Fermilab/CD
Beams-doc-1949-v3
December 2, 2005
Version 3

Main Injector Beam Position Monitor Upgrade Software Specifications for Data Acquisition

Luciano Piccoli, Stephen Foulkes, Margaret Votava
Fermilab, Computing Division, CEPA

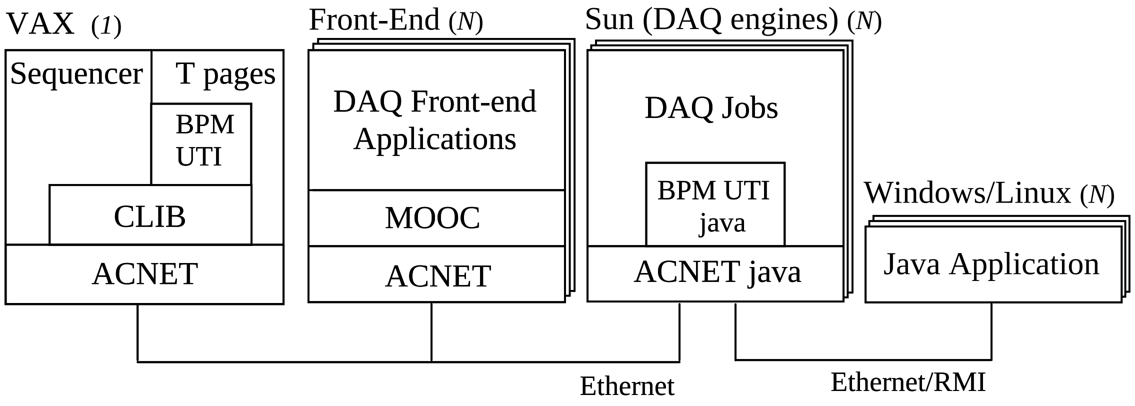
Brian Hendricks
Fermilab, Accelerator Division, Accelerator Controls Department

Abstract

This document contains the specifications for the Main Injector BPM upgrade's front end software. Expected operating modes and interactions with the BPM hardware are described. Data structures for communication with the online software via ACNET are also defined.

1 Overview

In compliance with the existing Accelerator Controls software architecture, the software pieces surrounding the BPM hardware are divided into 3 layers: the front end software running on the individual front-end computers in the readout creates in the service buildings (usually referred to as "houses"), user applications running on analysis nodes (Windows or Linux), and online software running on a central server (VAX) and a few DAQ engines (Sun) providing the primary (but not exclusive) bridge between user applications and the front end software. This document will focus on the front end software.



In the remaining part of this section, we will briefly outline the required measurement types, the hardware configuration and the requirements on the front end software. In section 2 we will describe the controls, configurations and data buffering within one front-end create. In section 3 we will define the interface to the online software. In sections 4 through 7 we will elaborate on calibration, debugging, alarms and monitoring.

1.1 Measurement Types

The different types of measurements to be handled by the upgraded Main Injector BPM system are summarized as follows:

- **User Defined Turn-by-Turn:** A measurement of the orbit on every turn (588 53Mhz RF buckets) for a specified number of turns, performed in high bandwidth mode.
- **Injection and Extraction Turn-by-Turn:** A measurement of the orbit on every torn of each portion of the beam injected into the machine. It also must provide the capability to measure the extraction turn of at least one portion of the beam extracted from the machine.
- **Flash Frame:** A single orbit measurement, performed in high bandwidth mode. Flash frames are collected from the injection and extraction turn-by-turn buffers.

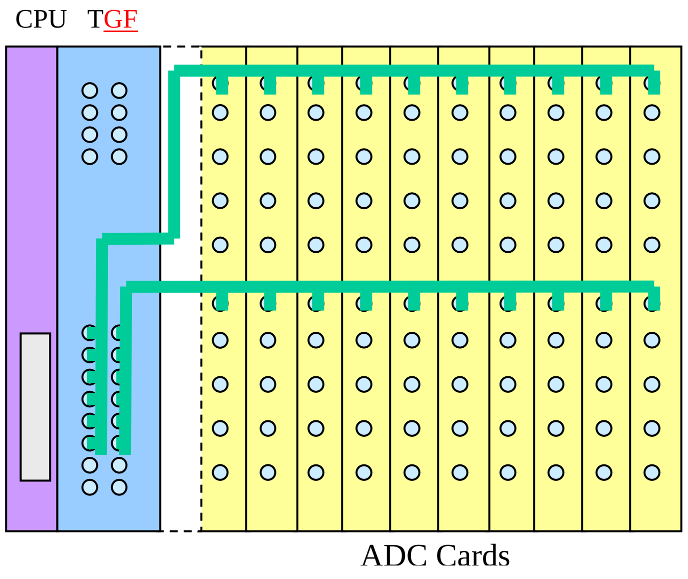
The first data sample in the injection/extraction buffer that has beam is the flash frame.

