C++ meta programming and declarative programming in embedded software

Paris Embedded Meetup
2024-10-24 - Paris
Patrick Boettcher YAISE

About me

- Kernel developer since 2004 – (media drivers)
- Embedded C++ and C (most Linux and RTOS)
- Freelancer with YAISE

Overview

1) Introduction
2) Example in C
3) Example in C++
4) Example of C++ Evolution
5) Future

Intro

- What are we looking for in embedded software?

- Fast
  - Minimal amount of instructions to do the job
- Small
  - Small files, memory is still expensive
- Code quality
  - Readable, extensible, maintainable, modular source code

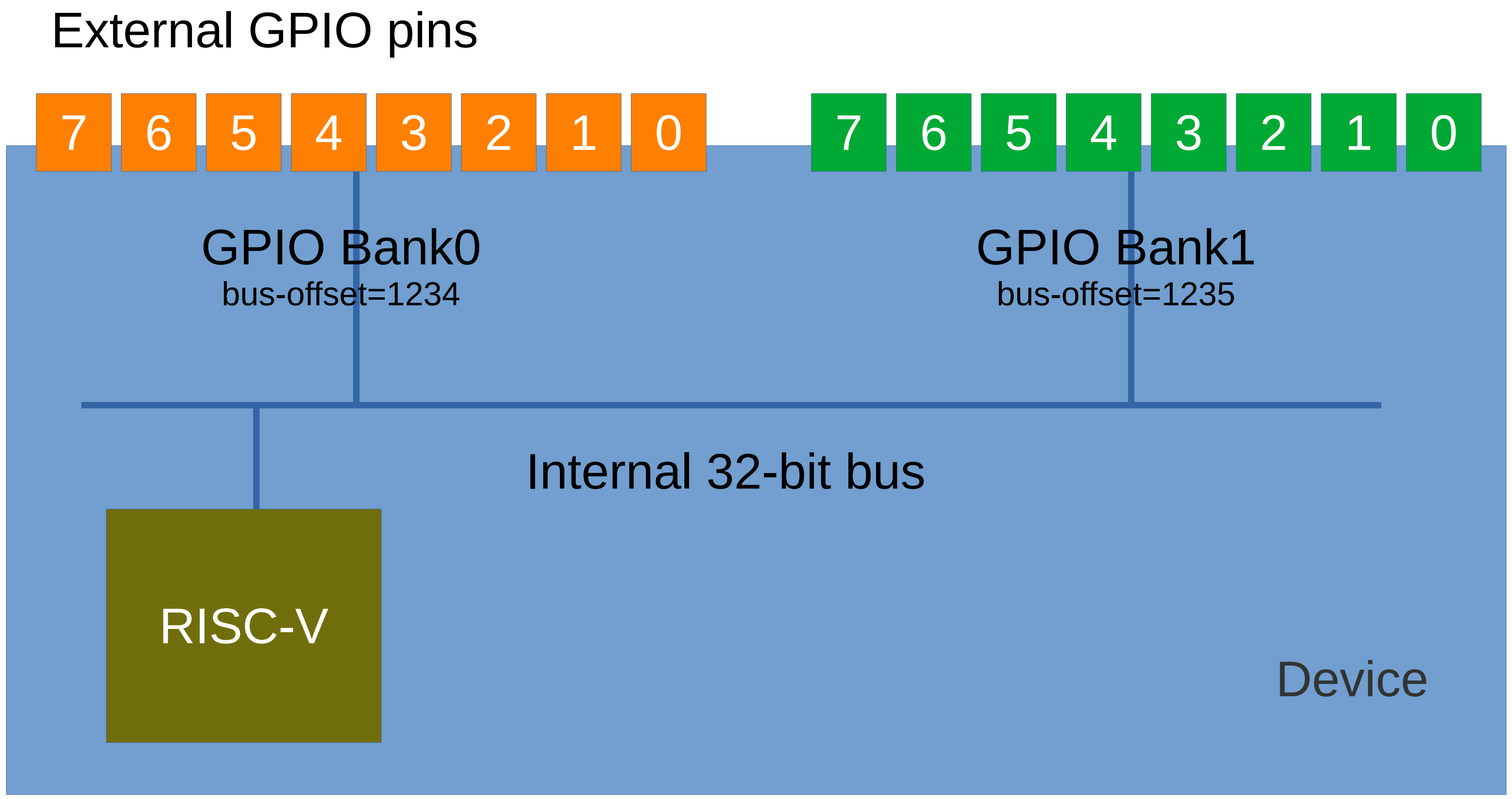
Intro

- Still looking for the magical equation to optimize the result
- Many solutions exists for *Fast* and *Small*
- *Code* quality is often the victim of these solutions
- Compilers are tools which are often misunderstood (or not understood at all)
  - e.g. aliasing: when the compiler cannot proof something is constant, it will load it each time.

Example – a GPIO controller

- There are two GPIO-controllers in our device
- Each one controls 8 bits
- Each one has a different address (32-bit), but the bit-mapping is identical
  - same hardware module is instantiated 2 time
- In our example, multiple GPIOs need to set at the same time (same clock-cycle)

GPIO controller schema



Example – Millenium C

- start.cpp
- step-1.cpp
- step-2.cpp
- step-3.cpp
- step-4.cpp

Example – classic C++

- Create a class of a GPIO-bank
  - *Associate data and functionality in one structure!*
  - Base-offset is a member variable
  - Member functions for setting and resetting GPIO
- Two static instances are created, one for each bank

→ Difficult to share these instances between files

→ The compiler can optimize, but it not all the time

→ we lost control, partially due to not fully understanding compiler constraints

Example – sophisticated C++

- Using the template-mechanism
  - Template-arguments are ctor args - only static methods and members - variadic template functions and folding
  - Templates are classes, typedefs are creating objects!
- Declarative approach (specializing templates)
  - Done in one file for a product. Usable as intrinsic for every code-file, unused stuff, simply removed w/o warnings
- One Definition Rule (ODR) and Visibility related to templates help the compiler tot expand and to inline
  - Sometimes forced to `__attribute__((noinline))`

Summary and future

- Runtime dynamic programming is a killer for embedded software.
- Dynamic and modular programming should be evaluated at compile-time for any product.
  - This pushes constant propagation to pre-compile-time (templates, meta-programming)
- Next step: Intervene at compilation time
  - by using LLVM plugins for module or function passes.

Thanks.

p@yai.se

@PatBoeFra

https://github.com/pboettch

https://yai.se/