HDMC: An object-oriented approach to hardware diagnostics

U. Pfeiffer, V. Schatz, C. Schumacher (e-mail: schumacher@asic.uni-heidelberg.de)
Kirchhoff-Institut für Physik, Ruprecht-Karls-Universität Heidelberg, Germany

M. P. J. Landon
Queen Mary and Westfield College, London, UK

Abstract

A software package has been developed which provides direct access to hardware components for testing, diagnostics or monitoring purposes. It provides a library of C++ classes for hardware access and a corresponding graphical user interface. Special care has been taken to make this package convenient to use, flexible and extensible. The software has been successfully used in development of components for the pre-processor system of the ATLAS level-1 calorimeter trigger, but it could be useful for any system requiring direct diagnostic access to VME based hardware.

I. Introduction

Developing electronics involves a fair amount of testing, where direct access to hardware via a computer is required. In addition to low-level test tools like oscilloscopes or logic analysers, higher level diagnostic facilties are essential for more complex tests. This includes software to access the developed hardware in an extensive and easy-to-use way to perform diagnostics and monitoring of individual or complete groups of components. Similar functionality is required for later integration in extended hardware and software frameworks.

The software package presented, called HDMC (Hardware Diagnostics, Monitoring and Control), addresses these needs. It provides a library of components for accessing hardware objects like registers, memories or FPGAs on VME modules or within devices not directly accessible to VME, but located on a VME module. It's also possible to access a VME bus via a network connection in a client/server configuration. A graphical user interface based on this library provides hardware access without requiring special knowledge about software development. The library can also be used for more direct access based on compiled or scripting programming languages for testing or integration into other software environments.

II. Hardware Access Framework

HDMC is implemented as a set of C++ classes, representing hardware components in a common framework. This is used to provide common ways to access similar components, to transmit data between components and to handle them in a uniform way. A simple and clean interface for direct hardware access is provided as well as a more abstract one for access through a graphical user interface. Register descriptions are loaded from human-readable configuration files in such a way that a lot of hardware development can be made without the necessity to recompile the software.

The basic HDMC framework is formed by two kinds of classes, one representing hardware objects like registers and memories and the other representing the access to these objects, for example VME bus access.

A. Hardware Objects

Hardware objects are modelled as a hierarchy of classes, separating the interface to access the hardware from the implementation of the hardware access. That means that from a software point of view e.g. a register can always be accessed in the same way by calling read and write functions regardless of whether it is implemented as a VME register, inside an ASIC or anything else. This separation of interface and implementation is one benefit of the object-oriented approach.

Another advantage of object-orientation is the reuse of code. Functions which are useful to several implementation are only implemented once in a base class and can then become part of different implementations of hardware objects by inheriting this base class. For example the configuration protocol of a Xilinx FPGA is implemented in a base class `FpgaXilinx` and is then used by all objects accessing Xilinx FPGAs, which is on different modules in different ways.

By making these abstractions of the hardware objects in an object-oriented class hierarchy two goals are met. A uniform and compact interface to access hardware objects is built, and the addition of new hardware components to the software model is made independent of code accessing the hardware by these interfaces. This has the effect that new hardware implementations gain all the functionality present in the software framework for the common interface of this implementation. For example, the graphical user interfaces and test algorithms of registers and memories work for all kind of registers and memories regardless of their actual implementation. They just use the common programming interfaces of all register and memories, which provide functions to read and write the register or a certain address of the memory.

The upper part of figure 1 labeled hardware components shows some of the classes representing hardware objects. The FeAsic is an ASIC developed in the Heidelberg ASIC laboratory, which was used in hardware tests peformed with the HDMC software package.

B. Hardware Access

The access to the hardware components, which were decribed in the previous section, is done by a separate tree of classes. These implement the underlying functions required for hardware access and are not directly available to the user. The

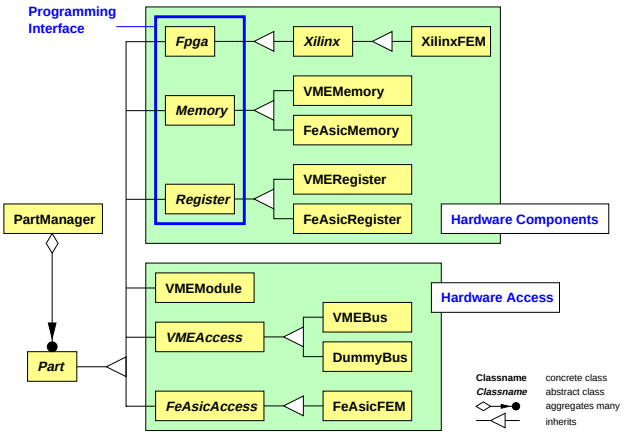

Figure 1: Parts class hierarchy

connection between the component and access classes is done by so-called *dependencies*, which are special object references manipulable by the user. They represent the dependencies of hardware objects in terms of access.