- **Averaged Orbit:** An average of a specified number of turn-by-turn measurements
- **Display Frame:** A narrow bandwidth measurement triggered by the display frame TCLK ($7B). This measurement can occur at most once per cycle.
- **Profile Frame:** A narrow bandwidth measurement triggered by the profile frame TCLK ($7A). This measurement can occur at most 128 times per cycle.

It is important to note that for all variations of the Turn-by-Turn measurement, all BPMs should be triggered off the same bunch passing with the same time advances.

1.2 Hardware Configuration

Based on the experience from the Recycler BPM system and the Tevatron BPM system, the Main Injector BPM system is designed to be a VME crate holding a crate controller with a PCMUCD daughter board, a Timing Generator Fanout (TGF) board and up to 10 ADC boards. There are seven houses around the Main Injector ring, each having one BPM crate to handle up to 40 BPMS. The specific models and main functions of the boards are briefly described in the following.



- The Main Injector TGF Board is based off of the Tevatron TGF Board. It is designed to perform the following:
    - o Phase lock to the Main Injector RF frequency which varies from 52.8-53.1 MHz and generate a clock signal 10/7 times the RF clock for up to 10 ADC boards
    - o Decode the events transmitted through the TCLK and MDAT systems to provide advanced arming signals to the crate controller so that the hardware can be configured in time for different measurement types or

present pre-defined requests for data transfer from the crate controllers to the online software.
    - Decode the events transmitted through the Recycler Beam Sync, Main Injector Beam Sync and Booster Beam Sync to derive accurate timing and triggering signals to synchronize up to 10 ADC boards with respect to the different beam arrivals.
- The ADC Boards have been chosen to be model ECDF-GC814-FV-2 from the EchoTek Corporation. The digitized output can be raw ADC count or digitally down-converted and filtered to extract the strengths of the 53 MHz and 2.5 MHz components. The output of each channel is represented by two components: a real (I for in-phase) and an imaginary (Q for quadrature) part.
- The crate controller has been decided to be an MVME5500 from Motorola. This model was chosen over the MVME2400 used in the Tevatron BPM and Recycler BPM due to its larger memory capacity, gigabit ethernet capability, faster processor and longer service lifetime. This card is responsible for the communications between the online software, TGF and EchoTek ADC boards. All controls and data transfers to/from those boards are preformed by the crate controller.
- The PMCUCD daughter board is from TechoBox Inc. It also decodes the TCLK signal and triggers the standard ACNET/MOOC software applications accordingly. Since this board belongs to the Accelerator Controls Infrastructure, it will not be discussed any further in this document.

1.3 Requirements on the DA Software

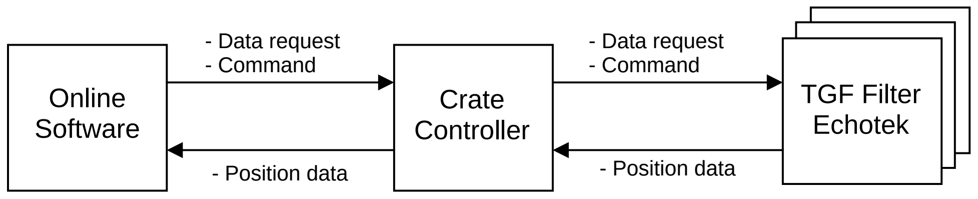
In order to comply with the existing Accelerator Controls software architecture and minimize the time needed for development and debugging, the DA software must meet the following requirements:

- All communications between the online software and the front-ends will be conducted via ACNET devices. This includes data readout as well as setting acquisition and readout parameters. Internal diagnostics, however, do not have this constraint.
- Data acquisition happens asynchronously from data readout, i.e., the BPMs can be configured to take data continuously on certain triggers, but the data will not be read out until later. Not all data that is collected is read out. This implies that the front end must have buffers for each measurement type and manage readout requests sent down from the online software.
- “Event assembly” is done by the online software, not by the front end. Therefore a given BPM crate does not need to have any knowledge of any other BPM crate(s). The data sent by the crate will however include information for having the data synchronized by the online software. That information includes time stamps and turn counts from the TGF boards.
- The front-ends should detect state changes via TCLK reset events. This will help to reduce the complexity of the timing board, allow for faster builds and testing of software changes and push the knowledge of the BPM behavior to the BPM themselves.

- The front-ends will use the Tevatron BPM software and the EchoTek driver as a starting point.
- The crate controller will run VxWorks.
- The front-end software will be responsible for making sure that all hardware is configured in such a way that all timing margins are met.

2 Data Acquisition

The front-end data acquisition system is located between the online software and the BPM digitizing hardware. Any access to information and controls on the electronics boards will pass through the front-end processor. The information includes beam position data, calibration data and diagnostics data among other configuration parameters.



