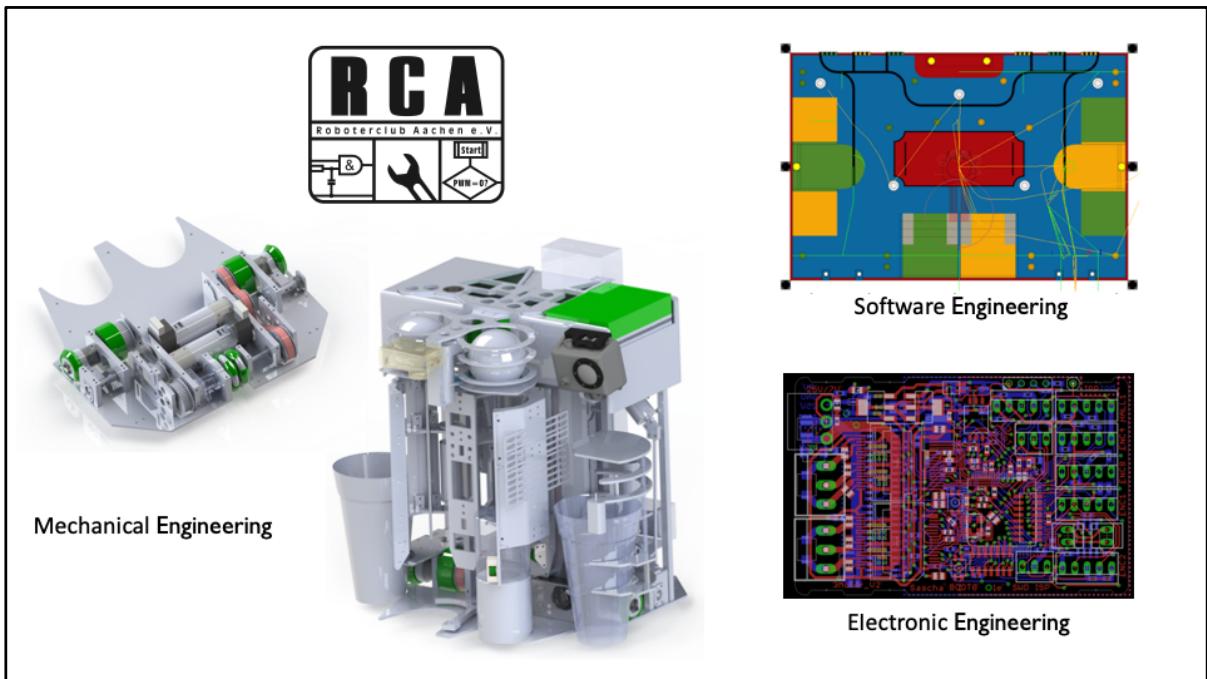
# Improving Embedded Software with Data Science

Niklas Hauser

DLR Erfahrungsaustausch 2018

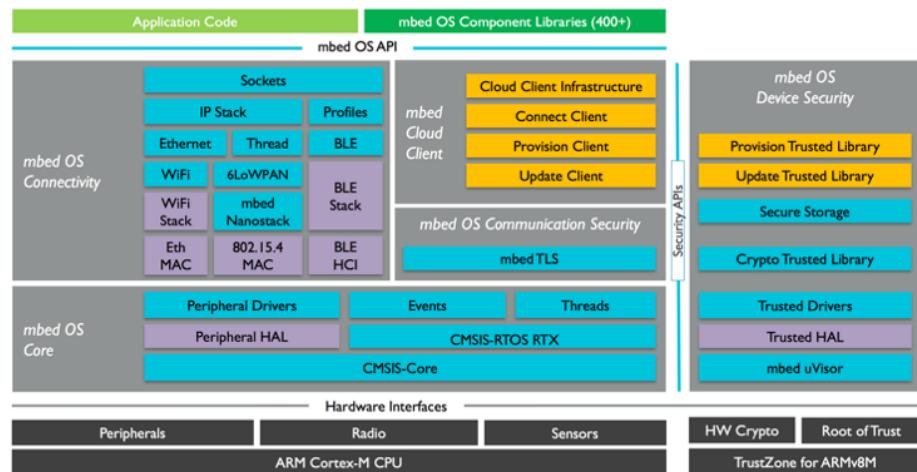


According to my parents I study something with computers.



But I also build autonomous robots in my spare time.

