Martin Gergeleit, Hermann Streich[1]

Implementing a Distributed High-Resolution Real-Time Clock using the CAN-Bus[2]

Many time critical applications, e.g. measurement devices, require a real-time clock with an accuracy in the order of microseconds. In a centralised system this is easy to implement with standard timer devices, but in a distributed system (like a number of sensor and actor nodes connected via the CAN-bus) this is more difficult as there is no global system tick. This problem can be solved by synchronising the local clocks of all nodes with a sufficient accuracy. The tight timing guaranties of a CAN-network offer a simple and cheap possibility to provide such a global clock without additional hardware.

The real-time group of GMDs CREW Project has designed and implemented a clock synchronisation protocol on the CAN-bus that provides a global time base with an accuracy of about 20 microseconds. The protocol is simple and hardware-independent. It uses only a small amount of bandwidth (< 20 messages/second) and works with a single, arbitrary CAN-object. If necessary, e.g. in large scale networks, the protocol can be synchronised with an external time-base, like a GPS satellite receiver.

Introduction

Many time critical applications, e.g. measurement devices, require a real-time clock with an accuracy in the order of microseconds. In a centralised system this is easy to implement with standard timer devices. But in a distributed system, where accurate time may be needed in different physical locations, this is more difficult to achieve as there is no global system tick. A protocol that synchronises accurate local clocks in the cooperating nodes provides a solution to this problem. A number of such protocols for networks of workstation-like computers have been proposed so far (like TEMPO [Gus86] and DCNET [Mil81]). As these solutions have to deal with different network topologies and a-priori unknown message latencies, they seem to be too heavy-weight to be ported to an environment of embedded low-cost controllers. For these applications a solution has to be provided that takes advantage of the properties of the existing system and thus requires no extra hardware and only a small amount of additional resources.

Today's microcontrollers typically have a build-in timer that provides a local clock with a resolution in the order of microseconds and with an accuracy of about 10^{-5} to 10^{-6}. This means that an unsynchronised local clock may drift 1 to 10 microseconds per second compared to an absolute external time. Thus, in a network the differences in the local clock values can reach 1/100 of a second within several minutes. For applications where cooperative activities have to occur simultaneous or where distributed measurements have to be coordinated via local time-stamps such a clock behaviour is unacceptable. The problem can be solved by synchronising the local clocks of all nodes with a sufficient accuracy. In an environment of embedded controllers this could be done with an additional wire, but if the devices are connected by a field-bus anyway, it is the simpler and more flexible alternative to utilise this connection. In addition, the tight timing guaranties of a field-bus network like the CAN-bus[Bosch] assist in the implementation of a cheap and robust synchronisation protocol that can provide the necessary accuracy.

---

[1]GMD, Schloß Birlighoven, 53754 Sankt Augustin, Germany, e-mail: {gergeleit, streich}@gmd.de

[2]published in: Proceedings of the 1st international CAN-Conference 94, Mainz, 13.-14. Sep. 1994, Can in Automation e.V., Erlangen

The System

This paper proposes a simple protocol that synchronises the local clocks of nodes connected to a field bus, especially a CAN-bus. All participating nodes have local clocks with possibly different accuracies. The local clocks are probably implemented using the usual timer facilities of the standard microcontrollers. Thus, the typical accuracy of these clocks is in the order of microseconds. One clock on the network is assumed to be the dedicated 'master' clock. The remaining clocks in the network, called 'slaves', are synchronised to the value of the master clock. The master can be a free running clock if only an internal synchrony of the connected nodes is desired. If instead synchrony with the absolute time of the real world is needed, the master has to be synchronised to an external time provider, like e.g. a GPS satellite receiver.

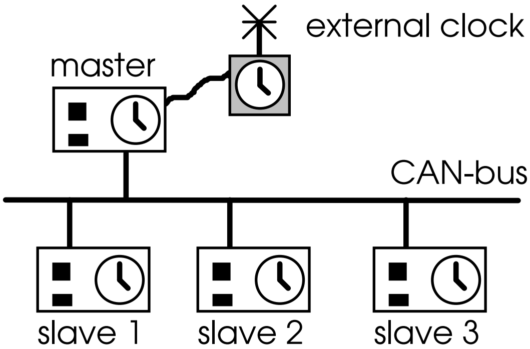

Figure 1: The system configuration

At least the slave clocks have to provide some means for adjusting their time value with a delta-value. In the most simple case this can be done by simply adding or subtracting the delta-value, but this results in non-monotonous time values and sudden jumps of the clock. In applications where this is not acceptable a smooth clock adjust has to be provided. This means that the clock virtually speeds up or slows down until the delta-value has been compensated. This can be implemented either by really changing the clock rate or, more simply, by leaving out or repeating single clock ticks (e.g. during timer overflow processing). In our test environment we are using adaptive clocks. This means the clocks try to adjust themselves, if they detect that they have a certain drift. They account for their adjust values. If these values are always positive, which means the clock runs too slow (or always negative; the clock runs too fast), then the clock increases or decreases its speed permanently. This is done again by leaving out or repeating single ticks.

