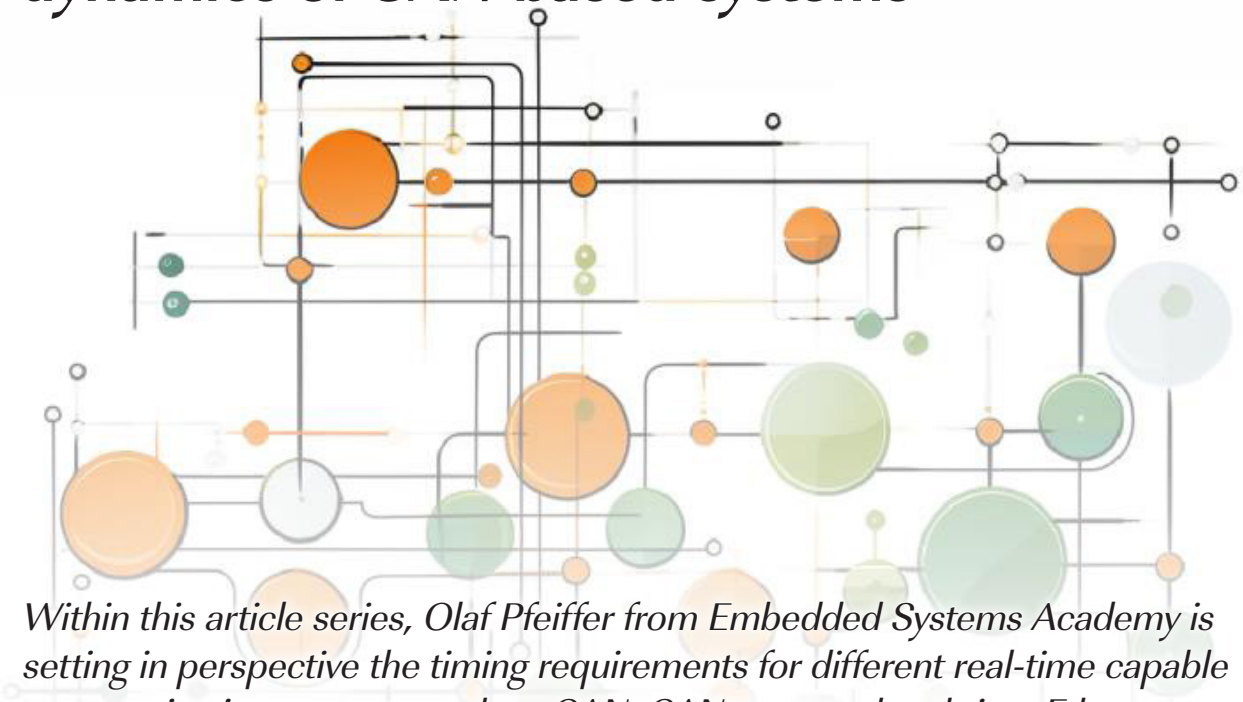
Real-time communication: Part III - Temporal dynamics of CAN-based systems



(Source: Embedded Systems Academy)

Within this article series, Olaf Pfeiffer from Embedded Systems Academy is setting in perspective the timing requirements for different real-time capable communication systems, such as CAN, CANopen, and real-time Ethernet.

In this third article, the author examines the real-time capabilities and limitations of CAN and CANopen in greater detail. The Part I, published in the December 2023 issue of the CAN Newsletter magazine, describes how to select the right real-time timeframe for certain applications. The Part II, published in March 2024, shows the different timeframes required by different applications and reviews what this means for the communication system used. The last article "Part IV – From theory to practice: CANopen source code configuration" shows which optimization options are typically available when working with CANopen source code, here, on example of Micro CANopen Plus from EmSA.

The Controller Area Network (CAN) protocol serves as the foundation for numerous applications across a wide range of industries, each with its own distinct real-time demands. Prominent examples like CANopen and J1939 highlight the diverse adaptations of this protocol to meet specific needs. It's important to note that the real-time requirements for these applications are not uniform across the board. While some applications require reaction times measured in milliseconds, many others operate effectively under more relaxed criteria. Factors such as physical constraints, network topologies, and computational tasks play a crucial role in shaping these requirements. As we explore tighter real-time constraints, the complexity of communication configurations and code handling increases. However, when real-time requirements are more relaxed, it opens up opportunities for simpler, more streamlined system designs without sacrificing functionality or reliability.

