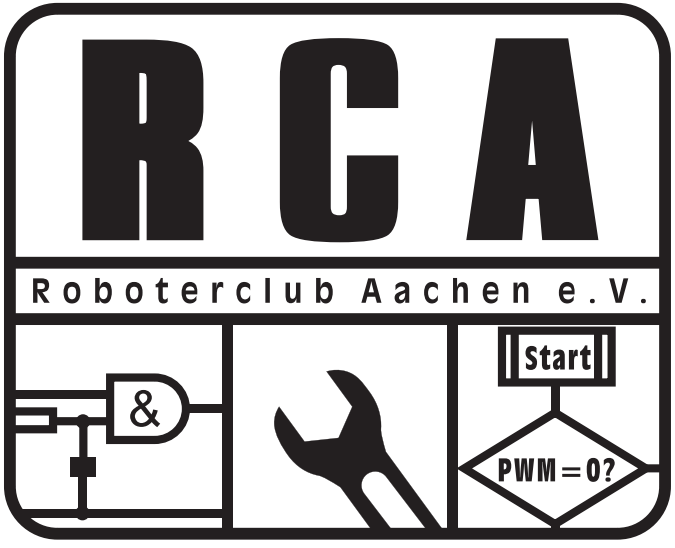
The xpcc microcontroller framework

An efficient object-oriented approach to embedded software development.

Niklas Hauser, Kevin Läufer
Roboterclub Aachen e. V.

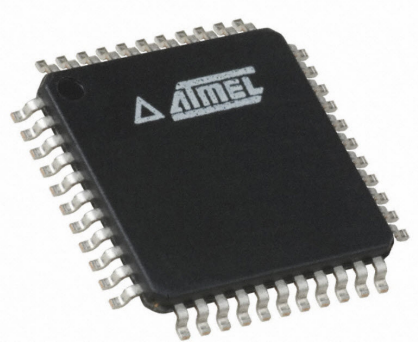
FOSDEM 2014

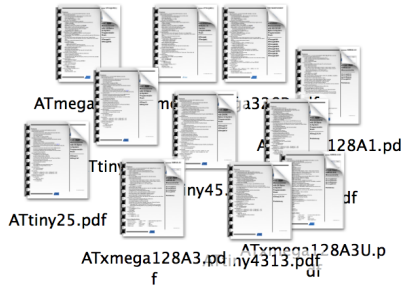
Who? What?



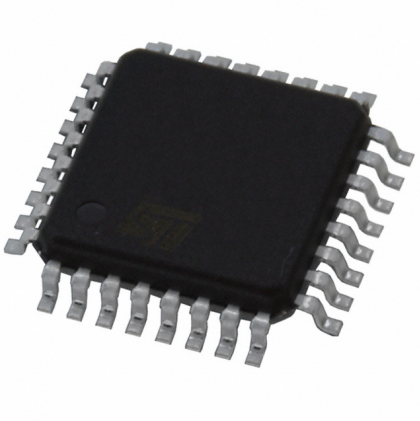
@salkinium • @ekiwi

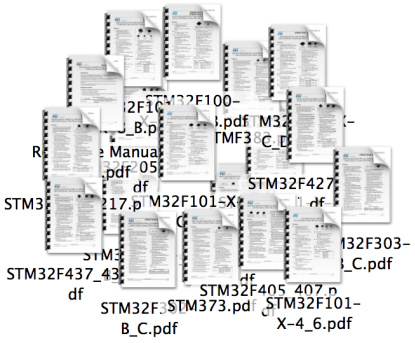
Motivation






```c
int main(void) {
    /* use PortB for output (LED) */
    DDRB = 0xff;
    /* enable timer overflow */
    TIMSK = (1 << TOV0);
    /* set timer counter initial value */
    TCNT0 = 0x00;
    /* start timer without prescaler */
    TCCR0 = (1 << CS00);
    /* enable interrupts */
    sei();
    /* intial led value */
    led = 0x00;
    while(1) {
        /* loop forever, timer interrupts will
         change the led value */
        PORTB = led;
    }
}
```






```c
int main(void) {
    /* use PortB for output (LED) */
    GPIOB->OTYPER = 0xffff;
    /* enable timer overflow */
    TIM1->DIER |= TIM_DIER_UIE;
    /* set timer counter initial value */
    TIM1->CNT = 0x00;
    /* start timer without prescaler */
    TIM1->PSC = 1;
    TIM1->CR1 |= TIM_CR1_CEN;
    /* enable interrupts */
    sei();
    /* intial led value */
    led = 0x00;
    while(1) {
        /* loop forever, timer interrupts will
         change the led value */
        GPIOB->ODR = led;
    }
}
```

Same task, different code!

