Modern C++ for Embedded Systems

Part 1 – Compile-Time Device Models

Alexander Graf

January 25th, 2019

Outline

Terms and Conditions :)

Defining "Embedded"

- interact with physical environment via sensors and actuators
- cost and resource constrained
- talk assumes typical MCUs
  - $<$ 1MB internal program memory
  - AVR, ARMv7M based (e.g. STM32, LPC5), etc.
  - bare-metal or small RTOS
  - or true kernel space programming on larger systems

Terms and Conditions :)

Todays Focus

- The Register Access Layer

Terms and Conditions :)

Todays Focus

- The Register Access Layer

Example Code for STM32F107

https://github.com/alibabashack/cxx-device-models

C-World Problems

Typical Register-Level Mistakes

- Intention: Set Pin B5 high
- ODR = Output Data Register
- IDR = Input Data Register

C-World Problems

Typical Register-Level Mistakes

- Intention: Set Pin B5 high
- ODR = Output Data Register
- IDR = Input Data Register

Mask Confusion 1

```
1 #define GPIO_ODR_ODR_5 ((uint32_t)0x00000020)
2 #define GPIO_IDR_IDR_5 ((uint32_t)0x00000020)
3 #define GPIOB          ((GPIO_t *) GPIOB_BASE)
4
5 GPIOB->ODR |= GPIO_IDR_IDR_5;
```

C-World Problems

Typical Register-Level Mistakes

Mask Confusion 1

```
1 #define GPIO_ODR_ODR_5 ((uint32_t)0x00000020)
2 #define GPIO_IDR_IDR_5 ((uint32_t)0x00000020)
3 #define GPIOB          ((GPIO_t *) GPIOB_BASE)
4
5 GPIOB->ODR |= GPIO_IDR_IDR_5;
```

- ✓ compiles
- ✓ works (by accident)
- ✗ but it is not correct!
- ✗ And it is non-optimal (on STM32)

C-World Problems

Typical Register-Level Mistakes

- Intention: Enable clock for blocks SPI2 and GPIOB
- RCC = Reset and clock control
- APB = Advanced Peripheral Bus

C-World Problems

Typical Register-Level Mistakes

- Intention: Enable clock for blocks SPI2 and GPIOB
- RCC = Reset and clock control
- APB = Advanced Peripheral Bus

Mask Confusion 2

```
1 RCC->APB1ENR |=
2     (RCC_APB1ENR_SPI2EN | RCC_APB2ENR_GPIOBEN);
```

C-World Problems

Typical Register-Level Mistakes

Mask Confusion 2

```
1 RCC->APB1ENR |=
2     (RCC_APB1ENR_SPI2EN | RCC_APB2ENR_GPIOBEN);
```

- ✓ compiles
- ✗ but does not work
- ✗ may introduces side effects on existing behavior
- ✗ may invoke undefined hardware behavior
  (e.g. unexpected power consumption)

C-World Problems

Resulting Issues

✗ Creates lots of problems for future-Homer!

C-World Problems

Resulting Issues

- time wasted for
  - flashing to target
  - creating stimuli/test fixtures
  - lots of manual work
- remaining hard-to-find non-functional bugs
- undetected usage of undefined behavior
  - catastrophic if the chip die changes
  - YES, this happens!

C-World Problems

Resulting Issues

- time wasted for
  - flashing to target
  - creating stimuli/test fixtures
  - lots of manual work
- remaining hard-to-find non-functional bugs
- undetected usage of undefined behavior
  - catastrophic if the chip die changes
  - YES, this happens!

Our aim:

The compiler shall detect illegal/non-optimal register usage!

C-World Problems

Reasons for Compiler Blindness

GPIO memory map definition

```
1 typedef struct
2 {
3   volatile uint32_t CRL;
4   volatile uint32_t CRH;
5   volatile uint32_t IDR;
6   volatile uint32_t ODR;
7   //...
8 } GPIO_TypeDef;
```

- everything is of the same data type (integer)
- thus, we can assign apples to oranges

C-World Problems

Reasons for Compiler Blindness

GPIO Output Data Register bit definition

```
1 #define GPIO_ODR_ODR0 ((uint16_t)0x0001)
2 #define GPIO_ODR_ODR1 ((uint16_t)0x0002)
3 #define GPIO_ODR_ODR2 ((uint16_t)0x0004)
4 // ...
```

- bits of the same field are unrelated language-wise

C-World Problems

Reasons for Compiler Blindness

ODR, IDR: different bits for the same logical pin

```
1 #define GPIO_ODR_ODR_5 ((uint32_t)0x00000020)
2 #define GPIO_IDR_IDR_5 ((uint32_t)0x00000020)
```

- no differentiation between
  - logical and physical structures
  - mechanism and policy[1]

[1] see Linux Device Drivers – Corbet et al.

