UserGuide

Doc Ref: UG032 Rev. D

Generation date: Oct 23, 2026

FullScale

Analog Solutions

MD-MZ-I313 Driver - User Guide

Driver version 2.0.0

Online documentation :

https://docs.fullscale-labs.com/UG032/v2.0.0/

Contents

1 References 1

2 Abbreviation table 2

3 Introduction 3

4 Driver Context 4
4.1 How the Driver Appears in Linux . . . 4
4.2 Media Format and Link Assumptions . . . 4

5 Getting Started 5
5.1 On a Raspberry Pi . . . 5
5.2 On a Jetson Orin Nano developer kit . . . 5

6 Control Model 6

7 Control Reference 7

8 Using v4l2-ctl 9

9 Using the V4L2 ioctl API 10

10 Revision history 14

11 Disclaimer 15

1. References

Driver version: 2.0.0

Document reference	UG032
Document revision	D
Generation date	23 Oct. 2026

Written by	F. DESCAMPS	22 Oct. 2026
Validated by	-	-
Approved by	K. TOUTAIN	23 Oct. 2026

2. Abbreviation table

API	Application Programming Interface
ISP	Image Signal Processor
V4L2	Video4Linux2

3. Introduction

This guide introduces the MD-MZ-I313 Linux driver and explains how to interact with its custom V4L2 controls from user space.

The driver is intended to be installed from the Debian package provided for the platform (refer to document `UG034` for instructions). Once installed and enabled, it registers a V4L2 sub-device node and exposes board-specific controls and Sensor/Sensor ISP access.

This document is focused on user-space usage. It gives the operational context needed to understand where the controls live, how they are expected to be used, and which access patterns are important during bring-up and validation.

4. Driver Context

4.1. How the Driver Appears in Linux

When the package is installed on a Raspberry Pi or on a Jetson Orin Nano developer kit and the corresponding overlay is enabled, the board appears in the Linux media graph as a V4L2 sub-device. User-space control access therefore happens on a device such as `/dev/v4l-subdev0`. The receiver can have sub-device nodes too. The board's node is the one whose name in `/sys/class/video4linux/v4l-subdev*/name` starts with `md-mz-i313`, for example `md-mz-i313 9-003d`.

> **Note**
>
> The custom controls documented here are exposed on the V4L2 sub-device node. On a Jetson the capture node exposes them too; on a Raspberry Pi, capture nodes such as `/dev/video0` do not.

4.2. Media Format and Link Assumptions

The driver reports a fixed media-bus format:

- Bus code: `MEDIA_BUS_FMT_RGB888_1X24` (with R = LSB, G = MSB and B = 0)
- Colorspace: `V4L2_COLORSPACE_SRGB`
- Frame size: `352 x 240`
- Field: `V4L2_FIELD_NONE`

The capture node stores each 24-bit pixel the way the platform's receiver stores RGB888: in 3 bytes (`V4L2_PIX_FMT_RGB24`) on a Raspberry Pi, in 4 bytes (`V4L2_PIX_FMT_RGBA32` , the fourth one 0xFF) on a Jetson, whose receiver offers no other format for this bus code. On both, the 16-bit sensor word is bytes 0 (LSB) and 1 (MSB) of each pixel.

5. Getting Started

5.1. On a Raspberry Pi

A typical deployment and validation sequence is:

1. Install the provided Debian package, which brings in the board device tree overlay (package `fullscale-mipi-overlays-rpi`).
2. Enable the board device tree overlay.
3. Reboot.
4. Confirm that the sub-device node exists.
5. Inspect the media graph and the list of custom controls.
6. Issue control requests on the sub-device node.