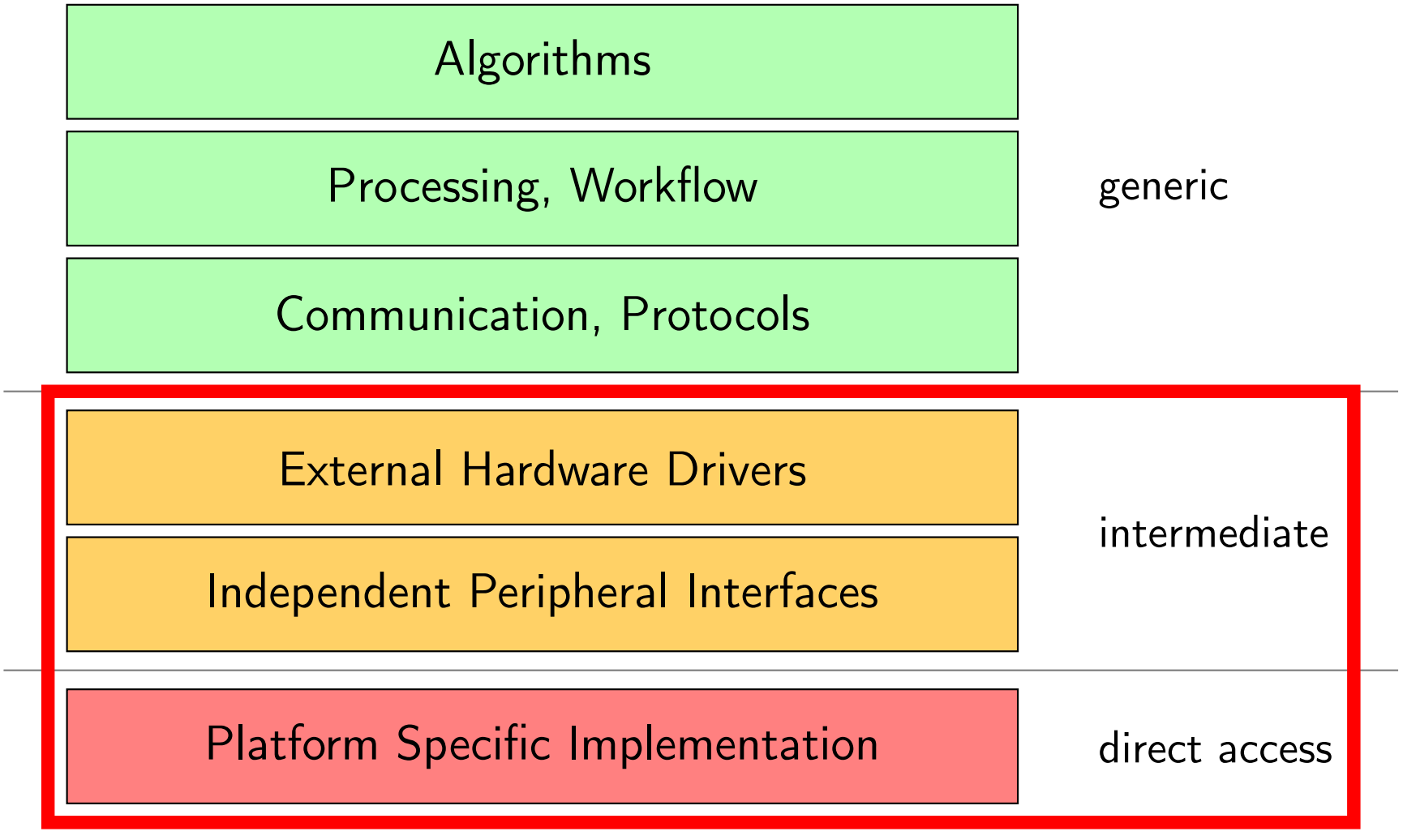
What is xpcc?

An efficient **object-oriented** microcontroller framework written in **C++** that enables you to create **identical code for multiple targets**.

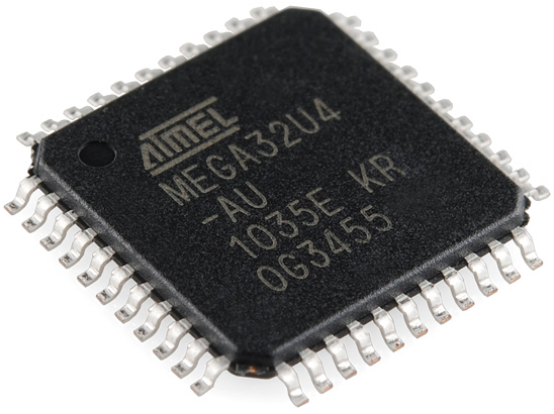
Talk structure

- concepts and interfaces of xpcc
- external hardware drivers
- build system
- future work

Library overview and talk focus



Visual appearance of microcontrollers



Black box with lots of pins

These pins are used for some form of Input/Output.

General Purpose Input/Output

```
class GpioInput
{
    void setInput();
    bool read();
}
```

```
class GpioOutput
{
    void setOutput();
    void set();
    void reset();
    void toggle();
}
```

Any platform should be capable of satisfying this interface.

General Purpose Input/Output

Implementation using C-like function calls:

```
pinMode(12, OUTPUT);
digitalWrite(12, HIGH);
```

xpcc provides a **customized class for each pin**:

```
GpioOutputB4::setOutput();
GpioOutputB4::set();
```

Inlined code **enables atomic GPIO operation**.

General Purpose Input/Output

Simply **connect alternate functions to pins**:

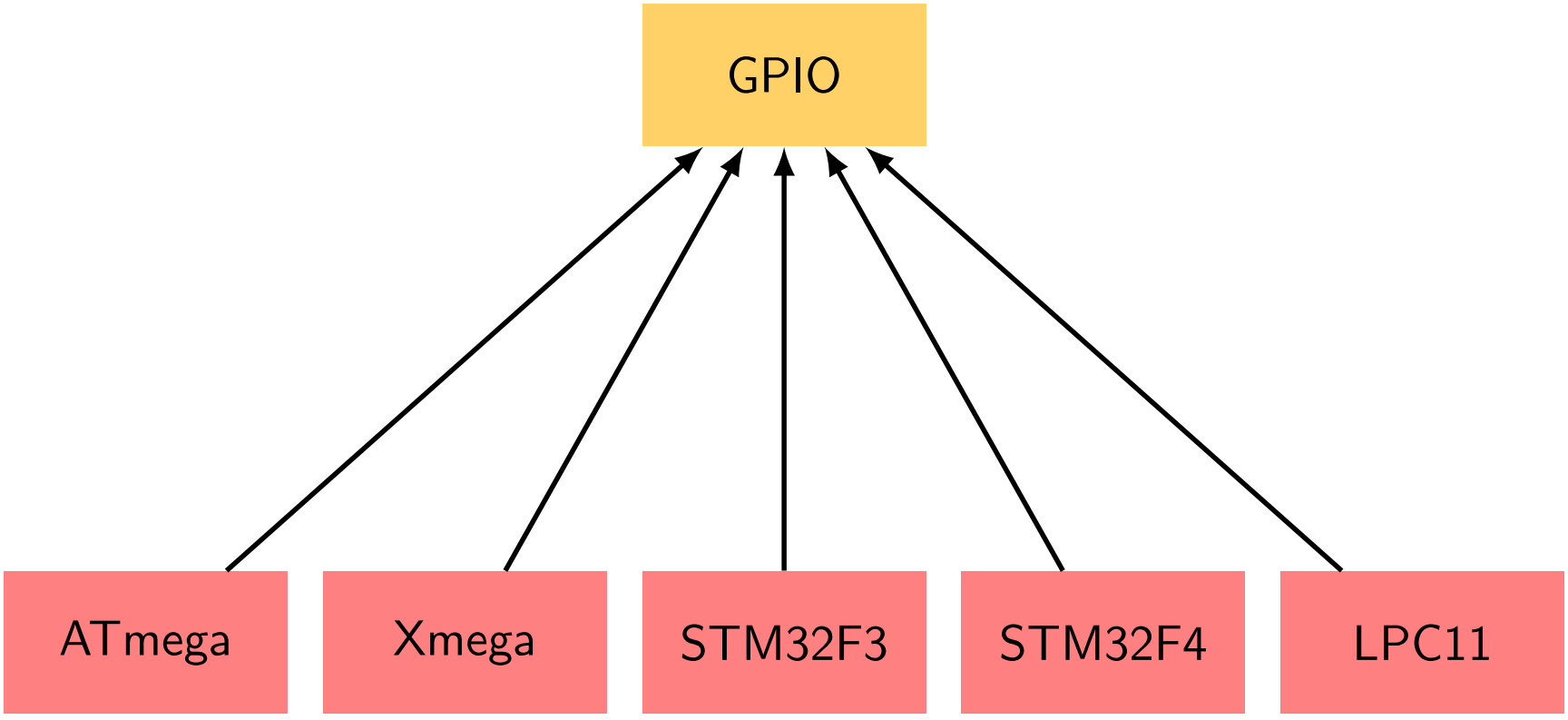
```
GpioInputD0::connect(Uart0::Rx);
GpioOutputD1::connect(Uart0::Tx);
```