The communication between the online software and the front-end processor is ACNET/MOOC based, while the communication between the front-end processor and other electronics happens over the VME backplane.

During a measurement, the digitized and filtered data is first stored in the memory on the Echotek boards and then transferred to the crate controller at some point before the end of the cycle. The data is saved in a set of buffers on the crate controller so that it may be accessed at another point in time. The depth and number of logical buffers that reside in the crate controller will vary depending on configuration and total physical memory [3]. The actual buffer implementation will be discussed in the Software Design Document [4].

2.1 Main Injector Cycles

Each Main Injector cycle is uniquely identified by a machine state. This state information is transmitted on the MDAT system and is valid at the TCLK reset for the main ramp. Each state must have a unique set of readout specifications that is re-configurable by the user. The format of the readout specification is discussed in greater detail in section 3.2. Table 1 lists the operational modes of the Main Injector as of 10/1/05 [1].

Description	**TCLK Reset Event**	**MI State**	**Extraction Energy**
Pbar stacking	$14, $29, $8E, $80	3	120 Gev
Pbar stacking cycle to MI abort	$14, $29	3	120 Gev

Protons to Tevatron	$15, $2B, $4D	4	150 Gev
Tevatron proton cycle to MI abort	$15, $2B	4	150 Gev
Slip stacking and NuMi target	$14, $19, $23, $8D, $8E, $A5	5	120 Gev
Pbar stacking and NuMi target	$14, $19, $23, $8D, $8E, $A5	7	120 Gev
NuMi target cycle to MI abort	$19, $23	8	120 Gev
Pbars from Recycler to Tevatron	$2E, $2A, $40, $E4	11	150 Gev
Protons from Tevatron	$2E, $2A, $5D	12	150 Gev
Slip stacking and SY120	$14, $13, $21, $80, $30	14	120 Gev
Proton Beam from Booster to MI Abort	$13, $2D	15	8 Gev
Proton Beam from Booster to Accumulator	$16, $2D, $93	16	8 Gev
Proton Beam from Booster to Debuncher	$16, $2D, $85	16	8 Gev
Stacking and SY120	$14, $13, $21, $80, $30	17	120 Gev
Pbar Beam from Recycler to Recycler	$2D, $E4, $E0	19	8 Gev
Pbar Beam from Accumulator to Recycler	$91, $2D, $E0	20	8 Gev
NuMi target	$19, $23, $A5	21	120 Gev
Pbars from Accumulator to Tevatron	$91, $2E, $2A, $40	23	150 Gev
SY120	$13, $21, $30	24	120 Gev
Proton Beam from Booster To Recycler	$2D, $E2	25	8 Gev
Proton Beam from Recycler to MI abort	$2D, $E3	25	8 Gev
Pbar slip stacking	$14, $29, $8E, $80	28	120 Gev
Tevatron pbar cycle to MI abort using protons	$2E, $1C, $2A	30	150 Gev

2.2 Timing

The DA system will be configured to operate in one of two basic running modes: wide bandwidth and narrow bandwidth. The former will be used for closed orbit measurements, while the later will be used for the various types of TBT measurements.

2.2.1 Timing Margins

It is understood that this system will require periods of time when data acquisition is not possible. There are currently three identified periods:

- Start of cycle – There will be at least 50ms between the TCLK reset event and the arrival of beam. This is to allow the front end to configure the timing boards and EchoTeks
- Mode Change – The front end must be able to change modes in less than 10ms, i.e. - going from Closed Orbit to Turn By Turn.
- End of Cycle – There will be at least 500ms between the end of beam signal and the beginning of the next cycle in order for the front end to read data off of the EchoTeks.

2.2.2 Delays

It should be noted that in a given running state of the Main Injector, the time delay from the Main Injector turn marker (MIBS $AA) as seen by the TGF until the actual bunch passing is fixed at each BPM location. In order to ensure that the ADCs are triggered at the same time relative to the actual bunch passing of the same turn, 4 layers of delays are implemented:

- Global Delay – A delay applied to all bpms
- House Delay – A delay applied to all bpms in a given house
- Board Delay – A delay applied to all bpms connected to a particular EchoTek board
- Channel Pair Delay – A delay applied to a single bpm

There will be different delays for different particles that circulate in different directions. Delays can be set using the diagnostics console application, and are effective at the start of the next Main Injector cycle.

2.2.3 Narrow Band

For narrow band measurements, the front-ends do not need to be synchronized to the same turns/bunch passing; they only need to cover the same length in time. In this mode, the TGF simply decimates the Main Injector revolution/turn markers, generates an interrupt to inform the crate controller of a pending data acquisition, applies appropriate time delays to account for the different bunch arrivals at different BPM pick-ups so that data from all the BPMs have the same phase, and generates trigger (SYNC) signals for the EchoTek boards. When the pre-set number of samples (burst count) has been collected, the first EchoTek board will issue a collection-done interrupt to notify the crate controller for data transfer.

