Intel Integrated Sensor Hub (ISH)

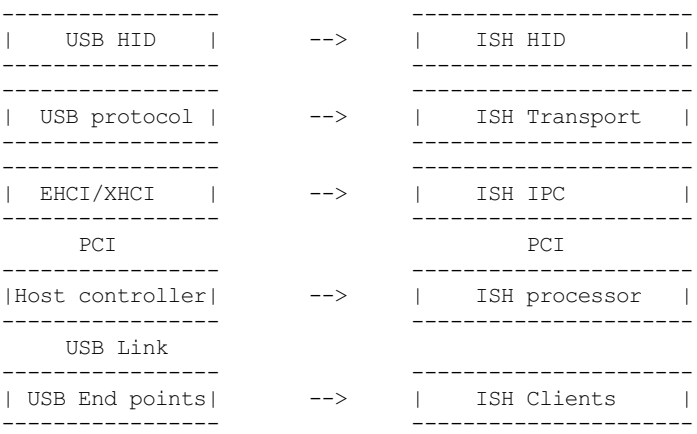
A sensor hub enables the ability to offload sensor polling and algorithm processing to a dedicated low power co-processor. This allows the core processor to go into low power modes more often, resulting in increased battery life.

There are many vendors providing external sensor hubs conforming to HID Sensor usage tables. These may be found in tablets, 2-in-1 convertible laptops and embedded products. Linux has had this support since Linux 3.9.

IntelÂ® introduced integrated sensor hubs as a part of the SoC starting from Cherry Trail and now supported on multiple generations of CPU packages. There are many commercial devices already shipped with Integrated Sensor Hubs (ISH). These ISH also comply to HID sensor specification, but the difference is the transport protocol used for communication. The current external sensor hubs mainly use HID over I2C or USB. But ISH doesn't use either I2C or USB.

1. Overview

Using a analogy with a usbhid implementation, the ISH follows a similar model for a very high speed communication:

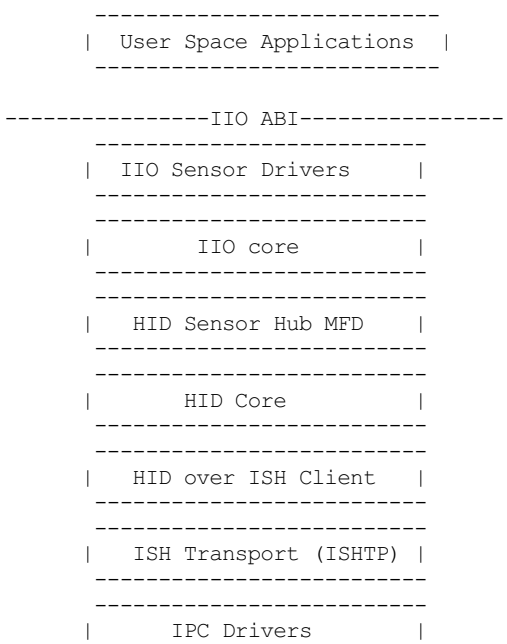


Like USB protocol provides a method for device enumeration, link management and user data encapsulation, the ISH also provides similar services. But it is very light weight tailored to manage and communicate with ISH client applications implemented in the firmware.

The ISH allows multiple sensor management applications executing in the firmware. Like USB endpoints the messaging can be to/from a client. As part of enumeration process, these clients are identified. These clients can be simple HID sensor applications, sensor calibration applications or sensor firmware update applications.

The implementation model is similar, like USB bus, ISH transport is also implemented as a bus. Each client application executing in the ISH processor is registered as a device on this bus. The driver, which binds each device (ISH HID driver) identifies the device type and registers with the HID core.

2. ISH Implementation: Block Diagram



```
        -------------------------
OS
---------------- PCI -----------------
Hardware + Firmware
        ---------------------------
       | ISH Hardware/Firmware(FW) |
        ---------------------------
```

3. High level processing in above blocks

3.1 Hardware Interface

The ISH is exposed as "Non-VGA unclassified PCI device" to the host. The PCI product and vendor IDs are changed from different generations of processors. So the source code which enumerates drivers needs to update from generation to generation.

3.2 Inter Processor Communication (IPC) driver

Location: drivers/hid/intel-ish-hid/ipc

The IPC message uses memory mapped I/O. The registers are defined in hw-ish-regs.h.

3.2.1 IPC/FW message types

There are two types of messages, one for management of link and another for messages to and from transport layers.

TX and RX of Transport messages

A set of memory mapped register offers support of multi-byte messages TX and RX (e.g. IPC_REG_ISH2HOST_MSG, IPC_REG_HOST2ISH_MSG). The IPC layer maintains internal queues to sequence messages and send them in order to the firmware. Optionally the caller can register handler to get notification of completion. A doorbell mechanism is used in messaging to trigger processing in host and client firmware side. When ISH interrupt handler is called, the ISH2HOST doorbell register is used by host drivers to determine that the interrupt is for ISH.