## ARM mbed OS: architected platform security

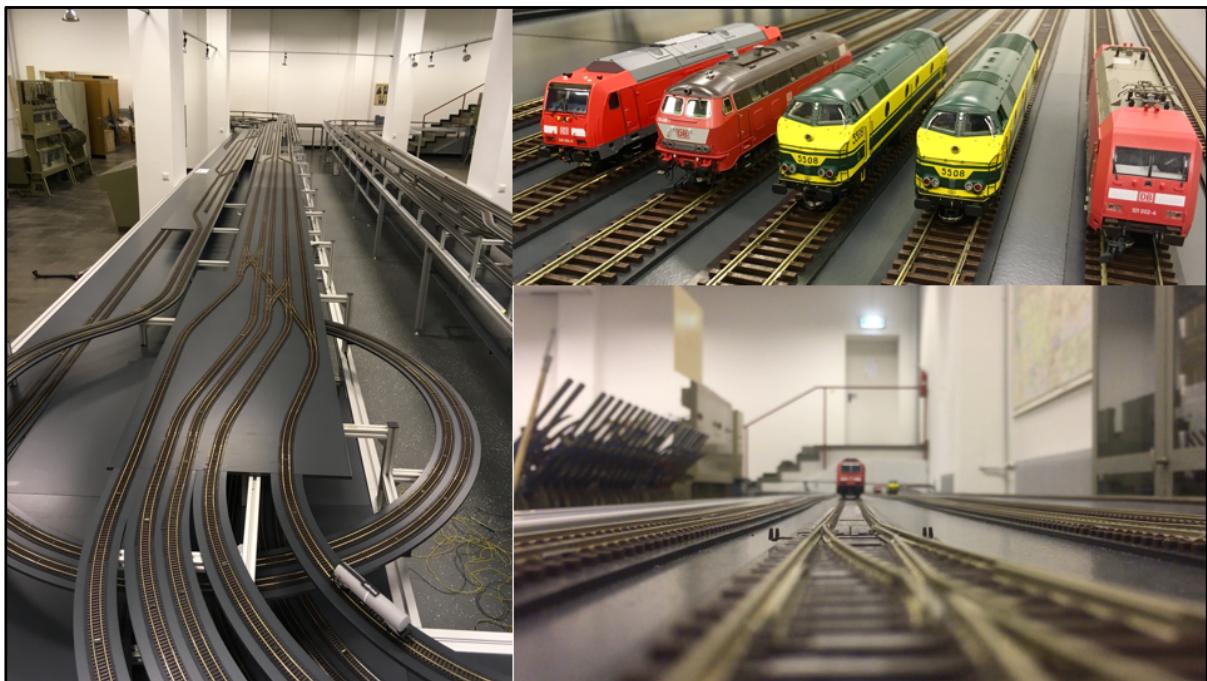


©ARM 2016

<https://github.com/ARMmbed/uvisor/raw/docs/uVisorSecurity-TechCon2016.pdf>

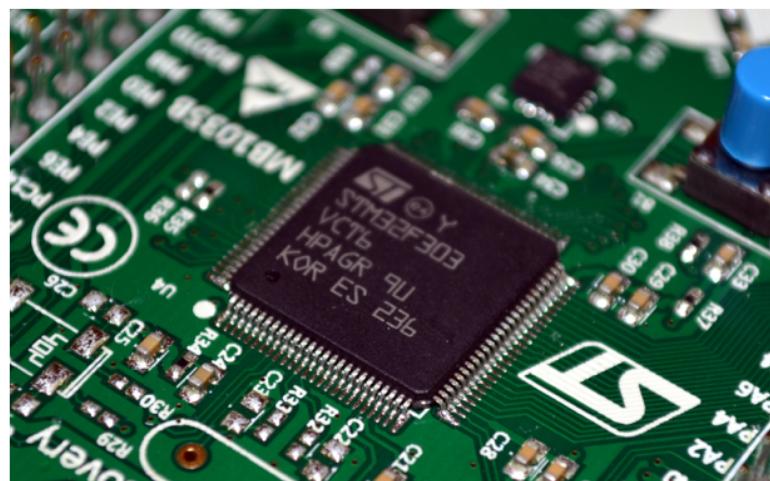
ARM

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

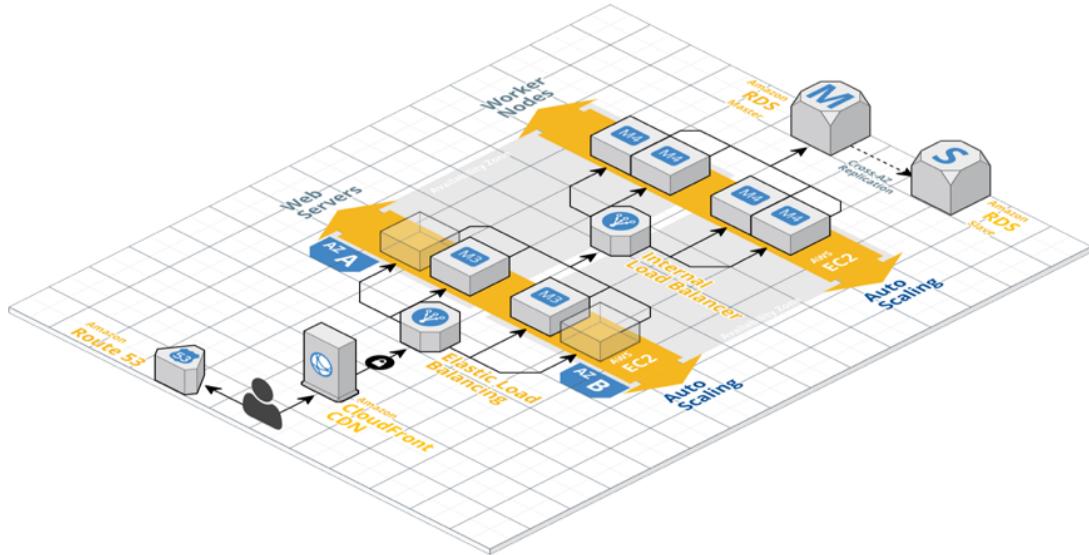


There many types of automation.

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.

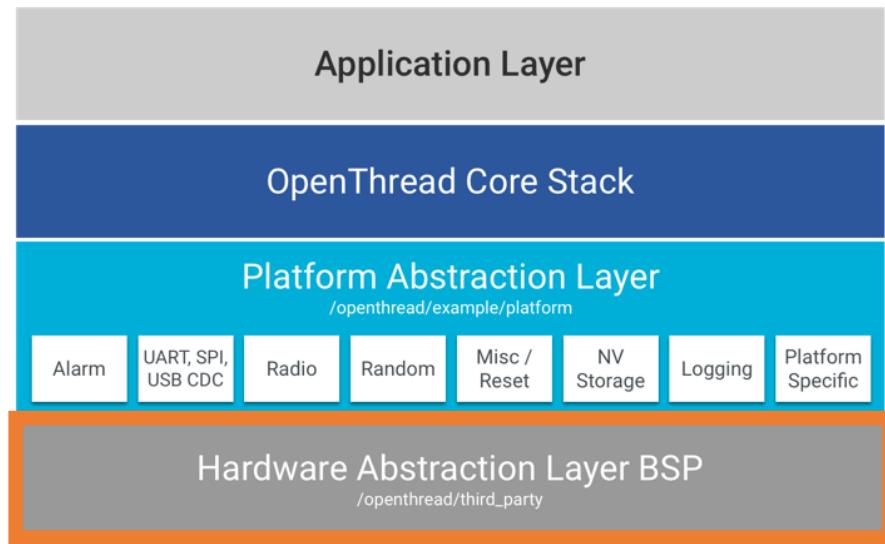
## The Web is Automation on Steroids



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™.

# Automation for Embedded Software



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

What needs to be scaled in embedded software? PORTING

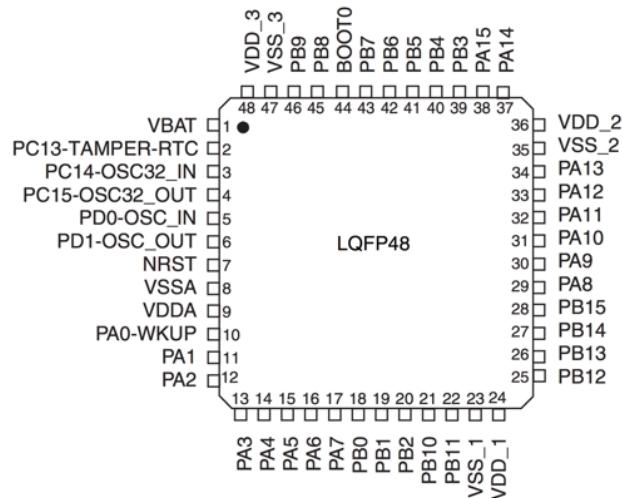
HALs are usually provided by the vendor, to abstract hardware differences between devices.