2.2.4 Wide Band

For wide band measurements, the front-ends in all the houses will need to be synchronized to the same specific turn. This is achieved by setting the TGF to wait for a particular “start event”. Upon the occurrence of this “start event”, the TGF generates an interrupt to inform the crate controller of a pending data acquisition, then repeats the following steps for the desired number of turns: wait for the next “turn event”, be it the turn markers, apply appropriate time delays and generate trigger signals for the EchoTek boards. When the pre-set number of triggers has been received, the first EchoTek board will issue a trigger-counter interrupt.

2.3 Data Acquisition Modes

Corresponding to the various measurement types it needs to handle, the BPM data acquisition system has a few different operation modes. These modes are mutually exclusive due to the need for specific configurations of the triggering and filtering in the EchoTek boards for a specific mode. These modes are:

- Closed Orbit
- Turn by Turn
- Flash

2.3.1 Closed Orbit Mode

This is a narrow bandwidth mode of the BPM system and is triggered by the data acquisition specification. Several buffers of closed orbit data will be kept by the front end:

- **Profile Frame Buffer** – 128 points deep. Data is taken from the most recent closed orbit measurement and placed in this buffer every time a TCLK $7A is received.
- **Display Frame Buffer –** A single measurement. Data is taken from the most recent closed orbit measurement and placed in this buffer every time a TCLK $7B is received.
- **Fast Time Plot Buffer –** Data is taken from the most recent closed orbit measurement every 2 milliseconds. This buffer is independent of state. It tracks several pieces of data useful for plotting the following:
  - A particular BPM’s position
  - A particular BPM’s intensity
  - A particular input’s I value
  - A particular input’s Q value
  - Sum signal for each channel (magnitude of A + magnitude of B)

The front end software must be able to have a profile frame buffer and display frame buffer for each state. At the beginning of each Main Injector state its profile and display frame buffers will be erased. If the front end is not collecting closed orbit data and a profile frame or display frame or fast time plot is called for an error value will be placed into the appropriate buffer.

The measurements are triggered at a 500 Hz rate. This rate is sourced by the TGF board through a decimate-by-*N* counter to down-sample the 90 kHz Main Injector turn marker (MIBS $AA). The value of *N*, and thus the trigger rate, can be adjusted through the turn modulus/decimation register on the TGF board. At 500 Hz, the DA system has about 200 us idle time between triggers.

2.3.2 User Defined Turn by Turn

User defined turn by turn measurements can occur at any time during a cycle, but can only be taken once per cycle. A turn by turn measurement will be taken when it is enabled in the command list and when the ACNET device that is passed down with the command has a non zero value. It will overwirte any flash turn measurements that may have been taken earlier in the cycle. Upon the successful completion of the turn by turn measurement, the data will be transferred off of the EchoTeks to the crate controller. The positions and intensities of the proton/pbar bunch at each BPM location will be calculated in the same fashion as closed orbit data. The position and intensity as well as the raw I and Q values will be stored for later readout. There must be a separate turn by turn buffer for each state.

2.3.3 Flash Turn by Turn

The front end software will be able to take turn by turn measurements for particle injection and extraction, triggered by the relevant beam sync clock event. The system must support taking 512 turn per measurement, and be able to store up to 20 data sets for each state.

If there are multiple injections in the same cycle, the system must provide the capability to measure the *first* turn for each portion of beam injected in the machine. Successive injections from the Booster happen at a maximum rate of 15 Hz.

If there are multiple single turn extractions in the same cycle, the system must provide the capability to measure the extraction turn of at least one portion of beam extracted from the machine.

The system must generate an Averaged Orbit by averaging the injection turn-by-turn data for the first injection into the Main Injector in each cycle. The Averaged Orbit will be the average of the first 16 turns with beam. This data will be used for machine injection closure.

2.4 Data Processing

After the EchoTek boards have completed data acquisition, the crate controller will read out the I-Q pairs. The modulus of each channel (M) is then calculated:

$$M = G \times (\sqrt{I^2 + Q^2} - M_0)$$

Where G and M_0 are the gain and offset of the electronics. Finally the beam position (D) and intensity (S) can be determined according to:

$$D = g \times \frac{M_A - M_B}{M_A + M_B} - D_M$$
$$S = M_A + M_B$$

Where g is a scale factor to convert the unit-less quantity to millimeters (nominally 26 mm), D_M is the mechanical offset that was surveyed relative to the BPMs electrical center before the BPM was installed in the ring.

For each BPM, the position (D) and intensity (S) are to be stored along with the raw I and Q pairs.

3 Interface to Online Software

This section defines how data and commands are exchanged between the front-end DAQ software and the online software. ACNET will be the means of transportation of data and commands between the front-end DA and the online software.

