Time Scheduled CAN Systems

by
Lars-Berno Fredriksson
CTO CanKingdom Int.
030612

Abstract

Distributed embedded control systems, especially safety critical ones, have some common requisites:

1. They should act and react on events by messages via a network communication
2. There are three classes of events
   a) Events predictable in time
   b) Events predictable in sequence but not in time
   c) Events unpredictable in time
3. The system behavior should always be predictable in the time domain

CAN has been used successfully for several years for distributed embedded control systems thanks to its robustness, efficient error control, efficient handling of colliding messages, message consistency throughout the system, etc. It handles event-triggered messages very efficiently. However, it has not been regarded as suitable for safety critical systems as it has no built-in services for guaranteeing the timeliness of messages. Therefore other concepts based on time-triggered communication protocols have been suggested, such as TTP and FlexRay. From a time predictability of messages appearing on the bus, this direction is understandable but basically, the timing problem is related to the system behavior, not the communication. A time-triggered communication is not well suited for rapid deliveries of messages triggered by unpredictable or sequential events such as 2 a) and b) above. An event-triggered communication is much more efficient. Further, an event-triggered communication can easily be turned into a time-triggered behavior at the system level by having events generated by applications related to one or more clocks. The communication layer is event-triggered and guarantees that transmitted bits and bytes are received correctly. The timing of the transmissions and the interpretation of the bits and bytes are taken care of at the system layer.

This article will describe a design concept for dependable and safety critical systems using CAN for the communication. The main steps in the concept are:

1. Modeling the system using virtual clocks
2. Verifying that the model fits the timing requirements of the system
3. Finding solutions that make the system work according to the timing of the virtual clocks
4. Designing the real system
5. Verifying and validating behavior of the real system

The result of applying the concept is that modules can be developed without any knowledge of the target system. The development work on the system and modules can to a great extent be separated. The timing of the communication can be fully controlled with no or limited time support at the module level. Some examples are provided to illustrate the concept and direct the minds toward efficient solutions.

Distributed embedded control systems, especially safety critical ones, have some common requisites:

1. They should act and react on events by messages via network communication
2. There are three classes of events
    a) Events predictable in time
    b) Events predictable in sequence but not in time
    c) Events unpredictable in time
3. The system behavior should always be predictable in the time domain

CAN has been successfully used several year now for distributed embedded control systems thanks to its robustness, efficient error control and handling of colliding messages, message consistency through out the system, etc. It handles event-triggered messages very efficiently. However, it has not been regarded as suitable for safety critical systems as it has no built-in services for guaranteeing the timeliness of messages. Therefore other concepts based on time-triggered communication protocols have been suggested as TTP and FlexRay. From a time predictability of messages appearing on the bus, this direction is understandable but basic the timing problem is related to the system, not the communication. A time-triggered communication is not well suited for rapid deliveries of messages triggered by unpredictable or sequential events as 2 a) and b) above. An event-triggered communication is much more efficient. Further, an event-triggered communication can easily be turned into a time-triggered behavior at the system level by having events generated by applications related to one or more clocks. The communication layer is event-triggered and guarantees that transmitted bits and bytes are received correctly. The timing of the transmissions and the interpretation of the bits and bytes are taken care of at the system layer.

This article will show a design concept for dependable and safety critical systems based on CAN for the communication. The main steps of the concept are:

1. Modeling the system by using virtual clocks
2. Verifying that the model fits the timing requirements of the system
3. Finding solutions that make the system work according to the timing of the virtual clocks
4. Designing the real system
5. Verifying and validating the real system behavior

The result of applying the concept is that modules can be developed without knowledge about the target system. The development work on the system and modules can to a great extent be separated. The timing of the communication can be fully controlled with no or limited time support at the module level. Some examples are provided to illustrate the concept and direct the minds toward efficient solutions.

Time Scheduled CAN Systems

by

Lars-Berno Fredriksson

CTO CanSystem Int.

