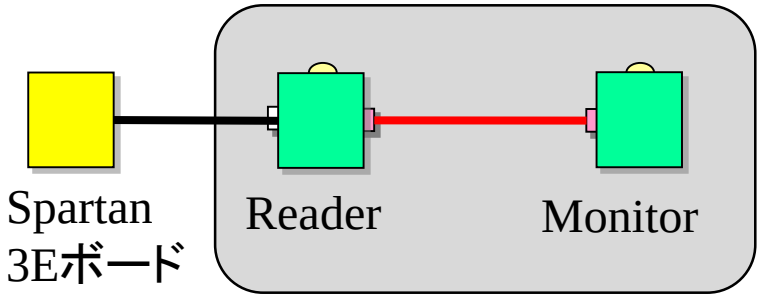
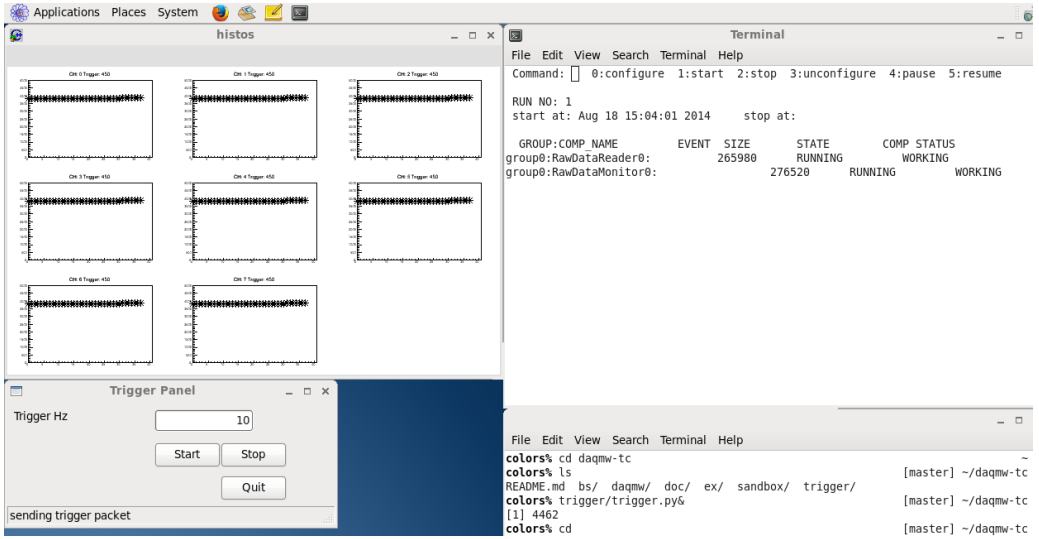
DAQ-Middleware トレーニングコース実習

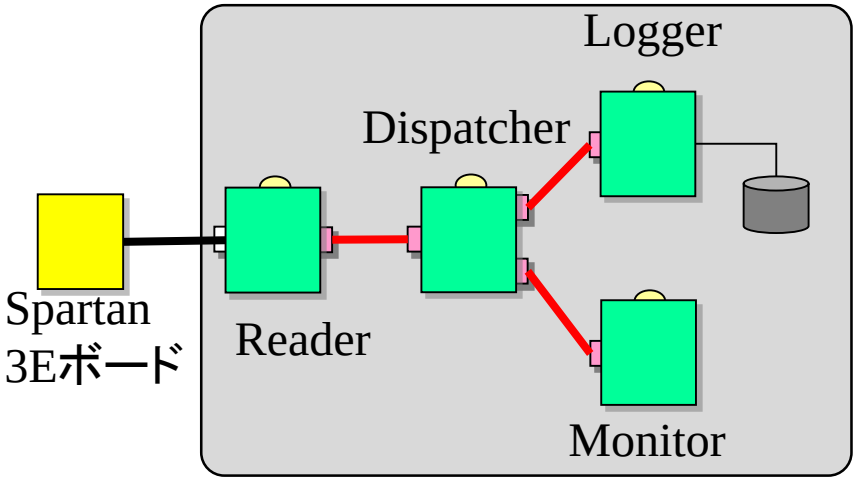
濱田英太郎
高エネルギー加速器研究機構
素粒子原子核研究所

実習最終目標

Spartan 3Eボードからデータを読んでグラフを画面に表示するシステムを作る







実習で行う事項

- セットアップ
  - Spartan3E評価ボードセットアップ
  - 実習用ファイルダウンロード
- 実習1 （DAQ-Middlewareを利用しない）
  - ex01 実習環境確認
  - ex02 C++の簡単な復習(クラス)
  - ex03 ネットワークバイトオーダー
  - ex04 char bufferからの数値の取り出し
  - ex05 バイナリファイルの読みだし
  - ex06 ファイルを読んでデコード
  - ex07 ROOTを使ってグラフを書く
  - ex08 ファイルを読みながらグラフを画面に表示する
  - ex09 ネットワークからデータを読みデコードする
  - ex10 ncコマンドでデータを読みグラフを画面に表示する

実習で行う事項

- 実習2　（DAQ-Middlewareを利用する）
  - ex11 DAQ-Middleware付属サンプルコンポーネントを動かしてみる
  - ex12 Webモードでシステムを動かす
  - ex13 ログの確認
  - ex14 ボードを読むシステム(DAQ-Middleware使用)を動かしてみる (Reader - Logger)
  - ex15 ボードを読んでモニターするシステムをDAQ-Middlewareで作る (Reader - Monitor)

  - ex16 追加課題：Mergerを利用して複数台のPCからデータを収集する

実習環境確認

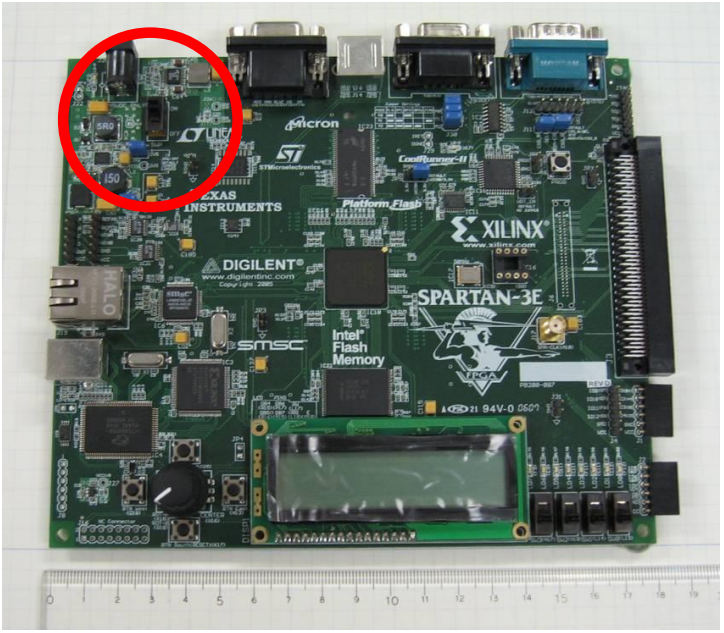
- VirtualBoxのセットアップ

以下のコマンドを実行して、インターネットに接続できることを確認してください。

```
% ping www.yahoo.co.jp
```

- Spartan 3Eの配布

ACアダプタ、LANケーブルをさすだけ。
電源スイッチはACアダプタコネクタそば



以下のコマンドを実行して、ボードに接続できることを確認してください。

```
% ping 192.168.10.16
```

実習ファイルダウンロード

- 実習ファイルダウンロード

（下記はwebページに記載されています。）

```
%  cd
%  git clone https://github.com/e-hamada/daqmw-tc.git
```