For example, to access a register on a VME module the base address of the VME module has to be known, i.e. the register object is dependent on a module object. The module has to know how to access the VME bus on the platform the program runs on, i.e. the module object is dependent on a VME bus object. Figure 2 illustrates these kinds of relationships.

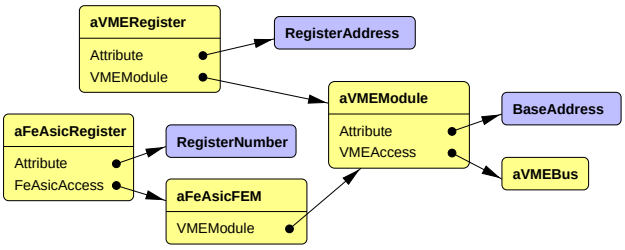

Figure 2: Parts dependencies example

In addition to dependencies, HDMC classes can have *attributes*. These attributes define certain properties of the software objects, which are required for operation, and are used to parametrize them. For example, a register has an address attribute, a module a base address, and a memory a base address and a size attribute.

Like the hardware component classes the access classes are also organized in a hierarchy separating interfaces from implementations. That means that a hardware object can be used with different implementations of access to it, because it relies only on the interface common to all implementations of the relevant access type. The VME register class for example can be used on different hardware platforms by using the appropriate VME bus access implementation, which handles the access to the VME drivers in a platform-dependent way, but provides platform-independent functions to the outside.

One advantage of the way of handling dependencies is that they can be changed at run-time. This makes it possible for the user to create or reorder the software model of the hardware to be accessed without compiling or restarting software. This allows, for example, temporarily replacing a real VME bus access by a dummy access.

C. Handling of hardware classes

All hardware classes, components and access, inherit from a common base class called *Part*. This Part class provides some basic functionality to all classes of the HDMC framework like a name, type identification and management of dependencies. It also allows handling of creation, deletion and manipulation of Part objects in a uniform way independent of the concrete class. These tasks are performed by a class called *PartManager*, which is responsible for the management of all Part objects in a running instance of HDMC.

This handling of arbitrary hardware component and access classes in a uniform way is a prerequisite for adding a graphical user interface on top of the basic framework. It makes it possible to let the user of HDMC create the software objects needed to access the hardware under test at run-time with a graphical user interface in a convenient and flexible way. The underlying hardware access classes remain independent of the graphical user interface and can also be used to build hardware drivers or other programs not directly accessible by the user.

D. Software Components

The HDMC package includes a rich set of components. It supports all kinds of VME registers and memories, Xilinx FPGAs, a simple I^2C master emulation, and a lot of custom hardware used in the environment of the Pre-Processor system of the ATLAS Level-1 calorimeter trigger, for example the PipelineBus [3]. Two sets of components are described below in more detail, bus access and data movement.

E. Bus Access

HDMC includes several implementations of VME bus access. There is support for a number of different drivers on PowerPC, Motorola or Pentium-based single-board computers and a dummy implementation emulating a VME bus without performing hardware accesses.

In addition there is a network-transparent VME bus implementation, which makes it possible to run HDMC on one computer and perform the actual hardware accesses on another one. A VME client/server combination transmits requests and results of bus accesses over the network using a format on top of TCP/IP. This networked bus allows the control of hardware sitting in different VME crates from a central location. Another advantage is that it allows to use a combination of simple VME single-board computers with standard PCs to control VME hardware instead of expensive full-featured computers for VME crates.

F. Data Movement

A basic requirement for testing and diagnosing hardware is the ability to move data through the hardware. It has to

be possible to load and read memories, to compare data from different origins or at different times and to view the results. HDMC provides a general mechanism for this purpose based on so-called *I/O-Frames*.