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 (!)

```
86 // TIM4 cannot be used because already used by the us_ticker
87 MBED_WEAK const PinMap PinMap_PWM[] = {
88     {PA_1,  PWM_2, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 2, 0)}, // TIM2_CH2 - Default
89     {PA_2,  PWM_2, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 3, 0)}, // TIM2_CH3 - Default (warning: not connected)
90     {PA_3,  PWM_2, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 4, 0)}, // TIM2_CH4 - Default (warning: not connected)
91     {PA_6,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 1, 0)}, // TIM3_CH1 - Default
92     {PA_7,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 2, 0)}, // TIM3_CH2 - Default
93     // {PA_7,  PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 6, 1, 1)}, // TIM1_CH1N - GPIO_PartialRemap_TIM1
94     {PA_8,  PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 1, 0)}, // TIM1_CH1 - Default
95     {PA_9,  PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 2, 0)}, // TIM1_CH2 - Default
96     {PA_10, PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 3, 0)}, // TIM1_CH3 - Default
97     {PA_11, PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 4, 0)}, // TIM1_CH4 - Default
98     {PA_15, PWM_2, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 8, 1, 0)}, // TIM2_CH1_ETR - GPIO_FullRemap_TIM2
99
100    {PB_0,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 3, 0)}, // TIM3_CH3 - Default
101    // {PB_0,  PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 6, 2, 1)}, // TIM1_CH2N - GPIO_PartialRemap_TIM1
102    {PB_1,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 4, 0)}, // TIM3_CH4 - Default
103    // {PB_1,  PWM_1, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 6, 3, 1)}, // TIM1_CH3N - GPIO_PartialRemap_TIM1
104    {PB_3,  PWM_2, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 8, 2, 0)}, // TIM2_CH2 - GPIO_FullRemap_TIM2
105    {PB_4,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 7, 1, 0)}, // TIM3_CH1 - GPIO_PartialRemap_TIM3
106    {PB_5,  PWM_3, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 7, 2, 0)}, // TIM3_CH2 - GPIO_PartialRemap_TIM3
107    // {PB_6,  PWM_4, STM_PIN_DATA_EXT(STM_MODE_AF_PP, GPIO_PULLUP, 0, 1, 0)}, // TIM4_CH1 - Default (used by ticker)
108    // {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 commented out lines.

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

# There are 1173 STM32 devices

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

## Mbed OS supports 55 STM32 devices