C-World Problems

Reasons for Compiler Blindness

mechanism

the intention of an action

policy (Verfahrensweise)

how the action is achieved

- no differentiation between
  - logical and physical structures
  - mechanism and policy
- self-documenting code

C-World Problems

Lets Discuss

What can we do to engage the compiler?

- one type per register
- cluster related bits
- separate mechanism from policy
- check invalid value combinations

C-World Problems

The True Challenge

- `enum` and `typedef` are unsafe: implicit conversions
- bit fields also suffer from implicit conversions
- no mechanism to convert types (except casting)
- non-trivial value checks must be performed at runtime

C-World Problems

The True Challenge

- `enum` and `typedef` are unsafe: implicit conversions
- bit fields also suffer from implicit conversions
- no mechanism to convert types (except casting)
- non-trivial value checks must be performed at runtime

Conclusion

The C type system is to weak for our purpose.

C++ Solutions

New Tools

- `enum class` (Scoped Enumerations)
  - define types without default operators or conversions
- user-defined operators
  - define exactly the operations you need
- `constexpr` – running code at compile-time
- `static_assert` – checking conditions at compile-time

C++ Solutions

Sorting Apples and Oranges

GPIO registers with individual types

```
1 struct GpioRegs {
2     PortConfigLowReg CRL;
3     PortConfigHighReg CRH;
4     InputDataReg IDR;
5     OutputDataReg ODR;
6     SetResetMaskReg BSRR;
7     // ...
8 };
```

C++ Solutions

Separating Mechanism and Policy

```
1  class OutputDataReg {
2  public:
3
4      OutputDataReg& operator=(PinId pid) {
5          rawReg = (1<<static_cast<size_t> (pid));
6          return *this;
7      }
8
9      OutputDataReg& operator=(PinMask pm) {
10         rawReg = static_cast<uint32_t> (pm);
11         return *this;
12     }
13
14 private:
15     volatile uint32_t rawReg;
16 };
```

C++ Solutions

Defining Value Types

```
1  enum class PinId {
2      Px0, Px1, Px2, Px3, Px4, Px5, Px6, Px7, Px8,
3      Px9, Px10, Px11, Px12, Px13, Px14, Px15,
4  };
5
6  enum class PinMask: uint32_t {
7      // PinMasks are created indirectly from
       PinId using logic operators
8  };
9
10 gpioERegs->ODR = PinId::Px2;
11 gpioERegs->ODR = (PinId::Px2 | PinId::Px3);
```

C++ Solutions

Converting Value Types

```
1  constexpr PinMask operator|(PinId l, PinId r) {
2      return static_cast<PinMask> (
3                  (1 << static_cast<size_t> (l)) |
4                  (1 << static_cast<size_t> (r))
5              );
6  }
7
8  constexpr PinMask operator|(PinId l, PinMask r)
       {
9      return static_cast<PinMask> (
10                 (1 << static_cast<size_t> (l)) |
11                 static_cast<size_t> (r)
12             );
13 }
```

C++ Solutions

Checking Value Ranges

```
1  template<PinId p, PinMode m, PinConfig c>
2  constexpr PinSetupHigh createPinSetupHigh() {
3      static_assert(
4          static_cast<size_t> (p) > 7,
5          "selected PinId invalid");
6      // ...
7  }
8
9  gpioERegs->CRL = createPinSetupLow<
10             PinId::Px2,
11             PinMode::Output2Mhz,
12             PinConfig::GeneralPushPullOutput>();
```

C++ Solutions
What did we gain?

- Compiler can throw errors for
  - ✓ mixed-up bit masks
  - ✓ illegal value ranges
  - ✓ suboptimal register access
- ✓ impossible to use reserved/undefined behavior
- ✓ zero runtime overhead
- ✗ Need to invest in a lot of new code

Restraints in the Industry

C as Dominating Language

- 70% of Embedded Software is written in C [2]
- 23% is written in C++
- Industry is havily focused to support C
- everything else is optional
- primary solution: building on top of C