The idea behind this is that each component able to be a source or a destination of data is surrounded by a frame providing ports for input and output of data and where appropiate a trigger port, which is used to initiate a data transfer at the output. By connecting outputs of one I/O frame to inputs of other I/O frames a chain is built suitable for movement of data between objects.

In figure 3 such a chain is shown, where a data source is connected to the input of a memory and the memory's output is connected to a histogram. By triggering the data source the memory is filled with data and by triggering the memory the histogram is filled with memory data. By connecting a periodic signal to the memory trigger port it is possible to establish a simple kind of data aquisitation suitable for testing and monitoring hardware.

Data is moved in blocks and controlled by the state of the participating I/O frames. That means that on each I/O frame connection data is moved until either the sender runs out of data or the receiver runs full. No addressing is needed. The data is moved in the sequence provided by the I/O frame.

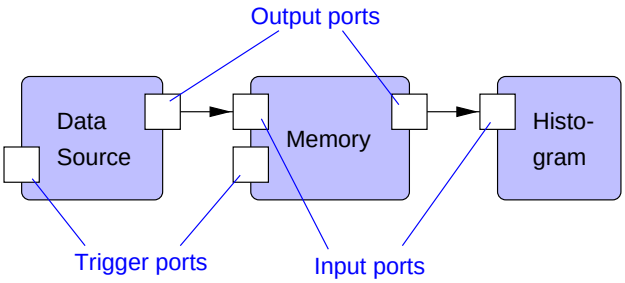

Figure 3: Data movement mechanism

Because the I/O frame classes are integrated in the Part class hierarchy, all the features of creating or manipulating Parts become also available for them. This allows the user of HDMC to dynamically arrange data sources, sinks and connections to efficently adapt to the changing needs of hardware diagnostics.

III. Graphical User Interface

The graphical user interface (GUI) allows construction, manipulation and access to VME modules and other components in a convenient and uniform way. Access to hardware configurations can be built using the interface and changed at run-time. There is also a plot and histogram component and facilities to present special views of hardware configurations like modules and crates. The graphical user interface is based on the cross-platform Qt toolkit by Trolltech [4].

Figure 4 shows the HDMC main window. It displays a tree list of the existing Part objects representing hardware components in the system under test. The user interface

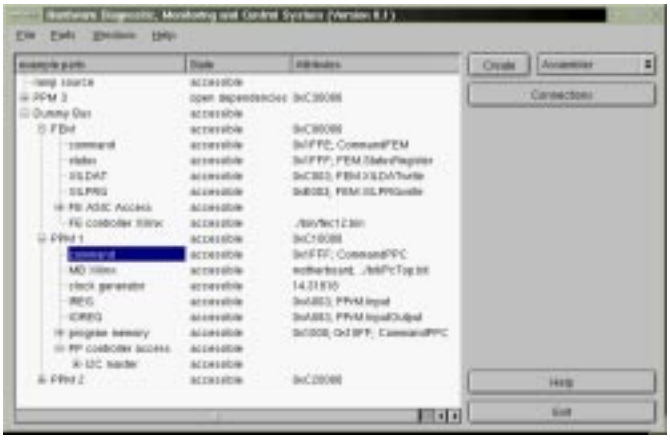

Figure 4: HDMC main window displaying the current Part hierarchy

provides facilities to add new components, to change existing ones and their relationships, to load and save the resulting configurations to disk and to access individual hardware components.

Each Part can have an associated GUI class, which provides access to this Part and its subclasses. By providing GUIs for the abstract base classes that form the programming interface of the hardware access framework, hardware components get a consistent interface. For example all registers get the same interface. One type of register interface is shown in figure 5.

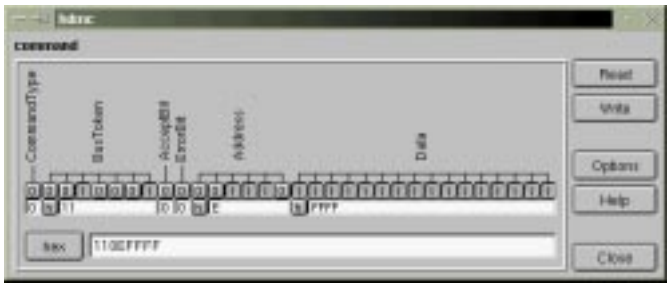

Figure 5: Graphical user interface for a register

The graphical user interface also provides access to the I/O frames used for data movement, which were described in section F. Figure 6 shows a data plot produced by connecting a data source producing random data to the histogram I/O frame.

