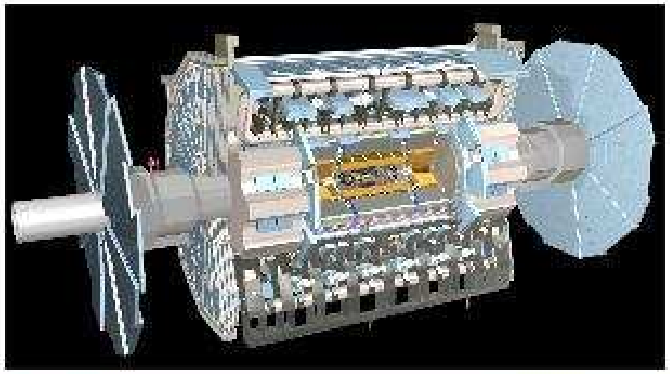
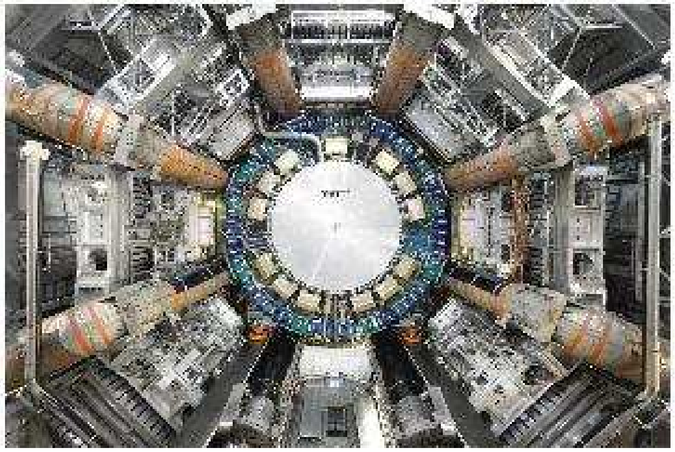




