Timestamps and Time Synchronization

Item	Content
Applicable product	OG（OmniGlove / H743VI main controller）
Software /firmware version	H743VI APP 1.10.3
Owner	Embedded team
Last updated	2026/10/24
Change history	Initial version: documents current firmware behavior using the "Timestamps and Time Synchronization" template

1 Data Frame Timestamps

1.1 Semantics Before and After Synchronization

Table 1

Field	Before Sync	After Sync
tick_ms（uint32）	Monotonic milliseconds from `HAL_GetTick()` since device power-on (uptime)	Unchanged; still device uptime
time_ms（uint64）	**Always 0 (time not set)**	Unix epoch milliseconds (since 1970-01-01 UTC)
time_us（uint16）	0	Microsecond remainder 0..999 carried by the synchronization anchor (see §2; not advanced locally at sub-millisecond resolution)

Business frames (POST, REMOTE, RAW header, POSE, and so on) carry both `tick_ms` and `time_ms`:

`time_ms = AgTime_EpochMsAtTick(tick_ms)`, which maps the uptime frozen for that cycle to UTC.

1.2 When Each Data Stream Is Stamped

Table 2

Data Stream	Stamped At	Notes
Main capture cycle (125 Hz / 8 ms）	`HAL_GetTick()` latched when the cycle closes	USB_Proto freezes HostTickMs / HostEpoch; RAW/POSE/POST/REMOTE in the same cycle share them
POST / REMOTE subscription reports	The shared cycle tick_ms above	The derived fields time_ms / time_us are computed from the anchor
Single read	The current shared cycle or current tick at packet assembly	Same source API as subscriptions: USB_Proto_HostTickMs() + AgTime_EpochMsAtTick
0x0D BOOT_TIME response	`HAL_GetTick()` at the moment of the response	Does not report UTC; does not change the anchor

1.3 Inheritance Rules for Derived Data

- POST, REMOTE, RAW headers, and other frames sent within the same 8 ms cycle share the same tick_ms, so the host can align multiple streams.
- Derived products such as IK and tactile data attach to the snapshot of that cycle and inherit its timestamp. They are not stamped separately.
- Before time has been set, time_ms = 0. The host should use tick_ms for relative ordering and interval calculations.

2 Clock Model

Table 3

Item	Behavior
RTC	**No battery-backed RTC**； UTC depends entirely on the time anchor set by the host software.
Power-on	`HAL_GetTick()` starts from 0. The AgTime synchronized flag is cleared, and time_ms reads 0.

Item	Behavior
Local time base	`Cortex SysTick` → `HAL_IncTick()`, **1 ms** resolution
UTC calculation	Computed at read time：unix_at_sync + (now_tick - tick_at_sync)（unsigned 32-bit subtraction）
Wraparound	`HAL_GetTick` wraps about every 49.7 days. The calculation is correct for 49.7 days after the most recent synchronization.
Restart	The anchor is lost. `SYNC_TIME` must be sent again.
Sub-millisecond	The request may carry time_us. The local clock remains at 1 ms; `AgTime_EpochUsPart()` keeps the **anchor remainder unchanged**. In the `0x0D` response, time_us currently returns **0**.

3 Synchronization Mechanism

3.1 How Client Time Setting Works

- The host is the time authority. It sends Unix time through the AG-Link common command set, `cmd_set=0x00` / `cmd_id=0x0D` (`AGILINK_CMD_ID_NORMAL_SYNC_TIME`).
- The device latches the anchor and then extrapolates with its local SysTick. The device does not perform NTP-style multi-packet round-trip estimation.

3.2 Command Types (agilink_normal_time_type_t)

Table 4

Type	Meaning	Device Behavior
1 UNIX_TIMESTAMP	Set the time	AgTime_Sync(time_ms, time_us), responds with the currently computed time_ms

Type	Meaning	Device Behavior
3 BOOT_TIME	Read uptime in milliseconds	Responds with `HAL_GetTick()`; **does not change the anchor**
0 NONE / 2 NORMAL_READ	Read the current global time	Synchronized: `OK + time_ms`; Not synchronized:`FAIL_1 + 0`

For the request and response structures, see `agilink_normal_cmd_defines.h` (`agilink_normal_sync_time_req_t` / `_resp_t`).