Type is checked at compile time:

```
GpioInputD1::connect(Spi::Miso); // Compiler Error!
```

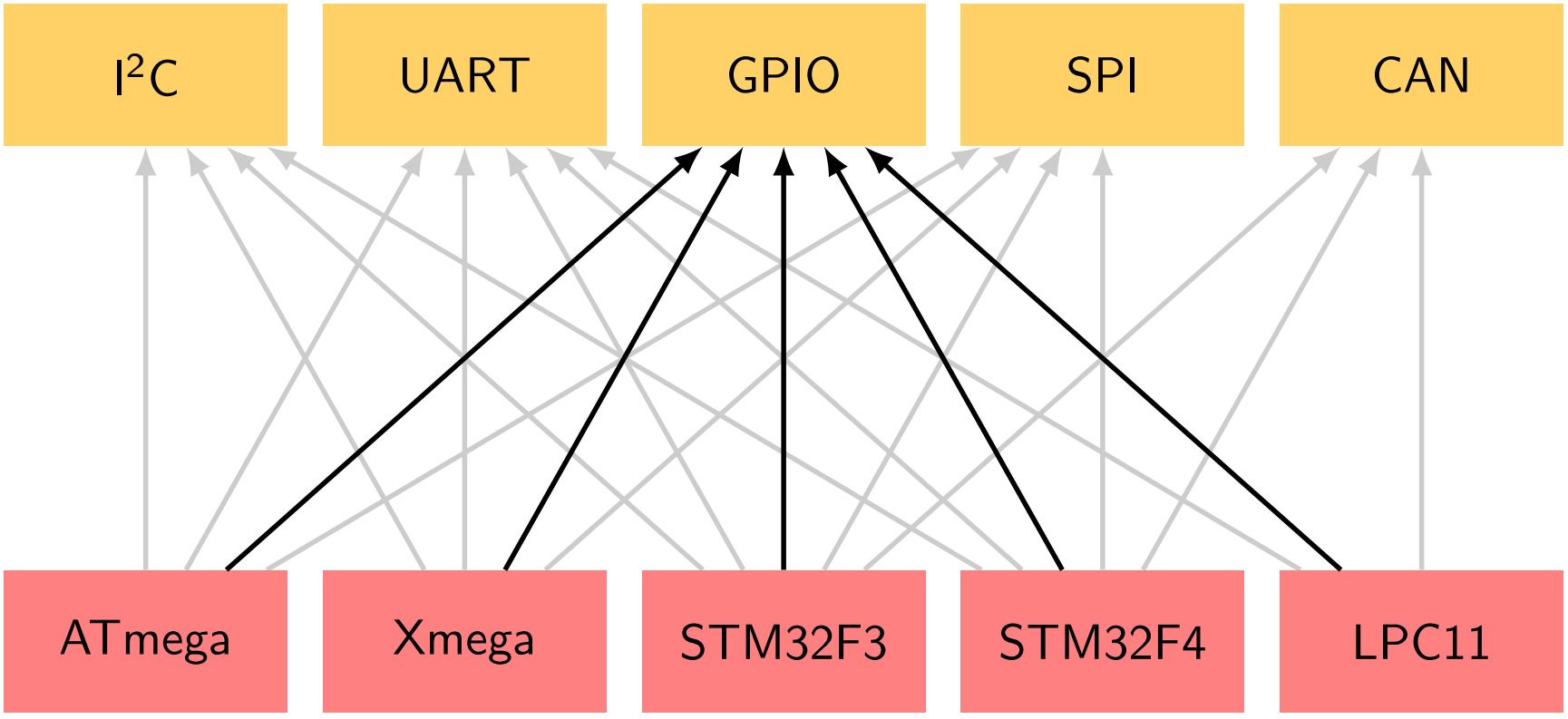
Bonus: **The code is your documentation!**

Peripheral



All targets provide these GPIO methods.

Peripherals



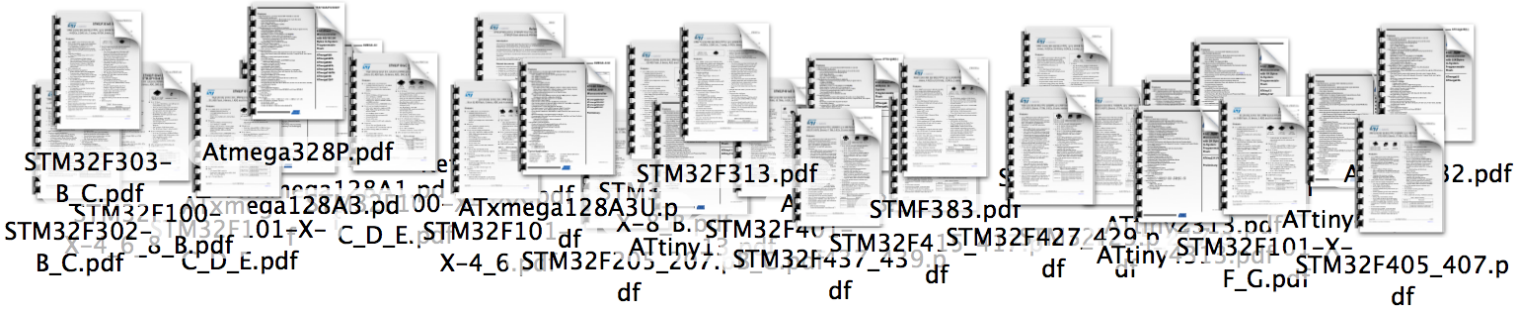
Targets also satisfy other interfaces when available.

