Tom Himel

Last revised 10/26/10

Original version 1/21/10

# EVG upgrade requirements

## Very top level requirements

Enough functionality will be moved from the MPG to the LCLS EVG to allow the LCLS to run without use of the MPG.

If the MPG is running, it will be possible to keep the EVG synchronized with it including sharing the same PulseIDs.

## Scope

Note that these requirements are not given in full detail. It will still be necessary to be familiar with what the MPG does and how the modifier bits are used by LCLS in order to write a more detailed functional requirements document and/or create the detailed design. For example, it does not tell the number of modifier bits or which ones are used by the LCLS or describe exactly how they are pipelined.

The reader is assumed to be familiar with the functionality of the MPG and EVG.

## More detailed requirements and some top level design

The main new task for the EVG will be to generate the beam code and modifier bits that it presently simply receives from the MPG via PNET to use and rebroadcast.

1. There is a new VME module whose design is being finalized by John Dusatko. This will provide a 360 Hz interrupt and information on which of those interrupts are timeslot 1. Once the design progresses far enough, writing the driver and device support for it is part of this project. As completion of the module is not expected until December, design and coding for the rest of this project should proceed before then and its interface should be separated enough so that this can be done.
2. The EVG shall generate the beam code (PP) and modifier bits at 360 Hz as needed for LCLS. (Note that YY is not needed by the LCLS as there are no CAMAC BPMs left in use.)
3. The names of these modifier bits shall be settable by the operator as shall the patterns at which they come out. We only expect these patterns to be changed a few times per year. Note that with LCLS-II we are likely to have a second beam code and extra modifier bits.
    a. The user should be able to define a new rate group setup while the LCLS is running without disturbing that running. This could be done with an EPICS DB, a java app or by

editing a file or… A *rate group* is the pattern of beam codes and modifier bits for several different rates for one or several related beam codes. A rate group runs on one or more time-slots. More than one rate group can be running at the same time as long as they use disjoint time-slots. For example, in the MPG right now LCLS and NLCTA use different time slots and have independent rate states and hence would be different rate groups in the new implementation. (NLCTA does not use the EVG, so this is not a real example. LCLS-II may be though.)

   b. After making a new rate group setup (which may involve several related changes), the operator shall be able to save it for future use (this could be to a SCORE configuration, to a disk file or to Oracle. I will call it a config.)
   c. Users shall be able to load a saved config for viewing and possible further changes.
   d. Users shall be able to activate a rate group setup that they have edited or loaded from a config so that it makes the EVG start using it for the generation of the PP and modifier bits.
   e. It is NOT necessary to use the same beam definition language (BDL) that was used to describe the modifier bits setup for the MPG. In fact a simpler, easier to program and understandable method is greatly preferred. At a meeting with Mike Stanek we discussed the possibility of using rates, starting pulse and inclusion and exclusion masks to define the patterns, but the exact method is an option left for the designer. An example interface is available in an Excel spreadsheet stored with this Word file.
   f. There shall be display(s) that allow an operator to see what bit patterns his setup would produce or is producing.

4. There will be buttons in the user interface to allow the beam rate to be easily changed. There will be different instances of the interface for different rate groups. Start with same rates available on the MPG (1, 10, 60? 120 Hz). Implement this in a way so if other rates are later wanted, the changes are not too difficult to make.
5. There are hardware buttons in MCC used to change the beam rate. As part of the Unique Devices AIP project, these have been moved to EPICS readout. The EVG will make use of these to change the beam rate.
6. Presently the MPG changes the patterns of modifier bits as part of changing beam rates. It gets these rates from the above mentioned software and hardware buttons and from the 1553 MPS system. LCLS will no longer use that MPS system, but will be using the link node MPS system. Stephanie is implementing the use of that MPS system and is simply adding the MPS mitigation device bits as extra modifier bits. She is not changing the patterns of other modifier bits. This means that inclusion and exclusion masks for things like reading BPMs will have to use the MPS bits to know when there is beam.
7. The single bunch and burst mode recently implemented in the EVG must continue to work.
8. The MPG has a way to handle rate transitions where modifiers are done differently for a short time after a rate transition. Unless this is easy to do, it should be left as a later upgrade as it presently is not needed.
9. There will be a method to keep the EVG synchronized with the MPG.

