Calice packets

Contents

1 Introduction

What: Managing packets content.

This library is a copy of the base software made by David Decotigny still available in the CVS's *svn/calice/online-sw/trunk/daq/calice_packets/* directory. It is now copied at:

- *libLDA/include/packets/*
- *libLDA/packets/*

From *svn/calice/online-sw/trunk/daq/calice_packets/calice_packets.hpp* we can read:

```
* Base class. Defines the notions of SDU and PDU common to all the
* accessor objects.
*
* Typical topology of a SDU (aka. packet):
*
*  +-----------+--...-------...-----+--------+
*  | Header    |       PDU          | Footer |
*  +-----------+--...-------...-----+--------+
*
* The PktAccessor_Base class defines the access to the SDU and PDU
* and allows to process the packet before it is actually sent. The
* children classes would alow to access the various fields inside
* the Header and Footer, and manipulate the packet attributes such
* as the PDU size.
```

All the accessor objects inherit from the PktAccessor_Base ABC, which defines 2 notions: the SDU (Service Data Unit) and the PDU (Protocol Data Unit). The idea is that the whole packet is called the "SDU". It encapsulates a unique PDU, this PDU corresponding to the arbitrary data being transmitted by the packet. So packet=SDU, and data_carried_by_the_packet=PDU. The SDU is supposed to contain protocol data which have to be updated once the packet is ready to be sent, such as a CRC: the PktAccessor_Base interface declares the sync_crc() method to do just that.

2 API

2.1 PktAccessor_Base

```
 class PktAccessor_Base
  {
  public:
    virtual ~PktAccessor_Base() { }
    virtual void * get_sdu() const =0;
    virtual unsigned int get_sdu_nbytes() const =0;
    virtual void * get_pdu() const =0;
    virtual unsigned int get_pdu_nbytes() const =0;
    virtual void sync_crc() =0;

  protected:
    PktAccessor_Base() { }
  };
```

2.2 PktAccessor_LDA_pkt

```
 class PktAccessor_LDA_pkt
    : public PktAccessor_ODR_pkt
  {
  public:
    PktAccessor_LDA_pkt();
    PktAccessor_LDA_pkt(void const* sdu,
unsigned int sdu_nbytes,
bool sdu_is_garbage = false);
    PktAccessor_LDA_pkt(unsigned int min_pdu_nbytes);
    virtual void attach(void const* sdu,
unsigned int sdu_nbytes,
bool sdu_is_garbage = false); /* throw() */
    virtual void detach();
    virtual void * get_pdu() const;
    virtual unsigned int get_pdu_nbytes() const;
    virtual void set_pdu_nbytes(unsigned int min_pdu_nbytes);
    inline bool is_fast_command_request() const;

    /* Normal packets: */
    inline unsigned char get_LDA_subsystem() const;
    inline void set_LDA_subsystem(unsigned char);
    inline unsigned char get_LDA_optype() const;
    inline void set_LDA_optype(unsigned char);

    /* Same fields */
    inline unsigned short get_LDA_Modifier() const;
    inline void set_LDA_Modifier(unsigned short);
    inline unsigned short get_LDA_from_diflink_id() const; /* base 1 */
    inline void set_LDA_from_diflink_id(unsigned short /* base 1 */);
    inline unsigned short get_LDA_to_diflink_id() const; /* base 0 */
    inline void set_LDA_to_diflink_id(unsigned short /* base 0 */);
    inline unsigned short get_LDA_PktID() const;
    inline void set_LDA_PktID(unsigned short);
    inline unsigned short get_LDA_DataLength() const;
    inline void set_LDA_DataLength(unsigned short);

    /* Fast-command packets: */
    void init_LDA_FastCommand(unsigned short dif_mask,
```

```
    unsigned char comma,
    unsigned char data);
  inline bool unpack_LDA_FastCommand(unsigned short /*out*/& cmdw /*0xfa57*/,
     unsigned short /*out*/& dif_mask,
     unsigned char /*out*/& comma,
     unsigned char /*out*/& data) const;

private:
  struct LDA_pkt *   _pkt;
  unsigned int       _pdu_nbytes; // Related to ODR::_pdu_nbytes
  uint16_t _compute_FC_parity() const; // Host byte order
  PktAccessor_LDA_pkt(PktAccessor_LDA_pkt const&); // Not implemented
};
```

