Detects, analyzes, and fixes sporadic errors

Inchron offers a tool-suite for CAN networks. It is designed to find sporadic real-time errors as early as possible.

Screenshot of the tool-suite supporting 15 requirement types (Photo: Inchron)

The appearance of sporadic errors in a late project phase or after start of production can be costly. This is true for functional errors, but it also applies to real-time errors. To detect, analyze, and fix the entire spectrum of real-time errors in a systematic manner is essential. For this purpose, the tool-suite by Inchron supports 15 different requirement types. These requirement types have been tailored to the needs of customers. Measurement results can be checked against the requirements based on trace logs, either post mortem, or “on-the-fly” during test execution. In the latter case, sporadic errors can be detected as soon as these occur.

Within mission-critical systems, the net slack-time requirement is widely used to ensure and confirm, that the architecture is robust and unintended watchdog resets are prevented. For autonomous driving, the maximum end-to-end latency for functions and the data age are the most important requirements.

Example, in which an end-to-end latency requirement is violated (Photo: Inchron)

The shown example illustrates a scenario, in which end-to-end latency requirement violations happen due to small variations in the system that comprises several activities sharing scarce resources. The first event chain (the series of arrows connecting events from the top line to the bottom line, indicating consecutive events that depend on each other) starts in the upper left corner at t = 0 ms, goes all across tasks running on CPU-1, the CAN network, and interrupts and tasks running on CPU-2. The event chain terminates within an end-to-end latency of 15 ms, which fulfills the real-time requirements in this particular example. A second event chain of the same kind starts at t = 20 ms. The T_10ms task that starts at t = 22 ms, however, takes longer to execute, such that it is being pre-empted by the T_05ms task starting at t = 25 ms. This even further delays the execution of the T_20ms task on CPU-1. Moreover, the T_20ms task takes more time to execute as well, such that it runs into further preemptions. “As a consequence, this event chain terminates at the I_CanRx interrupt on CPU-2, far too late to be taken into account by the periodic task T_20ms starting at t=31 ms on CPU2,” explained Olaf Schmidt from Inchron. “Depending on the way the software is implemented, this issue can either result in data loss, or unintended data reuse.”

hz