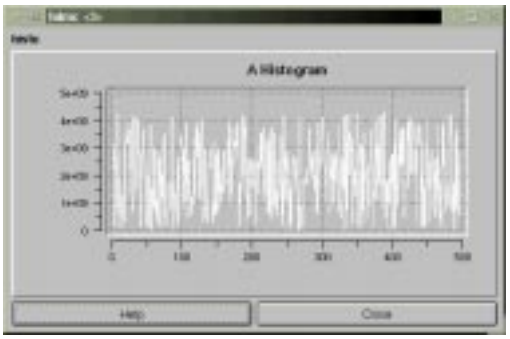

Figure 6: Histogramming and Data Plotting Part

In addition to the access-oriented way to view hardware configurations presented by the main window, HDMC provides other views on the hardware, which are more based on the physical appearance of the hardware. Examples are a module view and the crate view shown in figure 7.

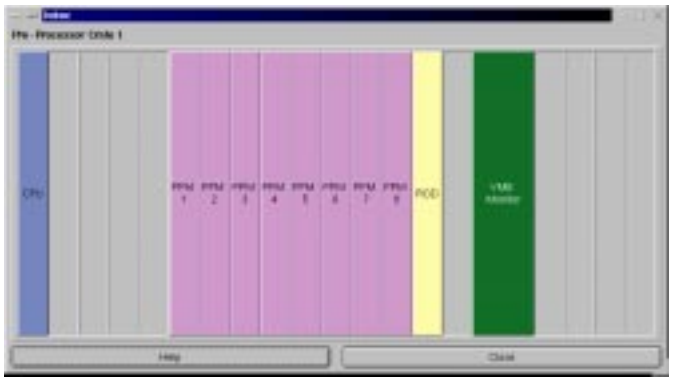

Figure 7: Crate View

It is also possible to combine the user interfaces of several hardware components in a single window. This makes it possible to have for example a compact view of the same register on different modules.

IV. Scripting

In addition to the graphical user interface, the hardware access framework of HDMC formed by the Part class hierarchy can also be used directly from C++ prgrams. Figure 8 shows an example of how a register is accessed from C++, which gives an impression of the programming interface of the Part classes.

```
#include "HDMC.h"

BitMapper::init("default.conf");

DummyBus bus;
Module mod ( &bus, AddressD8 ( 0xc00000 ) );
ModuleRegister32 reg ( &mod,"FEC.CtrlReg",
                       AddressD32 (0x100) )
if (reg.verify()) {
   reg.set("GenerateTrigger",1);
}
```

Figure 8: C++ code example for using the hardware Programming Interface

For hardware testing, access by a graphical user interface or a compiled language is sometimes not flexible, efficient or direct enough. In this case it is desirable to have scripting capabilities, which allow to perform tests in an interactive way which can gradually move on to a batched mode. Script languages like Python, Perl or Tcl provide these facilities and HDMC supports scripting from all of these languages.

The programming interface of the Part classes is exported to module libraries which can then be used from the scripting languages. The export process is done by a software package called SWIG (Simplified Wrapper and Interface Generator) [5], which automatically creates the wrapper code required for the language bindings from the C++ header files of HDMC. SWIG reduces the amount of work needed for extending the scripting language bindings to new classes to almost zero.

The C++ programming interface is reflected in the scripting languages. Only slight differences in the syntax are caused by the different languages. In figure 9 a piece of Python code is shown, which has the same functionality as the C++ code shown in figure 8.

```
from hdmc import *

BitMapper_init( "default.conf" )

bus = DummyBus()
mod = Module( bus, AddressD8( 0xc00000 ) )
reg = ModuleRegister32( mod, "FEC.CtrlReg",
                        AddressD32( 0x100 ) )
if reg.verify():
   reg.set( "GenerateTrigger", 1 )
```

Figure 9: Python example script accessing hardware

V. Word format definitions

An ever recurring task in writing software directly accessing hardware components is the definition of word formats for registers, memories, readout data etc. The approach taken for HDMC tries to minimize the work necessary to introduce new formats or change existing ones.