Declare **what** you want,
not how to get it.

Example: UART baudrate

Never set prescalers directly!

```
UBBR0 = 8;  // baudrate?
```



Real **runtime calculation result is unknown**:

```
Serial.begin(115200);  // actually 112300
```

(It also wastes time and program space.)

Example: UART baudrate

xpcc offers **calculation at compile-time**:

```
Uart::initialize<SystemClock, 115200>();
```

Calculated register values are **stored in program code**.

Bonus #1: **The code is your documentation!**
Bonus #2: Plausibility check for free!
Bonus #3: No need to open a datasheet.

Example: UART baudrate

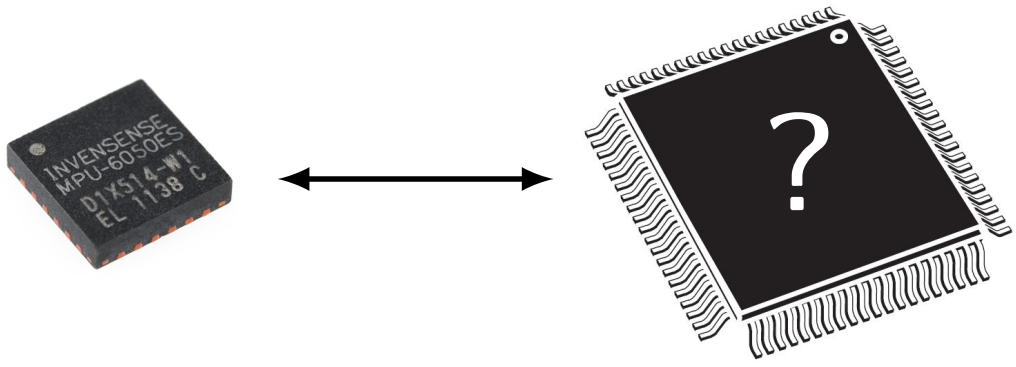
Declare your baudrate tolerance:

```
Uart::initialize<SystemClock, 115200,
                 xpcc::Tolerance::OnePercent>();
```

Using template magic, the **compiler provides you with the nearest alternative**:

```
In 'static void xpcc::Tolerance::checkValueInTolerance()
    [with long unsigned int reference = 112300ul;
    long unsigned int actual = 115200ul;
    short unsigned int tolerance = 10u]':
The actual value is exceeding the tolerance of reference!
```

External hardware drivers



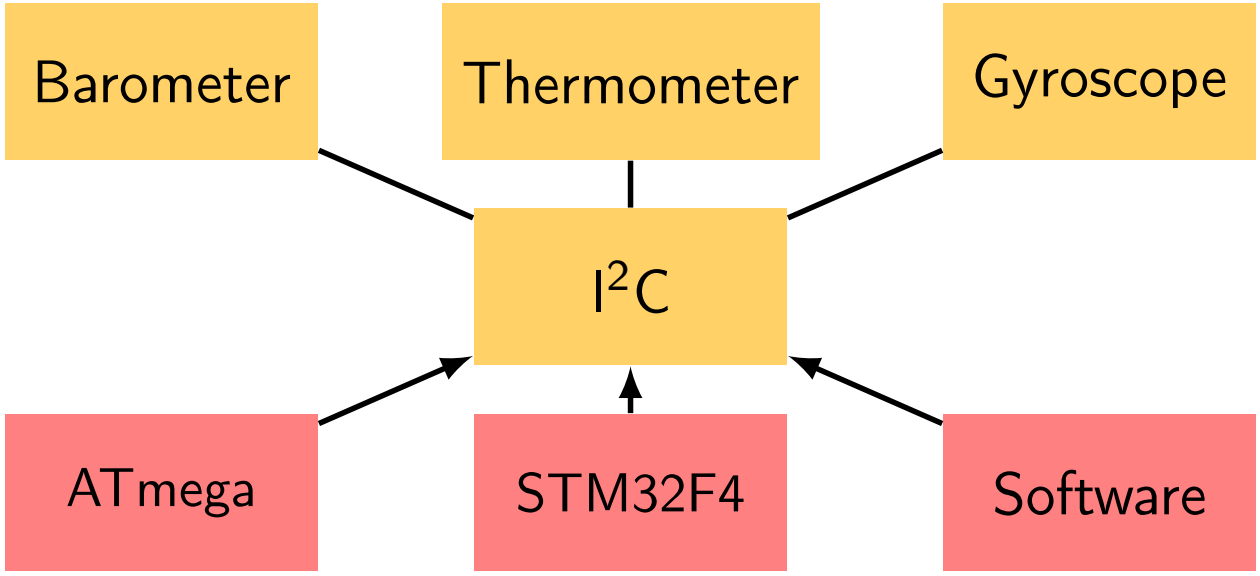
Can an external hardware driver be independent of platform?

Yes. **Think protocols, not platforms!**

External hardware drivers

These interfaces allow you to speak the protocols.

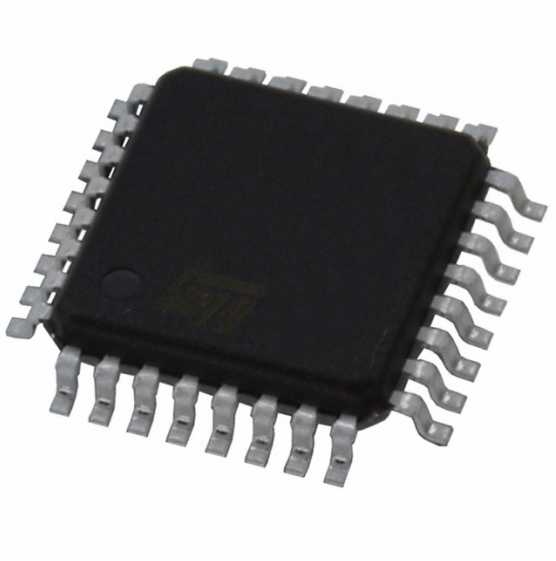
External hardware drivers



- Drivers only talk with interfaces
- Truly **platform-independent driver code**
- Software implementations use GPIO classes!

