Notes from RR BPM Meeting

November 1, 2012

Elliott McCrory

Present:

* Dave Capista
* Phil Adamson
* Roger Tokarek
* Charlie Briegel
* Nathan Eddy
* Elliott McCrory
* Brian Hendricks
* Peter Prieto

Dave began with a review of the 2005 MI BPM Requirements, which can be found at <https://beamdocs.fnal.gov/AD-private/DocDB/ShowDocument?docid=1904>.

High and low bandwidth: 2.5 and 53 MHz, respectively. It must switch between these modes in less than 10 milliseconds.

The measurement specification remains the same (except there are no anti-protons, of course).

The same set of DAQ will be supported:

* **Flash Frame**: A single turn orbit measurement, performed in high bandwidth mode.
* **Turn By Turn**: A measurement of the orbit on every turn (588 RF buckets)for a specified number of turns, performed in high bandwidth mode.
* **Averaged orbit**: An average of some number of Turn by Turn measurements.
  + A.k.a., “Injection Frame”
  + Average of 20 Flash Frames at injection
* **Display Frame**: A narrow bandwidth measurement taken at a specific time. This measurement occurs once per cycle.
* **Profile frame**: A narrow bandwidth measurement taken at a set of specific times during a cycle. This measurement can occur up to 128 times per cycle.
* **Fast Time Plots:** A narrow bandwidth measurement that occurs at 500hz when the system is in the closed orbit mode.
* **User Data Buffer** – not used and not actually needed. It is OK to keep it, but removing it is OK, too.

Greg Vogel has a list of events that are relevant for this system. Phil will see to it that Peter gets a copy of this list.

Injection: Always record 512-point buffers. There are 20 of them.

Nathan: Can we always count on a Main Injector cycle? Phil and Dave: Yes. It is conceivable that someday that answer may become “no,” but it is assumed that this will mean a re-write of the BPM software (which will be a big deal). We can count on the hundreds of millisecond dead time when the buffers can be flushed (which happens when the MI is ramping down).

The Recycler end-of-beam event is 0xE6.

Narrow-band and wide-band measurements are (and will always be) mutually exclusive.

The state machine is implemented in the front end and in the ACNET servers. It is crucial that these state machines match. State machine information is broadcast (multicasted?) on MDAT.

It looks like the Transition Board will need to be aware of the MDAT frames. This might not be simple.

When the system is in TBT mode, that is all it can do.

There was a discussion on the value of the gain that the system deals with. Dave points out that the high-level operations program has values of “High,” “Medium” and “Low.” Apparently, these values get translated to actual dB levels prior to the front end seeing it. This is the way it should be.

Peter points out that the dynamic range of the beam in the g-2 era is huge. These will be 2.5 MHz bunches, and Peter calculates that the raw voltage from the BPMs could be as high as 100 volts! There will be no gain adjustment for this mode.

Currently, the state machine information on MDAT is of the form:

* **Command: “**Set\_BPM\_Mode”
* **Datum1**: Enables and disables message
* **Datum2**: Sets 2.5Mhz proton, 2.5Mhz Pbar, 53Mhz proton, and 53Mhz pbar settings.
  + 2.5Mhz proton, 2.5Mhz Pbar.
  + 53Mhz proton, 53Mhz Pbar
* **Datum3**: Sets Wide band (TBT) or Narrow band (Closed orbit) mode.
* **Datum4:** Sets the filter attenuator.
  + 0db, -6db, -12db, -18db, -24db, -30db, -36db, -42db, -48db

Phil: We will need to store protons for several seconds, but we will never need to store them for several minutes, much less, several hours.

The Flash buffer must begin with one or more empty turns in order to assure that we got the first turn. Similarly for the end-of-beam mode: it has to end with empty turns just to be sure.

There are 6 new BPMs in the new 8GeV line (901, 902, ...). *This is where I was not able to follow the discussion very well any more. Please fill in details as to the understanding and the decisions, here—EM.*

Who will do the software needed here? Charlie? He says that he has not been assigned this task yet. Steve Volk implemented aspects of the original BPM code—maybe he can be drafted to do more work for us?

# Other Notes

Earlier, Capista told me that it is crucial that the internal data format in ACNET for the RR BPM readouts be exactly the same as for the present MI BPM system. Otherwise, the application programs (and probably other stuff) will break.

# Appendix: Events in the existing MIBPM code

From my perusing of the MIBPM code, I find the following events used in the MIBPM code

TCLK\_RESET -- An array of events: 20, 21, 23, 29, 2a, 2b, 2d and 2e.

ControlTask is run whenever any of these events is registered.

MIBPM\_PROFILE\_TCLK == 0x7a

MIBPMControl.cpp (line number 710): Constructor for profileGenerator.

MIBPMControl.cpp (768): Within a TRACE0()

MIBPMProfileTask.hpp (42): within method process(), it seems to catch this event and write the buffers with a recognizable value (888) if we are not in the state

MI\_CLOSED\_ORBIT\_MEASUREMENT.

MIBPM\_DISPLAY\_TCLK == 0x7b

MIBPMControl.cpp (718): Constructor for displayGenerator.

MIBPMControl.cpp (780): Within a TRACE0()

MIBPMDisplayTask.hpp (42): within method process(), it seems to catch this event and write the buffers with a recognizable value (888) if we are not in the state

MI\_CLOSED\_ORBIT\_MEASUREMENT.

MIBPM\_END\_OF\_BEAM\_TCLK == 0x26

MIBPMControl.cpp (722): Constructor for endOfBeamGenerator.

MIBPMControl.cpp (786 and 792): Within TRACE0()

MIBPMControlTask.hpp (139): Initiates a call to the method, MIBPMControl::endOfBeam().

MIBPM\_PREPARE\_FOR\_BEAM\_TCLK == 0x79

MIBPMControl.cpp (726): Constructor for prepareForBeamGenerator.

MIBPMControlTask.hpp (144): Initiates a call to the method, MIBPMControl::setPrepareForBeam().

MIBPM\_TBT\_ARM\_EVENT == 0xda

MIBPMStateManager.cpp(96): When the MIBPM\_STATE\_COMMAND becomes TURN\_BY\_TURN, this event becomes the armEvent. The armEvent otherwise comes from a struct called MIBPM\_STATE\_COMMAND, which contains 6 ints, the fourth of which determines this value.

MIBPM\_BSYNC\_TIMESTAMP\_CLEAR\_EVENT == 0xda

MIBPMTimingSystem.cpp (62): used like this:

timingSignalGenerator -> setRegister (TSG\_REG\_BSYNC\_CLEAR, MIBPM\_BSYNC\_TIMESTAMP\_CLEAR\_EVENT);

In the startup script, there are several lines that add events to a list at startup.

* bpmAddInjectionEvent(0x200);
* bpmAddInjectionEvent(0x7A);
* bpmAddInjectionEvent(0xD6);
* bpmAddInjectionEvent(0xD8);
* bpmAddInjectionEvent(0x1A3);
* bpmAddInjectionEvent(0x1A7);

The list being added to here is used to trigger the beginning of beam in several places in the code.

I see no evidence of any “end of beam” event list, only the single event listed above, 0x26.