

**NAME**

MPII – Canberra Multiport II USB/Socket MCA

**INTRODUCTION**

The Canberra Multiport II (MPII) is a NIM module containing from one to six independent multichannel analyzer (MCA) devices. The MPII can be used with `spec` over both socket and USB interfaces. Each MCA device can be configured for up to 16,384 channels and can be operated in either pulse-height analysis (PHA) or multichannel scaling (MCS) mode.

**CONFIGURATION**

Each MPII module interfaced to `spec` over a socket interface can be uniquely identified by its assigned IP address. USB units cannot be assigned addresses, so if more than one MPII is used with `spec` over USB, assignment of specific modules is done by including the module's unique serial number in `spec`'s hardware `config` file. If the serial number is entered as zero, `spec` will simply assign the MCA units in the order in which they are found.

In the `edconf` configuration editor, possible MPII configurations include:

MCA- and CCD-Acquisition Type Device Configuration

MCA-like	DEVICE	ADDR	<>MODE	<>TYPE
0 YES	host1:6000			Canberra Multiport II (Socket)
1 YES	192.168.1.5			Canberra Multiport II (Socket)
2 YES		12345		Canberra Multiport II (USB)
3 YES		0		Canberra Multiport II (USB)
4 YES		0		Canberra Multiport II (USB)

The first entry configures a socket-based MCA using a resolvable host name and an explicit port number. Note, the default port for the MPII is 6000, and doesn't need to be specified. The second entry specifies an IP address without the optional port number. Three USB configurations follow, the first with an explicit module serial number. The last two entries are also for USB units, but with the serial number set to zero. For those two units, it won't be possible to know in advance which MPII will be assigned to MCA unit number 3 and which to MCA unit number 4.

**ADDRESSING**

The `spec mca_get()`, `mca_put()` and `mca_par()` functions work with the default MCA unit. If there is only one MCA unit configured, that unit is the default. Otherwise, the `mca_sel()` function can be used to select the default unit number. The `mca_spar()`, `mca_sget()` and `mca_sput()` take an initial argument that specifies a specific MCA unit number, independent of the default MCA unit number. See the `mca` help file for more details.

Because each MPII can have up to 6 independent MCA devices, the MCA selection syntax for the commands `mca_spar()`, `mca_sget()`, and `mca_sput()` is expanded. To select a specific MCA device on an MPII, address arguments may be given as `0.1`, `0.2`, ..., `0.6`, where the first digit is the MCA unit number from the `config` file, and the digit right of the decimal point is the MCA device, where the devices are numbered from 1 to 6. The syntax `"0:1"`, `"0:2"`, ..., `"0:6"` is also recognized, where the arguments are strings.

For writing data to the MCA with `mca_sput()` and for setting parameter values with `mca_spar()`, if no subaddress is specified, the command will apply to all of the up-to-six MCA devices available. That is, the same data or parameters can be written to all MCA devices on a Canberra module with one command. For reading data or reading parameter values, only one MCA device can be accessed since only one set of data or a single value can be returned. Thus, a subaddress needs to be specified for commands that return data if there is more than one MCA device on the MPII unit.

If no subaddress is specified for any of the commands, subaddress 1 is assumed. If the subaddress doesn't specify a valid MCA device for any command, subaddress 1 is still assumed. If

there is only one MCA device on the unit, no subaddress is necessary. If the MPIO is the default MCA and there is only one MCA device, `mca_get()`, `mca_put()` and `mca_par()` functions can be used unambiguously. Although the `mca_sel()` function can select the default unit, it is not possible to assign a default subaddress.

## FUNCTIONS

The standard MCA commands supported by `spec` are described in the `mca` help file. Refer to that document for more details on usage of the `mca_get()`, `mca_sget()`, `mca_put()` and `mca_sput()` functions. Also, refer to the `mca` help file for a description of the standard `mca_par()` and `mca_spar()` options "info", "chans", "max\_chans", "disable", "auto\_run", "soft\_preset", "auto\_clear" and "native\_type".

The default settings for the MPIO are for "auto\_run" and "soft\_preset" to be disabled and "auto\_clear" to be enabled. These default settings are restored on start up and hardware reconfiguration. When "auto\_run" and "soft\_preset" modes are in effect, the available MPIO devices will be programmed with a time preset when counting to time and with the gross-counts preset when counting to monitor. When counting to time, the mode will be real-time or live-time depending on whether the MPIO is in real- or live-preset mode. That mode is set by using `mca_par()` with "preset\_live" or "preset\_real".

Of the preset modes, real, live and gross counts, only one can be active at a time.