```
$ find targets/TARGET_STM -name "PeripheralPins.c"
targets/TARGET_STM/TARGET_STM32F0/TARGET_DISCO_F051R8/PeripheralPins.c
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F030R8/PeripheralPins.c
targets $ find . -name "STM32*.ld"
targets/TARGET_STM/TARGET_STM32F0/TARGET_DISCO_F051R8/device/TOOLCHAIN_GCC_ARM/STM32F0xx.ld
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F030R8/device/TOOLCHAIN_GCC_ARM/STM32F030X8.ld
targets $ find . -name "startup_stm32*.S"
targets/TARGET_STM/TARGET_STM32F0/TARGET_DISCO_F051R8/device/TOOLCHAIN_ARM_MICRO/startup_stm32f0
targets/TARGET_STM/TARGET_STM32F0/TARGET_DISCO_F051R8/device/TOOLCHAIN_ARM_STD/startup_stm32f05
targets/TARGET_STM/TARGET_STM32F0/TARGET_DISCO_F051R8/device/TOOLCHAIN_GCC_ARM/startup_stm32f05
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F030R8/device/TOOLCHAIN_ARM_MICRO/startup_stm32f0
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F030R8/device/TOOLCHAIN_ARM_STD/startup_stm32f0
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F030R8/device/TOOLCHAIN_GCC_ARM/startup_stm32f0
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F030R8/device/TOOLCHAIN_IAR/startup_stm32f030x8
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F031K6/device/TOOLCHAIN_ARM_MICRO/startup_stm32f0
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F031K6/device/TOOLCHAIN_ARM_STD/startup_stm32f0
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F031K6/device/TOOLCHAIN_GCC_ARM/startup_stm32f0
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F031K6/device/TOOLCHAIN_IAR/startup_stm32f031x6
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F042K6/device/TOOLCHAIN_ARM_MICRO/startup_stm32f0
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F042K6/device/TOOLCHAIN_ARM_STD/startup_stm32f0
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F042K6/device/TOOLCHAIN_GCC_ARM/startup_stm32f0
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F042K6/device/TOOLCHAIN_IAR/startup_stm32f042x6
targets/TARGET_STM/TARGET_STM32F0/TARGET_NUCLEO_F070RB/device/TOOLCHAIN_ARM_MICRO/startup_stm32f0