Requests may be made in parallel by disjoint applications, and some mechanism for avoiding conflicts must be designed and implemented.

BPM data read by the online software will be organized according to the data structures defined in section 3.3. Online applications that make use of the old system must be changed to handle new data formats.

The supported ACNET protocols will be SETDAT, RETDAT and Fast Time Plot (FTP). The snapshot protocol (not to be confused with a BPM snapshot) will not be supported by the front-end DAQ. The front-end must be able to generate FTP data at a rate up to 500 Hz.

3.1 SSDN / ACNET Device Mapping

The following suggested SSDN numbers will be used to map to the appropriate ACNET devices. There is at least one ACNET device associated with each SSDN number.

The SSDN number is an 8 byte field split into 4 2-byte pairs. See the MOOC front-ends document for a detailed field description:

http://www-bd.fnal.gov/controls/micro_p/mooc_front_ends.html

For reading BPM data, this project will use the recycler BPM model of object id:

- 0x0020 for BPM Control
- 0x0021 for BPM RETDAT/SETDAT
- 0x0022 for BPM FTP data

3.1.1 SSDN Mappings and ACNET Devices for Command Lists

Each house will have an ACNET device for each possible machine state. The device will take the form I:xxCMyy where xx is the house number (10, 20, 30, 40, 50, 6S, 6N) and yy is the machine state. Note that the machine state is represented in hex, so machine state 11 on the MI40 front end would be I:40CM0B. These devices will be used to send command lists down to the houses, the data structures of which are discussed in section 3.2.1. The SSDNs for these devices take the form 0000/0020/yy00/2094 where yy is the machine state, which is also represented in hex.

3.1.2 SSDN Mappings and ACNET Devices for Timing

There are three ACNET devices for every house that are used for setting delays. I:BxxBRD (SSDN 0000/0020/0000/2087) is for setting the board delay, I:BxxHSD (SSDN 0000/0020/0000/2086) is for setting the house delay and I:BxxCHD (SSDN 0000/0020/0000/2089) is for setting the channel delay where xx is the house number (10, 20, 30, 40, 50, 6S, 6N). I:BxxBRD and I:BxxCHD are array devices.

3.1.3 SSDN Mappings and ACNET Devices for Diagnostics

There are two ACNET devices for diagnostics, I:BxxACQ (SSDN 0000/0020/0000/2082) for sending an acquistion specification to the front-ends and I:BxxADV (SSDN 0000/0021/0003/2100) for reading back the diagnositcs buffer.

3.1.4 SSDN Mappings and ACNET Devices for Front End Status

There are four devices that are used for collecting status from the front ends. I:xxHIST (SSDN 0000/0020/0000/2092) will return the software start time, the time of the last mode change, the last TCLK recevied and the time the last TCLK was received. I:xxSTAT (SSDN 0000/0020/0000/2091) will return the status of various timing signals as they are seen by the timing card. I:xxCONF (SSDN 0000/0020/0000/2093) will return the number of EchoTeks installed into the crate as well as the closed orbit frequency. I:xxVER (SSDN 0000/0020/0000/2094) will return the versions of the mibpm and gbpm libraries, as well as the firmware versions on the EchoTeks and TGF. The xx in the device name refers to the house number.

3.1.5 SSDN Mapping for FTP devices

Each channel can have an I and Q value. Data from a pair of channels (i.e. A and B plates) can be manipulated to generate intensity and position information. There will be a position and intensity measurement for the horizontal channel, and a position for the vertical channel for both proton and anti proton data. One EchoTek card contains eight channels, allowing 4 BPMs to be read out. A complete house with 10 EchoTek boards can read out 40 BPMs, each one with 2 channels. The total channels in a house is 80, and each channel has an I and Q value associated with it.

The following FTP devices have been defined:

- 0000/0022/0000/22**YY** : This device contains the I, Q, and sum signal information for each channel, where **YY** corresponds to channel 0x00-0x50. The first element in the device is I, the second is Q and the third is the sum.
- 0000/0022/0001/22**YY** : This device contains position and intensity information for protons or pbars, where **YY** corresponds to a BPM number 0x00-0x25. The first element is position while the second element is intensity.

The maximum number of FTP devices provided by one house is 120:

1 I/Q/SUM x 80 channels + 1 p/pbar * 40 position/intensities = 120

3.1.6 SSDN Mappings for RETDAT devices

The RETDAT protocol is used to retrieve most data read out by the BPM system. Through RETDAT it is possible to read the following BPM data:

- Profile frame buffer (array)
- Display frame buffer
- Flash Turn Buffer (array)
- Closed Orbit Buffer (array)
- Turn-by-turn Buffer (array)

The BPM data will contain information from all EchoTek cards present in the system, except when handling turn-by-turn requests that specify a single BPM. All array buffers can return any number of frames.

