

NAME

dxp – XIA DXP CAMAC MCA

DESCRIPTION

The X-ray Instrumentation Associates Digital X-ray processors (CAMAC models DXP 4C and DXP 4C2X and parallel port model X10P) have one to four MCA channels per module. Each channel of the device contains a digital signal processor (DSP). The DSP chips require firmware and parameter downloads for initialization.

As of spec release 4.05.10-8, spec must be linked with a special vendor-supplied application library to use the DXP modules. The XIA libraries have not yet stabilized, so one must be sure to use a version of the libraries compatible with the spec release. Contact CSS for advice on determining compatible versions.

In addition, the many XIA configuration files that will be read by routines in the DXP library must be installed on the system. Finally, the environment variable XIA_CONFIG must be set to the location of DXP system configuration file.

The source files for creating the DXP library can be obtained from the XIA Web page at <http://www.xia.com>, or in a slightly more convenient format at ftp://ftp.cer.tif.com/pub/misc/xia_dxp_css.tgz. The latter includes a *Makefile* to build a library called *libdxp.a*.

When running the spec *Install* script, you must indicate the location of the *libdxp.a* library in response to the “Extra library flags” query from the *Install* script. Use, for example */usr/local/xia/libdxp.a* or *-L/usr/local/xia -ldxp*.

The many XIA configuration files associated with the DXP modules need to be carefully constructed to avoid the code in the DXP library from failing. Please consult XIA documentation or XIA technical support for details related to the format of these files.

Note, the CAMAC slot numbers listed in the *dxp.module* (or equivalent) file must agree with those in the spec hardware *config* file. For the parallel port device, the hexadecimal base address of the I/O port listed in the *dxp.module* file must also agree with the value set in the spec *config* file.

FUNCTIONS

Since each DXP module can have up to four channels, the usual numbering scheme in spec for selecting MCA devices has been extended. For commands that require access to individual channels beyond channel zero, the *mca_spar()* and *mca_sget()* must be used (instead of *mca_par()* and *mca_get()*) where the first argument (that selects which MCA device) is a string. The string contains the number of the selected MCA, followed by a colon and a DXP channel number from 0 to 3. For the commands to set the DSP parameters, if the channel number is -1, the parameter will be set in all DXP modules. For example, *mca_spar("1:3", "finegain", 100)* would set the *finegain* parameter to the value 100 for MCA number 1, DXP channel 3, while *mca_spar("0:-1", "finegain", 100)* would set the *finegain* parameter in all channels in all DXP modules, not just those associated with module 0.

For commands that act on all channels, such as "run", assuming that MCA 0 and MCA 1 are DXP modules, the commands *mca_par("run")*, *mca_spar(1, "run")*, *mca_spar("0:3", "run")* and *mca_spar("0:-1", "run")* would each start all channels in all DXP modules.

Besides reading the spectrum data, the *mca_sget()* function can read the baseline values or the DSP parameters. Again, the first argument is a string, formatted as described above, but with 16 (0x10) added to the channel number to read back the baseline or 32 (0x20) added to the channel number to read back the parameters. For example,

```

ushort array pars[1024]
mca_sget("l:0x21", pars)

```

reads the current parameters into the `pars[]` array. Note, the array must be dimensioned to full size of the DXP data array, even though the parameter and baseline arrays are smaller (150 and 512 values, respectively).

Listed below are the functions available for the DXP device.

`mca_par("?")` – Lists the available commands, including the DSP parameters that can be individually examined or set.

`mca_par("disable" [, arg])` – With no arguments, returns nonzero if the selected MCA device has been disabled by the user, otherwise returns zero. If `arg` is 1, disables the MCA. If `arg` is 0, turns off the disabled mode. When the device is disabled, `spec` will not access the hardware. On startup, and after the standard `config` macro or the `reconfig` command is run, disabled mode is off.

`mca_par("auto_run" [, arg])` – With no arguments, returns nonzero if the selected MCA device has auto-run mode set, otherwise returns zero. If `arg` is 1, enables auto-run mode. If `arg` is 0, turns off auto-run mode. When auto-run mode is set the device is started and stopped with the counting functions `tcount()`, `mcoun()`, etc. When not set, the counting functions are ignored, but the device can be controlled with the "run" and "halt" options described below. In addition, the device can be halted with the `stop()` function and will be halted with `^C`. On startup, and after the standard `config` macro or the `reconfig` command is run, auto-run mode is on.

`mca_par("run")` – Starts acquisition.

`mca_par("halt")` – Halts the MCA device.

`mca_par("save_config", filename)` – Saves the current parameter configuration to the file `filename` using the DXP library routine `dxp_save_config()`.

`mca_par("restore_config", filename)` – Loads parameters from the DXP configuration file `filename` using the DXP library routine `dxp_restore_config()`.

`mca_par("calib_trkdac")` – Calls the `dxp_calibrate()` function with the TRKDAC argument.

`mca_par("calib_reset")` – Calls the `dxp_calibrate()` function with the RESET argument.

`mca_par("gate" [, arg])` – The DXP modules have an external gate capability. If the optional argument `arg` is nonzero, `spec` will program the modules to obey the external gate signal. Otherwise, the gate will be ignored. With no arguments, the current value of the gate parameter is returned.

`mca_par("modify_one_gain", factor)` – Scales the DXP amplifier gain for a single channel by `factor` by a call of the `dxp_modify_one_gains()` routine.

`mca_par("modify_gains", factor)` – Scales the DXP amplifier gains by `factor` by a call of the `dxp_modify_gains()` routine.

`mca_par("dump")` – Calls the DXP library routine `dxp_mem_dump()`, which prints the current values of all the DXP parameters.

`mca_par("show_files")` – Displays the complete path names of the three configuration files associated with the DXP system and the three configuration files associated with the designated channel as provided by the DXP library routines `dxp_locate_system_files()` and `dxp_locate_channel_files()`.

`mca_par("icr")` – Returns the input count rate.

`mca_par("ocr")` – Returns the output count rate.

`mca_par("real")` and `mca_par("elapsed_real")` – Returns the elapsed real time.

`mca_par("live")` and `mca_par("elapsed_live")` – Returns the elapsed live time.

`mca_par("dead")` – Return the percent dead time.

`mca_par(any_dxp_par [, value])` – Retrieves or sets (if *value* is given) any of the DXP parameters, such as `errinfo` or `livetime`. The parameter name can be given in either lower or upper case. Note, with long integer parameters such as `overflows` that are labeled as `OVERFLOWS0` and `OVERFLOWS1` in the DXP configuration files, the parameter name without the trailing digit will retrieve or set the long integer value in one call of `mca_par()`, although the DXP name can also be used to set the individual 16-bit portions of the long integers. Type `mca_par("?")` to get a list of all the parameters.