ホームディレクトリに「daqmw-tc」というディレクトリが追加されます。

実習ファイル　中身の説明

- ex

  実習で行う項目の解説、一部のコード

- sandbox

  このディレクトリにファイルをコピーする等して、実習してください

- doc

  Spartan 3Eが送ってくるデータのデータフォーマットを説明する資料がある

- trigger

  Spartan 3Eにトリガー信号を送るプログラム

- bs

  実習の解答例

- daqmw

  実習で使うDAQコンポーネント

ex01　コンパイル環境確認プログラム

ファイルをsandbox以下にコピーしてmakeを実行し、実行ファイルを作成。

（下記はREADMEやwebページに記載されています）

```
% cd ~/daqmw-tc/sandbox
% cp -r ../ex/ex01 .
% cd ex01
% make
```

下記を実行すると、hello, worldと画面に表示される。

```
% ./sample
```

ex02　C++の復習

クラスファイルを作りそれを利用するプログラムを作る
（下記はREADMEやwebページに記載されています）

```
% cd ~/daqmw-tc/sandbox
% cp -r ../ex/ex02 .
% cd ex02
```

ファイル

- MyClass.h (クラス宣言)
- MyClass.cpp (実装)
- main.cpp (MyClassを使うプログラム)

ex02　C++の復習

ファイルの説明

MyClass.h （一部）

```
class MyClass
{
public:
        MyClass();
        MyClass(int x, int y);
        virtual ~MyClass();
        int set_x(int x);
        int set_y(int y);
        int get_x();
        int get_y();
private:
        int m_x;
        int m_y;


};
```

コンストラクタ

デストラクタ

m_x, m_yをsetする関数

m_x, m_yを返す関数

メンバ変数

ex02　C++の復習

ファイルの説明

MyClass.h （一部）

```
class MyClass
{
public:
        MyClass();
        MyClass(int x, int y);
        virtual ~MyClass();
        int set_x(int x);
        int set_y(int y);
        int get_x();
        int get_y();
private:
        int m_x;
        int m_y;

};
```

コンストラクタ

デストラクタ

m_x, m_yをsetする関数

メンバ変数

MyClass.cpp(一部)

```
MyClass::MyClass(int x, int y): m_x(x), m_y(y)
{
    std::cerr << "MyClass ctor(int, int)" << std::endl;
}
```

MyClass.cpp(一部)

```
int MyClass::set_y(int y)
{
    m_y = y;
    return 0;
}
```

ex02　C++の復習

- ファイルの説明

main.cpp（一部）

```
MyClass a;
MyClass b(1, 2);

int x = b.get_x();
int y = b.get_y();
cerr << "b.m_x: " << x << endl;
cerr << "b.m_y: " << y << endl;

a.set_x(10);
a.set_y(20);
x = a.get_x();
y = a.get_y();
cerr << "a.m_x: " << x << endl;
cerr << "a.m_y: " << y << endl;
```

クラスMyClassをオブジェクト化

bのm_xとm_yを表示

aのm_xとm_yをset

aのm_xとm_yを表示

ex02　C++の復習

コードを見て結果を予想したあと、以下のコマンドで実行

```
% make
% ./main
```

以下のようにコードの変更して下さい。

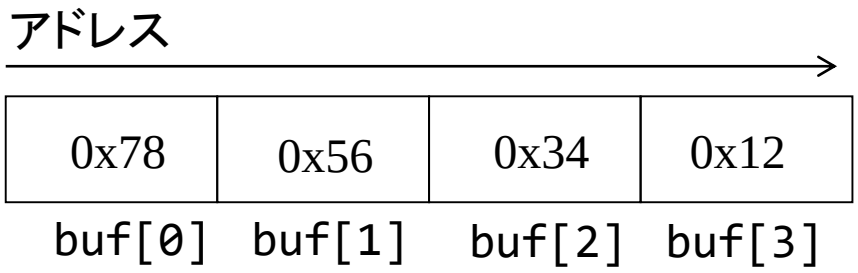
- MyClass.h、MyClass.cppにメンバー変数m_zを追加し、set_z()メソッド、get_z()メソッドを追加する。

- main.cppを変更し、set_z()、get_z()を使って値をセット、ゲットするプログラムを書く。

（解答は ~/daqmw-tc/bs/ex02_md/)

ex03 ネットワークバイトオーダー

0x 78 56 34 12 の順に送られてきたデータを



intとしての解釈

little endian　0x 12345678 = 305419896　（順序が逆）

bit endian　0x 78563412 = 2018915346　（そのままの順）

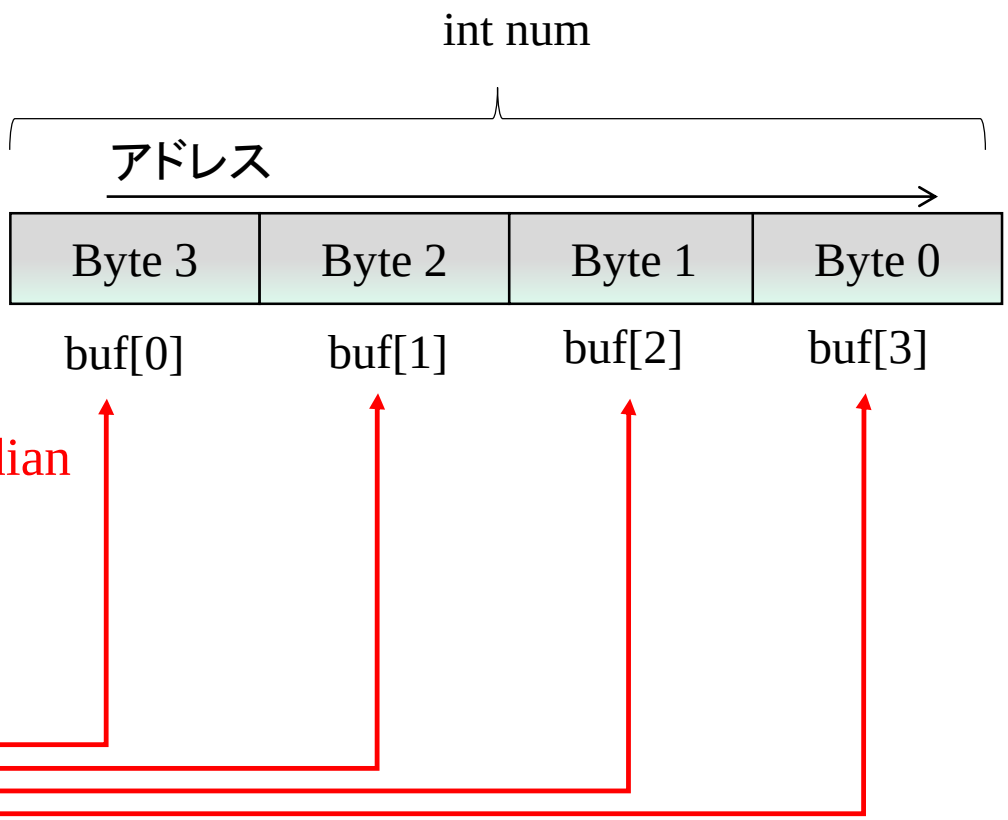
ネットワークバイトオーダーはbig endian

ex03 ネットワークバイトオーダー

union(共用体)は様々な型のデータを共通のメモリー領域で管理

byte_order.cpp (一部)

```
union my_num {
    int num;
    unsigned char buf[4];
};
```



byte_order.cpp (一部)

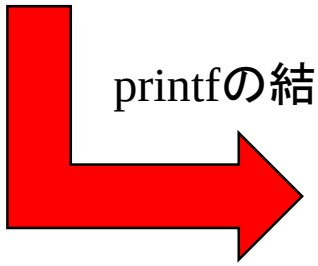
```
x.num = 0x12345678;
```

byte_order.cppではbuf[] のアドレスと格納されている値を表示する

ex03 ネットワークバイトオーダー

```
my_num x, y;
x.num = 0x12345678;

for (unsigned int i = 0; i < sizeof(x.num); i++) {
    printf("x: %p %d 0x%x¥n", &x.buf[i], i, x.buf[i]);
}
```



printfの結果例

```
% ./byte_order

x: 0x7fff78597440 0 0x78
x: 0x7fff78597441 1 0x56
x: 0x7fff78597442 2 0x34
x: 0x7fff78597443 3 0x12
```

