Describe common strategy for the representation of the devices and generic communication protocols for the access to the board resources

Current implementation on the LHCb Timing and Fast Control system is using the tools
- PVSS II – an industry SCADA system from ETM
- DIM – A light-weight inter-process communication package over TCP/IP

The ideas presented are not linked to the tools
- Not a publicity talk for PVSS and DIM but very powerful
General Aim

- Aspects of implementing generic and integrated remote control of electronics devices
  - Define a generic data structure which reflects the control configuration of a board and which interfaces to
    - Local control actions
    - GUI display
    - Overall expert system for automation
  - Provide an interface between the functional view and the hardware view of the system
  - Provide a simple and economical remote access protocol to any board resource type independent of the bus type.
  - Provide a simple and economical protocol which allows monitoring counter and status information
Control Architecture with PVSS and DIM

- Database and data manager – dynamic representation of all controllable resources
- Event manager
  - Associate functions in scripts or in API Managers to changes of values
  - Data subscription from GUIs and functions in scripts and API Managers
- Communication protocol based on server-client paradigm
- Control interface
Control Interface

- One-to-one relation between control interface and electronics device
  - Many advantages (parallel, distributed control, fault impact, cheap, etc)
  - In LHCb Credit card sized PC from Digital Logic AG, Switzerland
    - Real-time control using embedded micro-controllers, Niko Neufeld, RT 2005
  - Glue logic based on FPGA (or ASIC) to produce board busses

- Control Interface Server running on embedded controller compiled with libraries for
  - Network communication with Control System (DIM)
  - JTAG STAPL player for FPGA/configuration device programming over PCI (TCP/IP)
    - (Sukhanov PS1A003, Kaemmerling PS1C002, Koestner PS1D003)
  - JTAG Boundary scan over PCI
  - PLX Local Bus access over PCI
  - I2C access over PCI
  - etc

- Today rather a SOPC solution (e.g. Nios II)
Control Communication Protocol

- A set of fours pairs of generic commands and services per device to perform all control actions published by Control Interface Server
  
  - FPGA/Configuration device programming
    
    **DownloadFPGA:**
    ```c
    struct [] { identifier, code }
    ```
    
    **FPGADownloadStatus:**
    ```c
    struct [] { identifier, status }
    ```
    
    Identifier used to configure JTAG hub

  - Read/write dynamic set of registers located on any bus type (method)
    
    **ReadWriteRegisters:**
    ```c
    struct [] { method, addr, data, mask, write }
    ```
    
    **UpdateRegisters:**
    ```c
    struct [] { method, addr, data }
    ```

  - Read/write table of any length on any bus type
    
    **ReadWriteTable:**
    ```c
    struct { method, addr1, addr2, addr_start, nvalues, write, data [] }
    ```
    
    **UpdateTable:**
    ```c
    struct { method, addr1, addr2, addr_start, nvalues, data [] }
    ```

    Write/read interface in hw:
    
    - `addr1`: Address of address register
    - `addr2`: Address of data register

  - Subscribe and receive data at regular intervals
    
    **SubscribeRegisters:**
    ```c
    struct [] { method, addr, interval }
    ```
    
    **UpdateSubscribedRegisters:**
    ```c
    struct [] { method, data }
    ```

- All are dynamic in length to pack control actions – network/callback economy
A device type is represented by a dynamic data base structure
  - Each board is an instantiation of the structure
A full representation and storage of all resources for control and monitoring
  - Version
    - Cross-check with board/firmware version
  - FPGA code
    - File pointers
  - Hardware view – registers
    - Readings and settings to verify control operations on hardware (writing always followed by reading)
  - Functional view – parameters
    - Readings and settings to display separately in GUIs
  - State
    - Global status information used by expert system
  - Actions
    - Dynamic structures associated with the server commands and services

```
BoardType {
  struct Version {} 
  struct FPGAcode {} 
  struct State {
    int RunState 
    bool WriteError 
    bool StatusError 
    bool Monitored 
    bool Owner } }
struct Registers {
  struct Readings {} 
  struct Settings {} }
struct Parameters {
  struct Readings {} 
  struct Settings {} }
struct Action {
  struct ReadWriteRegisters {} 
  struct UpdateRegisters {} 
  struct ReadWriteTable {} 
  struct UpdateTable {} 
  struct SubscribeRegisters {} 
  struct UpdateSubscribedRegisters {} 
  struct DownloadFPGA {} 
  struct FPGAloadStatus {} }`
```
Device Representation - HW

Hardware view

```
struct Registers {
    struct Readings {
        struct Q1 {
            int R000
            int R004
            ...
        }
        struct Q2 {
            int R000
            int R004
            ...
        }
        struct I2C_40 {
            int R00
            ...
        }
    }  
    struct Settings {
        struct Q1 {
            int R000
            int R004
            ...
        }
        struct Q2 {
            int R000
            int R004
            ...
        }
        struct I2C_40 {
            int R00
            ...
        }
    }
}
```