Using the RCE as an FEE Simulator

RCE Training Workshop

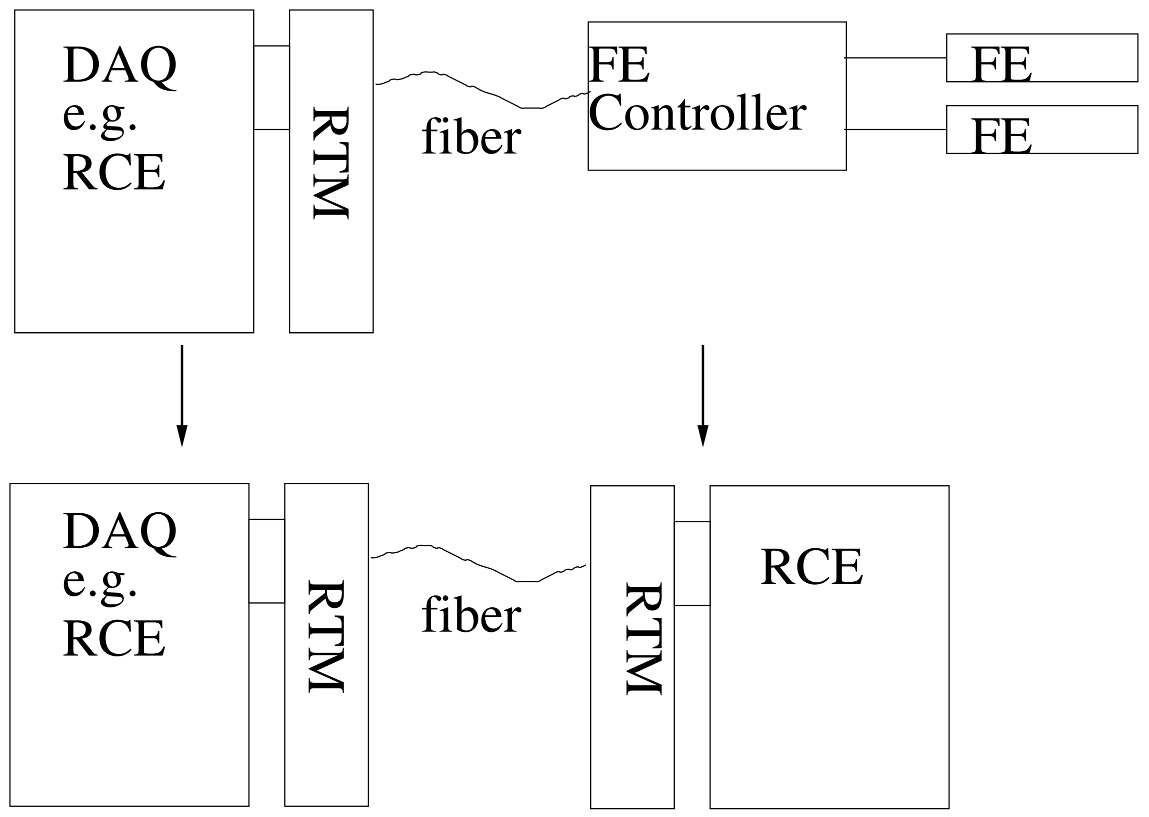
Martin Kocian, kocian@slac.stanford.edu

June 16, 2009

Introduction

- For DAQ development it is important to test the system under realistic conditions.
- FE ASICS are often available only late in the game.
- It can be hard to get realistic data without collisions.
- To do a real test of the DAQ frontend simulators are crucial.
- **The RCE can be used as a frontend simulator.**

Concept



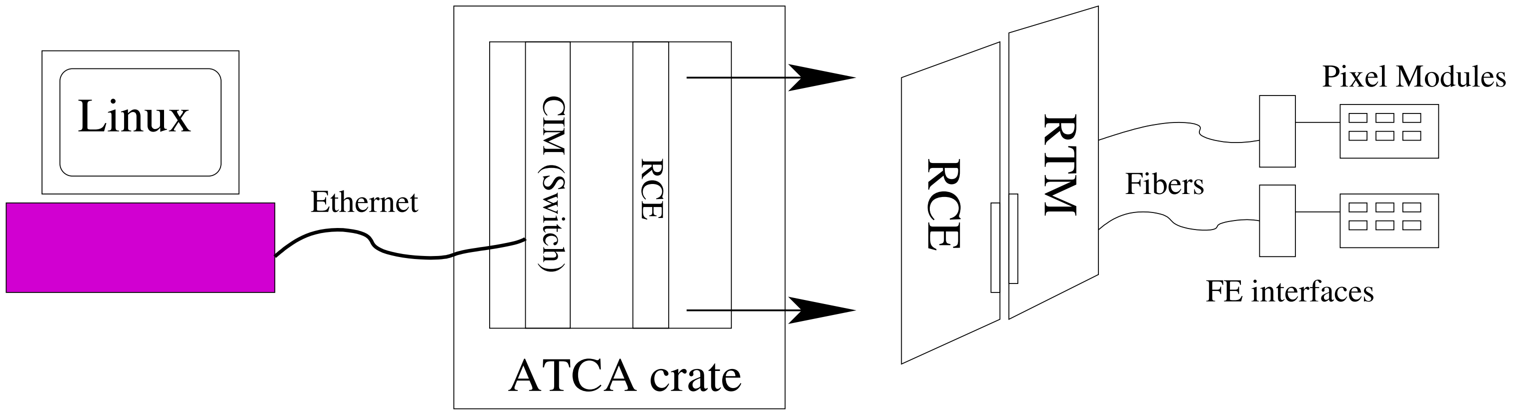
- The DAQ talks to the frontends through optical fibers.
- Idea: Replace FE by RCE.

Possibilities

- Simulate the ASIC behavior on the RCE in C++.
- Download data files to the RCE to be fed to the DAQ.
- Test frontend configuration.
- Test calibration software.
- Test data taking/trigger.

$\Longrightarrow$ Emulate frontend behavior in real time.

Pixel FE with PGP



- Use Pixel FE with PGP from the previous session as an example on how one could design a frontend simulator.
- Elements:
  - PGP RCDI interface.
  - HSIO (FPGA interface board) simulation.
  - Frontend simulation.
- A straightforward way to emulate the hardware's behavior would be to create classes that correspond to these elements.