Refer to the document `UG034` for instructions about first steps

Useful commands during validation are:

```
ls /dev/v4l-subdev*  # => Get exact subdevice node file (and confirm that the driver
# has been properly loaded)
v4l2-ctl --device=/dev/v4l-subdev0 --list-ctrls -L  # => List available controls
media-ctl -p  # => Confirm that the board's subdevice is linked to the CSI-2 receiver
```

5.2. On a Jetson Orin Nano developer kit

The driver builds against NVIDIA's VI and NVCSI drivers: the `nvidia-l4t-kernel-headers` and `nvidia-l4t-kernel-oot-headers` packages must be installed. The overlays come with the driver package, from `fullscale-mipi-overlays-jetson`: they are installed in `/boot` and selected with Jetson-IO, which lists them under the Jetson 22pin CSI Connector header as `FullScale MD-MZ-I313 on CAM0`, `FullScale MD-MZ-I313 on CAM1`, `FullScale MD-MZ-I313 on CAM0 and CAM1` and, with another FullScale board on the other connector, for example `FullScale DB-MIPI-A1 on CAM0 and MD-MZ-I313 on CAM1`:

```
sudo /opt/nvidia/jetson-io/config-by-hardware.py -l  # => List the headers and their overlays
sudo /opt/nvidia/jetson-io/config-by-hardware.py -n 2="FullScale MD-MZ-I313 on CAM1"  # => 2 is the␣
↪CSI connector header in that list
sudo reboot
```

The board streams on `/dev/video0`. Set the capture format to the sub-device's size, for example:

```
v4l2-ctl --device=/dev/video0 --set-fmt-video=width=352,height=240,pixelformat=AB24 \
         --stream-mmap --stream-count=100
```

With two boards, each board has its own capture node. A board's node is the one whose name in `/sys/class/video4linux/video*/name` ends with the board's sub-device name, for example `vi-output, md-mz-i313 9-003d`.

6. Control Model

The public API defines three main control styles:

- Read-only volatile Boolean controls used to fetch hardware live state
- Button controls used to trigger one-shot actions
- Integer read or write controls used to read hardware registers values or to set them

A few practical rules are worth keeping in mind:

- Button controls represent commands, so user space should write a value to trigger them, typically through `VIDIOC_S_CTRL` or `--set-ctrl`
- Volatile Boolean read controls are refreshed by the driver when user space reads them
- Sensor ISP related controls only make sense when the Sensor ISP has been started
- When the Sensor ISP is not started, some read operations intentionally return `0`, which is expected behaviour

7. Control Reference

Test Pattern Controls

`V4L2_CID_CUSTOM_G_TEST_PATTERN_ACTIVATED`

Returns whether the board test pattern is currently enabled.

Read-only boolean control.

`V4L2_CID_CUSTOM_S_TEST_PATTERN_ON`

Enables the board test pattern generator.

Button control.

`V4L2_CID_CUSTOM_S_TEST_PATTERN_OFF`

Disables the board test pattern generator.

Button control.

Data Mode Controls

These controls tune how the video data is presented on the MIPI CSI-2 interface by the board.

`V4L2_CID_CUSTOM_S_DATA_MODE_TONE_MAPPING_ON`

Enables tone mapping mode for the board data path.

Button control.

`V4L2_CID_CUSTOM_S_DATA_MODE_TONE_MAPPING_OFF`

Disables tone mapping mode for the board data path.

Button control.

`V4L2_CID_CUSTOM_S_DATA_MODE_MSB_FIRST`

Selects MSB-first ordering for the board data path.

Button control.

`V4L2_CID_CUSTOM_S_DATA_MODE_LSB_FIRST`

Selects LSB-first ordering for the board data path.

Button control.

Sensor ISP Controls

These controls allow communication with the ATI-320 sensor ISP.

V4L2_CID_CUSTOM_G_SENSOR_ISP_STARTED

Returns whether the Sensor ISP communication path is started.

Read-only boolean control.

V4L2_CID_CUSTOM_G_SENSOR_ISP_REGISTER

Reads a Sensor ISP register through a two-step access pattern.

Volatile 64-bit integer control. User space first writes the register address in bits [31:16], then reads the control back: bits [31:16] hold the address the value was read from and bits [15:0] the register value. The control is flagged inactive until the Sensor ISP is started and a first address has been written to it.