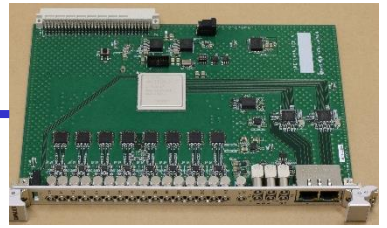
※アドレス値は環境によって異なるが、必ず+1されていく

htonl()関数を使うとどうなりますか？

（ex02と同様、プログラムをexからsandboxにコピーして、プログラムを起動してみて下さい）

ネットワークバイトオーダー

インテルCPU搭載



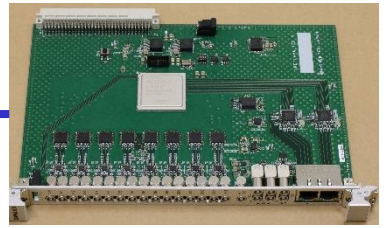
ホストオーダー:
リトルエンディアン

ビックエンディアンで送受信

- データ送信時にhtonl関数、htons関数を使って、
  リトルエンディアンからビックエンディアンに変換
- データ送信時にntohl関数、ntohs関数を使って、
  ビックエンディアンからリトルエンディアンに変換

ネットワークバイトオーダー

モトローラCPU搭載



ホストオーダー：
ビックエンディアン

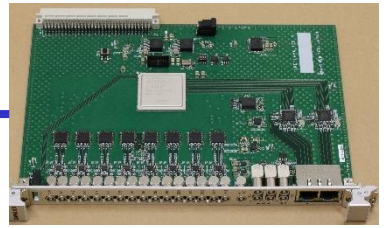
ビックエンディアンで送受信

- データ送信時にhtonl関数、htons関数を使って、
  ビックエンディアンからビックエンディアンに変換（つまり、変わらない）
- データ送信時にntohl関数、ntohs関数を使って、
  ビックエンディアンからビックエンディアンに変換（つまり、変わらない）

関数を使えば、ホストオーダーがどちらでも対応できる

ネットワークバイトオーダー

インテルCPU搭載



ホストオーダー:
リトルエンディアン

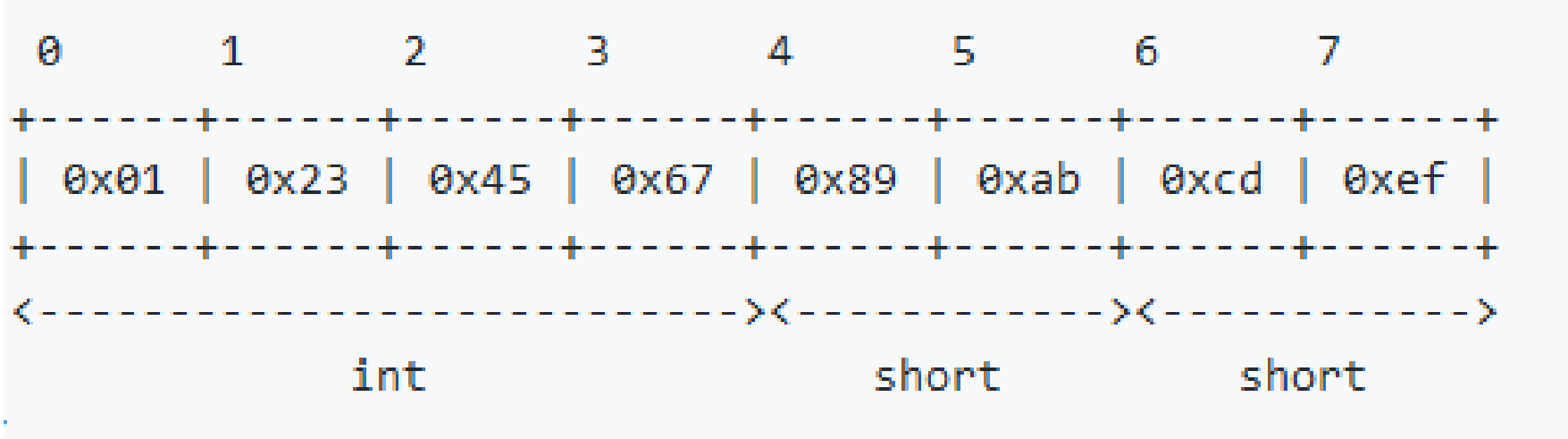
リトルエンデンィアンで送受信

注意！！
リトルエンディアンで送信することもある。
この時は、htonl関数などを使わない等の対応が必要。

仕様書や作成者に聞いて、
エンディアンを確認することが重要

ex04 char bufferからの数値の取り出し

デコードのときに必要になるのでchar buf[1024]のようなバッファからの 数値の取り出し方法を習得する。

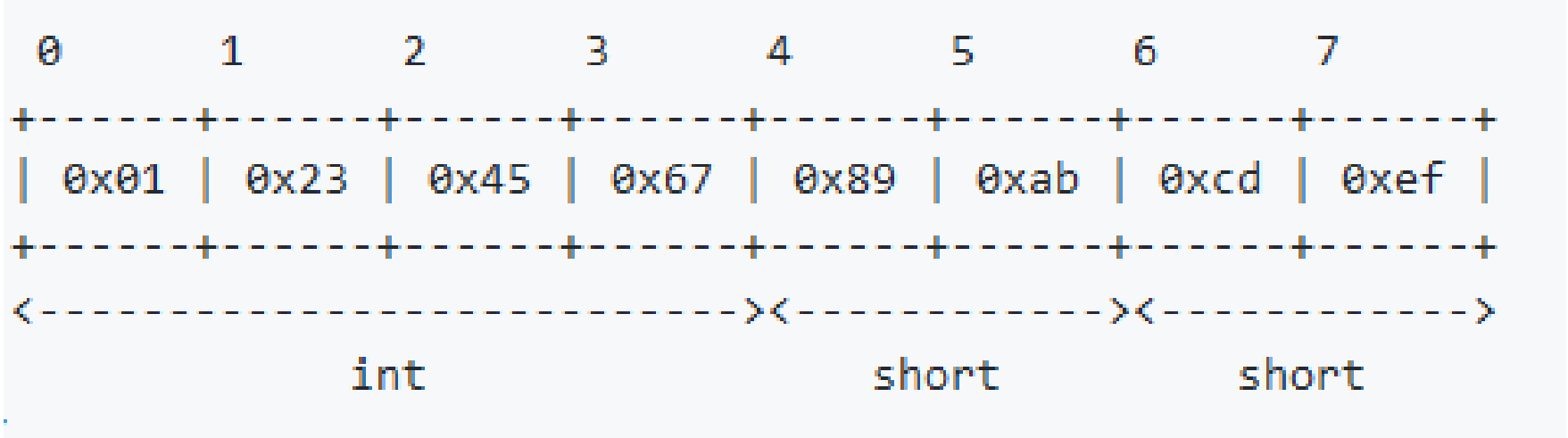



```
int_p = (unsigned int *)&buf[0];
                    // buf[0]のアドレスをセット。型が違うのでキャスト(unsigned int *)が必要

x = *int_p;         // *を付けると値が取り出せるのでxに代入している
                    // ネットワークバイトオーダからホストオーダに変更する必要があるなら
                    // x = ntohl(x); とする。
```