```

- 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



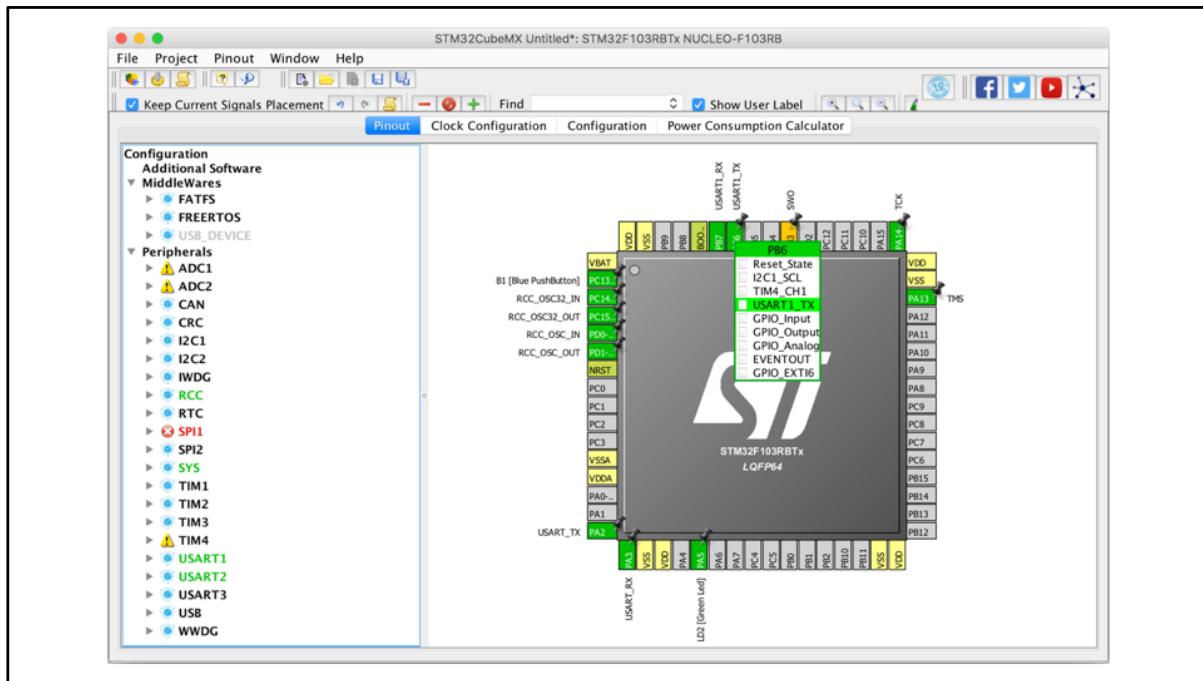
# TRIPLE FACEPALM

Not even double facepalm can explain how much you fail

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?



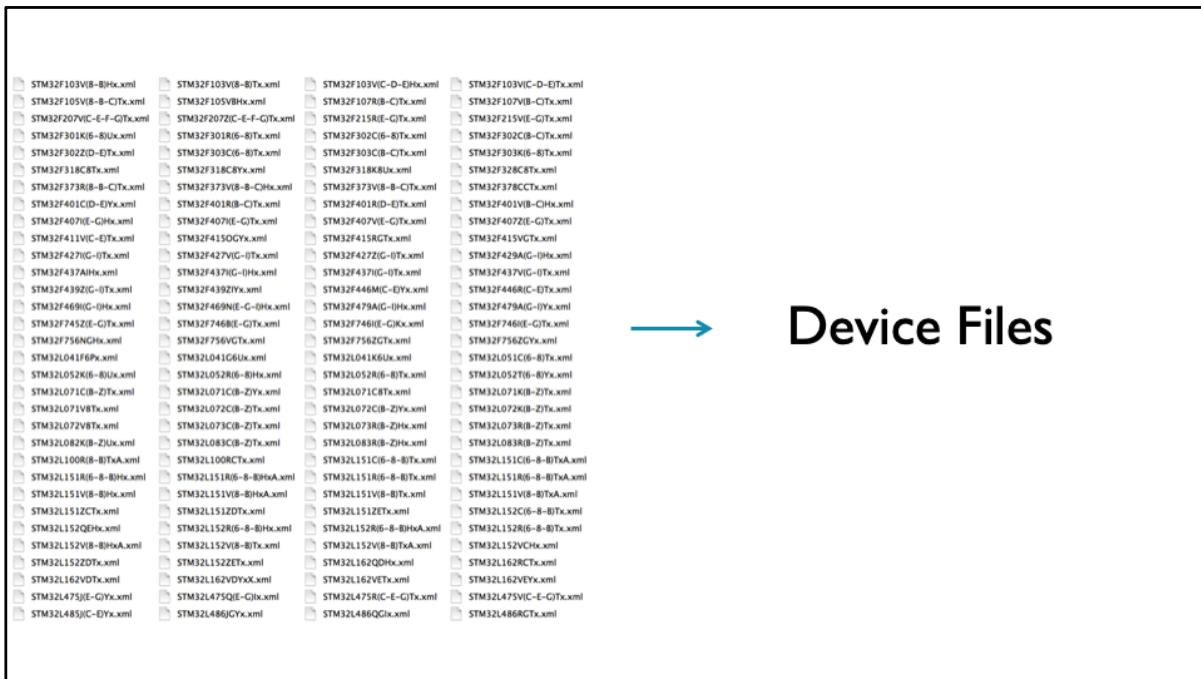
STMicro publishes a GUI App to initialize your STM32.

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!

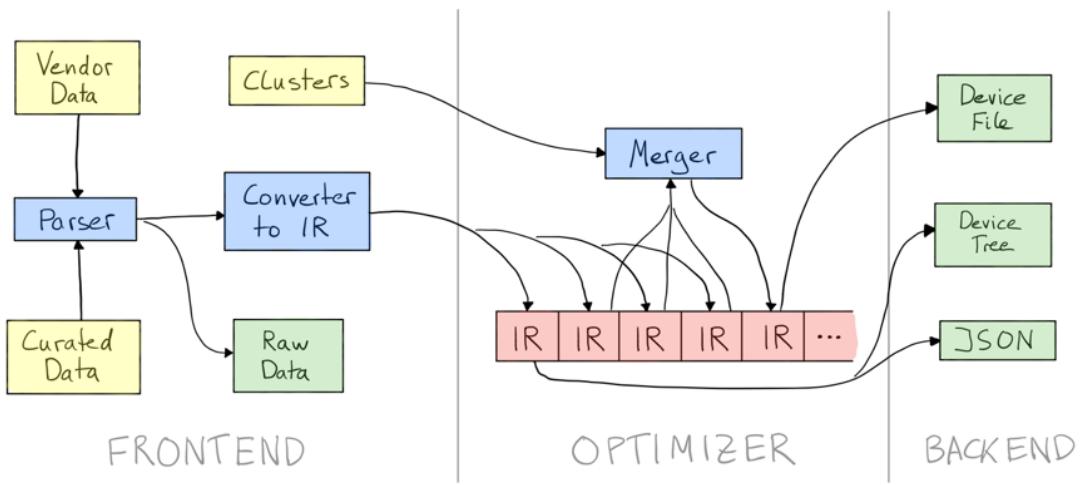


70MB of XML files, with a lot of device details.

## **modm-devices** contains

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

## [blog.salkinium.com/modm-devices](http://blog.salkinium.com/modm-devices)



The reality is a bit more involved.

**~1200 STM32s + ~200 AVRs**

modm-io / modm-devices

Code Issues 0 Pull requests 0 Insights Settings

Curated device data for all AVR and STM32 devices <http://blog.salkinium.com/modm-devices>

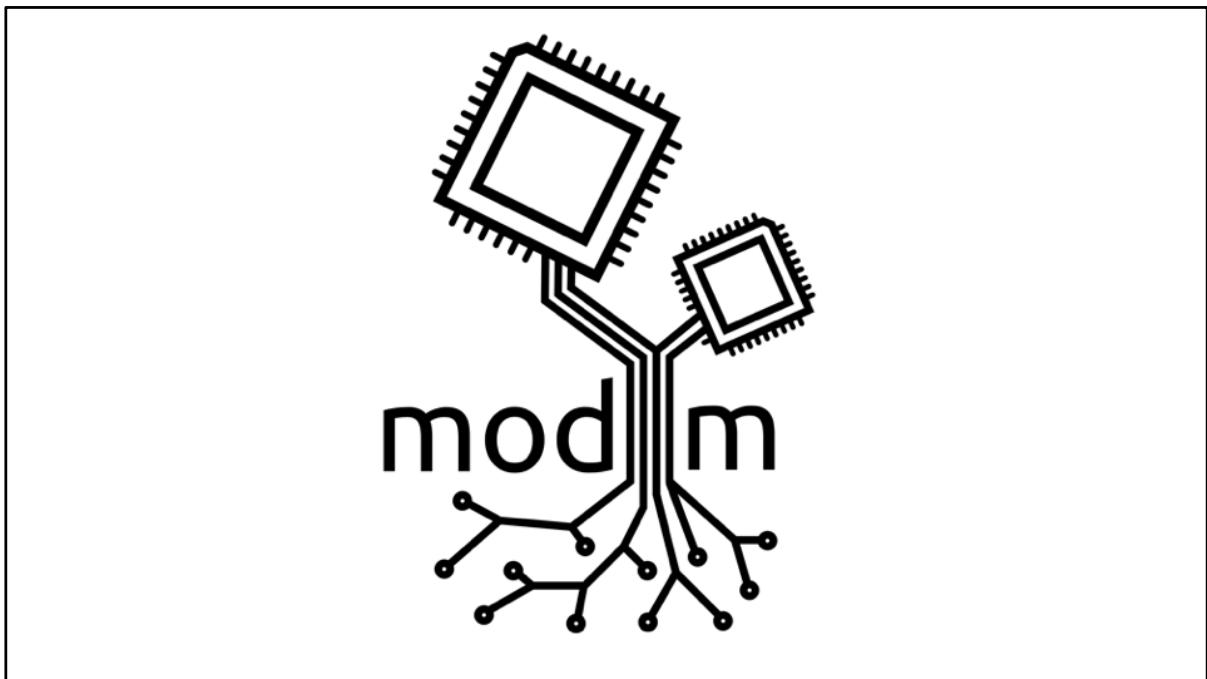
avr stm32 data microcontroller device-tree modm Manage topics

74 commits 3 branches 0 releases 2 contributors MPL-2.0

Branch: develop New pull request Create new file Upload files Find file Clone or download

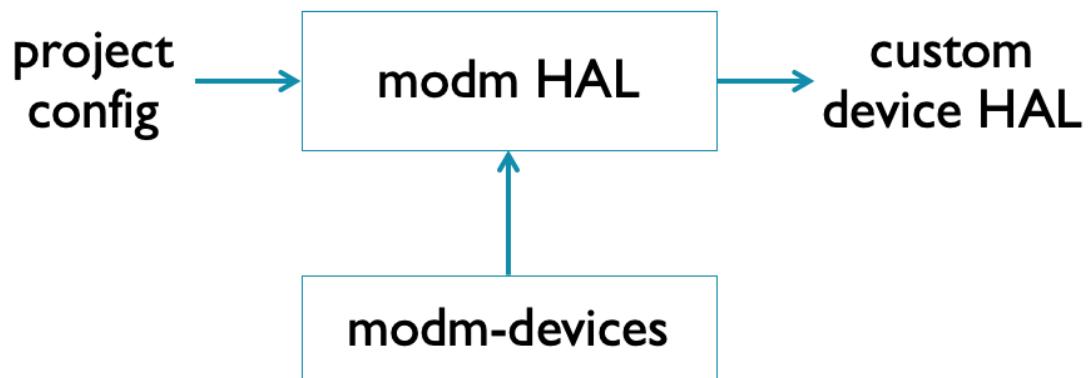
File	Description	Time Ago
salkinium	Update cmsis-header-stm32 submodule	Latest commit dd631c0 7 days ago
devices	Update STM32 and AVR device files.	3 months ago
tools	Update cmsis-header-stm32 submodule	7 days ago
.gitignore	Initial commit	2 years ago
.gitmodules	Add cmsis-header-stm32 submodule.	3 months ago
LICENSE	Initial commit	2 years ago

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.

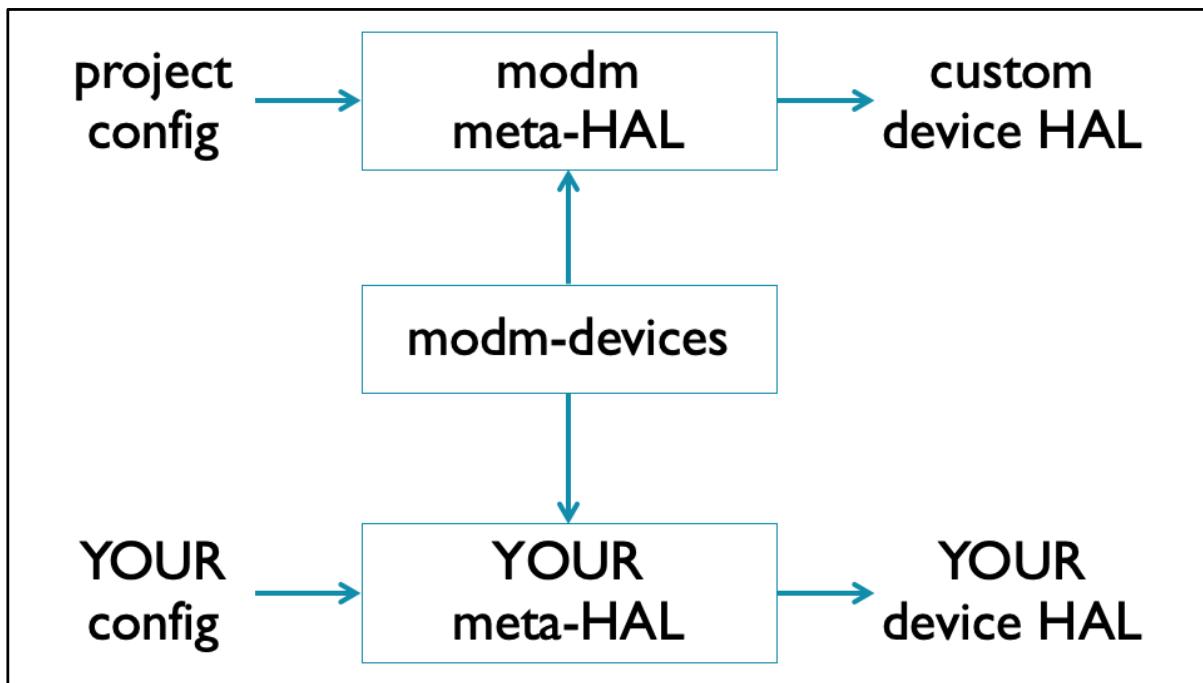
## **modm is our next generation toolbox**



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 AVR 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.

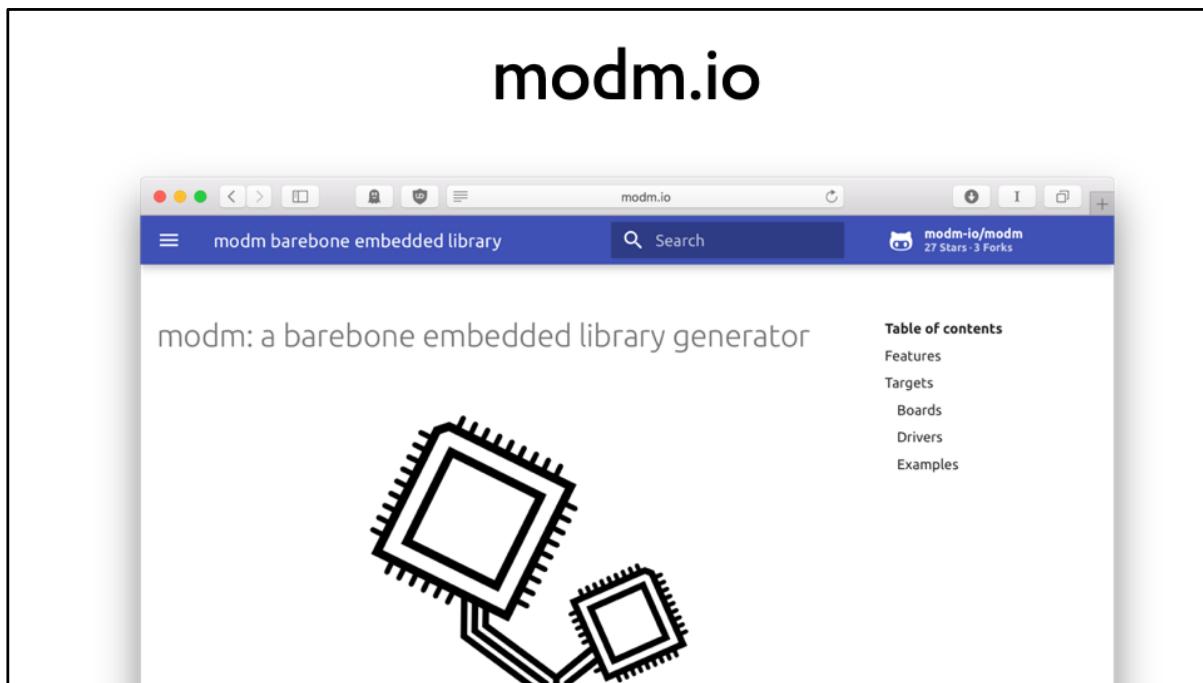
## Vertical vs. Horizontal Porting

Feature	STM32F0	STM32F1	STM32F3	STM32F4	STM32F7
<b>Core</b>	✓	✓	✓	✓	✓
<b>GPIO</b>	✓	✓	✓	✓	✓
<b>Clock</b>	✓	✓	✓	✓	✓
<b>UART</b>	✓	✓	✓	✓	✓
<b>SPI</b>	✓	✓	✓	✓	✓