   a. The proposed design for this is to have the EVG continue to receive the MPG PNET broadcast. Alternate designs can be considered, but the detailed requirements below in some cases assume this design.
   b. The EVG shall continue to broadcast the bits needed for running LCLS even if the MPG or PNET goes down.
   c. The MPG and EVG will each have their own timeslot 1 signal and by design of the electronics they should match. If they do not match, an error shall be signaled and the EVG will proceed as though the MPG were down.
   d. Lower frequency signals like 1 Hz, 0.5 Hz, and the pulseID cannot be simply derived from the power line like the timeslot 1 signal. Hence the only way to get them synchronized is for the MPG and EVG to communicate. As mentioned above, this will be handled by having the EVG read the PNET broadcast from the MPG.
      i. There will be three possible easily user settable modes for MPG synchronization: *off, on with manual resync*, *and on with auto resync*.
      ii. In the *Off* mode, the EVG will ignore the PNET broadcast from the MPG.
      iii. In the *on with manual resync* mode, the EVG will compare its internally calculated values for the pulseID and a user selectable set of modifier bits (e.g. the 1 Hz ,0.5 Hz bits and PDU resync bits). If they agree it will copy over an easily user settable set of modifier bits from the MPG to the EVG (e.g. bits that say the FACET klystrons are firing that we may want to use for diagnostic purposes). It will also copy over the MPG beam code if and only if the EVG beam code is zero. If the bits disagree or both beam codes are nonzero, the EVG will set an error flag , stop using data from the MPG, and issue an error. When the operator hits a manual resync button, the EVG will wait for the next full pulseID information from the MPG (takes up to 30 seconds), reinit all its internal counters so its pulseID and modifier bits will match those it compares to the MPG, and reset the error flag.
      iv. In the *on with auto resync* mode, the EVG performs as in the previous mode, but the resync happens automatically, without operator intervention, the next time the EVG sees the full pulseID information coming from the MPG. Note that each time this resync happens (typically because someone rebooted the MPG) there will be a glitch in the LCLS pulseID and other modifier bits. This is not desirable, but seems unavoidable given the planned design.

10. For other projects (feedback and PDU timing setup) Partha will be writing a java application that provides a user friendly interface to setup inclusion and inclusion masks. Where the EVG changes require inclusion and exclusion masks, this application should be used.
    a. The EVG code already uses inclusion and exclusion masks to determine when events should be generated. The user interface for entering these masks is not user friendly. At some point (not necessarily part of this project, but it could be convenient to do it at the same time), this user interface should be improved using the above application.
11. Before loading a new rate group for use in the EVG, a safety check must be made to ensure its use will not make any klystrons run at too high a rate. It also must make available to the klystron

application information about the rate on each beam code so the klystron application can avoid activating a klystron on a beam code with too high a rate. To enable this, each klystron has a PV (TMSK) which describes when it is allowed to fire. (It actually determines on which beam pulses the modulator will receive a standby trigger.) The new EVG interface will have similar PVs, one for each beam code that describes on which beam pulses that beam code can be issued for any rate in the presently active rate groups. (We'll call this the beamcode rate mask.) For starters we will allow only four beam codes (1-4) and that number (4) will be in a PV. It should require only a parameter change and recreation of the DB to increase the number of beam codes. The PV's containing this rate information for the klystrons are in the format described in appendix A and we suggest that the new EVG code use the same format. Given the information in the TMSKs, here are the actions the EVG user interface code must do before making a rate group active:

a. Check what are the beam codes in this file
b. Check what klystrons in any sector are activated on those beamcodes
c. For all Klystrons in b), check beamcode rate mask/pattern vs. Klys:LIxx:x:TMSK mask. (KLYS:LIxx:x:TMSK sets a mask that the PIOP uses to choose which of the 360 hz pulses on upper backplane channel 0 (a PDU REUS channel) should be taken as standby pulses for the modulator.)
d. If any bit from beamcode rate mask is not also in KLYS TMSK, then deny file load (and send error message indicating which KLYS does not fit).

## Appendix A: Format of TMSK

For the klystrons, the format of TMSK has been directly taken over from what was used in the SLC control system. It consists of 2 32 bit words. 36 of these bits represent 36 beam pulses and have a 1 in any location where the klystron will fire. In use, the 36 bit pattern is repeated 10 times each second and determines on which pulse the klystron will fire for each of the 10*36=360 pulses per second. With this representation, a klystron can only be set to fire at 0, 10, 20 …360 Hz. Another six bits in the TMSK tell how many of the other 36 bits are set. Taking this six bit number and multiplying by 10 gives number of times per second at which the klystron fires.

