C++ Access to Parts and Bitfields

Murrough Landon – 30 March 2000

Objectives

- HDMC GUI displays bit fields of complex registers
- . . . for DAQ, want that detailed access from C++
- HDMC PartManager knows which registers and memories comprise a module
- . . . for the DAQ, want C++ access to known registers as data members of Module classes
- In both cases, HDMC information comes from configuration files
- Dont want to duplicate maintenance, so aim to generate C++ from the configuration files
- Use existing HDMC code to parse them

HDMC Bitfield Access (1)

Typical Register Configuration Entry

```
MyReg {
  1 { SingleBit }
  4 { FourBitField }
  2 { BitMenu Value { 0x0 { ItemOne }
                      0x1 { ItemTwo }
                      0x3 { ItemThree }
                    } }
}
```

Desired Calling Syntax in C++

```
MyReg *myreg;
int value = myreg->SingleBit();
int value = myreg->FourBitField();
int value = myreg->BitMenu();
int oldvalue = myreg->SingleBit(1);
int oldvalue = myreg->FourBitField(6);
int oldvalue = myreg->BitMenu(MyReg::ItemTwo);
```

HDMC Bitfield Access (2)

Likely Implementation

```
class MyReg : public Register {
public:
  static const int ItemOne = 0x0;
  static const int ItemTwo = 0x1;
  static const int ItemThree = 0x3;

  int FourBitField (int value) {
    int regval = Register::read();    // Or use cached value?
    int oldval = (regval >> 1) & 0xF;
    regval &= (~0xF) >> 1;
    regval |= (value & 0xF) >> 1;
    Register::write (regval);         // Or caller does this?
    return oldval;
  }
}
```

HDMC Bitfield Access (3)

Problems and Unresolved Issues

- Register format is defined in configuration file, but read-only flags are in the parts file. Should/could they be moved? [NB these flags may change to support verify() method].
- Create one class per register format? Or one class per register part per module?
- Ideally, only define read functions for readable bits and write functions for writeable bits.
- Check that value for sparse “bit menu” type field is one of the known values; exception if not?
- Check that value for bit field is within range of bit field size; exception if not?
- Should Register (or suitable subclass) cache the last value read or written. Otherwise write functions need read/modify/write cycle.
- Should bit field manipulation functions actually write back the new register value; or use a separate MyReg::write() using the modified cached value? Or provide both options?!
- With a compiler, these choices are simple to change. . .

HDMC Module Parts (1)

Module Subclasses

- Can add Module data members for known components of specific Modules
- Can initialise these from PartManager. . .
- Some data members will be specific register subclasses created to allow access to their bit fields
- Useful methods for DAQ activities can use specific data members to initialise, download, read out specific module subclasses

HDMC Module Parts (2)

Possible Implementation

```
MyModule.h:              (write by hand)

#include "MyModule_include.h"
class MyModule : public Module {
  // ...module specific methods...
  #include "MyModule_parts.h"
}

MyModule_include.h:      (generated code)

#include "MyModule.ControlReg.h"
#include "MyModule.StatusReg.h"

MyModule_parts.h:        (generated code)

public:
void initialise_parts() {
  // Use inherited (to be written!) function to find known
  // Register Parts from PartManager. Exception if not found?
  m_ControlReg = find_part_by_name("MyModule.ControlReg");
  m_StatusReg = find_part_by_name("MyModule.StatusReg");
}
private:
MyModule.ControlReg *m_ControlReg;
MyModule.StatusReg *m_StatusReg;
```

HDMC Module Parts (3)

Possible Implementation

MyModule_Gen.h: (generated code)

```
  #include "Module.h"
  #include "MyModule_ControlReg.h"
  #include "MyModule_StatusReg.h"
  class MyModule_Gen : public Module {
  public:
  void initialise_parts() {
    // Use inherited (to be written?!) function to find known
    // Register Parts from PartManager. Exception if not found?
    m_ControlReg = find_part_by_name("MyModule_ControlReg");
    m_StatusReg = find_part_by_name("MyModule_StatusReg");
  }
  protected:
    MyModule_ControlReg *m_ControlReg;
    MyModule_StatusReg *m_StatusReg;
  }
```

MyModule.h: (write by hand)

```
  #include "MyModule_Gen.h"

  class MyModule : public MyModule_Gen {
  public:
    // ...module specific methods...
  }
```

HDMC Miscellany

Other Improvements Under Discussion

- Verify methods: add verify() method to Part. This should (optionally) iterate over its component Parts and verify() them all.
- For Registers and Memories, verify() should check all bits and some patterns. A few levels of testing as in the UK diagnostics?
- Status/message area to be added to main window?
- Split up default.conf into #included sections for easier maintenance? Makefile could use C preprocessor to recreate the complete file.

Further Suggestions?