RETDAT devices of the form 0000/0021/**ZZ**0**Y**/000**X** have been defined:

- **X** corresponds to the channel, which can be one of the following values:
    - **0** - Profile Buffer
    - **1** - Display Buffer
    - **2** - Flash Buffer
    - **3** - Turn By Turn Buffer
- **Y** corresponds to how the data is represented:
    - **0 -** Raw I/Q values
    - **1** - Scaled positions and intensities
- **ZZ** corresponds to the BPM number, and is only valid for the Turn by Turn channel. It can range from 0x00 to 0x25 (40 decimal).

3.2 Configuring the BPMS

For every MI state there is a sequence of such commands associated. These commands must be known by the front-end software before the MI cycle is started. The list of commands for any cycle can be changed at any time by online users. However if the cycle is currently running the front-end will wait until the end of the cycle to change the cycle command table.

The front-end will not preform any checking of the commands as they are sent down from the online software. The front-end will provide means to read back the cycle command configuration, but no history of commands for any cycle will be kept by the front-end software.

All commands received by the front-end that are marked as enabled will be executed in order which is specified in the command list. The only exception to this will be a command that specifies a turn-by-turn measurement. If this command is enabled, the front-end software will query an ACNET device that specifies a delay to begin the turn-by-turn measurement. If the ACNET device is set to 0, the measurement is not taken. Otherwise, the measurement will be taken at the time specified. Any measurements that are specified after the turn-by-turn will not be performed, and the turn-by-turn measurement will overwrite any flash measurements that had been taken previously.

3.2.1 Data Structures

The data sent to the front-end DAQ from the BPM library and/or applications is based on the following C data structures. All data is big endian. The front-end software will support a list of at most 32 commands during a given cycle. It also at this time supports the following four commands: FILTER, TURN_BY_TURN, CLOSED_ORBIT and FLASH.

```
const int MIBPM_MAX_COMMANDS_PER_CYCLE 32;

enum CommandType {
  FILTER,
  TURN_BY_TURN,
  CLOSED_ORBIT,
  FLASH
};

enum ParticleType {
  PROTON,
  PBAR,
};

enum FilterSetting {
  MIBPM_2_5MHZ_FILTER,
  MIBPM_53MHZ_FILTER
};

enum AttenuationType {
  DB0,
  DB6,
  DB12,
  DB18,
  DB24,
  DB30,
  DB36,
  DB42,
  DB48
};

typedef struct {
  int type;
  int delay;      // Specified in miliseconds
  union {
    int particle; // Proton or Pbar (FILTER)
    int enabled;  // (All other commands)
  } datum1;
  union {
    int frequency; // 53 MHz or 2.5 MHz (FILTER)
    int turnDelay; // (TURN_BY_TURN)
    int bsync;     // MIBS, RRBS or BES (FLASH)
  } datum2;
  union {
    int attenuation; // (FILTER)
    int bucketDelay; // (TURN_BY_TURN)
    int turnDelay;   // (FLASH)
  } datum3;
  union {
    int bucketDelay; // (FLASH)
  } datum4;
} MIBPM_STATE_COMMAND;

typedef struct {
  int size;
  MIBPM_STATE_COMMAND commands [MIBPM_STATE_COMMAND_SIZE];
} MIBPM_STATE_COMMAND_
```

3.3 Retrieving Data

3.3.1 Headers

The current version of the `MI_DATA_HEADER` is 1. The status variable inside the `MI_DATA_HEADER` structure will be set to a non-zero value if there is some problem at the crate level. Upon receiving a `MI_DATA_HEADER` with a non-zero status value the online software will ignore the data. All data is big endian.

The time data structure in the `MI_DATA_HEADER` is the time stamp of the first data frame.

```
const int MI_BPMS_PER_HOUSE 40;
const int MI_TURN_SCALED_BLOCK_SIZE 512;
const int MI_MAX_FRAMES 512;

typedef struct BPM_TIME
{
      ulong timestamp;        /* timestamp in second (GMT) */
      ulong nanoseconds;      /* nanoseconds after timestamp */
};
```

```
typedef struct MI_TRIGGER_INFO
{
      int type;   /* MIBS / RRBS / BES / Closed Orbit */
      int value;  /* trigger value */
};

typedef struct MI_DATA_HEADER
{
      int version;                  /* data structure version */
      int status;                   /* transaction status, this  is
                                       defined later in this document.
                                     */
      BPM_TIME time;                /* timestamp */
      unsigned int turn_number;     /* starting turn number */
      unsigned int num_turns;       /* number of turns in data */
      double time_in_cycle;         /* starting time in cycle */
      int data_type;                /* flash/profile/turn by turn… */
      MI_TRIGGER_INFO trigger_info; /* trigger information */
      int data_source;              /* beam or calibration */
      int particle_type;            /* proton/pbar */
      int scaled_data;              /* scaled/raw */
      int calibration_id;           /* calibration data ID number */
};

typedef struct MI_STATE_DATA
{
      int   reset_event; /* TCLK event which began the beam cycle */
      int machine_state;      /* MDAT / TCLK RESET */
      int bpm_state;          /* not yet defined */
};
```