Note, of the parameters described below, the following are saved in the state file and will maintain values between `spec` invocations: "xstart\_enable", "xstop\_enable", "mcs\_xsync", "mcs\_start\_chan", "mcs\_stop\_chan", "overflow\_enable", "preset\_passes", "preset\_real", "preset\_live" and "preset\_counts". When starting fresh, values will be restored to defaults. All other MPIO parameters are retained in the module's flash memory, and `spec` will use those values, including the values for "npts" (or "adc\_gain").

Additional options for the MPIO are described below. The string arguments are case insensitive.

### Basic Operation

`mca_par("clear")` – Clears the current data and the elapsed real-time, live-time, gross-counts and sweep-counts registers.

`mca_par("run")` – Sets the current acquisition parameters and starts acquisition. If "auto\_clear" mode is set, clears the existing data first. Otherwise, if the device hasn't been explicitly cleared, new data will be added to the existing.

`mca_par("halt")` – Halts acquisition.

`mca_par("pha")` – Sets pulse-height analysis mode.

`mca_par("npts" [, value])` – With no arguments, returns the current number of points. Otherwise, sets the number of points by setting the ADC gain to the *value* specified. Synonymous with "adc\_gain". Allowed values are 256, 512, 1024, 2048, 4096, 8192 and 16384.

`mca_par("preset_real" [, value])` – With no arguments, returns the current PHA-mode real (true) time preset in seconds. Otherwise, sets the real-time preset to *value*.

`mca_par("preset_live" [, value])` – With no arguments, returns the current PHA-mode live-time preset in seconds. Otherwise, sets the live-time preset to *value*.

`mca_par("preset_counts" [, value])` – With no arguments, returns the current PHA-mode gross-counts preset. Otherwise, sets the gross-counts preset to *value*.

`mca_par("overflow_enable" [, value])` – If no arguments, returns 0 if channel overflows are ignored and 1 if channel overflows halt acquisition. Otherwise sets the

overflow-enable mode according to *value*.

`mca_par("xstart_enable" [, value])` – With no arguments, returns nonzero if the XINPUT signal is to be used to start acquisition. Otherwise sets external-start enable mode according to *value*.

`mca_par("xstop_enable" [, value])` – With no arguments, returns nonzero if the XINPUT signal is to be used to stop acquisition. Otherwise sets external-stop enable mode according to *value*.

`mca_par("save_interval" [, value])` – With no arguments, returns the interval in minutes at which the Multport II saves its parameters to built-in flash memory. Otherwise sets the interval as specified by *value*. Allowed values are from 1 to 60 minutes.

### Status

`mca_par("dump")` – Displays all the parameters and settings.

`mca_par("elapsed_real")` – Returns the elapsed real (or true) time in seconds.

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

`mca_par("elapsed_counts")` – Returns the total elapsed counts.

`mca_par("start_time")` – Returns the starting time of the most recent acquisition in seconds since January 1, 1970, 00:00:00 GMT. The value can be converted to an ASCII string using `SPEC's date()` function.

`mca_par("stop_time")` – Returns the stopping time of the most recent acquisition in seconds since January 1, 1970, 00:00:00 GMT. The value can be converted to an ASCII string using `SPEC's date()` function.

`mca_par("stop_event")` – Returns a code with bits indicating the type of event that halted the most recent acquisition as follows:

- 0x00 None
- 0x01 Real-time preset reached
- 0x02 Live-time preset reached
- 0x04 Total-counts preset reached
- 0x08 External input
- 0x10 MCS-preset passes reached
- 0x20 Power lost
- 0x40 Channel overflow

`mca_par("overflow_chan")` – If acquisition was stopped by an overflow, returns the channel number that overflowed.

`mca_par("suspend_active")` – Returns nonzero if the XSUSPEND input is active, otherwise returns zero.

`mca_par("collecting")` – Returns nonzero if the MCA device is busy collecting data.

`mca_par("waiting_xinput")` – Returns nonzero if the MCA is waiting for the XINPUT to become active.

`mca_par("start_delay")` – Returns the internal delay time in seconds for the MCA to begin the most recent acquisition. This is a measure of latency within the MCA device.

`mca_par("changer_state")` – Returns the sample-changer state, where nonzero indicates the sample changer is active.

### External Connection Configuration

`mca_par("xcoinc_polarity" [, "low"|"high"|0|1])` – With no arguments, returns 0 (for active low) or 1 (for active high) to indicate the polarity of the XCOINC input signal. Otherwise, sets the polarity.

`mca_par("xdt_polarity" [, "low"|"high"|0|1])` – With no arguments, returns 0 (for active low) or 1 (for active high) to indicate the polarity of the XDT pileup-reject dead-time input signal. Otherwise, sets the polarity.

`mca_par("xlg_polarity" [, "low"|"high"|0|1])` – With no arguments, returns 0 (for active low) or 1 (for active high) to indicate the polarity of the XLG pileup-reject linear-gate output signal, which is active when the ADC acquires an input pulse. Otherwise, sets the polarity.