Each side has 32 32-bit message registers and a 32-bit doorbell. Doorbell register has the following format:

```
Bits 0..6: fragment length (7 bits are used)
Bits 10..13: encapsulated protocol
Bits 16..19: management command (for IPC management protocol)
Bit 31: doorbell trigger (signal H/W interrupt to the other side)
Other bits are reserved, should be 0.
```

3.2.2 Transport layer interface

To abstract HW level IPC communication, a set of callbacks is registered. The transport layer uses them to send and receive messages. Refer to struct ishtp_hw_ops for callbacks.

3.3 ISH Transport layer

Location: drivers/hid/intel-ish-hid/ishtp/

3.3.1 A Generic Transport Layer

The transport layer is a bi-directional protocol, which defines: - Set of commands to start, stop, connect, disconnect and flow control (see ishtp/hbm.h for details) - A flow control mechanism to avoid buffer overflows

This protocol resembles bus messages described in the following document:
http://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/dcmi-hi-1-0-spec.pdf "Chapter 7: Bus Message Layer"

3.3.2 Connection and Flow Control Mechanism

Each FW client and a protocol is identified by a UUID. In order to communicate to a FW client, a connection must be established using connect request and response bus messages. If successful, a pair (host_client_id and fw_client_id) will identify the connection.

Once connection is established, peers send each other flow control bus messages independently. Every peer may send a message only if it has received a flow-control credit before. Once it has sent a message, it may not send another one before receiving the next flow control credit. Either side can send disconnect request bus message to end communication. Also the link will be dropped if major FW reset occurs.

3.3.3 Peer to Peer data transfer

Peer to Peer data transfer can happen with or without using DMA. Depending on the sensor bandwidth requirement DMA can be enabled by using module parameter ishtp_use_dma under intel_ishtp.

Each side (host and FW) manages its DMA transfer memory independently. When an ISHTP client from either host or FW side

wants to send something, it decides whether to send over IPC or over DMA; for each transfer the decision is independent. The sending side sends DMA_XFER message when the message is in the respective host buffer (TX when host client sends, RX when FW client sends). The recipient of DMA message responds with DMA_XFER_ACK, indicating the sender that the memory region for that message may be reused.

DMA initialization is started with host sending DMA_ALLOC_NOTIFY bus message (that includes RX buffer) and FW responds with DMA_ALLOC_NOTIFY_ACK. Additionally to DMA address communication, this sequence checks capabilities: if thw host doesn't support DMA, then it won't send DMA allocation, so FW can't send DMA; if FW doesn't support DMA then it won't respond with DMA_ALLOC_NOTIFY_ACK, in which case host will not use DMA transfers. Here ISH acts as busmaster DMA controller. Hence when host sends DMA_XFER, it's request to do host->ISH DMA transfer; when FW sends DMA_XFER, it means that it already did DMA and the message resides at host. Thus, DMA_XFER and DMA_XFER_ACK act as ownership indicators.

At initial state all outgoing memory belongs to the sender (TX to host, RX to FW), DMA_XFER transfers ownership on the region that contains ISHTP message to the receiving side, DMA_XFER_ACK returns ownership to the sender. A sender need not wait for previous DMA_XFER to be ack'ed, and may send another message as long as remaining continuous memory in its ownership is enough. In principle, multiple DMA_XFER and DMA_XFER_ACK messages may be sent at once (up to IPC MTU), thus allowing for interrupt throttling. Currently, ISH FW decides to send over DMA if ISHTP message is more than 3 IPC fragments and via IPC otherwise.

3.3.4 Ring Buffers

When a client initiates a connection, a ring of RX and TX buffers is allocated. The size of ring can be specified by the client. HID client sets 16 and 32 for TX and RX buffers respectively. On send request from client, the data to be sent is copied to one of the send ring buffer and scheduled to be sent using bus message protocol. These buffers are required because the FW may have not have processed the last message and may not have enough flow control credits to send. Same thing holds true on receive side and flow control is required.

3.3.5 Host Enumeration

The host enumeration bus command allows discovery of clients present in the FW. There can be multiple sensor clients and clients for calibration function.

To ease implementation and allow independent drivers to handle each client, this transport layer takes advantage of Linux Bus driver model. Each client is registered as device on the transport bus (ishtp bus).