All word formats are defined in a human-readable configuration text file, which is read and interpreted by HDMC. The word format definitions are used by components like registers and memories to present a view to the user, which does not require remembering bit numbers or locations of various fields. Figure 10 shows a configuration file for a register format. The corresponding representation as graphical user interface is shown in figure 11.

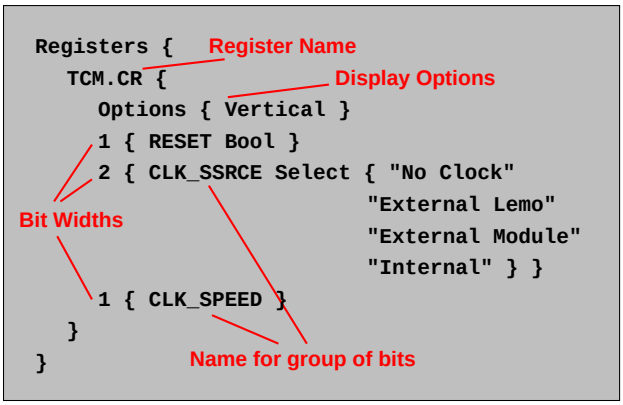

Figure 10: Word format definition

Figure 11: GUI generated from the configuration file shown in figure 10

Holding the word format definitions in a separate file means that it is not necessary to recompile HDMC in order to change word formats. Since modifications of register and other formats are a quite common situation in hardware development, the HDMC approach eliminates the need for many compilation cycles and makes it possible to adapt to a lot of hardware changes at run-time via the graphical user interface.

Also, many I/O frame Parts use the word format definitions. There is an Assembler Part, which converts textual commands in a binary representation suitable for loading in a memory. The Disassembler Part does the conversion in the opposite direction by converting the binary content of a memory in a textual representation. This works for all stateless command formats, i.e. formats where the interpretation of the different bit fields is not dependent on the sequence of words. A *DataExtractor* Part allows to extract the values of a single field of a word for example for histogramming or plotting.

When a very fast and efficient access to register fields is required the interpretation of a textual description of the format takes too much time. In this case it is desirable to have a direct definition of the elements concerned in C++, which can be resolved at compile-time in order to produce the optimal access to the hardware in terms of speed. For this case HDMC provides a set of conversion tools, which translate the word format definitions to C++ code. The generated files can be included by a program, which then can access register fields by native C++ constructs.

Another application of the word format definition files is the automatic generation of Verilog code. For digital logic designs this is a time-saving approach to create the necessary code for registers. HDMC produces synthesizable Verilog code for registers from the word format definiton files. This also means that the software to access the registers is ready at the same moment when the hardware implementation is done.

VI. Platforms and Development

HDMC supports a variety of UNIX platforms including Linux, Solaris and HP-UX, For VME access several VME single-board computers are supported, running Linux or LynxOS. Platform support could be extended to Windows without a major rewrite, and addition of other bus systems like CompactPCI is possible without change in the remaining framework or components.

An open-source process is used for development of HDMC. Source code and documentation is publicly available on the internet and it is open for contributions of any interested party.

VII. Conclusion

The software package has proven to be a useful and reliable tool for diagnosing hardware. It has been used for the pre-processor system of the ATLAS level-1 calorimeter system, whose current development activities are based on a flexible VME test system [2], but it is not limited to this system. Other systems in need of a software tool for hardware diagnostics could also benefit from the HDMC software.

The HDMC software package including source code and documentation can be obtained at the web pages of the ATLAS group at the *Kirchhoff-Institut für Physik* of the University of Heidelberg [1].

VIII. References

[1] HDMC homepage, `http://wwwasic.kip.uni-heidelberg.de/atlas/projects/hdmc.html`

[2] Volker Schatz, *Test of a Readout and Compression ASIC for the ATLAS Level-1 Calorimeter Trigger*, HD-KIP 00-13, Heidelberg, June 2000

[3] Cornelius Schumacher, *The Readout Bus of the ATLAS Level-1 Calorimeter Trigger Pre-Processor*, Fifth Workshop on Electronics for LHC Experiments, Snowmass, CERN/LHCC/99-13, 29 October 1999

[4] Troll Tech, `http://www.trolltech.com`

[5] Simplified Wrapper and Interface Generator (SWIG), `http://www.swig.org`