Beam Velocity Regulation

Local application

Robert Goodwin

Mon, Nov 2, 2009

Review of `VELO`

The Linac beam energy is adjusted to keep a measured beam velocity signal constant. This adjustment is made by local application `VELO` running in `node061E`, where a beam velocity signal is available. To adjust the Linac energy, a setting message is sent to `node0737`, where the Linac klystron# phase channel resides, which is in turn monitored by `KRFP` to adjust the hardware.

The velocity signal in `node061E` is calculated—on each beam cycle before `VELO` runs—by `BPMR`, which averages a 10 MHz digitized velocity waveform over part of the Linac beam pulse. (To sense the presence of actual Linac beam, a beam intensity signal is monitored during the same part of the beam pulse to confirm that each digitized point averaged corresponds to sufficient beam intensity. Without beam present, the velocity signal is invalid.

Local application `VELO` monitors HEP beam pulses, defined as those for which event `0x10` applies, but not Linac study pulses. For each beam pulse in a series of cycles, it sums the average velocity reading furnished by `BPMR`. When a non-beam pulse is reached, it uses the resulting average velocity to subtract a reference velocity that is specified as the "setting" of the `D4VELO` velocity channel. This difference is multiplied by a gain constant, currently set at -1.208, and the result is used as an adjustment in degrees for the klystron# 7 phase, `C7PHAS`. To avoid erratic behavior, the amount of each adjustment is limited to 0.1 degrees. Since the least bit for `C7PHAS` corresponds to 0.0175 degrees, the allowed range of adjustments, in raw units absolute value, is `0x0001`–`0x0005`. Any fractional residual adjustment is retained as a basis in building the next adjustment.

Although `BPMR` requires a significant beam presence, `VELO` does not. If `BPMR` does not see enough beam, it does not update its beam velocity result `D4VELO`. But `VELO` does not know that, so it accepts the last measured velocity value again. One way this could be changed is to have `VELO` also monitor the count of "good points" that `BPMR` puts out as a diagnostic. (This count is the number of digitized beam presence data points that exceed a threshold.) It could require this to be above a certain minimum in order to qualify the velocity result. If `VELO` determines it is not good enough, then it could ignore the velocity reading for that cycle. When it reaches a non-beam cycle, if no velocity data has been summed, it will not make any phase adjustment. This would ensure that `VELO` does not react to invalid velocity values.

Another change to `VELO` might include a diagnostic log of adjustments, perhaps only logging those that exceed `0x0001` in absolute value. In this way minimal adjustments of one least bit would not be included in the log.

Post-implementation of new `VELO`

Local application `VELO` has been modified as above so that it monitors the number of good data points in the beam presence signal, before using the beam velocity signal value. It also keeps an internal log of adjustments it makes, noting the time of day and the amount of adjustment. The threshold for logging such adjustments is set a 1; *i.e.,* one must make more than 1 least bit adjustment to qualify for being logged. But one can modify this threshold for a special study.

Remote operation for testing

In order to facilitate testing the new version, the code was modified so that it can acquire the beam-related data of interest remotely, then adjust a dummy phase which can be compared with the current version that adjusts the real phase. The beam data is collected from a given node, which does not have to be the local node, although in actual use, it will be. The data collected remotely is as follows:

1. Clock event bit map portion that includes HEP event `0x10`
2. Beam status Bit `0x9F`
3. Beam velocity reading and setting
4. Channel `NGOOD` reading of the #data points in the beam intensity waveform, via `BPMR`

Data that is always found in the local node are the gain parameters, so that one can try out different values without commiting to using them in the actual operating `VELO`. The klystron phase channels can also be in the local (test) node, which can then be compared with the real klystron phase to see whether they track.

Once testing is complete, the new `VELO` can be installed in the real node, with the gains and target parameters changed accordingly, but without changing any of the `VELO` code. This works because the underlying system support for data requests automatically takes care of the difference between remote and local access to the data requested. (This same approach was used earlier when testing `BPMR`.)

Internal details

The new parameter layout for `VELO` is as follows, using typical values:

Parameter		*Value*	*Meaning*
ENABLE	B	<00CB>	Usual enable Bit#
VELOCTY	C	<0093>	Beam velocity Chan#
OVERALL	C	<009A>	Overall gain Chan#
GAIN 2	C	<009B>	Secondary gain Chan#
NGOOD	C	<007F>	#good data points in beam waveform Chan#
TARGNODE		<061E>	Target node, as an aid for testing remotely
PH1NODE		<0737>	Target phase adjust #1 node,chan
PH1CHAN		<0004>	
PH2NODE		<0736>	Target phase adjust #2 node,chan
PH2CHAN		<0004>	

With this set of parameters, one can run `VELO` on remote beam data or on local data. The code understands that `TARGNODE` specifies where to obtain the four pieces of beam data listed above. The gains are always found locally, and the target phases are determined by the last 4 parameters.

During testing, good tracking was observed compared to the former version of `VELO`, when using the same `OVERALL` gain value.