The Protocol

The protocol relies on three basic assumptions about the network, the controller hardware, and the involved software. The presented approach is not restricted to the CAN-bus, but it can be used on every network that fulfils the following requirements:

1. A successfully sent frame arrives at all nodes with a fixed and known delay. This means, the underlying network has to be a broadcast network. The delay may be approximated as zero, constant, or even a function of the receiving node.
2. The delay from the transmission and reception of a frame to an interrupt service routine that time-stamps this event is known and has only a very small variance. Again, this delay and its variance may be functions of the receiving node.
3. A time bound for the maximum time between two valid synchronisation messages t_{max} can be guaranteed.

These assumptions hold for a CAN-bus network. The CAN-bus is a broadcast network and its physical specification guarantees, that the bus length is always significantly shorter than the length of one transmitted bit on the bus. This means, that the capacity of the CAN-bus is zero and that all connected nodes will see a logical bus level at about the same time. This property is inherent to all possible physical CAN-layers, as e.g. the CAN-bus arbitration method requires this degree of synchrony between all participants on the network. Thus the delay on a CAN network can be approximated as zero (this is not true for other networks, like e.g. Ethernet).

The second condition has to be ensured by the hardware design and the software of the connected nodes. Standard microcontroller-based solutions using the currently available CAN-bus controller chips [Int89, Phi90] can fulfil condition two, provided that the software guarantees a fixed interrupt latency. But even when a fixed latency can not be guaranteed under all circumstances but only with a certain variance, e.g. when another devices requires the highest interrupt priority, the clock synchronisation still works with a degraded accuracy.

Condition three has to be fulfilled by an appropriate capacity planning. In CAN terms this has to be expressed by the priority of the synchronisation message's identifier. Note, that this does not imply the highest priority for the synchronisation message. Also the number of unsuccessful transmission attempts (lost bus arbitrations or errors) is not critical to the protocol, as long as at least a maximum time interval between two successful transmissions can be guaranteed.

The most simple clock-synchronisation protocol one can think of (figure 2): the master takes its current time value (t1), broadcasts it (t2), all slaves receive this message (t3), and adjust their clocks accordingly (t4), has a major drawback: the complete path from getting the masters clock value, sending it over the network and obtaining the local clock value (t1 to t4) is time critical. The slaves have to know the latency of this path exactly in order correct the received time value. At first, the path must be deterministic, this means no interrupts or lost bus arbitrations must happen during its execution. But still the timing of the path execution is hard to predict, as it depends not only on static parameters of a certain configuration, such as the transmission speed of the network and the crystal speed and the local software of the master and the slave nodes, but also on the bit-pattern of every single message (because of the CAN bit-stuffing rules). In addition, the length of this path is huge compared to the accuracy of synchronisation that should be achieved with the protocol (several microseconds). The transmission of a time-stamp (8 bytes) with a typical bit-rate of 125 KBaud takes about 1 millisecond and even at maximum transmission speed of 1 MBaud the critical path is still at least a magnitude larger than the desired accuracy. Obviously it is hard to limit all possible inaccuracies on this path and to compute the correct latency from the master to the receiver. In the following we will describe a protocol, that relies on less stringent preconditions and a reduced critical path. It does not need more messages or computations that the one described above.

