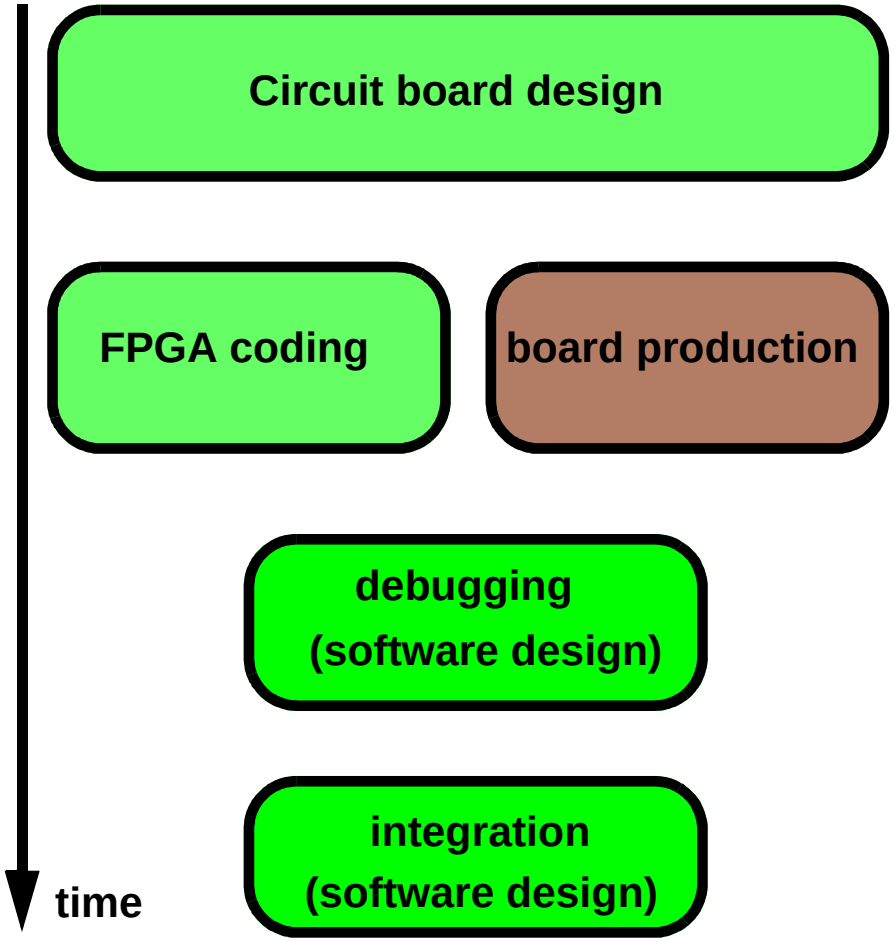
HAL (Hardware Access Library)

http://cmsdoc.cern.ch/~cschwick/software/documentation/HAL/index.html

- Motivation for this library
  - The problem
  - Requirements
- Features
  - what does this library do for me
- Structure of the library
- CMS related issues
- Documentation

Motivation

- Debug / Development phase
  - a lot of trial and error: first access to the card, play with configuration registers, find bugs in state-machines...
  - a lot of firmware changes (new registers, new state machines, changing initialization procedures)
  - hardware experts are rarly software experts; they need tools to test their cards under realistic conditions (data transfer rates, number of accesses per time,...) without writing linux kernel code or similar.



- Prototype comissioning
  - electrically the board works already
  - write procedures to operate the hardware in test environment (test bench, test beam) in order to measure the performance or find more sophisticated shortcomings.
  - module will have to work in changing hardware environment (new crate controller,...)
  - module will have to work in changing software environment (new bus-driver, new operating system, new application environment (stand alone -> XDAQ)
- Protype versions
  - functionality might change
  - functionality remains but technology might change (VME to PCI)