- Obviously requires some convention on the type of devices or rather the different access modes
  - E.g. Q1 means FPGA on Local Bus with base address 0x1000
  - E.g. I2C_40 means an I2C device with base address 0x40

- Created with Device Type Editor
Motivations for a device representation in terms of functional parameters

- Things depend on functional parameters, not physical registers
- Economic in terms of gate and access
- User interfaces – intuitive
- Saved configurations
- Settings of functions are applied together

Grouping of functional parameters in functional blocks to which they belong

- Applied together

Mapping between the functional view of a device and the hardware view

- Registers $\rightarrow$ Parameter decoding
- Parameters of function $\rightarrow$ Register encoding
Translation Registers ↔ Parameters

- Instead of hard-coded → Dynamic translation whenever there is a read/write operation using descriptor information
  - Register Descriptor
    - Same structure for any device type
    - One instantiation per device type
    - Information for Register → Parameter, Functional block → Registers, Data Subscription
    - Parameter info: {Addr, Method, Type, ParamName, FuncBlock, Width, Position, Check}

```
Device1 {
  struct Version {
  }
  struct Registers {
    struct Q1 {
    }
    struct Q2 {
    }
    struct Q3 {
    }
    struct Q4 {
      ...
      string R01C [ ] =
        {0x4034, 1, 1, P_IP_HDR_LEN, FrontEnd, 4, 0, 1}
        {0x4034, 1, 1, P_IP_VERSION, FrontEnd, 4, 4, 1}
        {0x4034, 1, 1, P_IP_SERVICE, FrontEnd, 8, 8, 1}
        {0x4034, 1, 1, P_IP_TTL, FrontEnd, 8, 16, 1}
        {0x4034, 1, 1, P_IP_PROTOCOL, FrontEnd, 8, 24, 1} }
      ...
    }
    struct FuncBlocks {
    }
    struct DataSubscribe {
    }
  }
}
```

- Descriptor Editor
- Generic Translation API Manager
- Device Type structure and Descriptor structure created and modified using a Descriptor Editor
Generic Translation Manager

- API manager in C++
  - Same for any device type
  - One or several per device type may be running depending on the number of devices
  - Creates “translation objects run time” for automatic
    - Register → Parameter translation
    - Functional Block → Register translation
    - Data Subscription of counters and status parameters
Summary: Control Flow 1

FPGA Programming

- Control System
  - GUI
    - Select FPGA
    - Display States
  - FPGA States
    - Read FPGA File
    - DownloadFPGA structure
  - FPGA LoadStatus structure
- Control network
- Command (DownloadFPGA)
- Service (FPGALoadStatus)
- Control Interface
  - SERVER
    - Select device → Program device with JTAG

ReadWriteRegisters

- Control System
  - GUI
    - Display Settings
    - Display Readings
    - Expert system
      - "Apply"
        - Parameter Settings
        - Translation
          - Regs → FuncBlock
          - Register Structure
          - UpdateRegister Structure
  - Translation Readings → Params
    - ReadWrite Structure
    - Register Readings
- Control network
- Command (ReadWrite Registers)
- Service (UpdateRegisters)
- Control Interface
  - SERVER
    - Read→Set→Write→Read to hardware
Summary: Control Flow 2

ReadWriteTable

Server

Command
(ReadWriteTable)

Control network

Control interface

SERVER
Write→Read to hardware

SubscribeRegisters

Server

Command
(SubscribeRegisters)

Control network

Control interface

SERVER
Append subscribed registers to list → Set timer → Update regularly

Expert system

"Apply"

"SubscribeRegister"

Parameter Settings

Display Settings

Display Readings

Update Table Structure

Parameter Readings

GUI

Control system

"Apply"

"Apply"

"Apply"

"Apply"

Counter & Status Readings

UpdateSubscribedRegisters Structure

Register Readings

GUI

Control system

Subscribe

Display Readings

Display

Subscribed Registers Structure

UpdateSubscribedRegisters Structure

Translation
Regs→Status &
Counters

R. Jacobsson, CERN
Conclusions

- General purpose strategy proposed to the middle-layers of a large control system
  - Generic control interface server
  - Dynamic communication protocol
  - Representation of resources and storage in control system
  - Automatic handling of parameters and registers
  - Interface to supervisor control system and automated expert system
- In use for the LHCb Timing and Fast Control system at CERN

- Special acknowledgment
  - Summer student Daniel Alonso Alvarez, University of Oviedo for working on the API Manager
  - Niko Neufeld, CERN, for providing the OS and the PCI library for the control interface
  - Clara Gaspar, CERN, for DIM and help with PVSS