V4L2_CID_CUSTOM_G_SENSOR_ISP_VTEMP_TEMPERATURE

Returns the Sensor ISP VTEMP value.

Read-only integer control. This readout is meaningful only once the Sensor ISP is started; the control is flagged inactive until then.

V4L2_CID_CUSTOM_S_SENSOR_ISP_START

Starts the Sensor ISP communication path.

Button control.

V4L2_CID_CUSTOM_S_SENSOR_ISP_STOP

Stops the Sensor ISP communication path.

Button control.

V4L2_CID_CUSTOM_S_SENSOR_ISP_REGISTER

Writes a Sensor ISP register.

64-bit integer control. Bits [31:16] hold the register address and bits [15:0] the value to write.

Defines

V4L2_CID_CUSTOM_BASE

8. Using v4l2-ctl

Control names shown by `v4l2-ctl` are derived from the driver strings and are typically normalized with underscores. Always trust the exact names returned by `--list-ctrls -L` on your target.

Typical examples are:

```
# Query current custom controls
v4l2-ctl --device=/dev/v4l-subdev0 --list-ctrls -L

# Enable or disable the board test pattern
v4l2-ctl --device=/dev/v4l-subdev0 --set-ctrl='s_test_pattern_on=1'
v4l2-ctl --device=/dev/v4l-subdev0 --set-ctrl='s_test_pattern_off=1'

# Read back the test pattern state
v4l2-ctl --device=/dev/v4l-subdev0 --get-ctrl='g_test_pattern_activated'

# Start Sensor ISP access and read its started state
v4l2-ctl --device=/dev/v4l-subdev0 --set-ctrl='s_sensor_isp_start=1'
v4l2-ctl --device=/dev/v4l-subdev0 --get-ctrl='g_sensor_isp_started'

# Select the Sensor ISP register to read (address in bits [31:16]), then fetch it
v4l2-ctl --device=/dev/v4l-subdev0 --set-ctrl='g_sensor_isp_register=0x00010000'
v4l2-ctl --device=/dev/v4l-subdev0 --get-ctrl='g_sensor_isp_register'  # => (address << 16) | value
```

Warning

Register-read and register-write interactions depend on the exact support level of the `v4l2-ctl` version available on the target. For robust production use, a dedicated application based on the V4L2 ioctl API is often more convenient than shell commands alone.

9. Using the V4L2 ioctl API

The public header provides the control identifiers that should be used by applications.

The example below illustrates a one-shot button command and a Sensor ISP register write.

```c
#include <fcntl.h>
#include <linux/videodev2.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <sys/ioctl.h>
#include <unistd.h>

#include "md-mz-i313_public.h"

static int trigger_action(int fd, unsigned int id)
{
   struct v4l2_control ctrl;

   memset(&ctrl, 0, sizeof(ctrl));
   ctrl.id = id;
   ctrl.value = 1;
   return ioctl(fd, VIDIOC_S_CTRL, &ctrl);
}

static int write_sensor_isp_register(int fd, uint16_t reg, uint16_t value)
{
   struct v4l2_ext_control ext_ctrl;
   struct v4l2_ext_controls ext_ctrls;

   memset(&ext_ctrl, 0, sizeof(ext_ctrl));
   memset(&ext_ctrls, 0, sizeof(ext_ctrls));

   // Register address in bits [31:16], value in bits [15:0]
   ext_ctrl.id = V4L2_CID_CUSTOM_S_SENSOR_ISP_REGISTER;
   ext_ctrl.value64 = ((int64_t)reg << 16) | value;

   ext_ctrls.count = 1;
   ext_ctrls.controls = &ext_ctrl;

   return ioctl(fd, VIDIOC_S_EXT_CTRLS, &ext_ctrls);
}

int main(void)
{
   int fd = open("/dev/v4l-subdev0", O_RDWR);

   if (fd < 0)
      return 1;

   // Enable test pattern
   if (trigger_action(fd, V4L2_CID_CUSTOM_S_TEST_PATTERN_ON) < 0)
      perror("enable test pattern");

   // Start sensor ISP
   if (trigger_action(fd, V4L2_CID_CUSTOM_S_SENSOR_ISP_START) < 0)
      perror("start Sensor ISP");

   // Wait for Sensor ISP to start (polling
   // V4L2_CID_CUSTOM_G_SENSOR_ISP_STARTED would work too)
   sleep(5);

   // Put sensor in trigger mode
   // /!\ NOTE : Don't forget to disable trigger mode
```