Calendar fields (year, month, day, and so on) and time zones are reserved in the request structure. The current firmware uses only `time_ms` + `time_us` to set the time.

3.3 Four Time Points and Offset (Recommended on the Host)

Single-packet time setting in the firmware is **equivalent to pinning the host's time_ms at the moment the request is received**. A more robust approach to the host:

1. Record `T1`: the time the host sends the request (host clock).
2. Record `T4`: the time the host receives the response.
3. Use the device time in the response (or the expected time filled in at send time) to estimate the one-way delay ≈ (T4 − T1) / 2.
4. If needed, send `UNIX_TIMESTAMP` again with the **corrected host UTC**.

The device **does not return** separate RTT or offset diagnostic fields. Quality metrics depend on host-side measurement (see §7).

3.4 Firmware Version Requirements

- Implementation files: `ag_time.c` / `ag_time.h`. Command handler: `handle_sys_sync_time`.
- For the current version of each product line, see the table at the top of this document.

The protocol test cases include `sync_boot_time` (`tools/protocol_test.py`).

4 Automatic Synchronization

Table 5

Item	Current Status
Automatic time setting in firmware	**None. The device does not connect to NTP on its own.**
First synchronization	The host or SDK sends `type=UNIX_TIMESTAMP` after enumeration succeeds and before it subscribes to data.
Periodic synchronization	Configured by the host. For long recordings, re-synchronize periodically to compensate for crystal drift. The firmware has no default interval parameter.
How to turn it off	The host simply stops sending. **The local SysTick cannot be turned off.**

5 Manual Synchronization

Table 6

Item	Description
When to use	Before a long recording; before aligning multiple devices; after a device reset or reconnection; when `time_ms` stays 0 for a long time or deviates too far from the host
Procedure	Send `0x0D` with `type=UNIX_TIMESTAMP`, filled with the host `UTC time_ms` (and optionally `time_us`)
Successful response	`return_code=0`, type echoed back, `time_ms` is the value computed immediately after synchronization
Readable fields	`BOOT_TIME` reads uptime; `NONE / NORMAL_READ` reads the current UTC (fails if not synchronized)

Item	Description
Not provided by firmware	Dedicated return values for offset, round-trip time, or synchronization wall-clock time. The host must calculate them from its own send and receive times.

6 Timestamp Monotonicity

Table 7

Clock	Monotonicity
tick_ms	Increases strictly by 1 ms after power-on (until 32-bit wraparound). Equal across streams in the same cycle; usually +8 ms between cycles (125 Hz).
time_ms	Extrapolated from `tick_ms` after time is set; monotonic between two time settings. Re-synchronization may cause a jump, which the host smooths according to its own alignment strategy.
Use	Prefer `tick_ms` for ordering and sampling intervals. Use `time_ms` (requires time to be set) for wall-clock alignment and cross-device logs.

The firmware does not smooth the clock rate (no PLL or servo). If you need smoothing, interpolate across re-synchronizations or limit the step size on the host.

7 Synchronization Accuracy and Quality

Table 8

Metric	Description	Magnitude/Threshold
Local Resolution	SysTick 1 ms	Business timestamps are quantized to 1 ms
USB/protocol latency	Time setting error includes one-way transmission and scheduling	Typically a few milliseconds to over ten milliseconds, depending on host load
Round-trip time(RTT)	Measured by the host as T4 − T1	**Typical value and upper limit to be provided after testing**

Metric	Description	Magnitude/Threshold
Offset / jitter / drift	Relative to the host NTP or system clock	**Acceptance thresholds to be provided after testing**
Recommendations	Set the time manually before long recordings; re-synchronize periodically if needed; use `tick_ms` to detect dropped frames	—

8 Multi-Device Synchronization

Table 9

Condition	Description
Shared time authority	Every glove receives `UNIX_TIMESTAMP` from the same host clock (or the same time service)
Recommended procedure	① Enumerate all devices → ② Set the time device by device or by broadcast → ③ Read back time_ms and spot-check → ④ Then start POST / capture subscriptions
With trackers or robots	If the external system uses the same host UTC, align with each side's `time_ms`. Otherwise, keep each side's uptime and align offline with event markers.
Capture epoch	**The 8 ms cycles of different gloves are not hardware-aligned.** Sub-cycle alignment across gloves needs an additional solution, which is outside the scope of UTC time setting in this document.