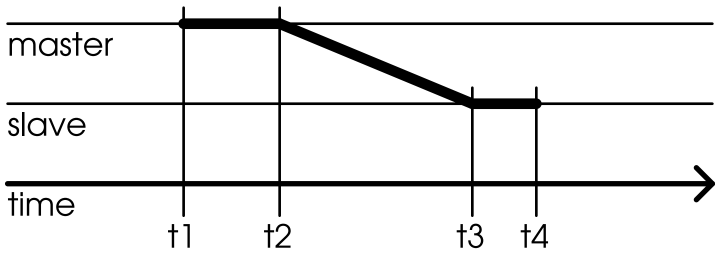

Figure 2: The simple synchronisation protocol

The main idea of a refined protocol is to use the synchrony on the CAN-bus, while minimising the length of the time critical path. Consider the following protocol (figure 3): An arbitrary node x broadcasts an certain indication message (t1). Each participant of the protocol receives and recognises this message (t2) and takes a local time-stamp right after the reception (t3) (Note, that figure 3 shows the special case, where the time needed for time-stamping the incoming message is the same on the master and the slave node.). Then the master node sends another message containing the masters time-stamp for the latest indication message (t4). Now each slave can compare its local time-stamp with the one received from the master. The difference between these values determines the amount of time for the adjust of the local clock (t5). In this protocol only the path starting with the correct reception of the indication message to getting the time-stamp for this is time critical (t2 to t3, denoted with thick lines). This path is a small sub-path of the one described for the simple protocol above. The latency of the path's execution only depends on parameters local to the slaves, such as the crystal speed and the local software. It is independent of network parameters and the masters properties. Thus, the time for this latency (t3 - t2) has to be determined once. Now it can be subtracted from the local time-stamps (taken at t3) in order to compute the corrected time for the reception of the indication message (t2).

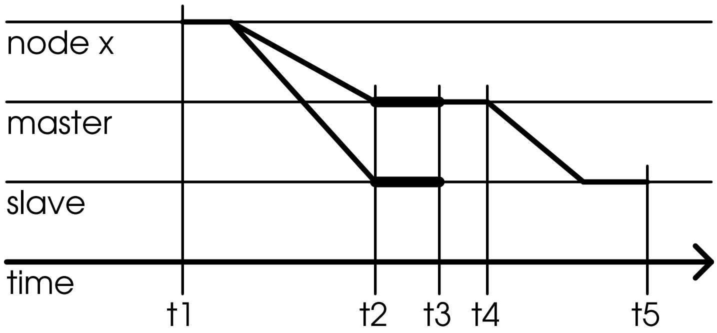

Figure 3: The refined synchronisation protocol

This protocol still needs about twice the bandwidth of the simple protocol to achieve a comparable accuracy, but this can be easily optimised. Our proposed protocol joins the two needed messages into one (figure 4). The masters time-stamp for the latest synchronisation round now serves as new indication message. The master does not take its time-stamp upon reception of an indication message but after a successful transmission of a synchronisation message. Again, this latency only depends on local parameters of the master node. In a CAN network the transmission success indication can be considered as synchronous to the reception of the message on the slave nodes (t2). Now the protocol needs the same amount of messages as the simple protocol. In both cases the messages contain the same contents, the master's time-stamp, but in the new protocol the time-stamp belongs to the previous message.

As shown in figure 4 the distances t3-t2 and t3'-t2' are fixed and known. Also t_{max} (i.e. t2' - t2) the maximum time between two synchronisation messages is given. The slaves clock is synchronised at t4 and t4'. As there are no assumptions made about the distance of the other points in time the interval between t4 and t4', the time where the slave clock may drift away from the masters time, is surely bound by 2(t2' -t2).

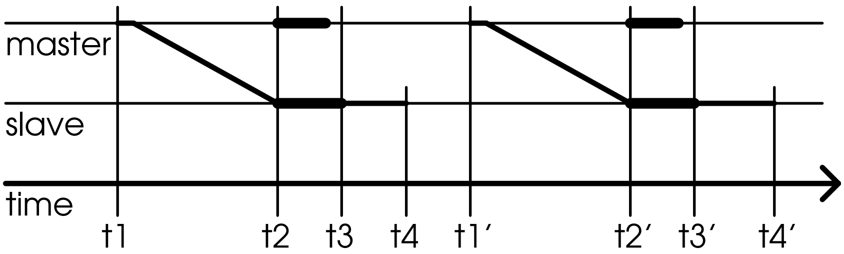

Figure 4: Two synchronisation rounds of the optimised protocol

Thus, after an initial synchronisation phase, the protocol guarantees, that

$$\left|t_s - t_m\right| < 2 t_{\max} \Delta_s$$

where:

t_s	is a time-stamp taken at absolute time t from slave clock s,
t_m	is a time-stamp taken at absolute time t from the master clock m,
$t_{\max}$	is the maximum time between two valid synchronisation messages, and
Δ_s	is the drift of the slave clock s.