ex04 char bufferからの数値の取り出し

デコードのときに必要になるのでchar buf[1024]のようなバッファからの 数値の取り出し方法を習得する。



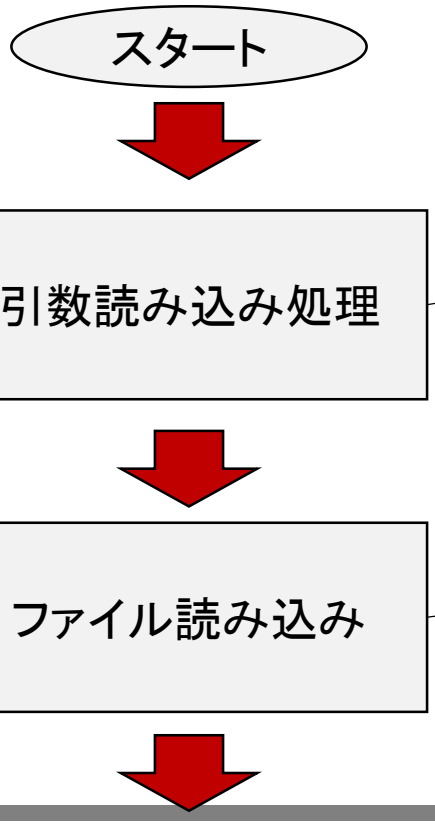
buf[6] buf[7]のshortの取り出しができるよう、
コードを修正してください。
extract_from_buf.cppを参考にしてください。

ex05 バイナリファイルの読みだし

バイナリファイルを読むプログラムを書く。

バイナリファイル→ ~/daqmw-tc/bs/sample.dat

参考用ファイルが入ったディレクトリ→~/daqmw-tc/bs/fread/



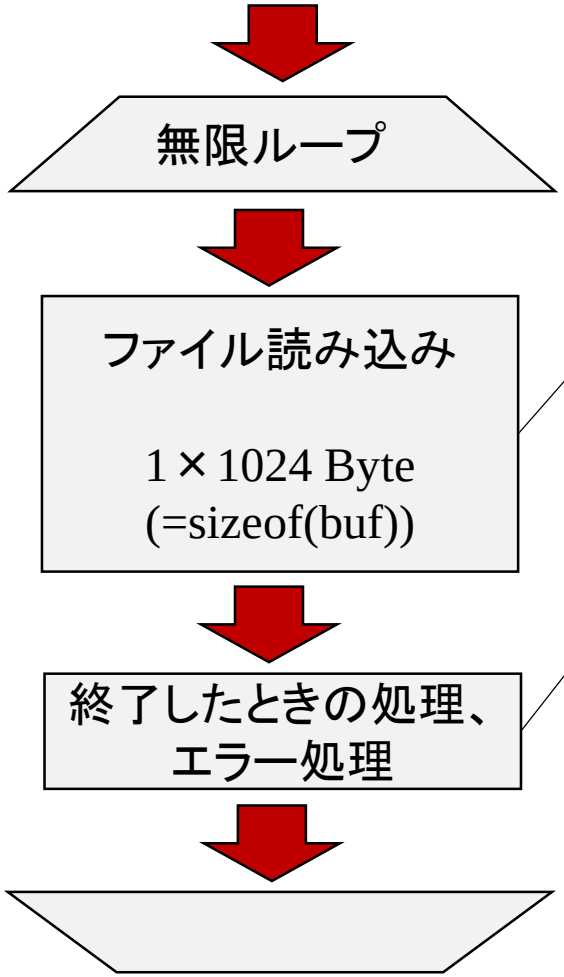
fread_sample.cpp （一部）

```
char buf[1024];

if (argc != 2) {
        usage();
        exit(EXIT_FAILURE);
    }

    FILE *fp = fopen(argv[1], "rb");
    if (fp == NULL) {
        err(EXIT_FAILURE, "fopen error");
    }
```

ex05 バイナリファイルの読みだし



fread_sample.cpp （一部）

```
for ( ; ; ) {
    n = fread(buf, 1 /*byte*/, sizeof(buf), fp);
    if (n == 0) {
        if (feof(fp)) {
            break;
        }
        else if (ferror(fp)) {   エラー処理（省略） }
    }

    if (n != sizeof(buf)) {   エラー処理（省略）}

    /* do something */
}
```

sample.datを読み取った場合、if (n != sizeof(but))の処理が実行し、無限ループが終了します。

/*do something*/以下にコードを追加して、sample.datの中身を表示するプログラムを作成して下さい。

ex06 バイナリファイルの読みだし

実習に使うボードからとったデータをデコードするルーチンを書く。

できたデコードプログラムは最終的にDAQ-Middlewareコンポーネントに 組み込むことになる。

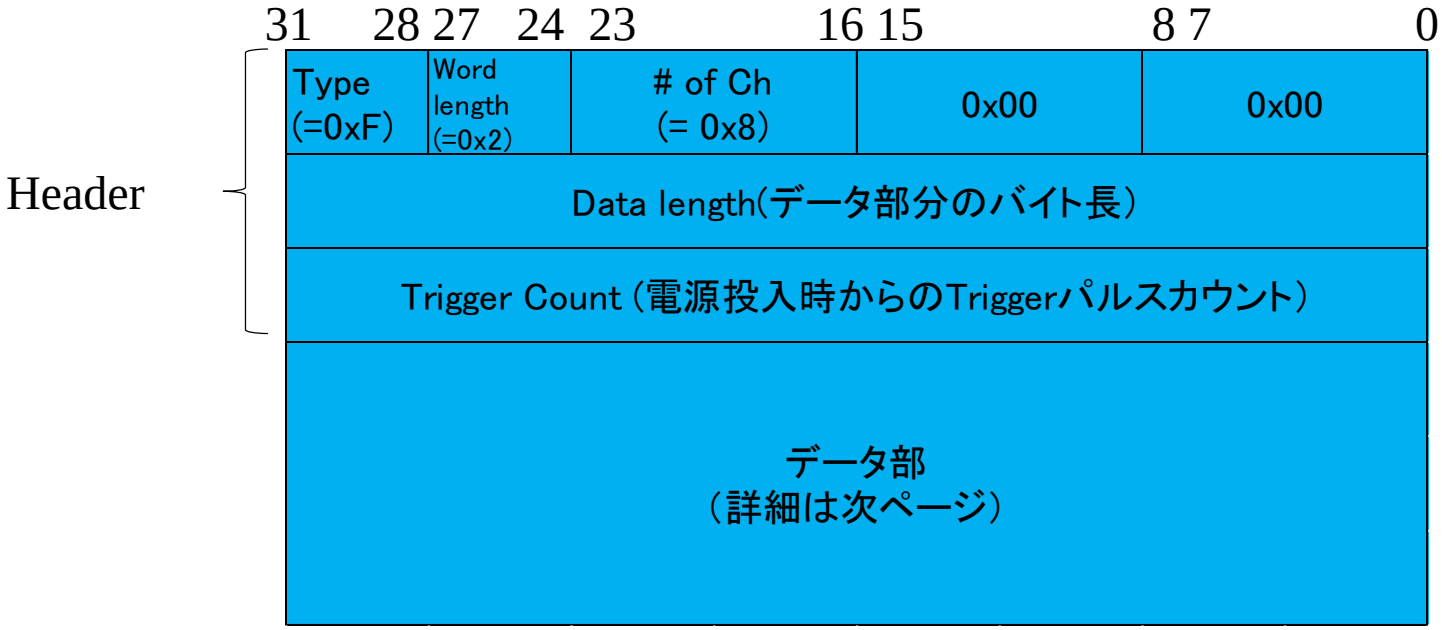
データフォーマット

~/daqmw-tc/doc/raw-data-packet-format.pdf

linux上では下記コマンドで、見ることができる

```
% evince ~/daqmw-tc/doc/raw-data-packet-format.pdf
```

