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

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

- Algorithms
- Processing, Workflow
- Communication, Protocols
- External Hardware Drivers
- Independent Peripheral Interfaces
- Platform Specific Implementation

Levels:
- Generic
- Intermediate
- Direct access
Visual appearance of microcontrollers

Black box with lots of pins

These pins are used for some form of Input/Output.
Any platform should be capable of satisfying this interface.
Implementation using C-like function calls:

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

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

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

*Inlined code enables atomic GPIO operation.*
Simply **connect alternate functions to pins:**

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

Type is checked at compile time:

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

**Bonus:** **The code is your documentation!**
All targets provide these GPIO methods.
Targets also satisfy other interfaces when available.
Declare **what** you want, not **how** to get it.
Example: UART baudrate

Never set prescalers directly!

$\text{UBBR0 = 8;}$  // 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:

```cpp
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:

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

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

```cpp
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!
```
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.
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

Everything is static.
Representing Peripherals

- TIM14: 1 channel as AF, RX, TX as AF, CTS, RTS as AF
- USART2: RX, TX as AF, CTS, RTS as AF
- USART3: RX, TX as AF
- UART4: RX, TX as AF
- UART5: RX, TX as AF
- SP2/I2S2: MOSI/SD, MISO/SD_ext, SCK/CK, NSS/WS, MCK as AF
- SP3/I2S3: MOSI/SD, MISO/SD_ext, SCK/CK, NSS/WS, MCK as AF
- I2C1/SMBUS: SCL, SDA, SMBA as AF
Representing Peripherals

- `class Timer14`
- `class Usart3`
- `class Spi2`
We want **one static C++ class for every peripheral**!

- We need to know which device needs which classes.
XML Device and Driver Files

- `stm32f407-i_v_z-e_g.xml`
- `peripheral/uart/stm32/driver.xml`
- 1 MB Flash
- 192 kB RAM
- Peripherals
  - ADC
  - Clock
  - SPI
  - UART
  - ...
- Linker
- ...
- `uart.hpp.in`
- `uart.cpp.in`
- `uart_base.hpp.in`
- ...
- `peripheral/uart/stm32/driver.xml`
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

uart.cpp.in
- uart1.cpp
- uart2.cpp
- uart3.cpp
- ...

uart.hpp.in
- uart1.hpp
- uart2.hpp
- uart3.hpp
- ...

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

Library Description

- Peripherals
- Math
- Sensors
- Graphics

Custom Library based on xpcc
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