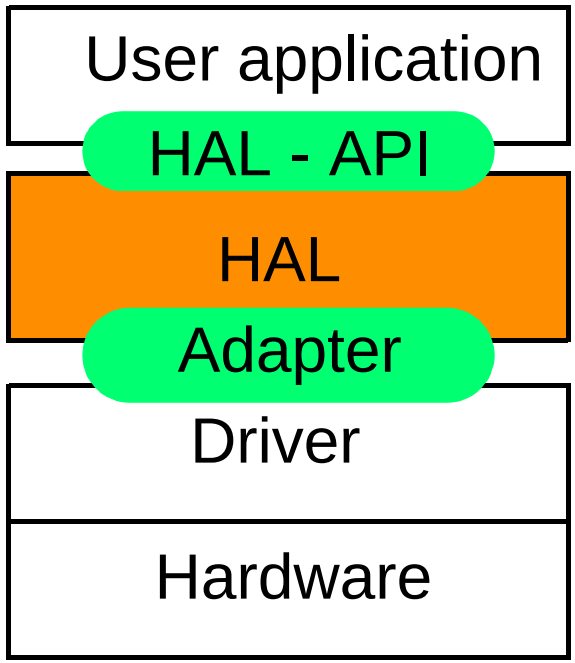
Requirements for the HAL

- User friendly usage
  - Non experts should be able to write test software “immediately”
  - Changes in hardware (new registers, change of initialization procedures) should be painless to implement
  - The address map should not be hardcoded in the software
  - Inputs to the library (address tables, sequences of commands) might use different techniques: ASCII-files, xml-files, database, ...
- The library should do something for you:
  - Anoying work should be done by the library:
    - masking and shifting around of bitfields
    - bookkeeping of the addresses and masks of the logical items to be accessed
  - Excecuting of simple sequences or scripts which might be parametrized
  - Debug option: read back and check every value written into a place (... which is also readable...)
- The Library should also allow for efficient data transfer

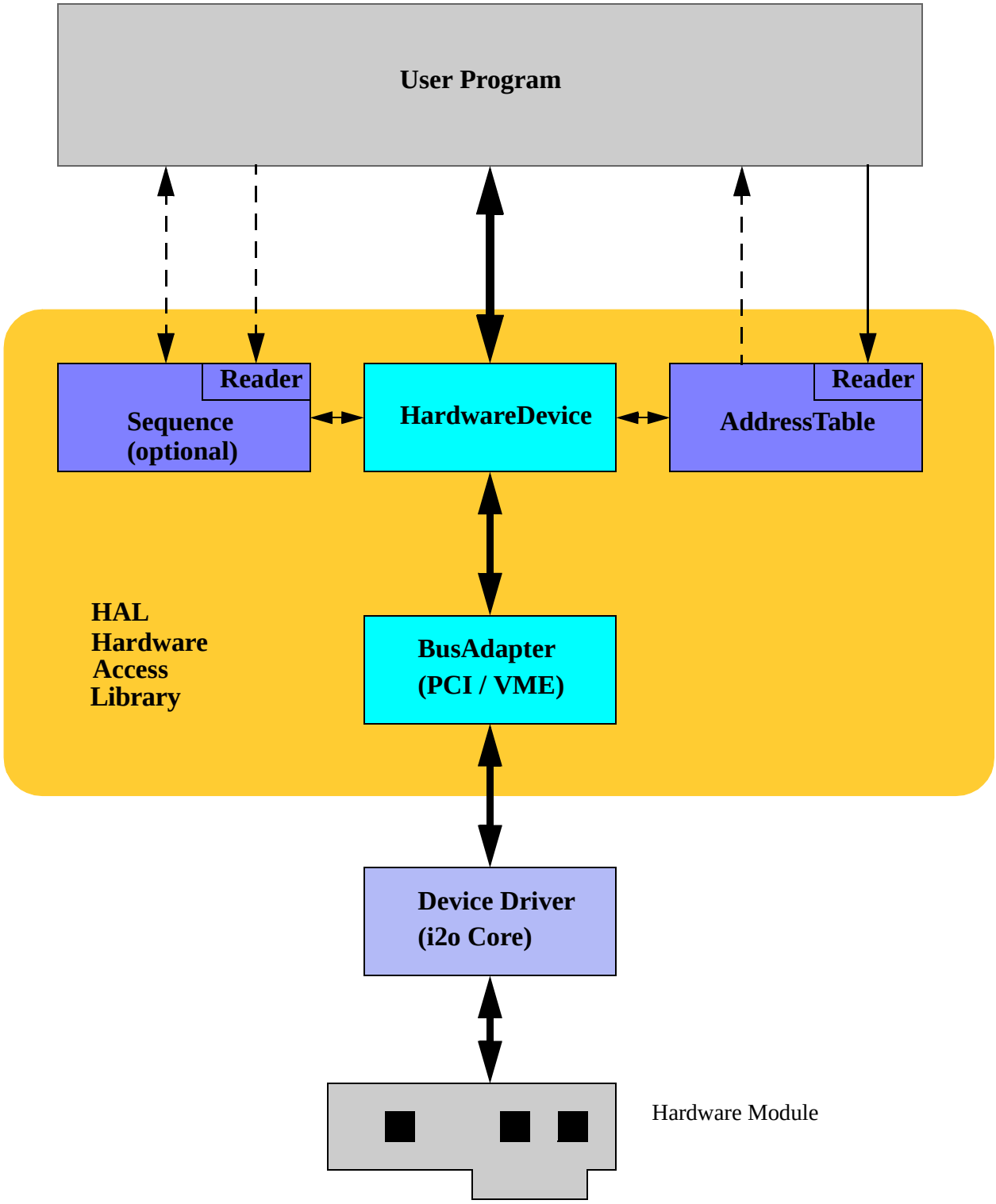
- Operating system independent
  - At least Linux and VxWorks
- Technology independent
  - (This means in practice today: VME and PCI)
- Do not implement functionality which is not needed
  - This helps to maintain the code.
  - Never implement something which may be somebody one day will use (... or not...)
- The library must be well documented
  - This allows non-experts to use it
- The Library must be compattible with CMS DAQ software environment (XDAQ)
  - It is used in the CMS online environment
  - It is actually a part of XDAQ
  - It must be usable also independent of XDAQ