[2] Barr Group – Embedded Systems Safety & Security Survey 2018

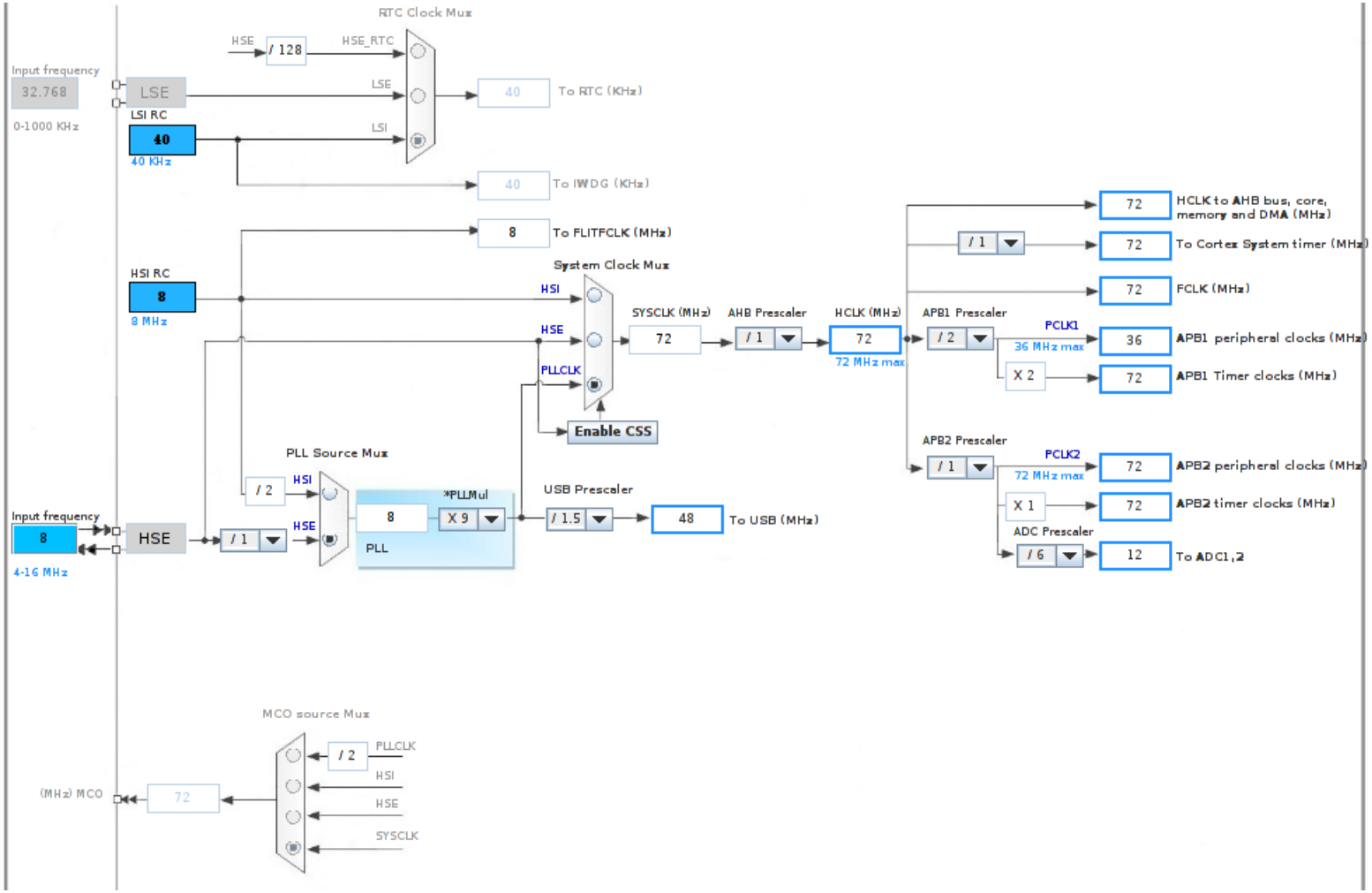
Restraints in the Industry

Building on Top of C

- Peripheral Library with assertions (e.g. ST)
  - assertions only thrown at runtime
  - runtime overhead
- Automatic Code Generation (e.g. ST CubeMx)
  - vendor-dependent tools in build
  - integration of automatic and manual code
  - support stops at the chip boundary

Restraints in the Industry

CubeMx



Conclusions and Discussion

- C++
  - is also suiteable for lowest code levels
  - can do powerfull things without run time overhead
  - can offer advanced diagnostics at compile time
- The industry will likely not support this approach.
- Lets make it free software then!

Conclusions and Discussion

- Q/A

Conclusions and Discussion

- Q/A
- possible future talks:
  - Test-Driven Development for Embedded Systems
  - Using Smart Pointers with Memory Pools
  - Device Driver Architecture