

Floating Point Analog Channels

Data without scaling

Feb 10, 2000

Current support for floating point data from an IRM node is based upon underlying raw 16-bit values for readings, settings nominal and tolerance values. This note explores possible approaches to support of raw data that is already in floating point form. It can be computed in a local application, or it can be found in shared memory that is maintained by another CPU.

Channel#

Should these data use up channel# space that is currently used by the ADATA and ADESC tables? If so, then there must be an additional form of ADATA that is parallel to the raw ADATA table, perhaps a new FDATA table. As an example, it might use 16-byte entries that can house 4 floating point values, which can provide values for reading, setting, nominal and tolerance. The alarm flags and trip count could be maintained in the associated ADATA entries, but the usual raw data fields in those entries would not be used. The alarm scan task would have to notice such cases during its analog scan, perhaps by a new bit in the alarm flags, or in the CONV byte of the ADESC entry. In the latter case, the Alarm scan logic would have to be able to find the parallel table entries so it can do its alarm scanning using floating point operations.

Another approach could be to use a different index besides a channel number. In this case an FDATA table would not be parallel with ADATA. Its size could be only large enough to house the number of floating point channels used in a node. This could fit in easily for RETDAT, because the listype would be different, as well as the index# being different. But for this case, the alarm flags would need to be kept along with the new table entries, so that 16-byte entries would not be large enough. If we had 32-byte entries, there would be more than enough space for the 4 floating point values (16 bytes), the alarm flags, the trip count, a motor countdown if needed, and a captured floating point reading if needed. In this case, however, how can we deal with the need for analog descriptors? Let's assume an FDATA table parallel to ADATA. For certain channels, marked via a flag bit in the alarm flags, FDATA values would be valid, but the corresponding ADATA values would not.

If desired, listypes 40–44 could be used to access floating point values for the special floating point channels. The code would need to check the flag bit to see whether it should access the floating point fields from FDATA, or as is done now, scale the values found in ADATA.

Alarm scanning

Check the special alarm flag bit that indicates the floating point case. If it is set, get a pointer to the corresponding fields in FDATA and perform the alarm checking logic using floating point operations.

When generating the Classic protocol alarm message output, find some way to include the floating point value, perhaps by overwriting the two words normally used for the raw reading and setting words. Whether it is important to also include the nominal and/or the tolerance values is debatable. Normally, only the reading word is included in the encoded alarm message. The alarm encoding will need to be changed accordingly to reflect the new format of analog alarm message.

Data access table

We will need to define a new type of entry suitable for copying floating point values from memory into the new FDATA reading fields. There may be additional entry types needed

for the purpose of forming new floating point values.

Local applications

For local applications that compute values to be assigned to analog channels, another routine will need to be written to simply copy an array of floating point values into the reading fields of consecutive FDATA entries.

Settings

A new analog control type is needed to perform settings of these floating point values. In this case, it is unlikely that hardware will get the values. Also, the setting values will have to be floating point 4-byte values. Currently, there are only 3 bytes to specify the target for the settings. If the target is merely memory for another CPU to find, it must be specified in some abbreviated fashion, not as a 32-bit address. For the other CPU case, the use of a command queue may be useful. There already is support for such queues for other CPUs.

Analog settings sent to a co-processor queue use the AUX byte in the ADESC entry analog control field to specify a 3-bit co-processor number, leaving 5 bits to specify flags for the co-processor to decode. The remaining 16 bits may be an index number to specify which thing is being controlled, again, something to be decoded by the co-processor.

Digital settings only have what is usually a memory pointer in the BADDR table, which has such an entry for each byte. But special values of this pointer are interpreted in a way that is not truly a memory address. Currently, high byte values of 0x80, 0x81, 0x82 mean special forms of encoded information to cover 1553, SRM, and PLC digital settings. It is proposed to use a high byte value of 0x83 to specify co-processor digital control. The other 3 bytes can be formatted similarly to the analog control specification. A 3-bit field gives the co-processor number, the remaining 5 bits of that byte specify other type info for the co-processor, and the other 16 bits are an index value that has meaning to the co-processor in identifying the target of the digital control setting.

Summary

For reading data, define a new data access table type that allows copying of memory to FDATA reading fields. For setting data use the co-processor interface that uses a simple queuing scheme for sending messages to the co-processor. Alarm scanning is done by noticing the bit flag in the alarm flags word that indicates that raw data is only in floating point format in FDATA entries, not in the usual ADATA fields.

FTPMAN

This is another topic to be addressed carefully. For continuous fast time plots, the time stamps are returned with a 2-byte or 4-byte data value. In this floating point case, 4-byte data would have to be provided in reply to a request. Some means of finding the data for a given channel is necessary, so that it can be copied to satisfy the request.

For snapshots, the time stamps need not be specified. To plot relative to a clock event plus delay, we must work out how this data can be collected. It may not be so easy, but FTPM already supports a number of types of digitizer interfaces, so this is just one more.