Techporn Bonus: non-blocking, callback-based implementation

From the Outside

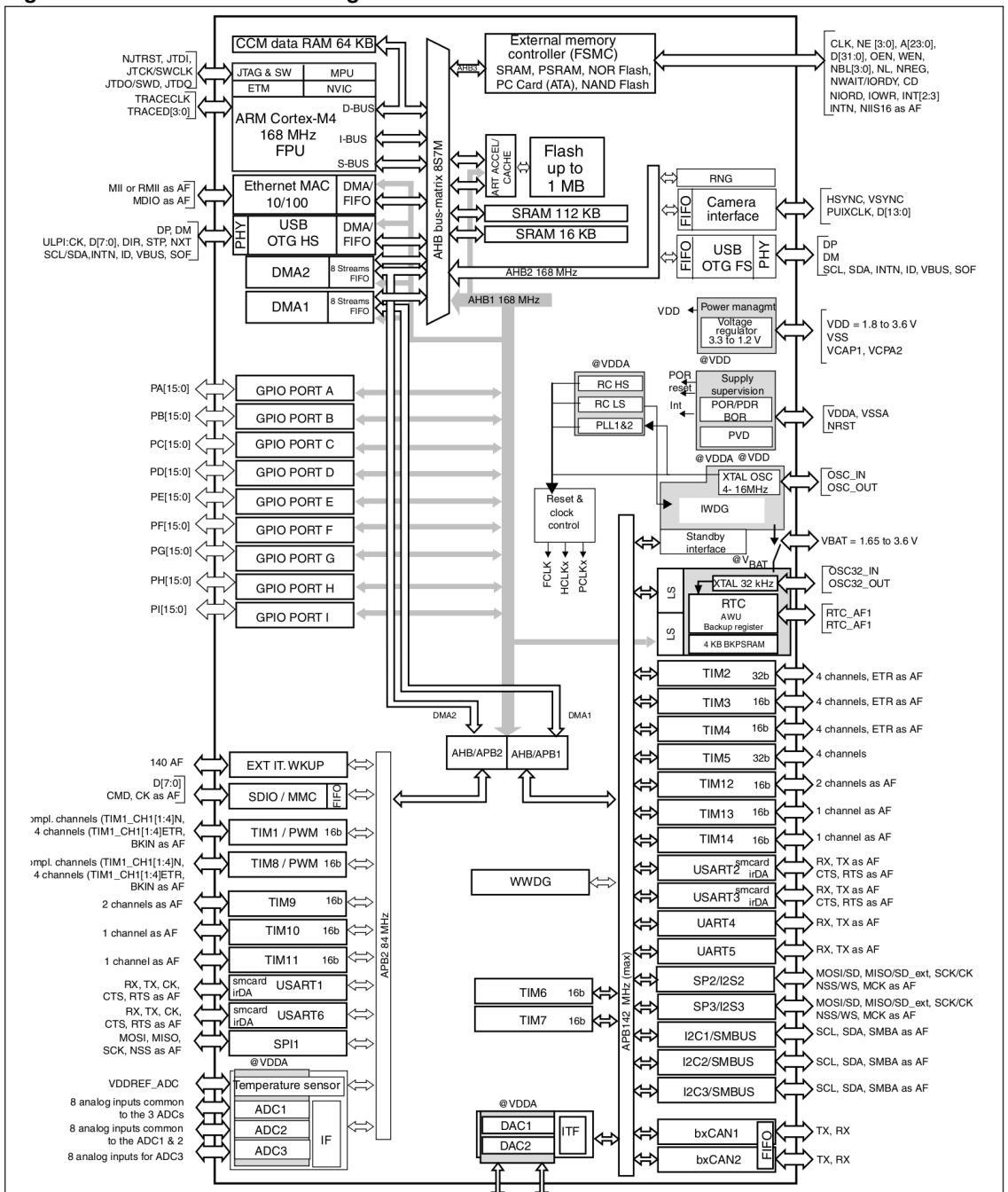


Black box with lots of pins

Everything is **static**.

