C++ Hardware Register Access

Safer and efficient MMIO

Common approach 1: raw volatiles

```
using reg_val_t = volatile unsigned uint32_t;
using reg_t = reg_val_t*;
reg_t device = reinterpret_cast<reg_t>(0xffff0000);

int main()
{
   *device |= 1;
   reg_val_t status = *device;
}
```

Features

- super common
- predictable runtime

- easy to misuse

Common approach 2: bitfields

```
struct reg_t {
   volatile uint32_t enabled: 1;
   volatile uint32_t flag : 1;
};

int main() {
   reg_t* reg = reinterpret_cast<reg_t*>(0xffff0000);
   reg->flag = 1;
   if (reg->enabled) {}
}
```

Features

- reads well
- performs well

- requires compiler support

The proposal

```
template<typename mut_t, uint32_t addr, int offset, int width>
struct reg_t
{
  static uint32_t read() { /* … */ }
  static uint32_t write(uint32_t val) {/* … */ }
};
```

read

```
static uint32_t read()
{
   mut_t::read(
      reinterpret_cast<volatile uint32_t*>(addr), offset,
      generate_mask_t(offset, width));
}
```

write

```
static void write(uint32_t val)
{
   mut_t::write(
      reinterpret_cast<volatile uint32_t*>(addr), offset,
      generate_mask_t(offset, width), val);
}
```

ro_t, a mutability policy

```
struct ro_t
{
   static uint32_t read(
      volatile uint32_t * device,
      int offset,
      int mask
   )
   { return (*device & mask) >> offset; }
};
```

Usage

```
using flag = reg_t<wo_t, 0xffff0000, 1, 1>;

flag::write(1);
flag::read() // error
```

Features

- volitional
- safer
- can easily translate from datasheet
- mutability policies provide opportunities for
  - unit testing (mock registers)
  - simulation
  - logging
  - profiling

Optimization is a requirement to elide func calls

Moar

- https://github.com/kensmith/cppmmio