The examples given below for the SLC control system should help clarify the format. Mike Zelazny can provide the names that have been chosen for the Klystron TMSK EPICS PVs.

```
MCCDEV> type UDSLC1:[mws]trbrtmsk.txt
!       :TMSK: = 00000001,01000000;   ! 10 Hz at 1st TS 1
!       :TMSK: = 00000002,01000000;   ! 10 Hz at 1st TS 2
!       :TMSK: = 00000004,01000000;   ! 10 Hz at 1st TS 3
!       :TMSK: = 00000008,01000000;   ! 10 Hz at 1st TS 4
!       :TMSK: = 00000010,01000000;   ! 10 Hz at 1st TS 5
!       :TMSK: = 00000020,01000000;   ! 10 Hz at 1st TS 6
!
!       :TMSK: = 00000040,01000000;   ! 10 Hz at 2nd TS 1
!       :TMSK: = 00000080,01000000;   ! 10 Hz at 2nd TS 2
!       :TMSK: = 00000100,01000000;   ! 10 Hz at 2nd TS 3
!       :TMSK: = 00000200,01000000;   ! 10 Hz at 2nd TS 4
!       :TMSK: = 00000400,01000000;   ! 10 Hz at 2nd TS 5
!       :TMSK: = 00000800,01000000;   ! 10 Hz at 2nd TS 6
!
!       :TMSK: = 00001000,01000000;   ! 10 Hz at 3rd TS 1
!       :TMSK: = 00002000,01000000;   ! 10 Hz at 3rd TS 2
!       :TMSK: = 00004000,01000000;   ! 10 Hz at 3rd TS 3
!       :TMSK: = 00008000,01000000;   ! 10 Hz at 3rd TS 4
!       :TMSK: = 00010000,01000000;   ! 10 Hz at 3rd TS 5
!       :TMSK: = 00020000,01000000;   ! 10 Hz at 3rd TS 6
!
!       :TMSK: = 00040000,01000000;   ! 10 Hz at 4th TS 1
!       :TMSK: = 00080000,01000000;   ! 10 Hz at 4th TS 2
!       :TMSK: = 00100000,01000000;   ! 10 Hz at 4th TS 3
!       :TMSK: = 00200000,01000000;   ! 10 Hz at 4th TS 4
!       :TMSK: = 00400000,01000000;   ! 10 Hz at 4th TS 5
!       :TMSK: = 00800000,01000000;   ! 10 Hz at 4th TS 6
!
!       :TMSK: = 01000000,01000000;   ! 10 Hz at 5th TS 1
!       :TMSK: = 02000000,01000000;   ! 10 Hz at 5th TS 2
!       :TMSK: = 04000000,01000000;   ! 10 Hz at 5th TS 3
!       :TMSK: = 08000000,01000000;   ! 10 Hz at 5th TS 4
!       :TMSK: = 10000000,01000000;   ! 10 Hz at 5th TS 5
!       :TMSK: = 20000000,01000000;   ! 10 Hz at 5th TS 6
!
!       :TMSK: = 40000000,01000000;   ! 10 Hz at 6th TS 1
!       :TMSK: = 80000000,01000000;   ! 10 Hz at 6th TS 2
!       :TMSK: = 00000000,01000001;   ! 10 Hz at 6th TS 3
!       :TMSK: = 00000000,01000002;   ! 10 Hz at 6th TS 4
!       :TMSK: = 00000000,01000004;   ! 10 Hz at 6th TS 5
!       :TMSK: = 00000000,01000008;   ! 10 Hz at 6th TS 6
!
!       :TMSK: = 01001001,03000000;   ! 30 Hz at 1st TS 1
!       :TMSK: = 02002002,03000000;   ! 30 Hz at 1st TS 2
!       :TMSK: = 04004004,03000000;   ! 30 Hz at 1st TS 3
!       :TMSK: = 08008008,03000000;   ! 30 Hz at 1st TS 4
!       :TMSK: = 10010010,03000000;   ! 30 Hz at 1st TS 5
!       :TMSK: = 20020020,03000000;   ! 30 Hz at 1st TS 6
!
!       :TMSK: = 40040040,03000000;   ! 30 Hz at 2nd TS 1
!       :TMSK: = 80080080,03000000;   ! 30 Hz at 2nd TS 2
!       :TMSK: = 00100100,03000001;   ! 30 Hz at 2nd TS 3
!       :TMSK: = 00200200,03000002;   ! 30 Hz at 2nd TS 4
!       :TMSK: = 00400400,03000004;   ! 30 Hz at 2nd TS 5
!       :TMSK: = 00800800,03000008;   ! 30 Hz at 2nd TS 6
!
!       :TMSK: = 41041041,06000000;   ! 60 Hz at TS 1
!       :TMSK: = 82082082,06000000;   ! 60 Hz at TS 2
!       :TMSK: = 04104104,06000001;   ! 60 Hz at TS 3
!       :TMSK: = 08208208,06000002;   ! 60 Hz at TS 4
!       :TMSK: = 10410410,06000004;   ! 60 Hz at TS 5
!       :TMSK: = 20820820,06000008;   ! 60 Hz at TS 6
!
```

```
!       :TMSK: = 49249249,0C000002;   ! 120 Hz at TS 1,4
!       :TMSK: = 92492492,0C000004;   ! 120 Hz at TS 2,5
!       :TMSK: = 24924924,0C000009;   ! 120 Hz at TS 3,6
!
!       :TMSK: = AAAAAAAA,1200000A;   ! 180 Hz at TS 1,3,5
!       :TMSK: = 55555555,12000005;   ! 180 Hz at TS 2,4,6
!
!       :TMSK: = FFFFFFFF,2400000F;   ! 360 Hz
!
!Note: The second word must have 5 zeros as follows:
!       :TMSK: = xxxxxxxx,xx00000x;
!
!Note: The first word must have a zero preceding any letter (as in
!      bigger than 9 in hex) in the left (8th from the right) column
!      as follows:
!       :TMSK: = 0Cxxxxxxx,xx00000x;
!
!Note: The second word must have the number of bits set via the other
!      fields in the leftmost 2 columns of the second word as follows:
!       :TMSK: = xxxxxxxx,NNxxxxxx;
!
!      (remember, each bit set represents 10hz, so this is the rate
!      divided by 10 in hex.)
```