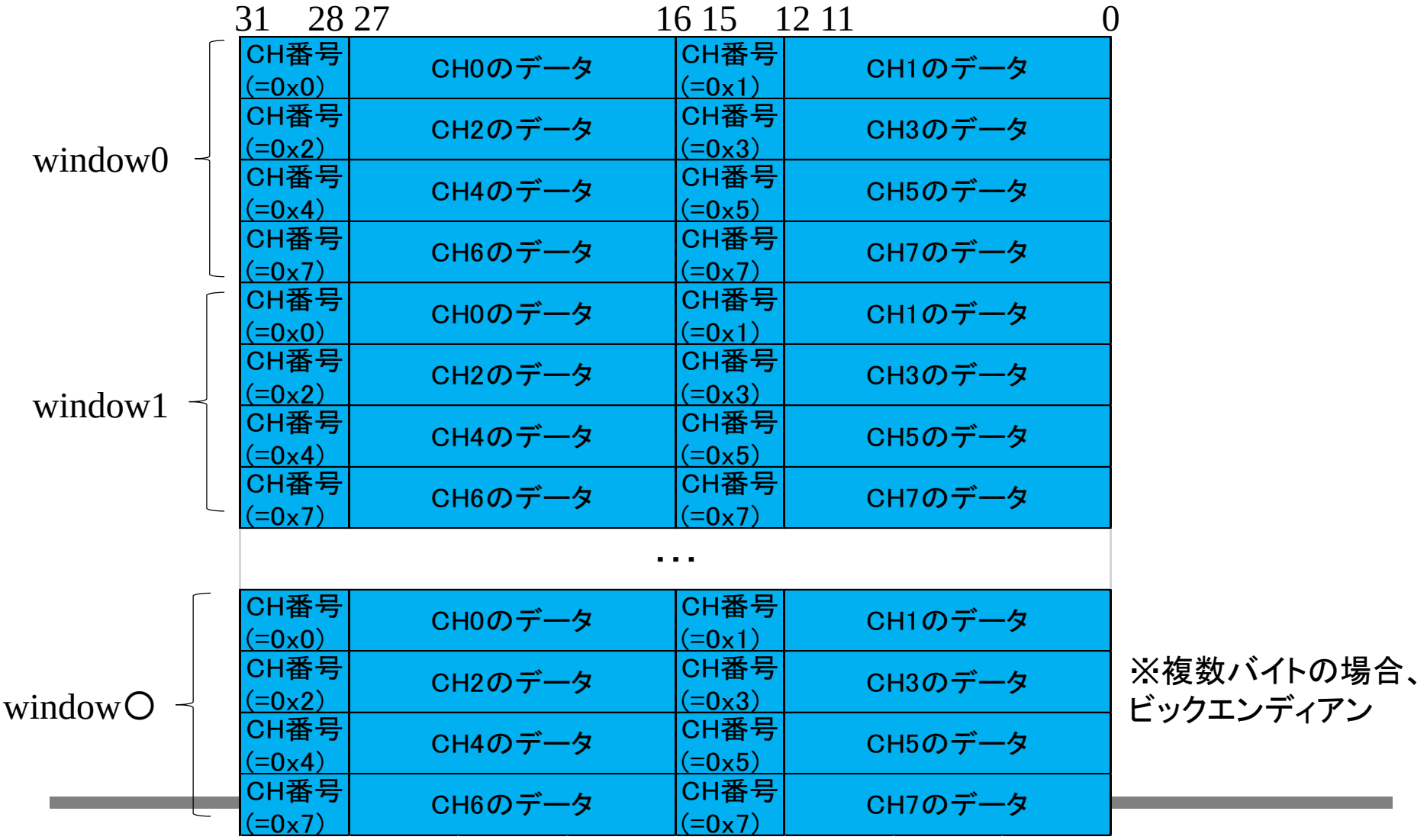
データ転送パケットフォーマット（全体）



※複数バイトの場合、ビックエンディアン

データ転送パケットフォーマット（データ部）

window数の分だけ、データが送られてくる



※複数バイトの場合、ビックエンディアン

データ量について

- 1windowあたりのデータ量
  = 2Byte (=1ch分のデータ)
  × ch数

- Data length（データ部分のバイト長）
  = 1windowあたりのデータ量
  × window数
  = 2Byte (=1ch分のデータ)
  × ch数
  × window数

Type (=0xF)	Word length (=0x2)	# of Ch (= 0x8)	0x00	0x00
Data length(データ部分のバイト長)				
Trigger Count（電源投入時からのTriggerパルスカウント）				
データ部				

CH番号 (=0x0)	CH0のデータ	CH番号 (=0x1)	CH1のデータ
CH番号 (=0x2)	CH2のデータ	CH番号 (=0x3)	CH3のデータ
CH番号 (=0x4)	CH4のデータ	CH番号 (=0x5)	CH5のデータ
CH番号 (=0x7)	CH6のデータ	CH番号 (=0x7)	CH7のデータ
CH番号 (=0x0)	CH0のデータ	CH番号 (=0x1)	CH1のデータ
CH番号 (=0x2)	CH2のデータ	CH番号 (=0x3)	CH3のデータ
CH番号 (=0x4)	CH4のデータ	CH番号 (=0x5)	CH5のデータ
CH番号 (=0x7)	CH6のデータ	CH番号 (=0x7)	CH7のデータ

…

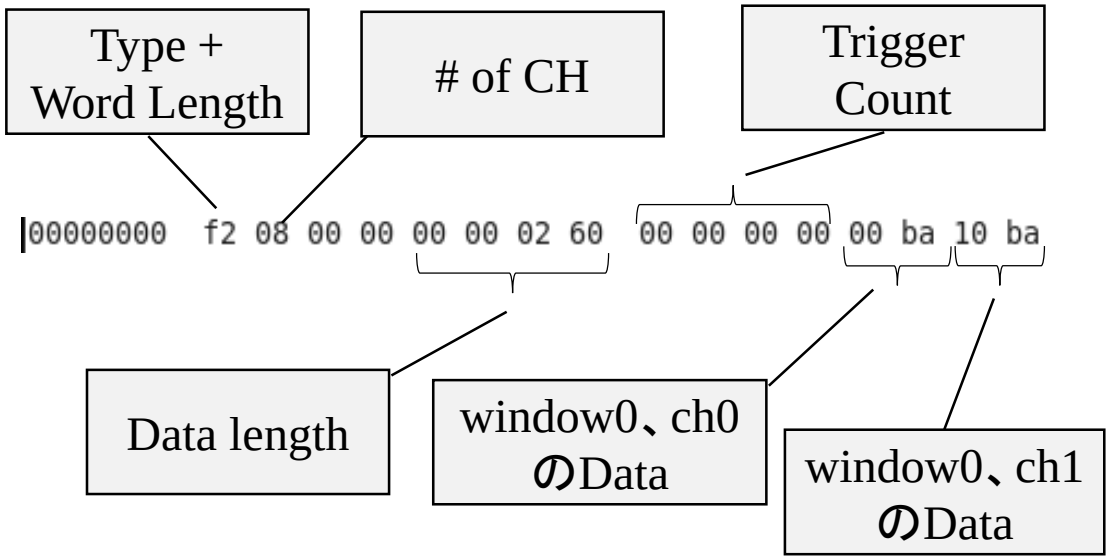
CH番号 (=0x0)	CH0のデータ	CH番号 (=0x1)	CH1のデータ
CH番号 (=0x2)	CH2のデータ	CH番号 (=0x3)	CH3のデータ
CH番号 (=0x4)	CH4のデータ	CH番号 (=0x5)	CH5のデータ
CH番号 (=0x7)	CH6のデータ	CH番号 (=0x7)	CH7のデータ

sample.datの確認

サンプルデータ(sample.dat)の確認

```
% hexdump -Cv ~/daqmw-tc/bs/sample.dat | less
```

サンプルデータの初めの数Byte



Type (=0xF)	Word length (=0x2)	# of Ch (= 0x8)	0x00	0x00
Data length(データ部分のバイト長)				
Trigger Count（電源投入時からのTriggerパルスカウント）				
データ部				

