Sensoray Model 2410 Command Line Protocol

Overview

Sensoray's model 2410 is a compact module that interfaces 48 general-purpose digital I/Os (DIOs) to Ethernet. The module has a telnet server and command line interpreter which can be used by a remote client to control and monitor the DIOs. This document describes the 2410 command line protocol.

Application developers are encouraged to use Sensoray's 2410 API because it combines and encapsulates a telnet client and 2410 command processor, thereby making it unnecessary to be familiar with telnet or 2410 command line protocols.

In some cases, users do need to know details of the command line protocol. Examples of this include interacting with a 2410 manually through a telnet client such as PuTTY, or using a telnet client library such as telnetlib.py for automation. This document is intended for such users.

Startup

The 2410 issues a sign-on message similar to this when a telnet session is opened:

```
Connected to Sensoray 2410 IoServer at 192.168.24.10
Sensoray Telnet Server v.1.0.24
>
```

The message is followed by a prompt (>), which indicates the 2410 is ready to receive a command.

Commands

The telnet client is allowed to invoke a 2410 command after receiving a prompt. To invoke a command, send the appropriate command line followed by a newline. If the command produces a reply, it will be sent to the client over the telnet connection.

A special reply is sent to the client if there is a problem with the command. This special reply is always sent when there is a problem, and it takes the place of any normal reply that would otherwise be sent:

- `?command` – indicates the command is not recognized.
- `?value` – indicates an illegal argument value or invalid command syntax.

After a command has finished executing and its reply, if any, has been sent, the 2410 will send a new prompt.

Numeric values sent to the 2410 may be expressed in decimal or hex (e.g., `10000` or `0x2710`). Numeric values returned from the 2410 are always expressed in hex. Commands are not case sensitive (except for the `0x` prefix on hex values).

Commands related to event capture and asynchronous notification are not covered in this document. Please see the API documentation for further information.

By default, the 2410 will echo all command line characters sent by the client. To disable this echo function, send a telnet IAC DONT ECHO command to the 2410.

Command: **quit**

Terminates the telnet session and closes the connection.

This is the preferred method of closing the connection. Note that `quit` does not immediately close the connection; it merely causes the client to schedule a deferred close. Consequently, the connection can remain open for an indeterminate time after `quit` is sent. For this reason, it is strongly recommended that you avoid opening and closing sessions frequently, as this could momentarily hold open the maximum number of connections allowed by the 2410 and thereby cause the next open to fail. Instead, the application should open a single session and use it for all interactions with the 2410, and close the session only when the application is terminating.

Command: **ver**
Reply: *version* {**pri** | **sec**}

Show firmware information.

`version` – firmware version string.
`pri` – indicates primary firmware is running.
`sec` – indicates secondary firmware is running.

Command: **rdi**
Reply: *hval mval lval*

Read debounced DIO input states.

Every DIO channel includes a monitoring circuit through which its physical pin state can be read. This function acquires a snapshot of the pin state.

Physical states are sampled in parallel (all 48 channels are sampled simultaneously) at one millisecond intervals. Sample data are passed through a debounce filter which may cause latency or, in the case of rapidly changing physical states, undetected state changes. Consequently, the returned values will not accurately indicate the instantaneous physical states of channels that are undergoing debounce.

Three values are returned (`0x0000` to `0xFFFF`) with each bit representing one DIO state (1/0 = on/off):

`hval` – DIO[47:32]
`mval` – DIO[31:16]
`lval` – DIO[15:0]

Command: **rdo**
Reply: *hval mval lval*

Read programmed DIO output states.

This function reads the programmed states of all DIO output drivers. Note that these states may differ from those returned by `rdi` because I/O pins can be driven by external signals as well as on-board drivers. Also, in the case of DIOs operating in PWM output mode, the instantaneous driver state is determined by a PWM generator.

Three values are returned (0x0000 to 0xFFFF) with each bit representing one DIO state (1/0 = on/off):

`hval` – DIO[47:32]
`mval` – DIO[31:16]
`lval` – DIO[15:0]

Command: **rtime**
Reply: *time*

Read timestamp generator.

`time` – snapshot of the timestamp generator (`0x00000000` to `0xFFFFFFFF`).

Command: **wdo** *hval mval lval*
Reply: –

Program all DIO output states.