Architecture and Components

- Layered architecture in order to separate:
  - User application (high abstraction level)
  - Procedures to access hardware (HAL)
  - Driver level
  - Physical layer (the hardware system)
- This decouples User application from :
  - Hardware access method
  - Operating System
  - Driver Software
  - Hardware technology



The principal components



Essentials of the API

- All accesses are based on the Address Map of the module
  - Example : VME AddressMap

Table entry	description
item	logical object to be accessed in module
address	relative to baseaddress
mask	define bits relevant to the item
readFlag	determines of item is readble
writeFlag	determines if item is writable
AM	addressModifier
dataWidth	in bytes (2 or 4)
description	documents the item

- for PCI you have ‘accessMode’, ‘BAR’ instead of ‘AM’ and ‘dataWidth’
- One AddressTable for n equivalent modules (with of course different baseaddresses)

- To access the hardware: choose a BusAdapter for your system
  - this is the interface to the hardware driver.
  - It has a simple API to the towards the HAL
  - BusAdapters exists for :
    - PCI on Linux PCs and Vxworks Motorola CPUs with PMC slots
    - VME for Vxworks Motorola CPUs, SBS PC-VME Interfaca, National PC-VME interface (under development)
  - It is easy (and documented) to write a new BusAdapter to your custom system.

Functions to access the hardware

function	description
write, read	write (read) logical item into (from) hardware (do not change bits which do not belong to the item, shift the bits to the correct bitfield position)
unmaskedWrite, unmaskedRead	write (read) a register (do not care about bits)
setBit, resetBit, isSet	set, reset, test a single bit
writePule, readPulse	like writing (reading) to a specific address in order to trigger a side effect (an action). This function makes programs readable.
writeBlock, readBlock	transfer blocks of data between module and host as fast as possible.

- optional arguments:
  - offset
  - verifyFlag for items which write data (exception: writeBlock)

A complete Program example

```
#include “VMEDevice.hh”
#include “VMEAddressTable.hh”
#include “VMEAddressTableASCIIReader.hh”
#include “SBS620x86LinuxBusAdapter.hh”
#include <iostream>

int main() {
  try {
    SBS620x86LinuxBusAdapter busAdapter();
    VMEAddressTableASCIIReader addressTableReader( “MyAddressTable.dat”);
    VMEAddressTable addressTable( “Test address table”, addressTableReader );
    VMEDevice myCard(addressTable, busAdapter, 0x84000000);
    unsigned long data;

    myCard.write(“threshold0”,0x34);
    myCard.read(“adcValue”, &data);
    cout << hex << “adcValue is : “ << data << endl;
  } catch ( HardwareAccessException e ) {
    cout << e.what() << endl;
  }
}
```