RCDI Interface

- Raw PGP is available through pgp driver on the RCE that emulates the frontend.
- Write a handler class RCDIslave that implements the functionality of the RCDI slave.
- To the DAQ RCE the FE RCE looks like the frontend vhdl RCDI slave.
- RCDI functionality:
  - Register R/W.
  - Command.
  - Data.
- The following slides are suggestions for an implementation of the RCDI slave.

Register Read/Write Block

- RCDI slave has to respond to the master's register read/write requests.
- Implementation through callback functions:

```
unsigned setRegister(unsigned address, unsigned value);
unsigned getRegister(unsigned address, unsigned& value);
```

- The return value would contain a user error and timeout to be returned to the master RCE.
- The user (in this case the class that implements the HSIO functionality) would implement the functions and register them with the RCDIslave.

Command Block

- RCDI slave has execute command requests.
- Implementation through callback function:

```
void executeCommand(unsigned char opcode, unsigned context);
```

- The command block does not reply to the master.
- Again the user (the class that implements the HSIO functionality) would implement the function and register it with the RCDIslave.

Data Block

- RCDI slave can send data to the master.
- Implementation through a function of RCDImaster:

```
void sendData(unsigned char *buffer, unsigned bufferSize);
```

- The user (the class that implements the HSIO functionality) would call the function.

PGP Initialization

- To use PGP:
  - Define transmit buffer parameters.
  - Create a pool of transmit buffers.
  - Get the pgp driver for the desired lane from the system.
  - Register your handler with the driver.

```
const RcePic::Params Tx = {
    RcePic::NonContiguous,
    16, // Header
    64*132, // Payload
    128 // Number of buffers };
PgpTrans::RCDIslave *rcdi=new PgpTrans::RCDIslave; // RCDI interface
RcePic::Pool *pool = new RcePic::Pool::Pool(Tx);
rcdi->SetPool(pool); // tell the handler so it can put back the buffers
RcePgp::DriverList *driverList = RcePgp::DriverList::instance();
RcePgp::Driver *pgpd = driverList->handler(RcePgp::RTM_0, rcdi);
```

HSIO

- In hardware the HSIO is a board with a Xilinx FPGA.
- Function:
  - Receive data from RCE.
  - Buffer it in a FIFO.
  - Serialize it for the FE.
  - Receive serial data from the FE.
  - Send data to the RCE.
- Create an HSIO class with this functionality.

HSIO Class

- Implement `setRegister(...)` function of RCDIslave to insert data into the buffer.
- Implement `executeCommand(...)` function of RCDIslave to send buffer to FE class.
- HSIO class calls RCDIslave's `sendData(...)` as handshaking like in the hardware implementation.
- To send data FE class calls a function in HSIO which then uses RCDIslave's `sendData(...)` to send data back to the master RCE.

FE Classes

- The implementation of the frontend classes depends on the ASIC to be simulated.
- Create classes that correspond to FE elements.
- In the pixel module there is 1 controller and 16 chips.
- Create configuration registers and emulate their functionality as needed.
- For the digital test presented in the previous session one would for example have to implement the pixel mask registers with their functionality.
- Return data on L1A. The data could come from file or be simulated in the FE classes.
- For the digital test the relevant data is basically just a “1” if the pixel fired.

FE Parser

- RCDI and HSIO do not know what the configuration and event data means.
- It is the FE controller and the 16 FE chips on the pixel module that parse the incoming bitstream and translate it into register settings.
- Write a parser class to decode the incoming configuration data and set the registers in the frontend classes accordingly.