2.3 PktAccessor_DIF_pkt_block_transfer

```
class PktAccessor_DIF_pkt_block_transfer
  : public virtual PktAccessor_Base
{
public:
  PktAccessor_DIF_pkt_block_transfer(void const* sdu,
     unsigned int buff_nbytes,
     bool buff_is_garbage = false,
     bool check_crc_ignore_dcc_nibble=false);
  explicit PktAccessor_DIF_pkt_block_transfer(unsigned int pdu_nbytes);
  virtual void * get_sdu() const;
  virtual unsigned int get_sdu_nbytes() const;
  virtual void * get_pdu() const;
  virtual unsigned int get_pdu_nbytes() const;
  virtual void sync_crc();
  inline unsigned short get_packettype() const;
  inline void set_packettype(unsigned short);
  inline unsigned short get_dcc_nibble() const;
  inline void set_dcc_nibble(unsigned short dcc_nibble);
  inline unsigned short get_pktID() const;
  inline void set_pktID(unsigned short);
  inline unsigned short get_type_modifier() const;
  inline void set_type_modifier(unsigned short);
  inline unsigned short get_data_length() const; // Number of 16b-words
  virtual void set_data_length(unsigned short num_16b_words);

private:
  struct DIF_pkt_block_transfer * _pkt;
  const unsigned int _buff_nbytes; // Allocated mem size (FIXED)
  unsigned int       _sdu_nbytes;  // Related to _pdu_nbytes
  unsigned int       _pdu_nbytes;  // Related to _sdu_nbytes
};
```

3 Compilation

3.1 Flags

There is two pre-compilation flags:

- CHECK_DIF_CRC: make the libLDA checking the CRC.
  Note: This will slow the DAQ acquisition and may imply data lost.
- DIF_FIRMWARE: make the libLDA compatible with both ECAL and DHCAL.

/include/packets/all.hh:

```
/* Check DIF CRC or not (faster) */
#define CHECK_DIF_CRC 0

/* Firmware version we use */
#define DIF_FIRMWARE DHCAL_v20
```

3.2 Ubuntu

```
# aptitude install g++ flex bison libboost-dev
$ make
$ make tests
```

3.3 SLC 5.5

```
# yum install gcc-c++ boost-devel
  Installing     : libstdc++-devel
  Installing     : kernel-headers
  Installing     : glibc-headers
  Installing     : glibc-devel
  Installing     : gcc
  Installing     : gcc-c++
  Installing     : libicu
  Installing     : boost
  Installing     : boost-devel

$ cd ~/libLDA/packets
$ make
$ make tests
```

3.4 Gcc flag used

It seems to me that these flags are not needed, as I didn't mention diference with or without.

- `-fno-strict-aliasing`:
  Allow the compiler to assume the strictest aliasing rules applicable to the language being compiled. For C (and C++), this activates optimizations based on the type of expressions.

- `-pipe`:
  Use pipes rather than temporary files for communication between the various stages of compilation. This fails to work on some systems where the assembler is unable to read from a pipe; but the GNU assembler has no trouble.

- `-fomit-frame-pointer`:
  Don't keep the frame pointer in a register for functions that don't need one. This avoids the instructions to save, set up and restore frame pointers; it also makes an extra register available in many functions. It also makes debugging impossible on some machines.

- `-fno-delete-null-pointer-check`
  Use global dataflow analysis to identify and eliminate useless checks for null pointers. The compiler assumes that dereferencing a null pointer would have halted the program. If a pointer is checked after it has already been dereferenced, it cannot be null. In some environments, this assumption is not true, and programs can safely dereference null pointers. Use -fno-delete-null-pointer-checks to disable this optimization for programs which depend on that behavior.

4 Unitary tests

Here we can test the CRC computation of packet. This unit test is optimized for copy/past from the `tcpdump` output (where DIF bytes are inversed).

```
$ libLDA/src/packets/computeCrc
1234
crc= 0xd219
5678
crc= 0xe70b
^C
```