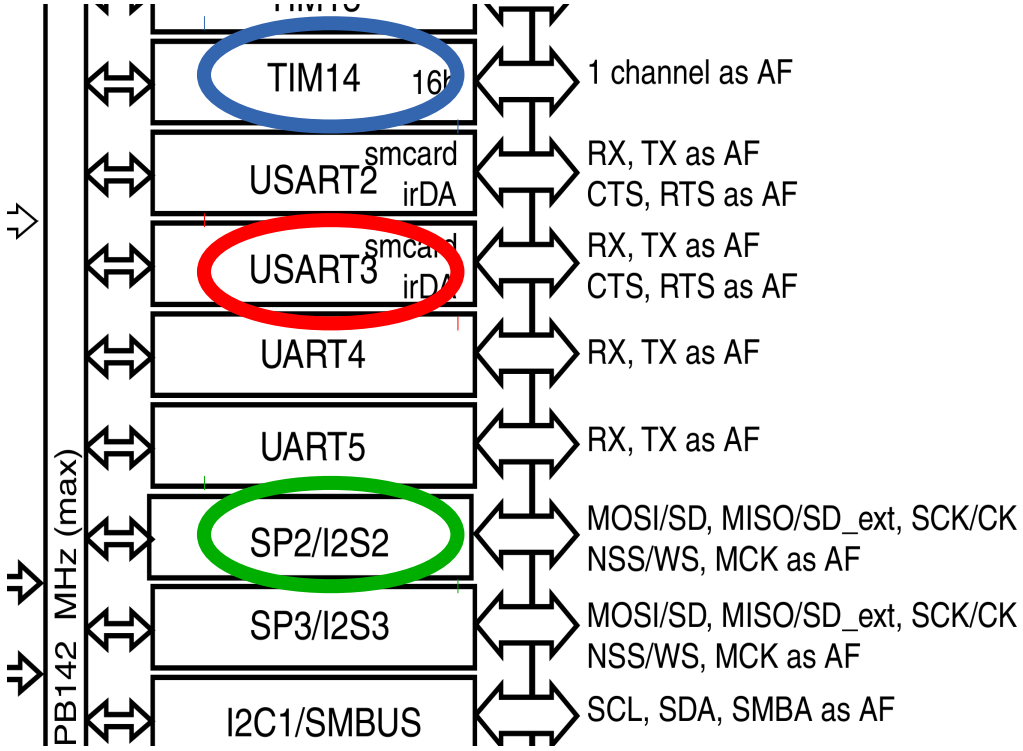
Underneath the Surface

Figure 5. STM32F40x block diagram

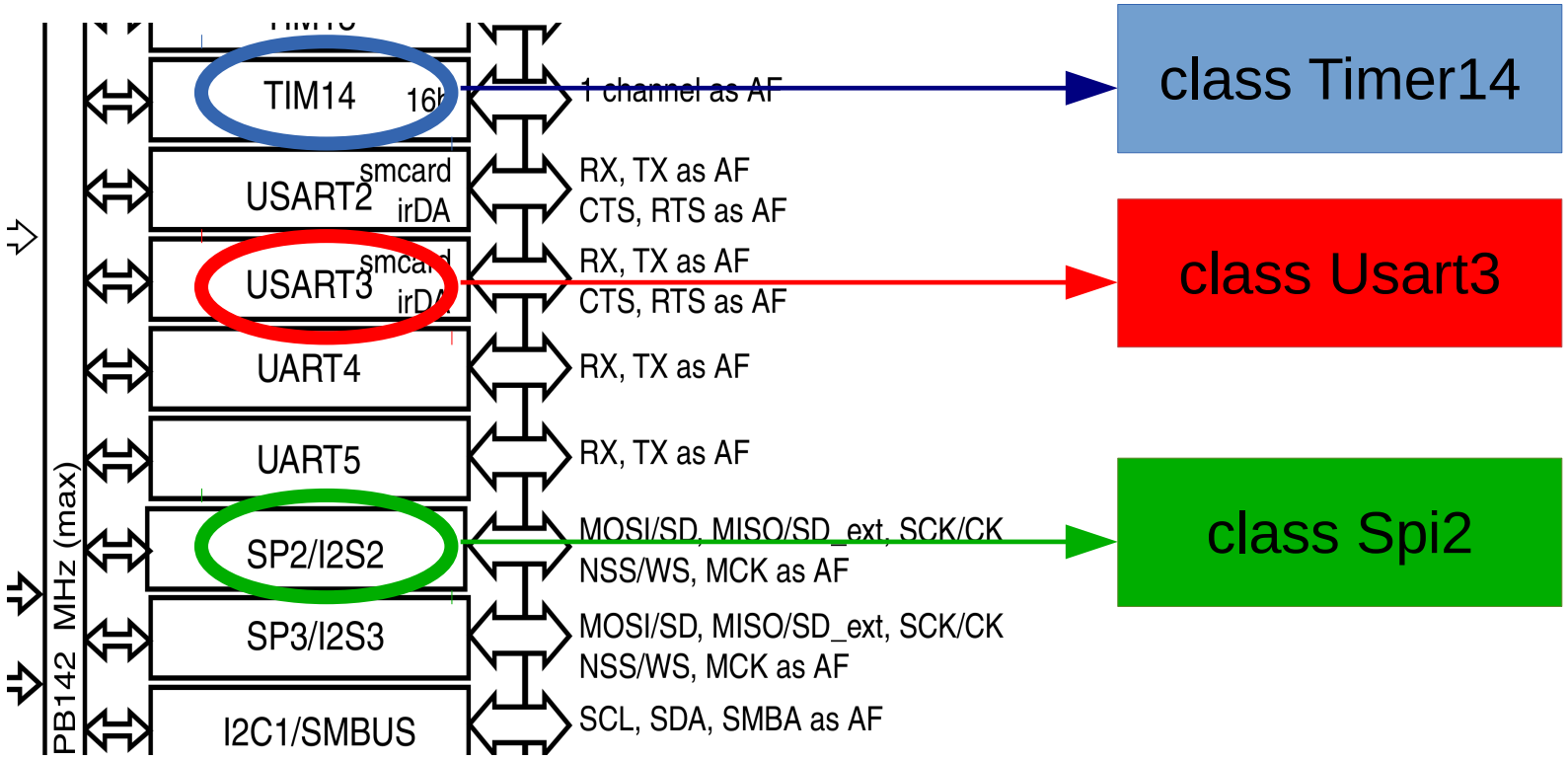


Everything is **static**.