Although both safety and security have been addressed with (by CANopen Safety and CANcrypt) there is currently no standardized solution that provides both. The CiA (CAN in Automation) user's group currently has multiple working groups reviewing various aspects of both safe and secure communication with CAN CC (classic), CAN FD, and CAN XL.

Real-time capabilities of CAN

CAN's real-time effectiveness is closely tied to its communication speed, and further affected by its priority-based arbitration mechanism. Calculating CAN frame transmission times is not a straightforward task; the time depends on both the number of data bytes and their content. This complexity arises because stuff bits may be added to a frame depending on its data. Therefore, the following determined values should be considered as approximations, providing a general sense of the scope at hand.

It's important to remember that your maximum bit rate also depends on the physical topology of the cabling, and depending on your application, the total transfer required for a single control cycle might include two transmission paths: one for input data to the control unit and another from the control unit to the outputs.

Though the author focuses on CAN CC here, most of the following considerations also apply to the CAN FD (flexible data rate) and CAN XL variants. Both of these protocols feature a dual bit rate mechanism, further enhancing their data throughput capabilities. However, when discussing timing-related dynamics, most of the considerations outlined here predominantly apply to the "nominal bitrate." This foundational bit rate essentially establishes the pace for control information such as arbitration, acknowledgment, and error signaling. For those using CAN FD and ▷

CAN XL, it's crucial to be aware of the additional complexities introduced by the "data phase bitrate," which governs the transmission of the actual data. One of the key concerns in these systems is determining the maximum duration a lengthy message might occupy the network and how much longer this delay might be compared to the longest CAN CC (classic) frame with 8 bytes.

At its maximum speed of 1 Mbit/s, CAN CC allows for the exchange of more than ten frames within a millisecond. Conversely, at a modest rate of 125 Kbit/s, it averages around one frame per millisecond. Beyond mere transmission times, signals or frames can experience delays if higher-priority communication is in the queue. To put it simply, the worst-case transmission time would be the sum of the frame's own transmission time and the delay expected from the longest sequence of higher-priority traffic in the system. This assumes that all communication happens on a single CAN network. If signals need to be forwarded via bridges or gateways, delays become longer and even more challenging to predict.

The system of message prioritization can be a double-edged sword. However, there is a mitigating factor: by strategically limiting the duration of sequential high-priority traffic, even communications with the lowest priority can be dispatched with minimal delay. This approach ensures consistent and timely data exchange throughout the system.

Looking at CAN (and the FD and XL variants) by itself, it is clear that “as is” it is not deterministic. A single device producing high priority frames can block the communication for all others. To make CAN deterministic, we need to ensure a controlled frame triggering – when may which CAN ID be used. To activate CAN's real-time capabilities, the following design goals should be considered. While these guidelines may vary based on application specifics, they serve as a reliable starting point:

A) Aim to keep the overall busload at a level where even low-priority frames have sufficient time to access the network. While the exact threshold can vary by application, the author’s initial recommendation is to stay below 75 % busload (less if communication is purely change-of-state-based).

B) Ensure that no individual node can generate an extended stream of consecutive messages. Some drivers offer a transmission "throttle" to limit the maximum transmission rate.

C) For those seeking finer control over transmission timing and sources, consider the SYNC mode of CANopen. This mode enables trigger messages, providing enhanced control over transmission schedules, allowing trigger modes like those used by time triggered systems.

Mastering temporal dynamics

After exploring the various use cases and their respective temporal demands of CAN-based systems, you can imagine that matching CAN configurations to specific time requirements is both an art and a science. Table 1 summarizes some of the main numbers and factors to consider. The first section of the table gives a summary of the CAN timings and throughputs that you can expect at various bit rates – this is all for CAN CC using 11-bit CAN identifiers. The fact is that even at the lower bit rates we are still talking about potentially thousands of CAN frames per second. There is sooo much room for communication that one can easily grasp that with some well-defined parameters on how to use all this “space”, one can very well design real-time capable systems.