`mca_par("xrej_polarity" [, "low"|"high"|0|1])` – With no arguments, returns 0 (for active low) or 1 (for active high) to indicate the polarity of the XREJ pileup-reject input signal. Otherwise, sets the polarity.

`mca_par("xcollectstatus_polarity" [, "low"|"high"|0|1])` – With no arguments, returns 0 (for active low) or 1 (for active high) to indicate the polarity of the XCOLLECTSTATUS external acquire status output signal. Otherwise, sets the polarity.

`mca_par("xinput_polarity" [, "low"|"high"|0|1])` – With no arguments, returns 0 (for active low) or 1 (for active high) to indicate the polarity of the XINPUT external start/stop and sample-changer ready input signal. Otherwise, sets the polarity.

`mca_par("xoutput_polarity" [, "low"|"high"|0|1])` – With no arguments, returns 0 (for active low) or 1 (for active high) to indicate the polarity of the XOUPUT advance-sample-changer output signal. Otherwise, sets the polarity.

### ADC Configuration

`mca_par("adc_gain" [, value])` – With no arguments, returns the ADC gain. Otherwise, sets the ADC gain (and the number of points) to the *value* specified. Synonymous with "npts". Allowed values are 256, 512, 1024, 2048, 4096, 8192 and 16384.

`mca_par("adc_LLD" [, value])` – With no arguments, returns the current setting of the low-level discriminator as a percent. Otherwise, set the low-level discriminator to the *value* given as a percent from 0 to 100.

`mca_par("adc_ULD" [, value])` – With no arguments, returns the current setting of the upper-level discriminator as a percent. Otherwise, set the upper-level discriminator to the *value* given as percent from 0 to 110.

`mca_par("adc_zero" [, value])` – With no arguments, returns the current zero-discriminator setting as a percent. Otherwise, sets the zero discriminator to the *value* given as a percent from -2.5 to 2.5.

`mca_par("adc_coincidence" [, value])` – With no arguments, returns the current ADC coincidence mode, where 0 indicates normal and 1 indicates anti-coincidence mode. Otherwise, sets the ADC coincidence mode as specified by *value*.

`mca_par("adc_high_pulse_action" [, value])` – If no arguments, returns zero if input pulses above the upper-level discriminator are rejected and nonzero if such pulses are added to the last channel. Otherwise sets the high-pulse action based on *value*.

`mca_par("adc_coinc_trigger" [, value])` – If no arguments, returns zero if the coincidence trigger is level sensitive and nonzero if the trigger is edge sensitive. Otherwise sets the trigger mode according to *value*.

**Multichannel Scaler**

- `mca_par("mcs")` – Set multi-channel scaling mode.
- `mca_par("dwell" [, value])` – With no arguments, returns the MCS-mode dwell time in seconds. Otherwise, sets the dwell time to *value*. A negative argument selects external dwell.
- `mca_par("mcs_start_chan" [, value])` – With no arguments, returns the start channel in MCS mode. Otherwise, sets the MCS start channel to *value*.
- `mca_par("mcs_stop_chan" [, value])` – With no arguments, returns the stop channel in MCS mode. Otherwise, sets the MCS stop channel to *value*.
- `mca_par("preset_passes" [, value])` – With no arguments, returns the current MCS-mode pass-preset value. Otherwise, sets the number of passes.
- `mca_par("elapsed_passes")` – Returns the elapsed number of passes in MCS mode.
- `mca_par("current_chan")` – In MCS mode, returns the current channel of the current sweep.
- `mca_par("elapsed_dwell")` – In MCS mode, returns the elapsed dwell of the current channel of the current sweep.
- `mca_par("mcs_xsync" [, value])` – With no arguments, returns 0 if MCS-mode external sync is disabled or 1 if enabled. Otherwise, sets the MCS external sync as specified by *value*.
- `mca_par("pulse_source" [, value])` – In MCS mode, with no arguments, returns zero if the input pulses are taken from the LLD/ULD filtered PHA input or nonzero if the input pulses are taken as TTL pulses from the XMCS PULSE input. Otherwise, sets the pulse source according to *value*.
- `mca_par("xmcsadvance_polarity" [, "low"|"high"|0|1])` – With no arguments, returns 0 (for active low) or 1 (for active high) to indicate the polarity of the XMCSADVANCE external channel advance input signal. Otherwise, sets the polarity.
- `mca_par("xmcspulse_polarity" [, "low"|"high"|0|1])` – With no arguments, returns 0 (for active low) or 1 (for active high) to indicate the polarity of the XMCS PULSE MCS input signal. Otherwise, sets the polarity.
- `mca_par("xmcsrestart_polarity" [, "low"|"high"|0|1])` – With no arguments, returns 0 (for active low) or 1 (for active high) to indicate the polarity of the XMCSRESTART MCS sweep advance input signal. Otherwise, sets the polarity.