There is a change in how you port your targets now:

Before: For each target implement the feature.

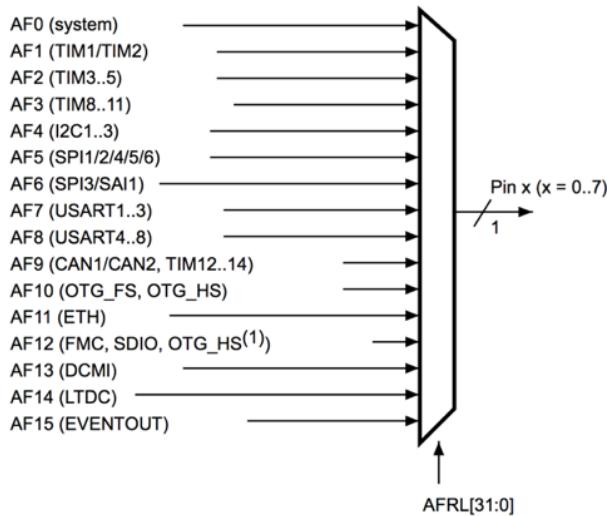
Now: for each feature port to all targets



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.

# STM32 GPIO Alternate Functions



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/ 1/2/3	USAR T6/UA RT4/5 7/8	CAN1/2/ TIM12/1 3/14/QU ADSP1/L CD	QUAD SPI/OT G2_HS /OTG1 _FS	ETH	FMC/SD IO/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	-	-	-	EVENT OUT	
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	USART2_ RTS	UART4_ RX	QUADSPI_BK1_IO3	-	ETH_MII_RX_CLUKETH_RM1_RX_CLK	-	-	-	LCD_R2 EVENT OUT	
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	USART2_T_X	-	-	-	ETH_MDIQ	-	-	-	LCD_R1 EVENT OUT	
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-	-	USART2_RX	-	LCD_B2	OTG_HS_ULPI_D0	ETH_MII_COL	-	-	-	LCD_B5 EVENT OUT	
	PA4	-	-	-	-	SPI1_NSS	SPI3_NSS/ I2S3_WS	USART2_CK	-	-	-	OTG_HS_SOF	DCMI_HS_YNC	LCD_VSY_NIC	-	EVENT OUT	
	PA5	-	TIM2_CH1/ TIM2_ETR	-	TIM8_CH1_N	-	SPI1_SCK	-	-	-	OTG_HS_ULPI_C_K	-	-	-	-	LCD_R4 EVENT OUT	
	PA6	-	TIM1_BKIN	TIM3_CH1	TIM8_BK1_N	-	SPI1_MISO	-	-	TIM13_CH1	-	-	-	DCMI_PIX_CLK	LCD_G2	EVENT OUT	
	PA7	-	TIM1_CH1N	TIM3_CH2	TIM8_CH1_N	-	SPI1_MOSI	-	-	TIM14_CH1	QUADSPI_CLK	ETH_MII_RX_DVETH_RMII_CRS_DV	FMC_SDN_WE	-	-	-	EVENT OUT
	PA8	MCO1	TIM1_CH1	-	-	I2C3_SCL	-	USART1_CK	-	-	OTG_FS_SOF	-	-	-	-	LCD_R6 EVENT OUT	
	PA9	-	TIM1_CH2	-	-	I2C3_SMBA	SPI2_SCKI/ 2S2_CK	USART1_T_X	-	-	-	-	-	DCMI_D0	-	EVENT OUT	
	PA10	-	TIM1_CH3	-	-	-	-	USART1_RX	-	-	OTG_FS_ID	-	-	DCMI_D1	-	EVENT OUT	
	PA11	-	TIM1_CH4	-	-	-	-	USART1_CTS	-	CAN1_RX	OTG_FS_DM	-	-	-	LCD_R4	EVENT OUT	
	PA12	-	TIM1_ETR	-	-	-	-	USART1 RTS	-	CAN1_TX	OTG_FS_DP	-	-	-	LCD_R5	EVENT OUT	
	PA13	JTMS-SWDIO	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENT OUT	
	PA14	JTCK-SWLK	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENT OUT	
	PA15	JTDI	TIM2_CH1/ TIM2_ETR	-	-	-	SPI1_NSS	SPI3_NSS/ I2S3_WS	-	-	-	-	-	-	-	EVENT OUT	

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);
```