Table 1: Ballpark figures for transmission delays (Source: EmSA)

CAN Timings and Throughput				
CAN Bitrate	1000kbps	500kbps	250kbps	125kbps
Shortest 11bit CAN Frame	50us	100us	200us	400us
Longest 11bit CAN Frame	130us	260us	520us	1040us
Max shortest frames per second	20000	10000	5000	2500
Max longest frames pers second	7700	3850	1925	960
Max data throughput 70% load	400kbps	200kbps	100kbps	50kbps
Possible transmission delays (ms)				
CAN Bitrate	1000kbps	500kbps	250kbps	125kbps
Network in use	0.13	0.26	0.52	1.04
Arbitration delay for 5 frames	0.5	1	2	4
Transmission throttle delay	0.5	1	2	4
Sum	1.13	2.26	4.52	9.04
MCU/CPU Processing delays				
Higher Layer Stack Processing	fractions of milliseconds			
App processing RTOS	fractions of milliseconds			
OS (non RT) minimal proc. Delay	single milliseconds			
OS (non RT) minor delay	some 50 to 100ms			
OS (non RT) mayor delay	can be above 1s			

The next section of the table shows potential transmission delays and depends on many factors. Therefore, it is only a rough estimate for a specific use case, you need to adapt it to your own use case. The first row shows the delay even the highest priority will have, if the bus is currently in use (arbitration already started, transmitter is too late to join). Transmission has to wait, until the current frame is completed. The second row shows an arbitration delay – if there are other devices also trying to transmit a frame, how long do we have to wait? Here is shown the delay for five other frames currently pending for transmission and having a higher priority followed by a line of further delays, if a throttle mechanism is used protecting from back-to-back transmissions. Further on in this article will be reviewed what can be done if the sum of delays shown is unacceptable in your application.

The last section of the table shows the potential processing delays caused by executing various code on the device handling the CAN communication. Here is assumed that a modern 32-bit MCU (micro-controller unit) with an integrated CAN CC interface is used, running at 80 MHz or faster. In such environments, the code execution directly related to handling the CAN frames is typically marginal. Potential delays come from “what else is happening” on that MCU.

Translating this knowledge into real-world system performance requires actionable strategies and considerations. With the previously established benchmarks from Part I – seconds, 100 ms, 10 ms, and single milliseconds – as the guideposts, let's review practical recommendations for optimizing your CAN-based systems.

Mastering CAN applications with response times beyond seconds

In the domain of applications operating with delays stretching into seconds or even minutes, designing ▷

CAN-based systems to meet these response times is not particularly challenging. Interestingly, even a device burdened by sub-optimal drivers or firmware might still be suitable, as even sub-optimal drivers will even still perform within 10 to 100 of milliseconds.

Figure 1: For applications operating with delays stretching into seconds, even a device burdened by sub-optimal drivers or firmware might still be suitable (Source: EmSA)

However, when working with devices that rely on non-real-time operating systems, the challenge lies not so much in countering communication delays, but rather in upholding consistent performance and avoiding the worst-case possible delay. Regular testing and thorough monitoring are essential to ensure that these devices never falter in allocating the necessary resources for seamless CAN communication. It is also crucial to proactively curb any potential system disruptions. Simple yet effective measures, such as ensuring the absence of updates or other resource-draining operations during active communication periods, can strengthen the system's responsiveness and reliability.

Mastering CAN applications with response times of 100 ms

This domain is where the potential of CAN, in synergy with higher-layer protocols like CANopen, truly comes into play. The CANopen PDO (process data object) communication mechanisms inherent in CANopen provide users with flexible control, simplifying the configuration of message content and triggering. These PDOs facilitate real-time data exchange between nodes, optimizing communication efficiency.

At this response time, CAN-ID assignment and overall busload remain critical, but not overwhelmingly so, as they are unlikely to cause delays approaching anywhere near 100 ms. The system architecture should be designed such that even frames with the lowest priority have timely bus access, ensuring their transmission within the stipulated timeframes. As we navigate this middle ground, it becomes increasingly important to review potential high-priority frame bursts. Back-to-back high-priority transmissions

Figure 2: While many non-real-time operating systems can still achieve a 100-ms response, it is advisable to lean towards a real-time operating system (Source: EmSA)

can dominate the bus, posing risks of delays for lower-priority messages. Effective strategies for avoidance or control, such as limits on what each node can transmit per timeframe or synchronized triggering, can be employed to mitigate these bursts, ensuring more predictable and harmonious network communication, even as the system scales.