9 Data Processing Recommendations

Table 10

Scenario	Principle
Ordering	Within one device, prefer `tick_ms`. Use `time_ms` when time is set and wall-clock time is needed.

Scenario	Principle
Buffering	Assemble frames by cycle number / tick_ms. Merge multiple streams in the same cycle by the shared tick_ms.
Interpolation	Interpolate on the `tick_ms` time axis. Do not interpolate on time_ms when it is 0 (time not set).
Dropped frames	A tick_ms jump larger than the expected interval (for example, > 8 ms) indicates a dropped cycle.
Out-of-order data	USB generally preserves order. If the host reorders packets, go by tick_ms.
Offline alignment	Use the time-setting event at the start of the recording or a shared trigger. Record a discontinuity at each re-synchronization point.
Time not set	**When time_ms == 0, never use it as UTC.** Use it for uptime analysis only.
Unrecoverable	After a device reset, time must be set again. A lost anchor cannot be restored from flash.

10 Semantics of SDK Output Time Fields

In addition to business data, every sample the SDK outputs carries a set of time and sequence fields. This section continues from Section 3.2 of the **OmniGlove SDK User Guide** and describes the time base and recommended use of these fields. Note that these fields are the raw host-side and device-side time bases of the SDK. They are independent of the host UTC time setting in Chapter 3 of this document, and the two cannot be converted directly into each other.

Table 11

Field	Time Base / Range	Recommended Use
sample_monotonic_ns	Host monotonic clock; nanoseconds	Ordering, interval, and continuity analysis of samples on the same host
publish_monotonic_ns	Host UDP publish time	Analysis of the publish stage against receive times on the same host

Field	Time Base / Range	Recommended Use
device_tick_ms	MCU uptime in milliseconds; uint32, wraps around	Tick tracking within the device; handle 2^32 wraparound
device_time_ms	Reserved device UTC in milliseconds	Not a business basis for synchronized UTC
device_time_us	Reserved microsecond remainder within the millisecond, 0–999	Not to be interpreted on its own as a full microsecond timestamp

The `_monotonic_ns` fields are host monotonic time, for ordering and interval analysis on the same host. Different hosts have different starting points, so you cannot subtract across hosts to get network latency. `device_tick_ms` is MCU uptime in milliseconds and wraps at 2^32. It cannot be compared directly with host monotonic time.

`device_time_ms` / `device_time_us` are reserved for compatibility, and their values may be 0. When `OMNIGLOVE_STREAM_SAMPLE_DEVICE_TIME_VALID` is set, it means only that the fields are valid, not that UTC has been synchronized. Use `sample_monotonic_ns` for business ordering and interval analysis.

Same-frame relationship: When `POSE components=ALL`, pose and tactile data belong to the same device tick and need no cross-stream pairing. This relationship does not apply across different gloves, between POSE and POST, or across different hosts. The `frame_id` of POST is a 32-bit frame number, interpreted separately from the shared `device_sequence` metadata. For the full field definitions, see Section 3.2 of the **OmniGlove SDK User Guide.**

11 Appendix: Implementation Index

Table 12

Module	Path
Time anchor	app/project/cmake/H743VI/Core/Src/ag_time.c
0x0D handling	ag_link_new_cmd_handler.c → handle_sys_sync_time
Protocol structures	app/include/agilink_normal_cmd_defines.h
Command ID	AGILINK_CMD_ID_NORMAL_SYNC_TIME（0x0D）

Module	Path
Shared cycle tick	USB_Proto_HostTickMs() / USB_Proto_HostEpoch()
Node capture synchronization (not UTC)	docs/sync_and_goertzel_epoch.md