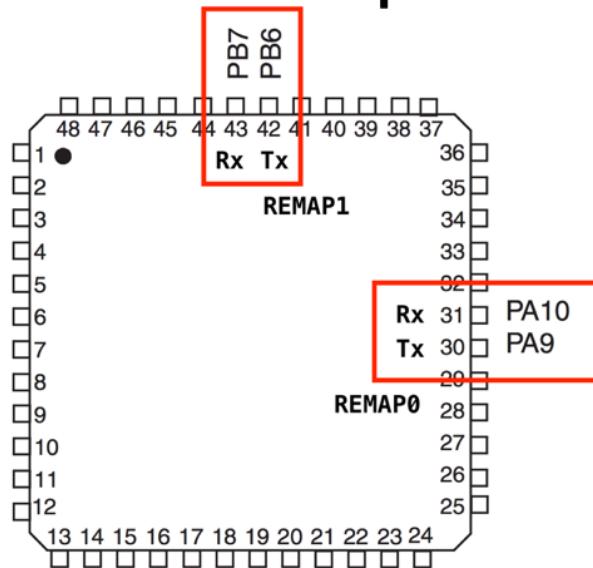
We came up with this API.

Just connect the pin to the peripheral.

Worked fine for a long time, until we ported to STM32F1!

Things suddenly broke.

# STM32F1 Remap Groups



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

f301c8t	f301e8y	f301k6u	f301kbu	f301r6f	f301r8t	f302e6t	f302e8t	f302e8y	f302e8t	f302ecet	f302k6u	f302k8u	f302r6t	f302r8t	f302zbt	f302ret	f302ret	f302vbt	f302vdt	f302vet		
f302vet	f302vet	f302vet	f302vet	f302vet	f303c6f	f303c8t	f303c8t	f303c8t	f303c8t	f303c8t	f303k6t	f303k8t	f303k8t	f303k8t	f303k8t	f303ret	f303ret	f303ret	f303ret	f303ret	f303ret	
f303vey	f303d4t	f303gpt	f318e8t	f318e8y	f318k8u	f328k8t	f334c4t	f334c6t	f334c8t	f334k6t	f334k6t	f334k8t	f334k8t	f334k8t	f334k8t	f334ret	f334ret	f334ret	f334ret	f334ret	f334ret	
f373ret	f373v8t	f373v8t	f373vbt	f373vbt	f373vch	f373vct	f378ret	f378ret	f378ret	f378ret	f378ret	f378ret	f378ret	f378ret	f378ret	f398vet	f401cbu	f401ccu	f401ccy	f401cdt	f401ceu	
f401ret	f401vbt	f401vbt	f401vh	f401vh	f401vdh	f401vdt	f401vh	f401vh	f401vh	f401vh	f401vh	f401vh	f401vh	f401vh	f401vh	f401vh	f401vh	f401vh	f401vh	f401vh	f401vh	
f410cbvt	f410r8t	f410r8t	f410r8y	f410r8y	f410by	f411ccu	f411ccy	f411ccy	f411ccy	f411ccy	f411ccb											
f412vgh	f412vgt	f412vgt	f412zej	f412zej	f412zgj	f412zgt	f415gey	f415grt	f415vgt	f415vgt	f415vgt	f415vgt	f415vgt	f415vgt	f415vgt	f415vgt	f415vgt	f415vgt	f415vgt	f415vgt	f415vgt	
f427iit	f427vgt	f427vgt	f427zgt	f427zgt	f427zgt	f429agh	f429ah	f429ah	f429ah	f429ah	f429bgt	f429bit	f429ieh									
f429zgt	f429zit	f429zyt	f437aih	f437igh	f437ih	f437ih	f437iit	f437iit	f437iit	f437iit	f437iit	f437iit	f437iit	f437iit	f437iit	f439ah	f439bgt	f439bgt	f439igh	f439igh	f439igh	f439igh
f439zgt	f439zit	f439zyt	f446mcy	f446mcy	f446ret	f446ret	f446ret	f446ret	f446ret	f446ret	f446vct											
f469iich	f469iit	f469iit	f469iit	f469iit	f469iit	f469iit	f469iit	f469iit	f469iit	f469iit	f469iit	f469iit										
f479iib	f479iit	f479iit	f479iit	f479iit	f479iit	f479iit	f479iit	f479iit	f479iit	f479iit	f479iit	f479iit										
f746igt	f746neh	f746ngh	f746vch	f746vet	f746vgh	f746vet	f746vet	f746vet	f746vet	f746vet	f746zey											
f765iik	f765iit	f767bgt																				
f769bgt	f769bit	f769igt	f769iit	f769iit	f769iit	f769iit	f777iit	f777iit	f777iit	f777iit	f777iit	f777iit	f777iit	f777iit	f777iit	f777iit	f777iit	f777iit	f777iit	f777iit	f777iit	f777iit

