

TEL2 Vacuum Interlock

Local application

Thu, Dec 21, 2006

A system of providing data for Tevatron vacuum that is needed for interlocking Tevatron electron lens beam is interfaced via the CEC protocol. A local application is needed to support this connection. It may be called TELV. (Another local application that used this same protocol for access to correction magnet power supply voltages was named CECV.) This note describes how TELV can work. A document written by Greg Saewert, called *TEL2 Vacuum Interlock Chassis*, describes the hardware interface for the related data.

The data accessed via CEC using TypeCode 0 consists of an array of 5 (including one spare) vacuum readings that are scaled as 0000–FFFF representing 0 to 10.24 volts.

An array of 10 analog settings of window comparator limits is accessed via TypeCode 1. Each of the 5 vacuum readings uses two limits, an upper and a lower limit, in that order. The scaling matches that used for the vacuum readings. These limits can be read or set, either all at once or individually. Note that setting limit values is accomplished via TypeCode 3.

An array of 5 status words, one per vacuum channel, is accessed via TypeCode 2. Each status word has the following bit usage:

<i>Bit#</i>	<i>Meaning</i>
0	Outside window HI (1=outside, 0=inside)
1	Outside window LO (1=outside, 0=inside)
2	First to trip (1=tripped first, 0=no trip or not first to trip)
3–15	spare

A single control word is accessed via TypeCode 4. It is used for resetting the trip latches. Only Bit# 0 of the word is used, where 1=reset latches for all channels, 0=do nothing.

Routine monitoring

The LA routinely reads data from the hardware, including the vacuum readings, the limits, and the status words. It will keep only a single request active at a time. The analog readings and the limits are assigned to local analog channels. The status readings are also assigned to analog channels, but they are established as status words. The BADDR table will have addresses of the bytes of the reading words, so that the status data becomes part of the binary data pool.

Setting of limits

The limits are accessible individually, and each is a separate Acnet device. (They have nothing to do with normal front end monitoring for alarm conditions, but are only used by the interlock hardware.) The CEC protocol allows access to each limit separately using the initial-element and number-of-elements fields. The local application will routinely monitor the current settings (in the ADATA table) of these limit “dummy” channels. (A setting to a dummy channel merely updates the setting word in the ADATA entry; it does not talk to hardware.) It maintains a reference set of limit settings internally, called the “standard” set here.) If there is a change, it implies that a user is trying to modify the limit value, so the new limit value must be passed to the hardware via the CEC protocol. Let there be a queue of such waiting limit changes. Each time the normal data acquisition completes, the queue is checked for pending settings, and if there is one, it is removed from the queue and a setting message is passed to the hardware box.

External access to limits

Since any computer can use the same CEC protocol to access the interlock hardware, we can

ask whether to allow other computers to make settings to the limit values. Normally, we would not permit this at all, since the front end keeps nonvolatile copies of all settings, which are used following a front end reset to remind one of the hardware setting values, since it normally does not have nonvolatile memory itself. But in this case, the hardware includes a nonvolatile flash memory that is used to maintain the latest limit settings in use. But if we are careful, we should be able to permit such settings, at least most of the time.

The limits are assigned dummy channels in the front end system. The job of passing the new setting value to the hardware rests with the LA. The LA can detect the need for such by looking for changed values in the nonvolatile setting fields. When it sees a changed value, it installs/updates an entry in a queue of waiting settings. It also updates the “standard” value so it won’t continue to notice the same change.

When a reply is received from the hardware in response to the routine request for limit values, the values received are compared against the standard set. If any is different, both the the standard and nonvolatile records are updated. (This alerts the front end of a change in a limit that was presumably made by another computer.) Then a scan is made for changes in the ADATA settings against the standard set. Again, if a change is detected, a record of intent is placed into the queue, and the standard set updated. The last step is to check the queue. If an entry is present, remove it and send out the appropriate setting. When the acknowledgment is returned from the setting, or if no settings had been sent, continue the routine monitoring.