This implies that the absolute difference between two clocks of the distributed system can be made as small as an application requires by simply increasing the rate of synchronisation messages. This is true in an ideal system, where the time-stamp for sending and receiving the messages can be obtained with an arbitrary accuracy. In any implemented system there are valuesδ_x^{sys}, that describe the system dependant inaccuracy when determining the reception or transmission time of a message, caused by the clock, the interrupt processing or the hardware. Thus, the possible synchrony between two clocks is bounded by $\delta_s^{sys} + \delta_m^{sys}$, which depends on the actual implementation.

$$\left|t_s - t_m\right| < 2t_{\max}\Delta_s + \delta_m^{sys} + \delta_s^{sys}$$

where:

δ_s^{sys} is the maximum error for obtaining a time-stamp for an incoming message at slave s and

δ_m^{sys} is the maximum error for obtaining a time-stamp for a sent message at master m.

The protocol also synchronises slaves, that join the system with an arbitrary clock value. The speed of their synchronisation depends only on the properties of the slaves local clock (how fast it can be adjusted to a given time value and whether it is allowed to jump (forwards or backwards)). The protocol can tolerate a crash and rejoin of any slave and also single message losses but not a failure of the master clock.

The Implementation

The described synchronisation protocol has been implemented at GMD, the German National Research Center for Computer Science. The test configuration consists of a number of Intel 8051-microcontroller boards connected via a 1 MBaud CAN-bus. The internal timer of the microcontroller is used as local clock. Some of the boards are running with 16 MHz crystal speed, others with 12 MHz, resulting in a maximum clock resolution of 0,75 and 1 microseconds. The master node is connected to a GPS satellite receiver, thus having the exact global time.

One board is used for measurement, only and

- generates events (CAN-messages), which are time-stamped by all other boards, including the master clock, and
- collects and compares the time-stamps.

Without the synchronisation protocol the clocks drifted up to 10 microseconds per second (Δ_s). So, with synchronisation using a frequency of 20 Hz, (resulting in a $t_{\max}$ of 50 milliseconds, if no errors occur and no higher priority messages are sent at the same time), $t_{\max}\Delta_s$ is 500 nanoseconds.

The measured differences of the local clocks are in the range of 20 microseconds. It follows that the main source of inaccuracy is $\delta_s^{sys} + \delta_m^{sys}$. In fact, the reading of the local clocks has an non-negligible inaccuracy. This is due the fact that

- each local clock generates a timer interrupt on overflow of a timer register. In our implementation the overflow occurs each 10 milliseconds. If the timer interrupt and the receive interrupt for the synchronisation message occur simultaneously, the receive interrupt is delayed. In such cases the local clock can be read as soon as the overflow interrupt handling has finished. In our implementation this leads to a variance of about 15 microseconds,
- read-errors influence the adjustment of the clocks. In can happen, that a well synchronised clock is readjusted to a worse speed because of a reading error.

Both problems are extermely hardware dependent and can be avoided by the usage of better clocks.

Current Work

The existence of a synchronised clock does not imply the possibility of getting high precise time-stamps for arbitrary events. Currently the interdependencies between bus-traffic, processor load, and the time-stamping mechanism is observed. Question like, "How relate message priorities to accuracy of measurement?" or "How does accuracy depend on capacity-planning?" should be answered. If e.g. an incoming CAN-message should be time-stamped and this message has a higher priority than the synchronisation messages, $t_{\max}$ becomes larger, because the synchronisation messages are lost. In order to guarantee a worst-case $t_{\max}$ either additional capacity planning is needed or the protocol gets more complicated (e.g. changes priorities) also by the cost of accuracy. Furthermore, if a synchronisation message arrives shortly after an other arbitrary CAN-message, it is possible, that the receive interrupt for the synchronisation message is delayed (but accepted). This must be detected by the protocol because the delay may be up to about hundred microseconds, which is spent by the controller to free its buffers.

References

[Bosch] Robert Bosch GmbH, CAN - Controller Area Network Funktionelle Beschreibung

[Gus87] Gusella and S. Zatti: "TEMPO - Time Services for the Berkley Local Network", Report No. UCB-CSD83-163, Computer Science Division, University of California, Berkley, CA, Dec. 1987

[Int89] INTEL 82526 Serial Communication Controller Architectural Overview, Order Number 270678-001, Jan. 1989

[Kle93] U. Kleinhans, J. Kaiser, K. Czaja: "Spearmints: Hardware Support for Performance Measurements in Distributed Systems", IEEE Micro, Vol. 13, No. 5, October 1993, pp. 69-78

[Mil81] D.L. Mills: "DCNET Internet Clock Service", RFC-778, Defense Advanced Research Projects Agency, Information Processing Techniques Office, April 1981

[Phi90] Philips PCA 82C200 Stand-alone CAN-Controller, Product specification, Oct. 1990