CH番号 (=0x0)	CH0のデータ	CH番号 (=0x1)	CH1のデータ
CH番号 (=0x2)	CH2のデータ	CH番号 (=0x3)	CH3のデータ
CH番号 (=0x4)	CH4のデータ	CH番号 (=0x5)	CH5のデータ
CH番号 (=0x7)	CH6のデータ	CH番号 (=0x7)	CH7のデータ
CH番号 (=0x0)	CH0のデータ	CH番号 (=0x1)	CH1のデータ
CH番号 (=0x2)	CH2のデータ	CH番号 (=0x3)	CH3のデータ
CH番号 (=0x4)	CH4のデータ	CH番号 (=0x5)	CH5のデータ
CH番号 (=0x7)	CH6のデータ	CH番号 (=0x7)	CH7のデータ
...			
CH番号 (=0x0)	CH0のデータ	CH番号 (=0x1)	CH1のデータ
CH番号 (=0x2)	CH2のデータ	CH番号 (=0x3)	CH3のデータ
CH番号 (=0x4)	CH4のデータ	CH番号 (=0x5)	CH5のデータ
CH番号 (=0x7)	CH6のデータ	CH番号 (=0x7)	CH7のデータ

ex06 バイナリファイルの読みだし

プログラムは ~/daqmw-tc/ex/ex06/ にあるのでこれをコピーして 使う

```
% cd ~/daqmw-tc/sandbox
% cp -r ../ex/ex06 .
```

ex06の中

- Makefile
- RawDataPacket.h
  デコードルーチンクラス ヘッダファイル
- RawDataPacket.cpp
  デコードルーチンクラス実装(各メソッドが書いてないので埋める)
- read_file_decode.cpp
  fread()を使ってファイルを読む
  (このなかでRawDataPacketで実装したメソッドを使っている。
  main()はこのなかにある)。

ex06 バイナリファイルの読みだし

read_file_decode.cpp 説明

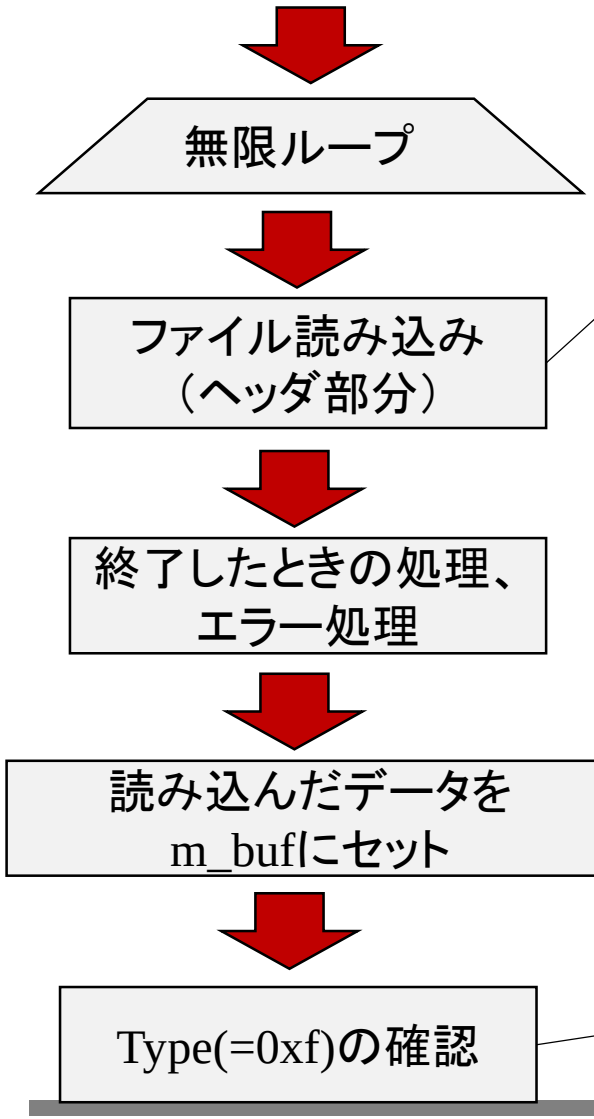
read_file_decode.cpp （一部）

```
RawDataPacket r;

if (argc != 2) {
        usage();
        exit(1);
    }

    filename = argv[1];
    fp = fopen(filename, "rb");
    if (fp == NULL) {
        err(1, "fopen for %s", filename);
    }
```

ex06 バイナリファイルの読みだし



read_file_decode.cpp （一部）

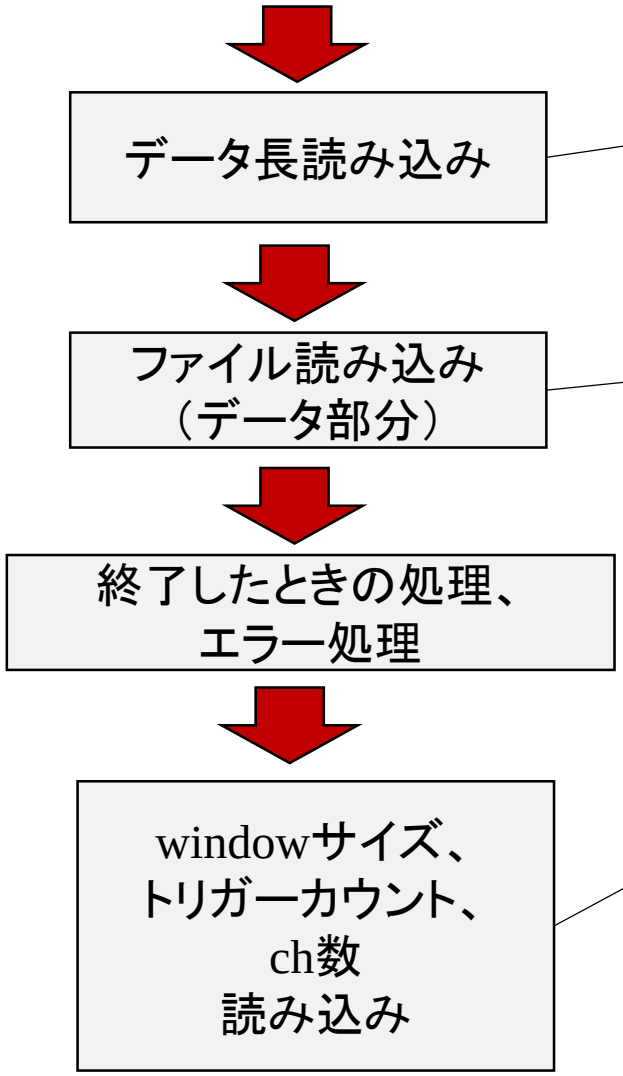
ヘッダサイズ
(RawDataPacket.hで定義）

```
for ( ; ; ) {
    // Read Header Part
    n = fread(buf, 1, RawDataPacket::HEADER_SIZE, fp);
    if (n == 0) {
        if (feof(fp)) {
            break;
        }
        else if (ferror(fp)) {   エラー処理（省略） }
    }

    if (n != sizeof(buf)) {   エラー処理（省略）}

    // Set header part to decode
    r.set_buf(buf, n);
    // Decode.  Verify Type
    if (! r.is_raw_data_packet() ) {
        cout << "Not a RawDataPacket" << endl;
        exit(1);
    }
```