(continues on next page)

(continued from previous page)

```
   // or to restart the sensor ISP in order to get video stream again
   if (write_sensor_isp_register(fd, 0x0102, 0x0001) < 0)
      perror("write Sensor ISP register");

   close(fd);
   return 0;
}
```

Note that the `md-mz-i313_public.h` file should be placed next to the above file with the following content :

```
/*****************************************************************************
 *                     ^                                                     *
 *                     |  @                                                  *
 *                @    _____    @    _ _ ____              _                 *
 *                |    |  ___|     _| | / ___|  ___ __ _| | ___              *
 *                |    | |_  | | | | | \___ \ / __/ _` | |/ _ \             *
 *             @  |    |  _|| |_| | | |___) | (_| (_| | |  __/             *
 *             |  |    |_|   \__,_|_|_|____/ \___\__,_|_|\___|             *
 *            ------------------------------------------------>            *
 *                     |             @  |     |     |    @                  *
 *                     |                |     |     |                       *
 *                                    @     |     @                         *
 *                                          @                               *
 * File: md-mz-i313_public.h                                                 *
 * Project: md-mz-i313-driver                                                *
 * Created 30/01/2026 by François Descamps                                   *
 *                                                                           *
 * Copyright (c) 2026 FullScale SARL. All Rights Reserved.                   *
 *                                                                           *
 * This project is licensed under GNU General Public License v2.0 only.      *
 * See the LICENSE file for details.                                         *
 *****************************************************************************/

#ifndef MD_MZ_I313_PUBLIC_H
# define MD_MZ_I313_PUBLIC_H

# include <linux/v4l2-controls.h>

// Base placed above the ranges already allocated in v4l2-controls.h to avoid ID collisions
# define V4L2_CID_CUSTOM_BASE (V4L2_CID_USER_BASE + 0x2000)

/**
 * @name Test Pattern Controls
 */

/**
 * @brief Returns whether the board test pattern is currently enabled.
 *
 * Read-only boolean control.
 */
# define V4L2_CID_CUSTOM_G_TEST_PATTERN_ACTIVATED (V4L2_CID_CUSTOM_BASE + 0x0100)

/**
 * @brief Enables the board test pattern generator.
 *
 * Button control.
 */
# define V4L2_CID_CUSTOM_S_TEST_PATTERN_ON (V4L2_CID_CUSTOM_BASE + 0x0181)

/**
 * @brief Disables the board test pattern generator.
 *
 * Button control.
 */
```

(continues on next page)