Enumeration sequence of messages:

- Host sends HOST_START_REQ_CMD, indicating that host ISHTP layer is up.
- FW responds with HOST_START_RES_CMD
- Host sends HOST_ENUM_REQ_CMD (enumerate FW clients)
- FW responds with HOST_ENUM_RES_CMD that includes bitmap of available FW client IDs
- For each FW ID found in that bitmap host sends HOST_CLIENT_PROPERTIES_REQ_CMD
- FW responds with HOST_CLIENT_PROPERTIES_RES_CMD. Properties include UUID, max ISHTP message size, etc.
- Once host received properties for that last discovered client, it considers ISHTP device fully functional (and allocates DMA buffers)

3.4 HID over ISH Client

Location: drivers/hid/intel-ish-hid

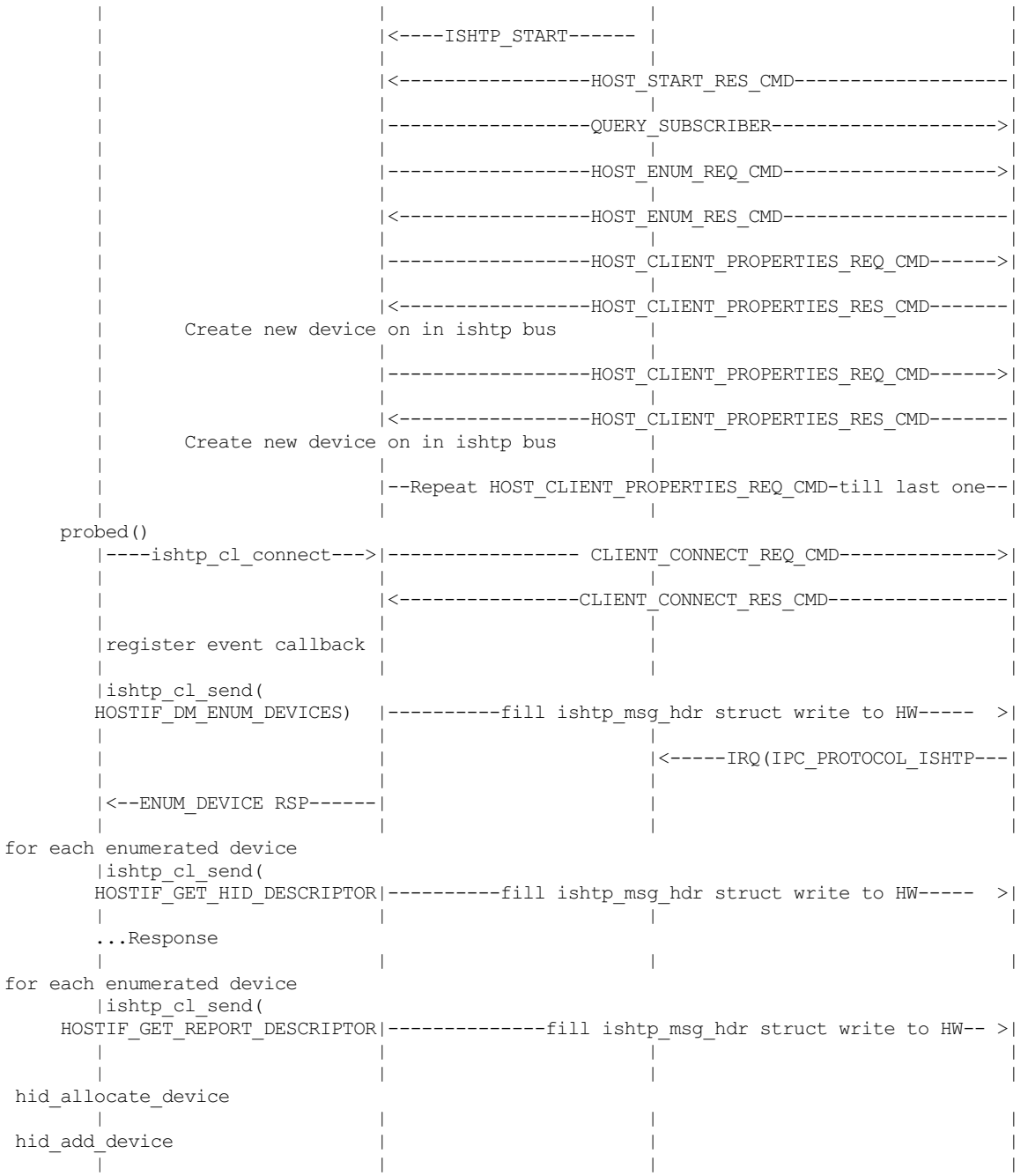
The ISHTP client driver is responsible for:

- enumerate HID devices under FW ISH client
- Get Report descriptor
- Register with HID core as a LL driver
- Process Get/Set feature request
- Get input reports

3.5 HID Sensor Hub MFD and IIO sensor drivers

The functionality in these drivers is the same as an external sensor hub. Refer to Documentation/hid/hid-sensor.rst for HID sensor Documentation/ABI/testing/sysfs-bus-iio for IIO ABIs to user space.