## 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

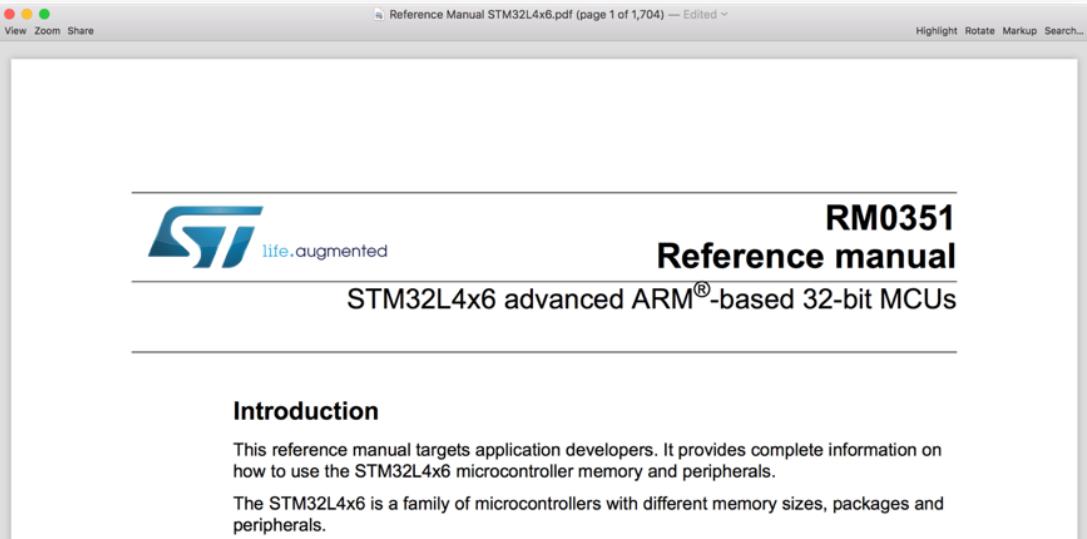
```
Uart1::connect(GpioB7::Rx,  
                GpioB6::Tx);
```

```
Uart1::connect(GpioB7::Rx,  
ERROR! GpioA9::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!



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 information.

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_RX	USAR-T3_RX	X	X	X	X
3	-	I2C3_RX	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 peripherals 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.

# There are lots of other PDFs

The screenshot shows a PDF document titled "LSM303DLHC" from STMicroelectronics. The document is a datasheet for an ultra-compact high-performance eCompass module. It features two images of the sensor package: a top-down view and a side-view cross-section. Below these images, under the heading "Features", is a bulleted list of applications: "Display orientation", "Gaming and virtual reality input devices", "Impact recognition and logging", and "Vibration monitoring and compensation". Under the heading "Description", it is mentioned that the LSM303DLHC is a system-in-package featuring a 3D digital linear acceleration sensor and a 3D digital magnetic sensor.

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

# The Plan

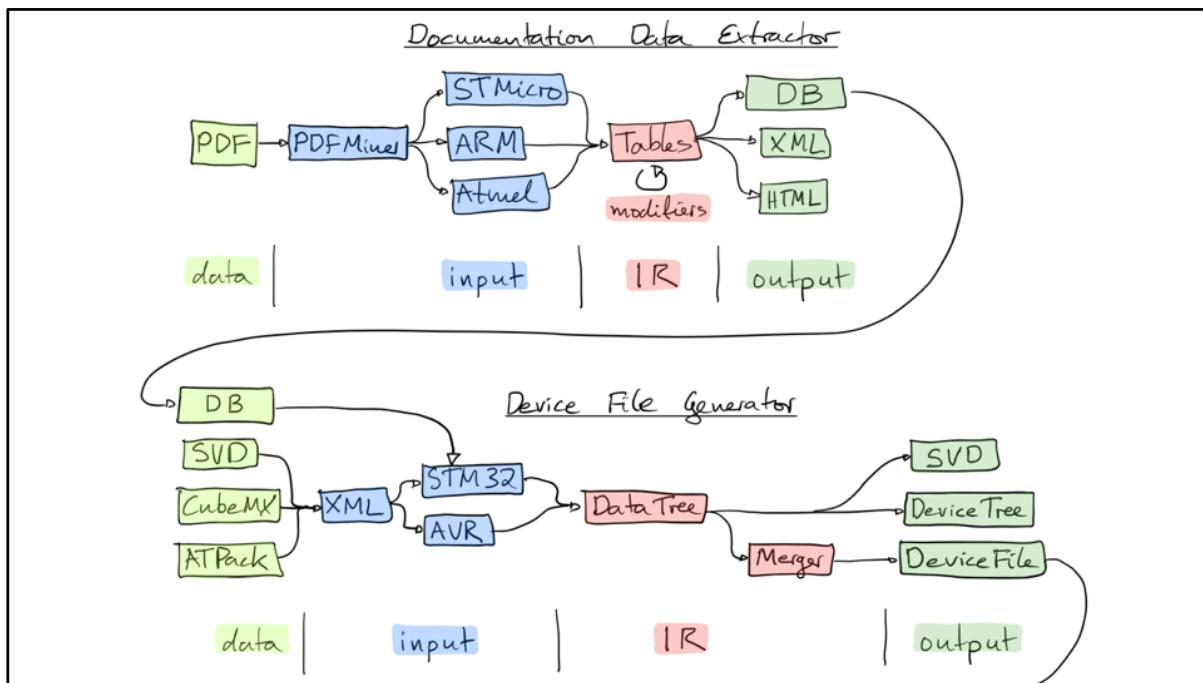
**PHASE 1    PHASE 2    PHASE 3**

Parse ALL  
Datasheets



Profit

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 \(GitHub/Twitter\)](https://github.com/salkinium)  
[\(blog.\)salkinium.com](http://blog.salkinium.com)

[modm.io](http://modm.io)  
[github.com/modm-io](https://github.com/modm-io)

EY MANN, WO IS' MEIN TSCHUNK?