ex06 バイナリファイルの読みだし

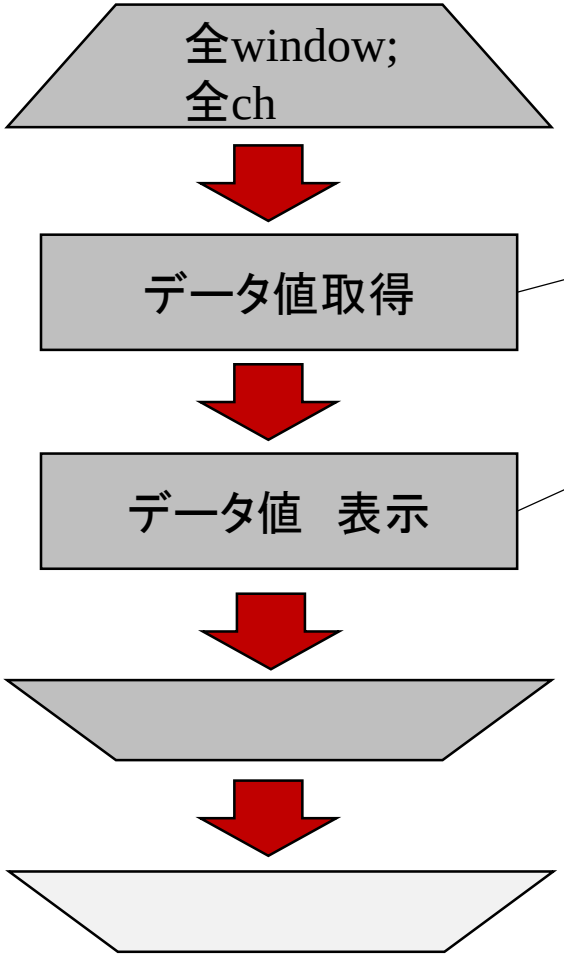



```
// Get Data length
      int data_length   = r.get_data_length();
      //cout << "data_length:   " << data_length   << endl;

      // Read Data Part
      n = fread(&buf[RawDataPacket::HEADER_SIZE], 1,
data_length, fp);
      if (n == 0) {  エラー処理（省略）}
      }
      else if (n != data_length) {   エラー処理（省略）}

      // Get window size, trigger count, number of channels
      int window_size   = r.get_window_size();
      int trigger_count = r.get_trigger_count();
      int n_ch          = r.get_num_of_ch();
```

ex06 バイナリファイルの読みだし




```
// Decode data
      for (int w = 0; w < window_size; w++) {
         for (int ch = 0; ch < n_ch; ch ++) {
            unsigned short data = r.get_data_at(ch, w);
            cout << "trg: "     << trigger_count;
            cout << " ch: "      << ch;
            cout << " window: " << w;
            cout << hex << " data: "   << data;
            cout << endl;
         }
      }
      r.reset_buf();
   }
```

ex06 バイナリファイルの読みだし

is_raw_data_packet()やget_word_size()等のメソッドを実装して、ファイルを。デコードできるようにしてください。

デコードして表示させたデータと下記のデータ（正解用データ）を比較してみてください。

~/daqmw-tc/bs/ascii.sample

is_raw_data_packet解答例

```
unsigned char format = m_buf[FORMAT_POS];
  format = (format & 0xf0);
  if (format == 0xf0) {
    return true;
  }
  else {
    return false;
  }
```

※FORMAT_POSは0
(RawDataPacket.hで定義）

実習2

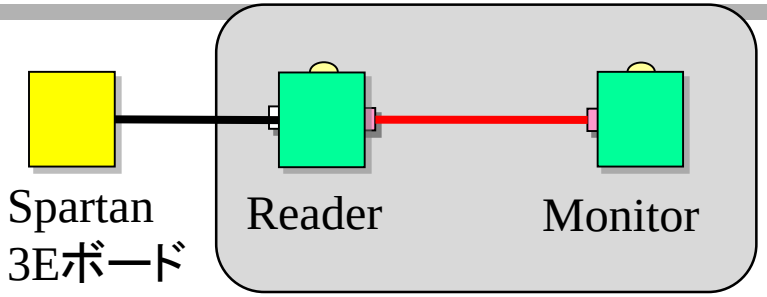
- 実習2　（DAQ-Middlewareを利用する）
  - ex11 DAQ-Middleware付属サンプルコンポーネントを動かしてみる
  - ex12 Webモードでシステムを動かす
  - ex13 ログの確認
  - ex14 ボードを読むシステム(DAQ-Middleware使用)を動かしてみる (Reader - Logger)

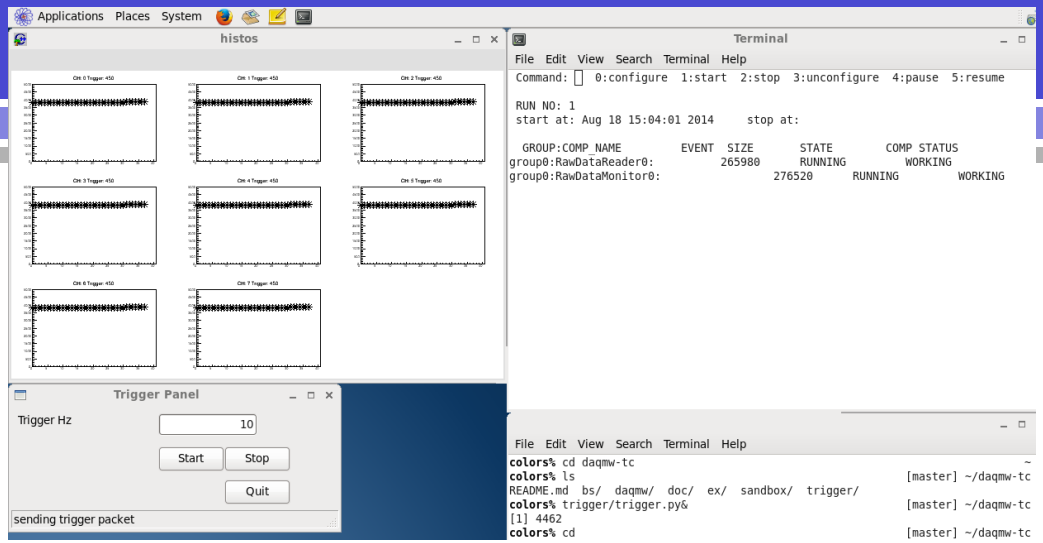
用意されたコンポーネントを動かす

  - ex15 ボードを読んでモニターするシステムをDAQ-Middlewareで作る (Reader - Monitor)

Reader、Monitorの理解が必要
Monitorの中身を変更して、目的のシステムを作る

ex15





- ex14で使ったReaderを利用。
  Readerは1イベントごと、データをMonitorに送っている。
  →read_data_from_detectors関数に処理内容が書かれている。

- Monitorはサンプルモニターを利用して自分で作る。
  DAQ-Middleware特有の関数があるので、理解が難しい箇所があります。
  →濱田に質問していただくか、マニュアルを参照してください。

コンポーネント間データフォーマット関連メソッド

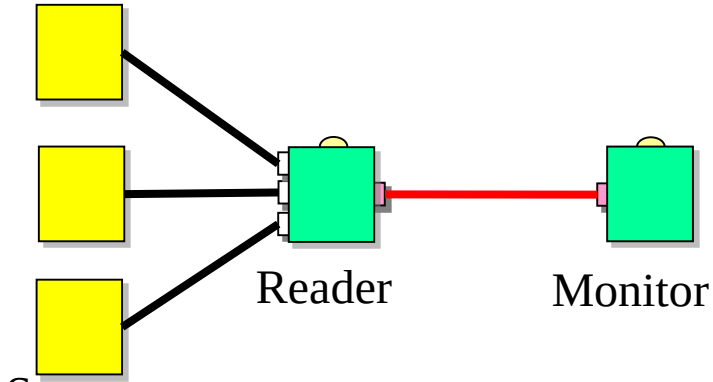
- inc_sequence_num()
- reset_sequence_num()
- get_sequence_num()

- set_header(unsigned char *header, unsigned int data_byte_size)
- set_footer(unsinged char *footer)

- check_header(unsigned char *header, unsigned received_byte)
- check_footer(unsigned char *footer)
- check_header_footer(const RTC::TimedOctetSeq& in_data, unsigned int block_byte_size)