3.6 End to End HID transport Sequence Diagram



3.7 ISH Debugging

To debug ISH, event tracing mechanism is used. To enable debug logs:

```
echo 1 > /sys/kernel/debug/tracing/events/intel_ish/enable
cat /sys/kernel/debug/tracing/trace
```

3.8 ISH IIO sysfs Example on Lenovo thinkpad Yoga 260

```
root@otcpl-ThinkPad-Yoga-260:~# tree -l /sys/bus/iio/devices/
/sys/bus/iio/devices/
â”œâ”€â”€ iio:device0 -> ../../../devices/0044:8086:22D8.0001/HID-SENSOR-200073.9.auto/iio:device0
â”‚Â Â  â”œâ”€â”€ buffer
â”‚Â Â  â”‚Â Â  â”œâ”€â”€ enable
â”‚Â Â  â”‚Â Â  â”œâ”€â”€ length
â”‚Â Â  â”‚Â Â  â””â”€â”€ watermark
...
â”‚Â Â  â”œâ”€â”€ in_accel_hysteresis
â”‚Â Â  â”œâ”€â”€ in_accel_offset
â”‚Â Â  â”œâ”€â”€ in_accel_sampling_frequency
â”‚Â Â  â”œâ”€â”€ in_accel_scale
â”‚Â Â  â”œâ”€â”€ in_accel_x_raw
â”‚Â Â  â”œâ”€â”€ in_accel_y_raw
â”‚Â Â  â”œâ”€â”€ in_accel_z_raw
â”‚Â Â  â”œâ”€â”€ name
â”‚Â Â  â”œâ”€â”€ scan_elements
â”‚Â Â  â”‚Â Â  â”œâ”€â”€ in_accel_x_en
â”‚Â Â  â”‚Â Â  â”œâ”€â”€ in_accel_x_index
```

```
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_accel_x_type
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_accel_y_en
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_accel_y_index
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_accel_y_type
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_accel_z_en
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_accel_z_index
â”‚  â”‚  â”‚  â”‚  â””â”€â”€ in_accel_z_type
...
â”‚  â”‚  â”‚  â”œâ”€â”€ devices
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ buffer
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ enable
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ length
â”‚  â”‚  â”‚  â”‚  â”‚  â””â”€â”€ watermark
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ dev
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_intensity_both_raw
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_intensity_hysteresis
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_intensity_offset
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_intensity_sampling_frequency
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_intensity_scale
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ name
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ scan_elements
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_intensity_both_en
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_intensity_both_index
â”‚  â”‚  â”‚  â”‚  â”‚  â””â”€â”€ in_intensity_both_type
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ trigger
â”‚  â”‚  â”‚  â”‚  â””â”€â”€ current_trigger
...
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ buffer
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ enable
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ length
â”‚  â”‚  â”‚  â”‚  â”‚  â””â”€â”€ watermark
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ dev
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_hysteresis
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_offset
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_sampling_frequency
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_scale
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_x_raw
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_y_raw
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_z_raw
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_rot_from_north_magnetic_tilt_comp_raw
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_rot_hysteresis
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_rot_offset
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_rot_sampling_frequency
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_rot_scale
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ name
...
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ scan_elements
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_x_en
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_x_index
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_x_type
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_y_en
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_y_index
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_y_type
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_z_en
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_z_index
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_magn_z_type
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_rot_from_north_magnetic_tilt_comp_en
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_rot_from_north_magnetic_tilt_comp_index
â”‚  â”‚  â”‚  â”‚  â”‚  â””â”€â”€ in_rot_from_north_magnetic_tilt_comp_type
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ trigger
â”‚  â”‚  â”‚  â”‚  â””â”€â”€ current_trigger
...
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ buffer
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ enable
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ length
â”‚  â”‚  â”‚  â”‚  â”‚  â””â”€â”€ watermark
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ dev
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_hysteresis
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_offset
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_sampling_frequency
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_scale
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_x_raw
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_y_raw
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_z_raw
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ name
â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ scan_elements
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_x_en
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_x_index
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_x_type
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_y_en
â”‚  â”‚  â”‚  â”‚  â”‚  â”œâ”€â”€ in_anglvel_y_index
```

```
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_y_type
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_z_en
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_z_index
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â””â”€â”€ in_anglvel_z_type
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ trigger
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â””â”€â”€ current_trigger
...
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ buffer
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ enable
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ length
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â””â”€â”€ watermark
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ dev
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_hysteresis
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_offset
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_sampling_frequency
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_scale
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_x_raw
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_y_raw
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_z_raw
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ name
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ scan_elements
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_x_en
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_x_index
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_x_type
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_y_en
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_y_index
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_y_type
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_z_en
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ in_anglvel_z_index
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â””â”€â”€ in_anglvel_z_type
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”œâ”€â”€ trigger
â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â”,Â Â  â””â”€â”€ current_trigger
...
```