Representing Peripherals



Representing Peripherals

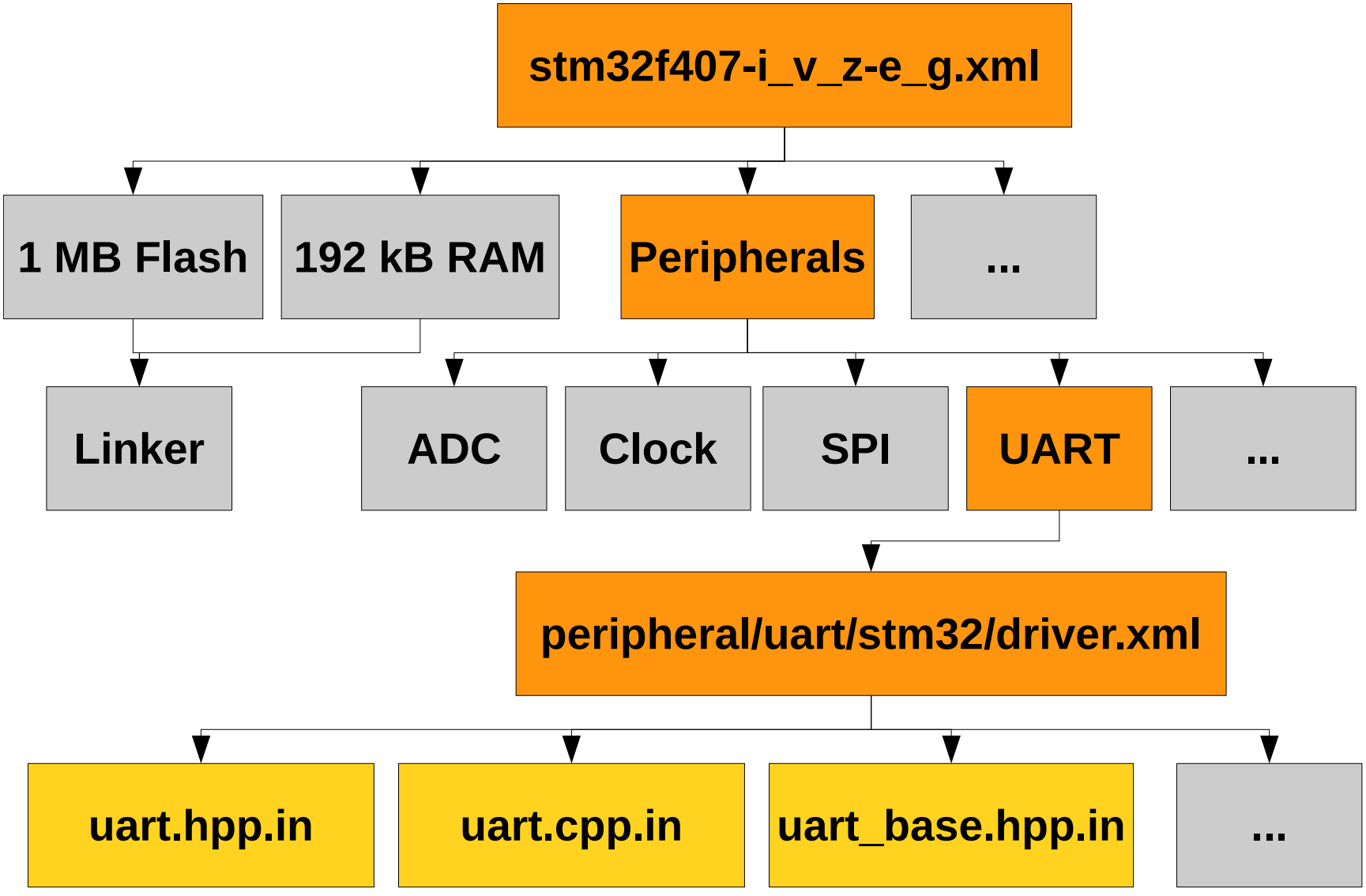


Representing Peripherals

We want **one static C++ class for every peripheral**!

- We need to know which device needs which classes.

XML Device and Driver Files

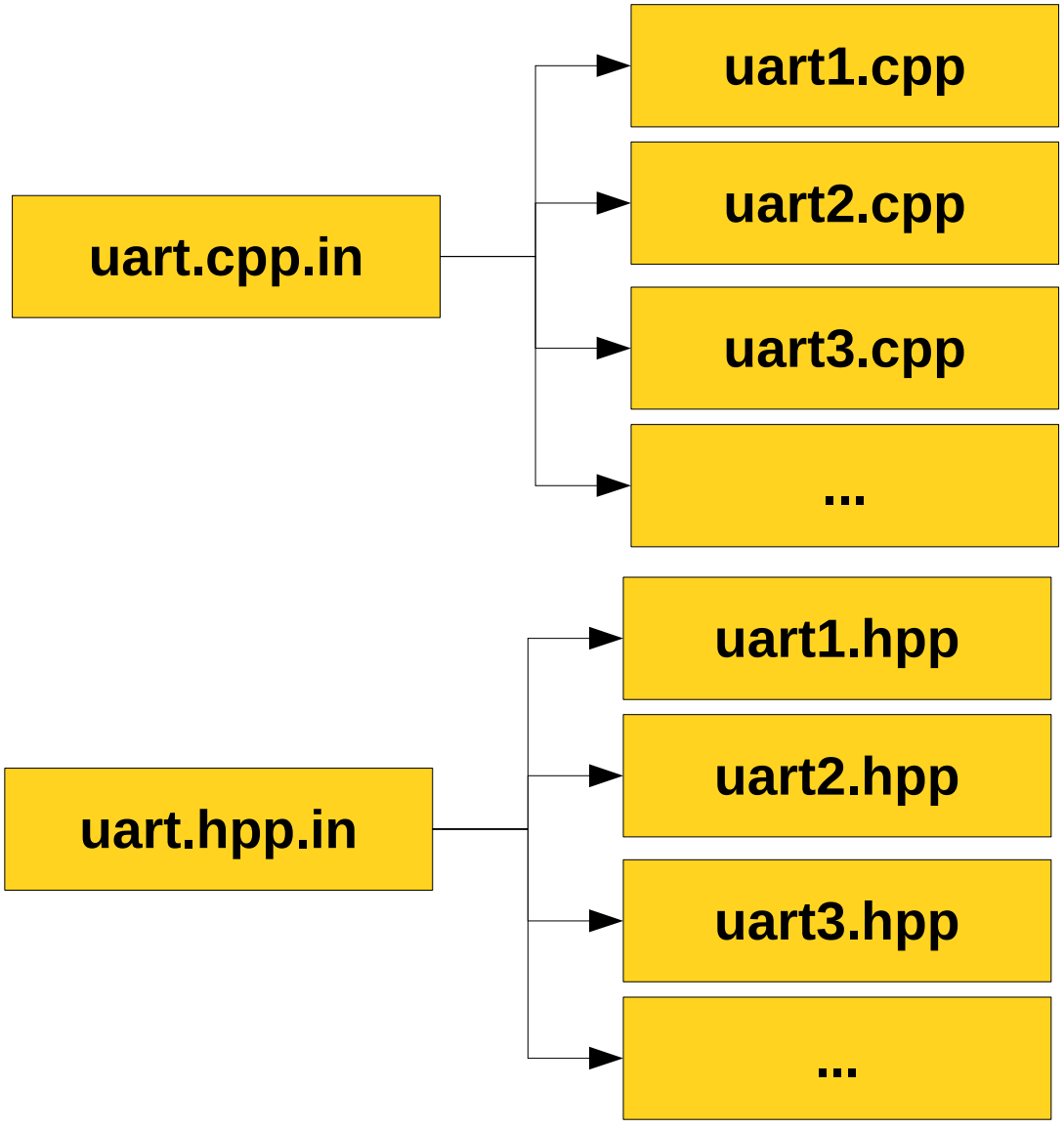


Representing Peripherals

We want **one static C++ class for every peripheral**!

- We need to know which device needs which classes.
- We are developers, we are lazy, we want to avoid duplicate code.