3.3.2 Non Turn-by-Turn Data

The front end will return either an `MI_ORBIT_DATA_SCALED` or `MI_ORBIT_DATA_RAW` data structure depending on the type of data requested for all measurements except Turn-by-Turn and FTP. All data is big endian. The `bpm_status` array inside the `MI_FRAME_DATA` data structures will be set to one of the following values:

- **0** – OK
- **1** – To little beam intensity
- **2** – Alarm level – if we want alarm limits
- **3** – Saturated
- **4** – Hardware error
- **5** – Channel not in use

```
typedef struct MI_ORBIT_DATA_SCALED
{
      MI_DATA_HEADER header;        /* generic bpm information */
      long num_frames;              /* number of frames returned */
      MI_FRAME_DATA_SCALED frame_data[MI_MAX_FRAMES];
};

typedef struct MI_ORBIT_DATA_RAW
{
      MI_DATA_HEADER header;        /* generic bpm information */
      long num_frames;              /* number of frames returned */
      MI_FRAME_DATA_RAW frame_data[MI_MAX_FRAMES];
};

typedef struct MI_FRAME_DATA_SCALED
{
      MI_STATE_DATA state_data;     /* machine state information */
      int frame_number;             /* ordinal number in front-end */
      int bpm_status[MI_BPMS_PER_HOUSE]; /* status information for
                                            bpms, defined later in this
                                            document. */
      float positions[MI_BPMS_PER_HOUSE]; /* position values */
      float intensities[MI_BPMS_PER_HOUSE]; /* intensity values */
};

typedef struct MI_FRAME_DATA_RAW
{
      MI_STATE_DATA state_data;     /* machine state information */
      int frame_number;             /* ordinal number in front-end */
      int bpm_status[MI_BPMS_PER_HOUSE]; /* status information for
                                            bpms, defined later in this
                                            document. */
      short i[MI_BPMS_PER_HOUSE*2]; /* Each channel has an I and Q */
      short q[MI_BPMS_PER_HOUSE*2]; /* value, and there are two  */
                                    /* channels per BPM. */
};
```

3.3.3 Turn by Turn Data

The front end will return either an `MI_TBT_DATA_SCALED` or `MI_TBT_DATA_RAW` data structure depending on the type of data requested for all Turn-by-Turn measurements. All data is big endian.

```
typedef struct MI_TBT_TURN_SCALED
{
      unsigned int turn_number;     /* turn number */
      float position;               /* position in mm */
      float intensity;              /* beam intensity */
};

typedef struct MI_TBT_TURN_RAW
{
      unsigned int turn_number;     /* turn number */
      short i[2];                   /* There is an I and Q value for */
      short q[2];                   /* every channel, 2 channels per */
                                    /* BPM */
};

typedef struct MI_TBT_DATA_SCALED
{
      MI_DATA_HEADER header;
      MI_STATE_DATA state_data;
      int status;
      int num_turns;
      MI_TBT_TURN_SCALED turn_data[MI_TURN_SCALED_BLOCK_SIZE];
};

typedef struct MI_TBT_DATA_RAW
{
      MI_DATA_HEADER header;
      MI_STATE_DATA state_data;
      int status;
      int num_turns;
      MI_TBT_TURN_RAW turn_data[MI_TURN_RAW_BLOCK_SIZE];
};
```

4 Interface to BPM Hardware

The front-end software interfaces to the BPM hardware (which consists of a Timing Generator Fanout Board and one or more EchoTek 814gc boards) over the VME bus. The front-end software is responsible for configuring the hardware as well as retrieving data from it over the VME bus.

4.1 Operation of the EchoTek ADC Boards

Each EchoTek board provides 8 receiver channels of 14-bit, 80 MHz (maximum) analog-to-digital conversion and digital processing in a single 6U VME slot. It supports 3 operating modes:

1. Gate Mode – Data is collected as long as the external sync signal is active.
2. Tigger & Free Run – Data collection begins at the rising edge of the external sync signal, or when the Software Sync Bit is written, and continues until the Trigger Clear bit is written.
3. Trigger & Counted Burst – A preprogrammed number of samples (burst count) is acquired and processed with each occurrence of an external sync pulse, or writing to the Software Sync Bit.

In the trigger modes, the external sync signal and trigger a delay specified in the SYNC_DELAY registers. The SYNC_DELAY registers are 12 bit counters that are clocked at the ADC sample rate. Each register controls a pair of receivers – 1 & 2, 3 & 4, 5 & 6, 7 & 8.