Initialization

When the front end resets, it delivers the nonvolatile ADATA table settings to all analog channels in order to update the hardware that may have just powered up. As stated earlier, the limit channels are dummy settings to the front end. The interlock hardware has flash memory that holds the current settings of all limits. When the LA is initialized, it should defer sending any settings to the hardware until it has collected the set of limits from the hardware and assigned them as the standard set and also updated the ADATA set. Only after this should the scanning of changes seen via the ADATA entries commence. This scheme allows other computers the chance to make changes in the limit settings whether TELV is running or not, and such changes will not be altered when the front end resets, or when TELV is initialized.

Reset control

There is one control bit that can be set to reset the latches in the hardware. When a user decides to perform such a reset, a dummy control bit is set. Again, the LA will have to notice this bit set and queue up the appropriate setting for the hardware. The dummy bit will be reset so the next scan will not see the bit still set and queue another setting. Nothing need be done to monitor whether another computer performs such a reset.

Retries

When sending setting commands to the hardware, there is an acknowledgment expected. If it is not received in a suitable time, another attempt should be made to deliver it before giving up. Setting the same limit twice, in the case that the setting was successful but the acknowledgment was lost, will not hurt anything. Even if a limit setting fails, the standard set of values is updated, so the front end will not continue to send such settings forever. Again, if the hardware were simply inaccessible or off, when it comes back up, the front end will first obtain the hardware version of the limits to “get up to speed” before allowing any changes to be made. The retry is used only for an unlikely network failure.

LA parameters

Here is the parameter layout for TELV:

Prompt		Size	Meaning
ENABLE	B	2	Enable Bit# for this LA
RESET	B	2	Reset control Bit# (TypeCode 4)
VACUUM	C	2	Chan 1 vacuum Chan# (TypeCode 0)
NVAC		2	#vacuum channels
LIMIT	C	2	Chan 1 upper limit Chan# (TypeCode 1)
STATUS	C	2	Chan 1 status Chan# (TypeCode 2)
IP ADDR		4	Target IP address

The number of limit channels is twice the number of vacuum channels, in pairs of (upper, lower). The number of status channels is the same as the number of vacuum channels. This means we have two spare words of parameters.

Post-implementation notes

The new LA TELV was written in C to run in the 68K-based IRM. It was organized according to the above plan. It is installed in two nodes: 067D (TEL1) and 067B (TEL2), as described in Greg Saewert's *TEL Vacuum Interlock Chassis* document (revised 12/18/06). Along the way, it was found that the acknowledgment, to a setting of the limits using TypeCode 3, is the same message returned, including the data word that was sent. (Requests to read these same limit data use TypeCode 1.) See the document by Greg Saewert and Brian Kramper, called *The Compact Ethernet Communication (CEC) Protocol*.

The hardware uses flash memory to keep the latest limit setting values, so that it comes up after power-on already using those values. But we learned that it takes about 700 ms to write to this memory, during which time the hardware is *incommunicado*, although its vacuum digitizing and limits checking logic continues to operate at 5 KHz via its FPGA. The software takes this into account by inserting a 2 second delay after sending a setting message, which the hardware acknowledges promptly with the same data value before starting its slow write access to flash memory. The LA processes this reflected data so that the act of making such a setting appears promptly to the user.

Typical responses to data requests vary from 5–15 ms, making it easy to operate at 15 Hz. The routine monitoring activity requests and receives vacuum readings on one cycle, limit values on the next cycle, and status readings on the following cycle. A setting queue is used to set aside setting commands until completion of the latest 3-cycle series of data requests. If the queue is not empty, an extra cycle is inserted in the sequence in order to send the setting command to the hardware; otherwise, the sequence repeats. The data is thus regularly updated at about 5 Hz.

Dummy channels are used to hold the vacuum, limit, and status data returned from the hardware. Setting such a limit channel is noticed by TELV, causing it to queue up a setting. Provision for another computer to set a limit is made by accepting a newly changed limit value as one that should be adopted by the LA and the ADATA fields. Also, when the LA is initialized, the first set of limits obtained from the hardware is accepted as the standard set.

A combined binary status word is built (for the convenience of Acnet) that includes the 3 status bits from each of the up-to-5 vacuum devices.

Both setting actions and occurrences of errors are logged internally by the LA as diagnostic aids. The LA currently uses 600 lines of source code and occupies about 4K bytes of executable code.