This function programs the output drivers of all DIOs operating in the Standard output mode. It has no effect on DIOs operating in the PWM output mode (which are being controlled by PWM generators).

Three values are specified (`0` to `0xFFFF`), with each bit representing one DIO state (1/0 = on/off):

`hval` – DIO[47:32]
`mval` – DIO[31:16]
`lval` – DIO[15:0]

Command: **wdom** *chan* {**std** | **pwm**}
Reply: –

Program DIO output operating mode.

Each DIO may be independently configured to operate in Standard or PWM mode. When operating in Standard mode, the DIO output state can be manually programmed via `wdo`. In PWM mode the state is automatically controlled by the I/O module, and the DIO output state will be toggled according to the timing specified in the latest `wpwm` command.

`chan` – DIO channel number (`0` to `47`).
`std` – standard (manual control) mode.
`pwm` – pulse width modulation mode.

Command: **wdbt** *chan time*
Reply: –

Program DIO input debounce time.

Physical input states are sampled periodically at one millisecond intervals and passed through a debounce filter. A digital input is regarded to be in a particular state only after it has held steady in that state for its debounce interval.

For example, consider the case of a digital input channel that has a 30 millisecond debounce interval. If the channel has been in the inactive state for a long time and then switches to the active state, `rdi` will not indicate the new (active) state until 30 milliseconds after the physical input became active. If the input goes active and then switches to inactive before the 30 milliseconds has elapsed, `rdi` will never indicate that the input is active.

Upon boot-up, all digital inputs have a ten millisecond debounce interval by default.

`chan` – DIO channel number (`0` to `47`).
`time` – debounce interval in milliseconds (`0` to `255`).

Command: **wpwm** *chan on off*
Reply: –

Program the duty cycle of a DIO that is operating in PWM mode.

This function applies to channels operating in PWM mode; it has no affect on channels operating in Standard mode.

The `on` and `off` arguments specify the amount of time that the DIO is to be in the active and inactive states, respectively. If `on` is zero and `off` is non-zero then the DIO output will always be inactive.

Similarly, if `off` is zero and `on` is non-zero then the output will always be active. The output state is indeterminate if both `on` and `off` are set to zero.

The designated DIO channel will switch to the active state and remain active until `on` has elapsed, then it will switch to the inactive state and remain in that state until `off` has elapsed. This sequence will repeat with the same duty cycle and frequency until one of these events occurs:

- The `on` and/or `off` is changed by calling s2410_WritePwm().
- The channel's operating mode is switched from PWM to Standard. The operating mode can be switched under software control by calling s2410_SetDoutMode() or s24xx_ResetIo(), and it may also be automatically switched in response to a module hardware reset.

`chan` – DIO channel number (0 to 47).
`on` – PWM on time in microseconds (0 to 65535).
`off` – PWM off time in microseconds (0 to 65535).

Command: **wtime** *time*
Reply: –

Jam a value into the timestamp generator.

`time` – value to be jammed (`0` to `0xFFFFFFFF`).

Command: **wto** *time* {**ms** | **s**} {**rst** | **norst**}
Reply: –

Set the communication timeout interval for the current session.

Each session employs a timer to detect loss of communication with the client. If no communication is received within this interval, the session will close and, if `rst` is specified, all DIOs will be reset.

By default, when a new session opens, the interval is five minutes (`5000 ms`) and the action is `norst`.

Upon executing this command, the new interval is effective immediately and the timer is started.

Setting `time` to zero will disable the timer. This should be avoided except during application development, as it could make it impossible to open a new session if earlier sessions failed to close properly.

`time` – time interval (`0` to `0xFFFFFFFF`).
`ms` | `s` – time units: milliseconds | seconds.
`rst` | `norst` – action to be taken upon time-out: reset | don't reset DIOs.

Command: **led {on | off |** *level***}**
Reply: –

Set brightness of DIO LED indicators.

This can be used to set LED brightness to a comfortable level or to decrease power consumption. LEDs can be disabled by setting level to 0. Upon boot-up, the LED intensity defaults to 16 (maximum intensity).

`level` – brightness value (`0` to `16`).
`on` – maximum brightness (equivalent to level 16).
`off` – dark (equivalent to level 0).

Command: **reset**
Reply: –

Reset all DIOs to their default power-up configurations and states.