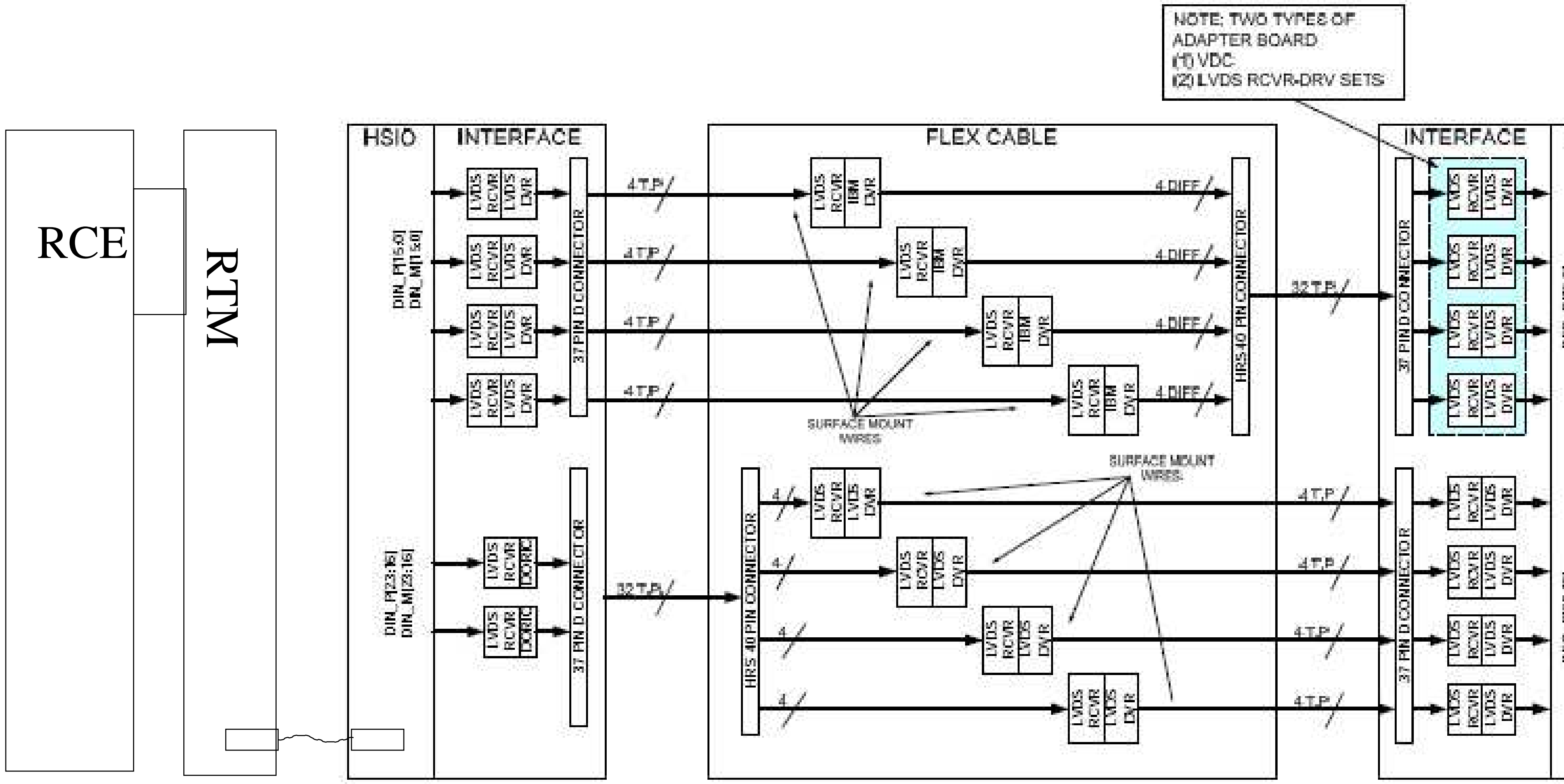
Data Encoder

- Collect data from the frontend classes.
- Encode the data into a bitstream.
- Transfer the bitstream buffer to the HSIO class.
- The data can be read into the slave RCE through ethernet or from nfs.

A Real Example

- At SLAC we want to do a realistic test of the data transmission for the insertable B layer.
- We are building an electrical prototype for one halfstave.
- 16 chips.
- Data readout at 160 Mbits/s.
- Configuration and clock at 40 Mbits/s.
- In order to measure transmission quality and cross-talk we have to run 16 readout chains in parallel.

IBL Transmission Test



Drawing: Dave Nelson

- Data transmission test for the insertable B layer.

Design

- 16 serializers and deserializers on the HSIO.
- HSIO connects to the stave through a custom interface board.
- Upload data from RCE to HSIO via PGP.
- Control serializers via PGP.
- Download transmission results to the RCE through PGP.

Summary

- The RCE can be used as a frontend simulator.
- Data can be sent to the DAQ at Gbit/s rates.
- C++ classes emulate hardware components.
- Data can be uploaded via ethernet.
- SLAC will use the RCE for frontend simulation for the IBL.