Header Magic	Header Magic	Reserved	Reserved	Data Byte Size	Data Byte Size	Data Byte Size	Data Byte Size
Footer Magic	Footer Magic	Reserved	Reserved	Seq. Num	Seq. Num	Seq. Num	Seq. Num

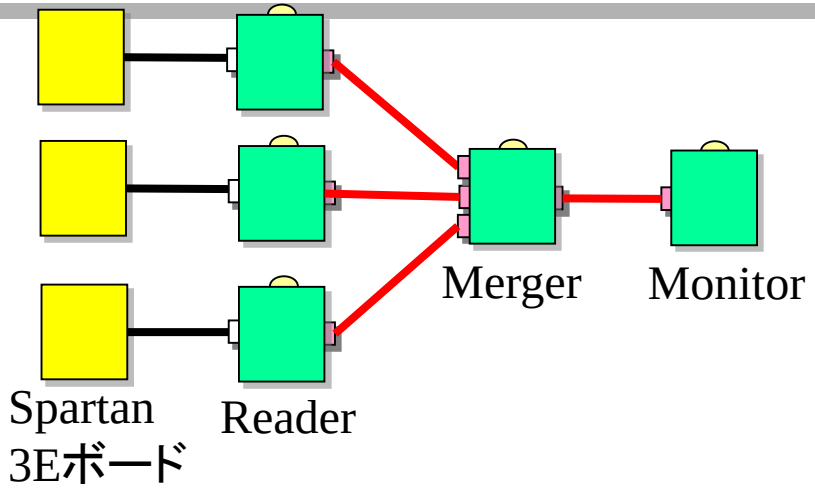
DAQ-Middleware　多重読みだしの例



例1　Readerでepoll等を利用して多重読み込みを行う

（メリット）
- コンポーネントが少ないので使用するリソースが少なくても済む

（デメリット）
- Readerの作成が難しい
- プロセスを分けないと、1CPUにReaderの分の負荷が大きくなってしまう

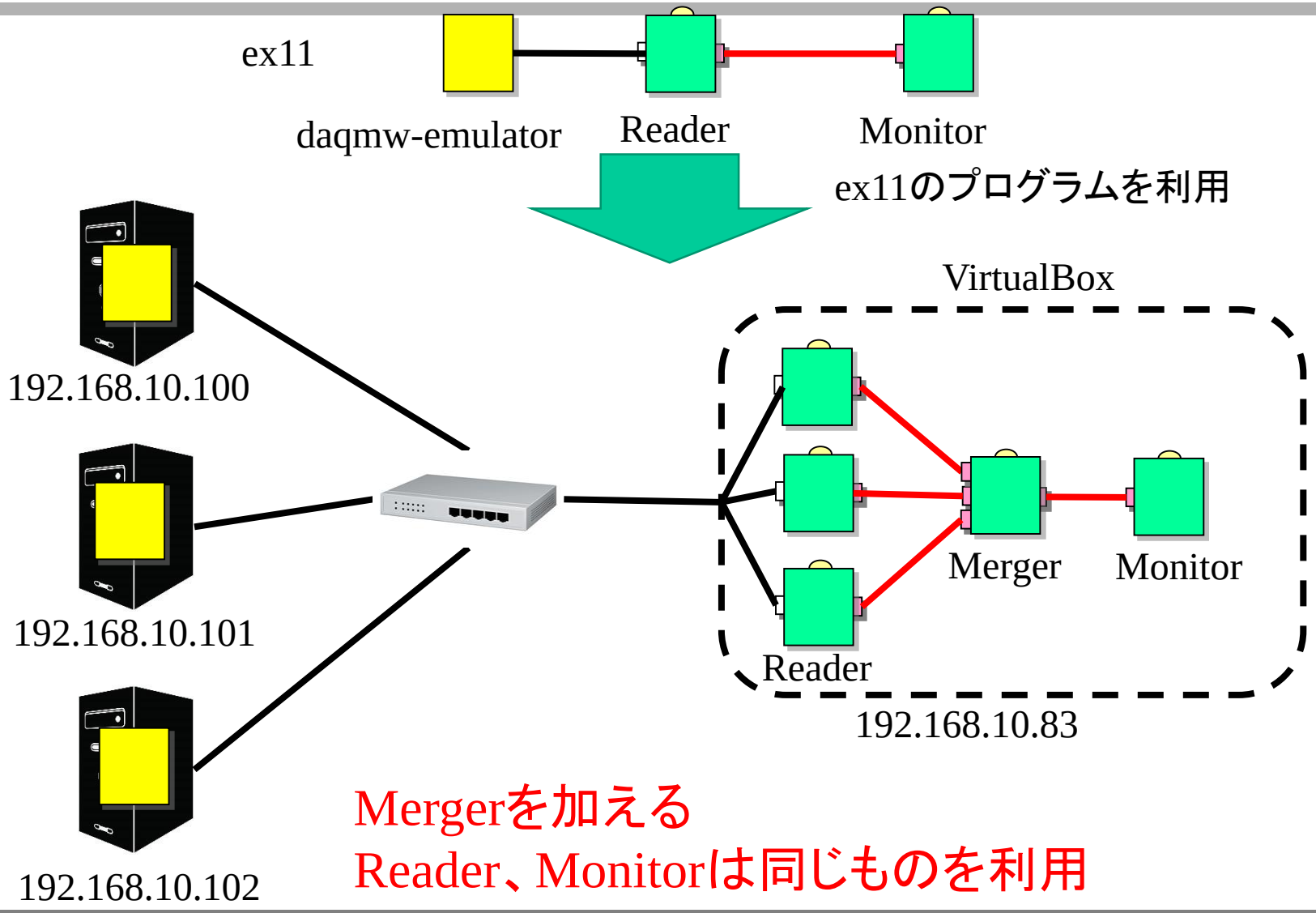


例2　複数のReaderとMergerを利用する

（メリット）
- Readerは全て1台の読み出しなので簡単に作れる。
- Readerの負荷を分散できる

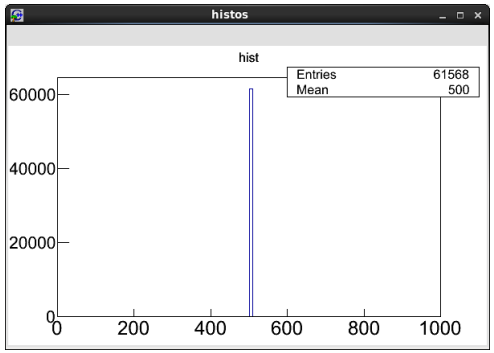
（デメリット）
コンポーネントが多いので使用するリソースが多くなる

ex16 Mergerを利用して複数台のPCからデータを収集する



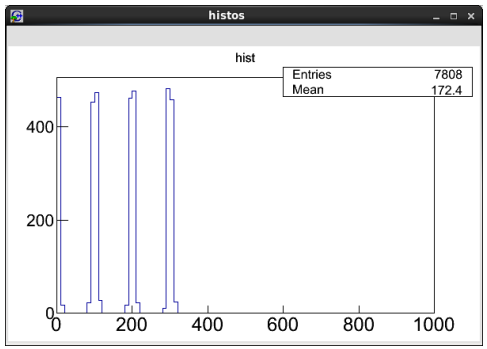
ex16 Mergerを利用して複数台のPCからデータを収集する

192.168.10.100



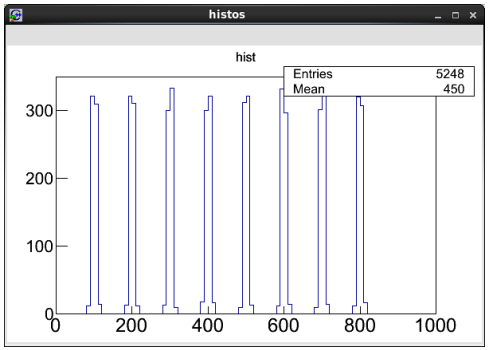
+

192.168.10.101



+

192.168.10.102



ALL

=