input: ASCII file

read AddressTable

create device

write

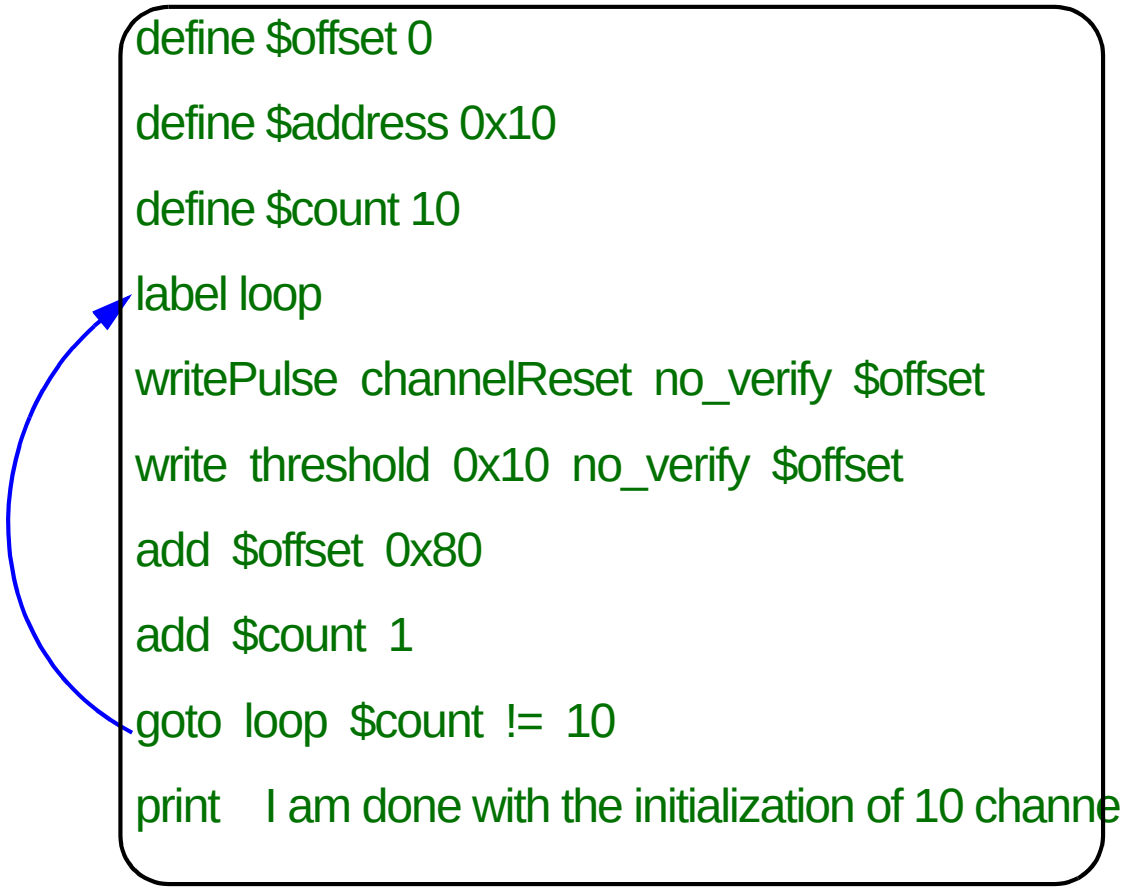
read

catch problems

The Sequencer

- Allows to store small procedures in ASCII files
  - equivalent to simple scripting language
  - parameters can be given by user application to the squence
- Example
  - Initialize 10 equivalent channels where only an address offset changes
  - The variables are accessible by the application.
  - This sequence can be used on different modules (the baseaddress is added automatically)
  - Another example: debug the programming of a DMA engine

A sample sequence to configure 10 channels



Conclusions

- Requirements are fullfilled
- Library in use since three years (Atlas 1st level trigger and CMS DAQ)
  - has been used for VxWorks, Linux, LynxOS
  - with PC, Motorola MV2304, CES-RIO2
  - With VME and PCI
- Since developed in house and fully documented it is easy to adapt for CMS needs
- It is part of XDAQ and therfore fully supported
  - FED developers will most probably use XDAQ in their test environments
- Complete documentation and download available

  http://cmsdoc.cern.ch/~cschwick/software/documentation/HAL/index.html
  - for drivers contact me