(continued from previous page)

```
# define V4L2_CID_CUSTOM_S_TEST_PATTERN_OFF (V4L2_CID_CUSTOM_BASE + 0x0182)

/**
 * @name Data Mode Controls
 *
 * @brief These controls tune how the video data is presented
 * on the MIPI CSI-2 interface by the board.
 */

/**
 * @brief Enables tone mapping mode for the board data path.
 *
 * Button control.
 */
# define V4L2_CID_CUSTOM_S_DATA_MODE_TONE_MAPPING_ON (V4L2_CID_CUSTOM_BASE + 0x0200)

/**
 * @brief Disables tone mapping mode for the board data path.
 *
 * Button control.
 */
# define V4L2_CID_CUSTOM_S_DATA_MODE_TONE_MAPPING_OFF (V4L2_CID_CUSTOM_BASE + 0x0201)

/**
 * @brief Selects MSB-first ordering for the board data path.
 *
 * Button control.
 */
# define V4L2_CID_CUSTOM_S_DATA_MODE_MSB_FIRST (V4L2_CID_CUSTOM_BASE + 0x0282)

/**
 * @brief Selects LSB-first ordering for the board data path.
 *
 * Button control.
 */
# define V4L2_CID_CUSTOM_S_DATA_MODE_LSB_FIRST (V4L2_CID_CUSTOM_BASE + 0x0283)

/**
 * @name Sensor ISP Controls
 *
 * @brief These controls allow communication with the ATI-320 sensor ISP.
 */

/**
 * @brief Returns whether the Sensor ISP communication path is started.
 *
 * Read-only boolean control.
 */
# define V4L2_CID_CUSTOM_G_SENSOR_ISP_STARTED (V4L2_CID_CUSTOM_BASE + 0x0300)

/**
 * @brief Reads a Sensor ISP register through a two-step access pattern.
 *
 * Volatile 64-bit integer control. User space first writes the register
 * address in bits [31:16], then reads the control back: bits [31:16] hold
 * the address the value was read from and bits [15:0] the register value.
 * The control is flagged inactive until the Sensor ISP is started and a
 * first address has been written to it.
 */
# define V4L2_CID_CUSTOM_G_SENSOR_ISP_REGISTER (V4L2_CID_CUSTOM_BASE + 0x0301)

/**
 * @brief Returns the Sensor ISP VTEMP value.
 *
 * Read-only integer control. This readout is meaningful only once the
 * Sensor ISP is started; the control is flagged inactive until then.
```

(continues on next page)

(continued from previous page)

```
 */
# define V4L2_CID_CUSTOM_G_SENSOR_ISP_VTEMP_TEMPERATURE (V4L2_CID_CUSTOM_BASE + 0x0302)

/**
 * @brief Starts the Sensor ISP communication path.
 *
 * Button control.
 */
# define V4L2_CID_CUSTOM_S_SENSOR_ISP_START (V4L2_CID_CUSTOM_BASE + 0x0383)

/**
 * @brief Stops the Sensor ISP communication path.
 *
 * Button control.
 */
# define V4L2_CID_CUSTOM_S_SENSOR_ISP_STOP (V4L2_CID_CUSTOM_BASE + 0x0384)

/**
 * @brief Writes a Sensor ISP register.
 *
 * 64-bit integer control. Bits [31:16] hold the register address and
 * bits [15:0] the value to write.
 */
# define V4L2_CID_CUSTOM_S_SENSOR_ISP_REGISTER (V4L2_CID_CUSTOM_BASE + 0x0385)

#endif
```

10. Revision history

Date	Author(s)	Version	Comments
04/2026	F. DESCAMPS	Rev. A	First document release
04/2026	F. DESCAMPS	Rev. B	Switch to generated documentation + Minor layout changes
04/2026	F. DESCAMPS	Rev. C	Fix example C program
10/2026	F. DESCAMPS	Rev. D	Driver 2.0.0: Raspberry Pi 5 and Jetson Orin Nano, overlays package, new control IDs, fixed examples

11. Disclaimer

FullScale reserves the right to make any changes without further notice to any products herein. FullScale makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does FullScale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

“Typical” parameters can and do vary in different applications. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts.

Use of FullScale products or services with statements different from or beyond the parameters stated by FullScale for that product or service voids all express and any implied warranties for the associated FullScale product or service and is an unfair and deceptive business practice. FullScale is not responsible or liable for any such statements.

FullScale products are not authorized for use as critical components in life support devices or systems without the express written approval of FullScale.

FullScale software and associated products cannot be used except strictly in accordance with a FullScale software license. The terms of the appropriate FullScale software license shall prevail over the above terms to the extent of any inconsistency.