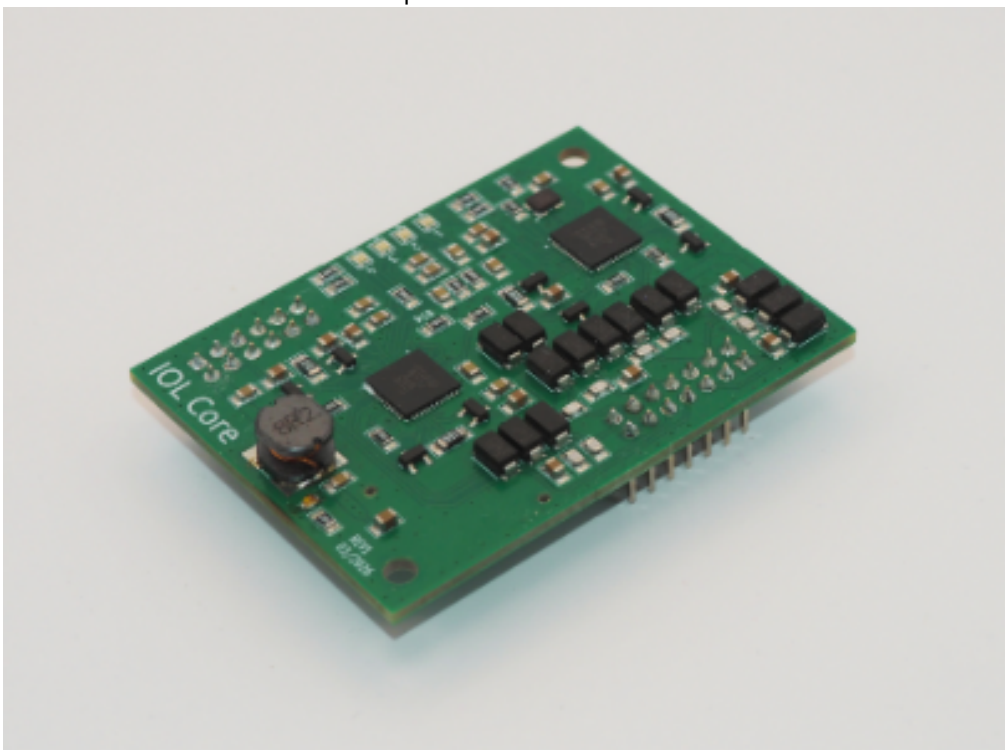
# 5 Timer Engine

## Hardware manual

www.pinetek-networks.com




Pinetek Networks GmbH, Bachstelzenweg 8, 88677 Markdorf
info@pinetek-networks.com

2026-11-22

# Table of Contents

5 Timer Engine ..... 5
*Timer Values* ..... 5
Timer Values ..... 5
Reset Timer ..... 5
*Timer Configuration* ..... 5
All Timer Configs ..... 5
Get Timer Config ..... 6
Set Timer Config ..... 6
*Engine Control* ..... 7
Start ..... 7
Stop ..... 7
Status ..... 7

# 5 Timer Engine

The Pinebox provides **8 timers** (indices `0..7`). Each timer measures elapsed time between a **start event** and a **stop event**, both sourced from IO-Link process data. Statistics (count, min, max, average) accumulate until explicitly reset.

The timer engine must be explicitly started and stopped.

## Timer Values

### Timer Values

`GET /timer/values`

Returns measurement statistics for all 8 timers.

**Query parameters**

| Parameter | Description |
|---|---|
| `?reset=true` | Atomically read all statistics and reset them to zero in a single operation |

**Response**

```
{
  "timer0": {
    "count": 5,
    "min_ms": 102,
    "max_ms": 310,
    "average_ms": 201
  },
  "timer1": { "..." },
  "..."
}
```

| Field | Type | Description |
|---|---|---|
| `count` | int | Number of completed start→stop cycles |
| `min_ms` | int | Minimum measured duration in milliseconds |
| `max_ms` | int | Maximum measured duration in milliseconds |
| `average_ms` | int | Average duration in milliseconds |

### Reset Timer

`POST /<num>/timer/reset`

Resets statistics for timer `<num>` to zero (count, min, max, average).

**Response:** Error envelope

## Timer Configuration

### All Timer Configs

`GET /timer/config`

Returns configuration for all 8 timers in a single response.

**Response**

```
{
  "timer0": {
    "enabled": true,
    "label": "Cycle time",
    "start_port": 0,
    "start_pd_index": 1,
    "start_edge": "rising",
    "start_filter_ms": 5,
    "start_threshold": 0,
    "stop_port": 0,
    "stop_pd_index": 1,
    "stop_edge": "falling",
    "stop_filter_ms": 5,
    "stop_threshold": 0
  },
  "timer1": { "..." },
  "..."
}
```

| Field | Type | Description |
|---|---|---|
| `enabled` | bool | Whether this timer is active |
| `label` | string | Human-readable timer name |
| `start_port` | int (0–3) | IO-Link port for the start event |
| `start_pd_index` | int | PD-in variable index for the start event |
| `start_edge` | string | Edge direction for start: `rising` or `falling` |
| `start_filter_ms` | uint32 | Debounce time for start event in milliseconds |
| `start_threshold` | number | Threshold value (for threshold-based start events) |
| `stop_port` | int (0–3) | IO-Link port for the stop event |
| `stop_pd_index` | int | PD-in variable index for the stop event |
| `stop_edge` | string | Edge direction for stop: `rising` or `falling` |
| `stop_filter_ms` | uint32 | Debounce time for stop event in milliseconds |
| `stop_threshold` | number | Threshold value (for threshold-based stop events) |

### Get Timer Config

`GET /<num>/timer/config/get`

Returns configuration for a single timer `<num>`. Field structure is identical to one entry in `timer/config`.

### Set Timer Config

`POST /<num>/timer/config/set`

Sets configuration for timer `<num>`. The request body uses the same fields as the config GET response.

**Response:** Error envelope

# Engine Control

## Start

`GET /timer/start`

Starts the timer engine. Timers begin evaluating IO-Link process data for start and stop events.

**Response:** Error envelope

## Stop

`GET /timer/stop`

Stops the timer engine. Accumulated statistics are retained.

**Response:** Error envelope

## Status

`GET /timer/status`

Returns the current run state of the timer engine.

**Response**

```
{ "running": true, "error": "none" }
```