

**NAME**

ILL 2D – ILL two-dimensional detector

**DESCRIPTION**

spec supports two-dimensional position-sensitive detectors used at the ADAM and the EVA instruments at the Institut Laue Langevin (ILL) in Grenoble, France. The support is by way of built-in C code in spec that communicates with CSS-supplied *Linux* kernel drivers. The drivers catch interrupts generated by trigger signals from the detector electronics, which also supply event addresses on the digital input lines. The driver decodes the event address and increments a corresponding element of an array that accumulates the data. There are drivers for both an ISA card (Measurement Computing CIO-PDMA16 and compatible) and a PCI card (Measurement Computing PCI-DIO24 and compatible – added in spec release 5.05.01-6).

The hardware is selected in the *config* file with the line

```
PC_ILL2D = device_name dimension @img_sel
```

where *device\_name* is the device node of the driver, as in */dev/pio*, *dimension* is the dimension of one side of the square array used to collect data and *sel* is the image-device number used as the first argument in the *image\_par()* and *image\_get()* functions below. The *dimension* argument is only supported by the PCI card. The ISA card is fixed at a 256×256 array. For the PCI card, the dimension of the array can be any power of two from 16 to 4,096. The native data type for both cards is unsigned long.

When running the configuration editor, switch to the MCA- and CCD-like device configuration screen to select the “ILL 2D Detector” as a CCD-like device.

The CSS-supplied kernel-level driver expects the external detector hardware to assert an IRQ request on pin 1 of the external connector when data is available. For the PCI card, the interrupt-enable signal on pin 2 must be pulled high. For the ISA card, the driver will acknowledge with a signal on pin 26. For the PCI card, there is no acknowledgment signal.

On each interrupt, data is read from the card’s input registers. For the ISA card, the column (x) and row (y) are taken from the A and B ports, respectively, yielding an address into the 256×256 data array:

```
x = A7 A6 A5 A4 A3 A2 A1 A0 (LSB)
y = B7 B6 B5 B4 B3 B2 B1 B0 (LSB)
```

For the PCI card, the C port adds additional bits as follows:

```
x = C3 C2 C1 C0 A7 A6 A5 A4 A3 A2 A1 A0 (LSB)
y = C7 C6 C5 C4 B7 B6 B5 B4 B3 B2 B1 B0 (LSB)
```

When the driver is configured (via software) for less than twelve bits per channel, the appropriate number of least significant bits (LSB) are ignored.

When an event is recognized, the array element at that address is incremented to indicate an event. The driver also maintains a counter of total events and total time of acquisition. There is also a feature for counting events and time over an interval.

To install the drivers, change to the *drivers* subdirectory of the spec distribution. For the ISA-card driver, type

```
./install_pio -b 0x300 -i 10
```

where the base address is chosen to match the digital I/O board’s jumpers and the IRQ is chosen not to conflict with any other devices. Type `cat /proc/ioports` and `cat /proc/interrupts` to see what resources are currently in use. For the PCI-card driver, simply type

```
./install_pio24
```

## FUNCTIONS

The `image_get()` functions retrieves the data from the driver. The `image_par()` function controls the detector interface behavior as follows. The `sel` parameter is the image-device number from the `config` file.

`image_get(sel, arr)` – Reads data into the array `arr`.

`image_par(sel, "clear")` – Clears the driver data.

`image_par(sel, "run")` – Starts data collection. Memory is not cleared.

`image_par(sel, "halt")` – Halts data collection.

`image_par(sel, "sum")` – Returns the total number of counts detected since last clear.

`image_par(sel, "elapsed_time")` – Returns the elapsed time of the run in seconds.

`image_par(sel, "rate")` – Returns the count rate in counts per second since the last call of `image_par()` with the "rate" argument.

`image_par(sel, "interval_sum")` – returns the total number of counts since the previous "rate" call.

`image_par(sel, "interval_time")` – Returns the elapsed time since the previous "rate" call.

The value returned by "rate" is, in fact, the ratio of the values returned by "interval\_sum" and "interval\_time".

The "sum", "elapsed\_time", "interval\_sum" and "interval\_time" parameters are read from the driver no more frequently than every 100 msec. That is, while acquisition is active, `spec` checks to see if it has read the parameters within the last 100 msec. If it has, it returns the last read values. Otherwise it reads all the values and saves them to be returned on subsequent calls within the 100 msec window.

## EXAMPLE

The following macros show how to obtain and display the count rate.

```
# display current count rate
def rate '{
    local r, t, s

    r = image_par(0, "rate")          # instantaneous rate
    t = image_par(0, "elapsed_time")
    s = image_par(0, "sum")
    printf("%6d in %s at %.2f0, s, hms(t), r)
}'
# convert seconds to hours:minutes:seconds
def hms(t) '{
    if (t > 0) {
        local h, m, s, d, r

        if (h = int(t/3600))
            r = sprintf("%d:", h)
        if ((m = int(t/60) - h*60) || r)
            r = r sprintf("%02d:", m)
        s=int(t) - h*3600 - m*60
        r = r sprintf("%02d", s)
```

```
        if (d = t - int(t))
            r = r sprintf("%.3d", int(d*1000))
        return(r)
    }
    return("0")
},
```