The 8 channels on each EchoTek board can be independently configured to output one of 3 types of data:

1. Count Data – The FPGA on board the EchoTek generates a continuous counting sequence and puts the data directly into memory. This bypasses the whole chain of analog-to-digital conversion and digital processing. This mode is intended for checking the data handling within the board and check VME interactions between the board and system controller.

2. Raw Data – The ADC counts are stored into the memory directly, bypassing the digital processing. The samples are right justified with the two least significant bits always set to zero. Samples are stored in pairs to form a 32 bit word.
3. Receiver Data – The ADC counts are digitally down-converted, decimated and filtered to output an interlaced sequence of 24bit I's and Q's. The 24bit I's and Q's can either be truncated to 16bit and then packed into 32bit words or directly output as 32bit words.

The memory for each channel is a 128K x 32 bit SRAM operating in a FIFO fashion.

The procedure to initialize the boards through the driver is as follows:

1. Read in all the setup files (*.ini and *.ch), parse the files and then create an array of "ghSetup" structures in the crate controller's memory by calling ecdr814gcReadSetup() in the startup script.
2. Install a driver for each board by calling ecdr814gcInstall()
3. Open the driver by calling open()
4. Allocate data buffers in the create controller's memory for each gray chip channel. The buffers must be 8 bit aligned to the address of the corresponding gray chip channel.
5. Map the allocated buffers to those on the EchoTek boards by calling ecdr814gcSetBufferChBrd()
6. Copy the desired setup from the "ghSetup" array to all the boards by calling ecdr814gcCopySetupAll()
7. Set the channel-pair delays by calling ecdr814gcIoctlCopySyncDelayAll()
8. Program all the Gray chips by calling ecdr814gcProgramGrayAll()
9. Set up the DAQ conditions by calling ecdr814gcRdSetupAll()
10. Set the trigger counters by calling ecdr814gcSetNumTrigsAll()
11. Reset the boards by calling ecdr814gcIoctlClearAll()

Steps 4 through 11 need to be repeated when changing to a different operation mode.

After initialization, the boards need to be enabled by calling ecdr814gcEnableSyncAll(). Then the boards cab be disabled and read out by calling ecdr814gcReadAll(). These two function calls form a complete DAQ cycle.

5 Calibration

The front-end software is able to return raw data as well as calculated beam position to the online applications. Raw data does not require any processing on the front-end whereas calculated beam position requires the use of calibration constants.

Calibration constants are defined by offline processing. Data used for calibration will be collected from the front-end systems, running on calibration mode, and will have its data type marked as calibration data.

At startup time the front-end downloads current calibration constants. The calibration constants are retrieved by the front-end system via ACNET variables. The use of ACNET insures that the front-end automatically receives the latest calibration set. The calibration set used by the front-end is identified by a database ID. This ID should be included in the metadata that is returned when calculated data is requested by an online application.

6 Diagnostics, Test Suite, and Simulation

6.1 Diagnostics

The front-end software will provide diagnostics data via ACNET devices. It will provide means for operators or programs to detect a bad or misbehaving BPM.

Some diagnostics operations follow:

- Generate closed orbit data and return known closed orbit values
- Generate turn by turn data and known values for a single turn by turn measurement
- Generate single turn data and return known values for a single turn measurement
- Check BPM hardware – Run test procedures on the BPM hardware, if supported
- Get Buffers – Return current contents of all (or selected) data buffers

6.2 Self-Testing Procedures

The front-end software should be able to perform tests on itself and on the associated BPM hardware. Results from self-tests should be available to user applications. Hardware tests will be performed if supported, i.e., the hardware should have the capability of receiving triggers from the front-end and generate data for self-tests. Software self-testing will be used for validating the data path from the time data is read out from the BPM until it is ready to be read via ACNET devices.

7 Monitoring

The front-end software should periodically send status and statistics messages to a monitor, via ACNET devices. There should be a central monitoring application that receives data from all BPM front-ends and points out BPMs that have problems.
Data from the front-end includes:

- Buffer Usage
- Up Time

- Available Memory
- Status of processes
- Number of requests
- Main Injector Status

References

[1] Dave Capista, ‘MI BPM Configurations for Operational States’, Beams-doc-1996-v5
[2] Alberto Marchionni et. al., ‘Requirements for the Main Injector BPM Upgrade’, Beams-doc-1786-v7
[3] Luciano Piccoli, ‘MIBPM Front-End Software Memory Requirements Review’, Beams-doc1955-v1
[4] Luciano Piccoli, ‘Main Injector BPM Software Design’, Beams-doc-2036-v0
[5] Brian Hendricks, ‘Main Injector BPM Console Application Software’, Beams-doc-1907-v2
[6] Charles Briegel, ‘Echotek Drivers’, Beams-doc-1041-v1

Change Log

Version	**Issue Date**	**Description Of Change**
1	11/15/05	Initial Revision
2	11/17/05	Updated data structures and ACNET devices
3	12/02/05	Miscellaneous fixes