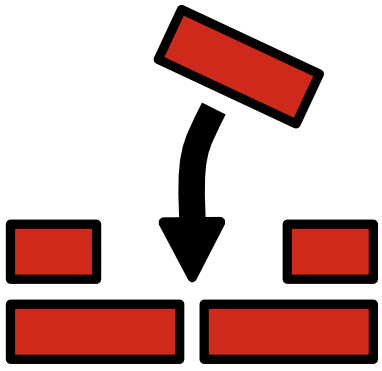
Jinja2 Templates



Jinja2 Templates


```
void
xpcc::stm32::Uart{{ id }}::write(uint8_t data)
{
%% if target is stm32f0 or target is stm32f3
    {{ peripheral }}->TDR = data;
%% elif target is stm32f2 or target is stm32f4
    {{ peripheral }}->DR = data;
%% endif
}
```

Build System



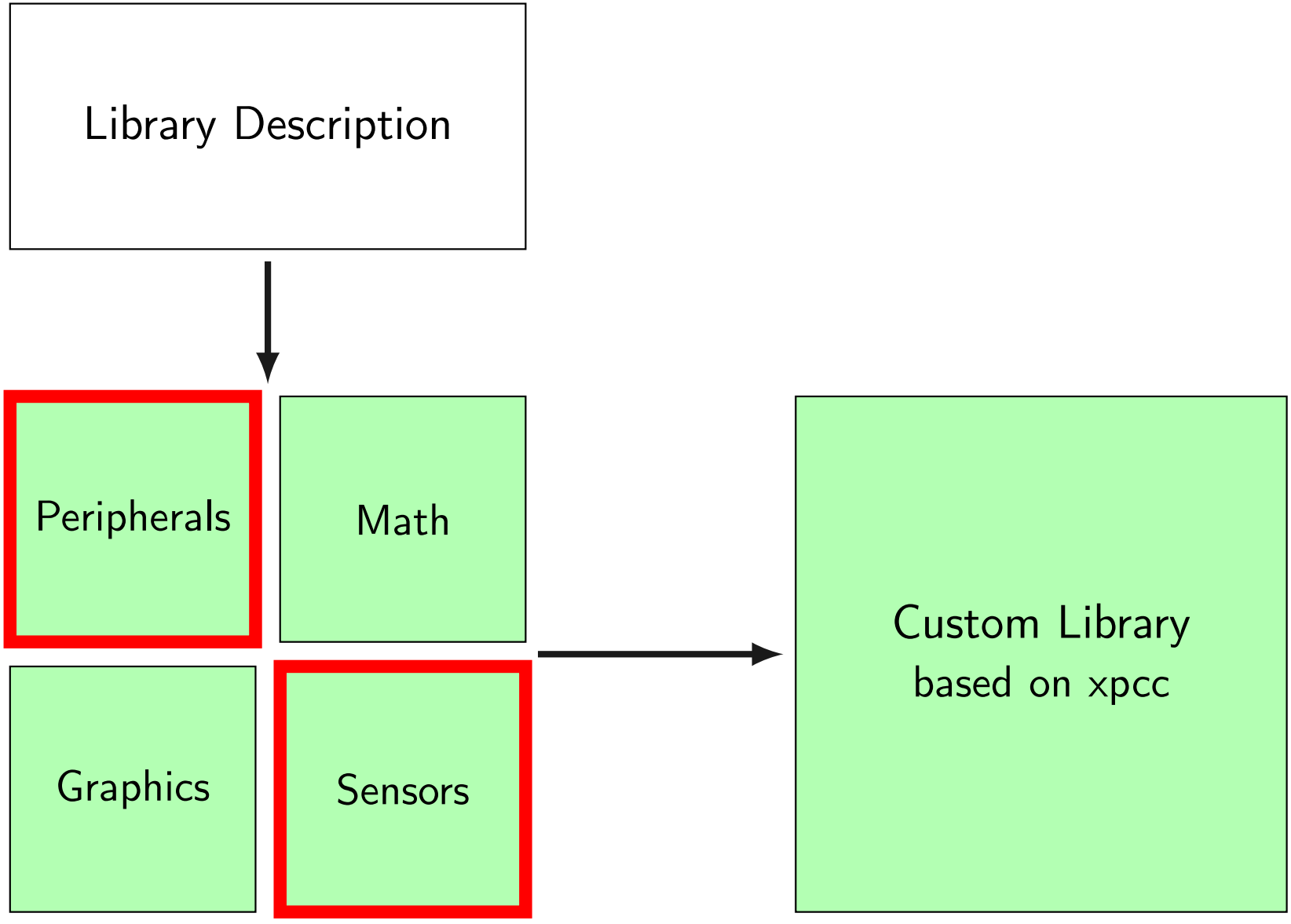
Constructs using

SCONS

with **custom**

python™

Long Term Goal: Library Generator



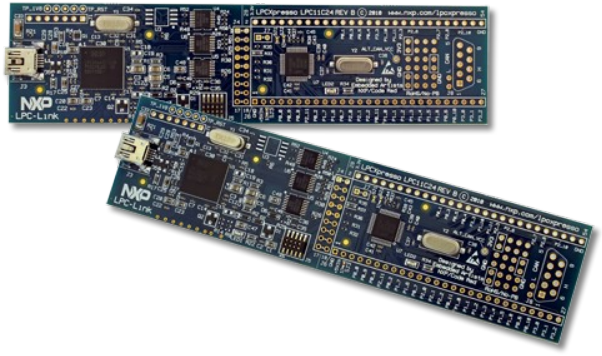
In the meantime ...

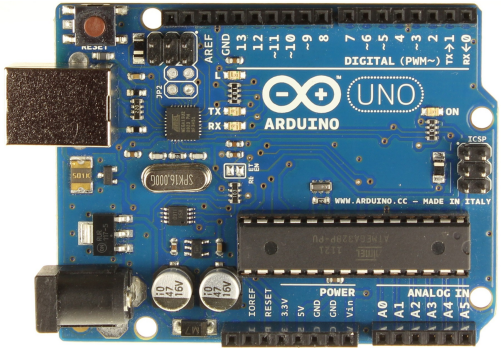
- **use xpcc** in your projects
- improve **documentation**
- **add** IC drivers
- **add** more peripheral drivers, **improve** existing ones
- port to **new platforms**: *Freescale K20, LPC, Atmel SAM D20, MSP430*

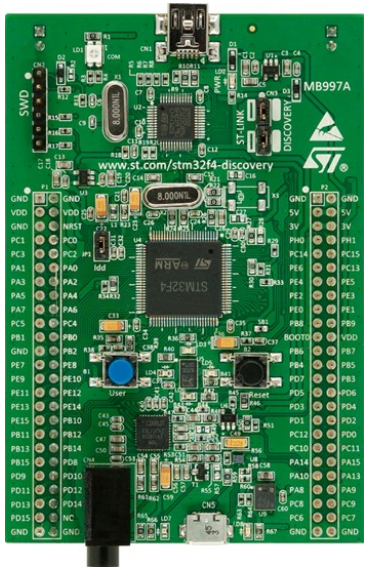
Support

Host OS

Boards







https://github.com/roboterclubaachen/xpcc

xpcc-dev@lists.rwth-aachen.de