While many non-real-time operating systems (OS) can still achieve a 100-ms response, it is advisable to lean towards an RTOS (real-time operating system) in such scenarios (if an OS is required at all, many simple I/O devices typically do not have an OS at all). Using an RTOS aligns naturally with the demands of a 100-ms response window. If a non-RTOS is chosen, rigorous and extended testing becomes imperative to ensure the OS consistently meets the desired response times under all conceivable operational circumstances.

Within this 100-ms response time framework, the software and firmware requirements remain relatively forgiving. Specific optimizations are often unnecessary; even drivers or stack implementations deemed sub-optimal in high-performance environments (for example not taking advantage of CAN controller hardware features for advanced filtering and buffering) can adequately serve the purpose.

Mastering CAN applications with response times of 10 ms

As we move into the 10-ms response time zone, precision and control over every system component becomes essential. This is where detailed scrutiny of network data flow is essential.

Time-triggered networks, optimized for hard real-time applications, are often the preferred choice in such demanding scenarios. The CANopen SYNC mode is an effective approach to mimic communication behavior as used in time-triggered communication systems. By utilizing SYNC triggering messages, it enables specific nodes to transmit their associated PDO messages at precise moments, bringing predictability and consistency to system communication.

While an RTOS might seem ideal for such tight timing requirements, it comes with its set of challenges. An RTOS offers a range of configuration options, and managing these tasks requires careful coordination. Within the tight 10-ms window, the process involves a sensor sending its current data, a control device with an RTOS receiving and processing this data, and then acting upon it.

However, simply implementing an RTOS does not guarantee the desired outcomes. Task prioritization and configuration must align perfectly with the system's stringent timing requirements. Additionally, a detailed review of driver functionality, firmware, and stack structures is crucial. Potential issues, such as priority inversion where a low-priority message in the queue might delay a higher-priority one, need to be addressed. Highly optimized drivers can address priority inversion, but this may cause changes in transmission sequences. A change in transmission sequences can be problematic for certain higher-layer protocols and needs to be reviewed carefully.

Mastering CAN applications with response times of 1 ms

Venturing into the 1-ms response time territory for CAN-based applications is akin to treading on the edge of the protocol's capabilities. These applications truly push the boundaries, requiring an unparalleled level of optimization and attention to every detail.

At this threshold, conventional approaches and tools often prove inadequate. Even some RTOSs, which typically excel in managing real-time tasks, may struggle to consistently adhere to this tight window. This necessitates reliance on micro-controller-specific implementations, where most tasks are handled directly within interrupt service routines, bypassing the typical layers of an RTOS.

Figure 3: Venturing into the 1-ms response time territory for CAN-based applications is akin to treading on the edge of the protocol's capabilities (Source: EmSA)

The extreme precision required at this level means that many system configurations may need to be hardcoded, potentially bypassing higher-layer protocol stacks like CANopen that would otherwise delay processing. This also helps avoid potential delays introduced by configuration handling, ensuring maximum predictability. Every component, message, and byte transmitted on the network must be judiciously managed. Given the stringent requirements, if a 1-ms response time is a necessity for your application, it is wise to review if other communication solutions beyond CAN might be better suited to your needs. This domain requires significant commitment in terms of development time, testing, and optimization. If this endeavor is taken on, one should be fully prepared for a time-consuming project journey.

Conclusion

The exploration of the temporal dynamics of CAN-based systems has underscored the adaptability and capabilities of the CAN protocol across various response time requirements. For applications with response times extending beyond seconds, there is less emphasis on precise timing, and decisions regarding CAN-ID usage, higher-layer protocols employed, or the operating system selected generally have a less pronounced impact on performance.

Figure 4: In conclusion, understanding the balance between application requirements, the inherent strengths of CAN, and the related temporal constraints is vital (Source: EmSA)

However, as we reach into tighter time constraints of 100 ms and 10 ms, system design considerations become of greatest importance. These include total busload, message priorities, and the strategic employment of functionalities like CAN-open's SYNC mode. When navigating the demanding 1-ms response time domain, every element of the system requires meticulous attention and may even prompt a re-evaluation of the network system selected.

In conclusion, understanding the balance between application requirements, the inherent strengths of CAN, and the related temporal constraints is vital. It's this knowledge that empowers CAN system designers to make informed decisions across diverse temporal scenarios. ◀

Author

Olaf Pfeiffer
EmSA (Embedded Systems Academy)
info@esacademy.com
www.esacademy.de