030201

1 Introduction

A definition of a Hard Realtime System is one where the maximum delay of any information transfer is predictable and calculable. It has been shown that this requirement can be obtained in any event-triggered CAN system if the maximum repetition rate of any message is known[1]. The theory is based on the calculation of worst case, i.e., at an instance when the bus is free and every message in the system would be initiated for transmission at the same time. The probability of this occurring decreases with the number of modules in the system and the number of messages and applying the theory results in poor utilization of the available bus bandwidth. The remedy is to switch to a time scheduled message transfer. CanKingdom supports a global clock, so when this is implemented, it is no problem to schedule messages according to traditional techniques. The bandwidth can be further increased by taking advantage of the collision resolution mechanism in CAN to allow some unscheduled messages, e.g., alarm messages, to take part in the communication, and using the Action/Reaction option in CanKingdom can allow clock-less modules to be utilized in the system. Some solutions are sketched in the article “CAN for Critical Embedded Automotive Networks” in the July-August 2002 issue of IEEE Micro[2]. The purpose of this article is to take the time scheduled concept with CAN yet another step forward and show different ways to use the inherent properties of CAN, resulting in an efficient use of the bus bandwidth using modules supporting different clock and clock synchronization alternatives, ranging from highly sophisticated clock solutions down to no clock support at all.

The idea using time scheduled communication in CAN systems is not new. Bosch has developed Time-triggered CAN (TTCAN) and this is soon to be a part of the ISO 11898 standard. It is generally believed that it is better for hard realtime systems to be time-triggered than event-triggered, i.e., that measurements, event detections and message transmissions are made at specified points in time. Further, time-triggered systems are regarded as synonymous to time scheduled systems. These two beliefs will now be questioned: An event-triggered but time scheduled approach is a better concept. The main goal of *time scheduling* CAN messages is to make the communication behavior predictable, to shorten message latency and to make best possible use of the bandwidth. To reach this goal, some generally accepted rules for *time-triggered* communication should be violated. We can safely do so by using some CAN features and the spirit of CanKingdom: The system designer should set the most efficient rules for his system. We will end up with a handful of building blocks for setting up such rules for an event-triggered but time scheduled system rather than a time-triggered system.

The key objective of control systems is to detect and react on events. The reaction on events is to create new events. In distributed embedded control systems, the communication must at the very least transfer information about events in the nodes fast enough for them to react in a proper way to keep the system stable. Thus, for control systems, the basis for message scheduling should be primarily based on events related to time, not on time related to events. In any control system, mechanical as well as electronic, there is a delay between the detection of an event and the system response to the event. The delay can be divided into parts, an undesired part and a desired part. The undesired part is given by the selected construction elements and it is a design task to get the desired delay with the right accuracy, taking in account the undesired delay. This is obvious for

mechanical control systems. Construction material properties such as strength, stiffness, density, speed of sound, etc. contribute to the undesired dely and sets the limits for the design possibilities for structures as linkages, springs, gearboxes, etc. in systems to obtain the proper delays in sequences controlling a machine, e.g., a steam or gas engine. The mechanical system also clearly shows the difference in nature between the desired and undesired latency. The desired latency is often a function of some kind of a reference speed but the undesired latency is only a function of the physical time. Most distributed embedded control systems are designed to replace mechanical systems. The problems are the same but the means are different. Construction material properties are replaced by digital protocols, signal propagation, clock accuracies, etc.

2 Time Scheduled CanKingdom (TSCK)

CanKingdom is based on a design concept for distributed embedded control systems with a clear separation between system design and module design. Although there has been some support for time-triggered systems[1] in CanKingdom, there has not been any real support for time scheduling in the concept. This will now be added.

2.1 The TSCK concept

A basic idea behind TSCK is to organize events in time. **Figure 1** shows the model. An event at module 1 generates some kind of a signal that can be processed by a computer task. In this case it is a temperature sensor that generates a signal that the task finds is above a threshold. The task

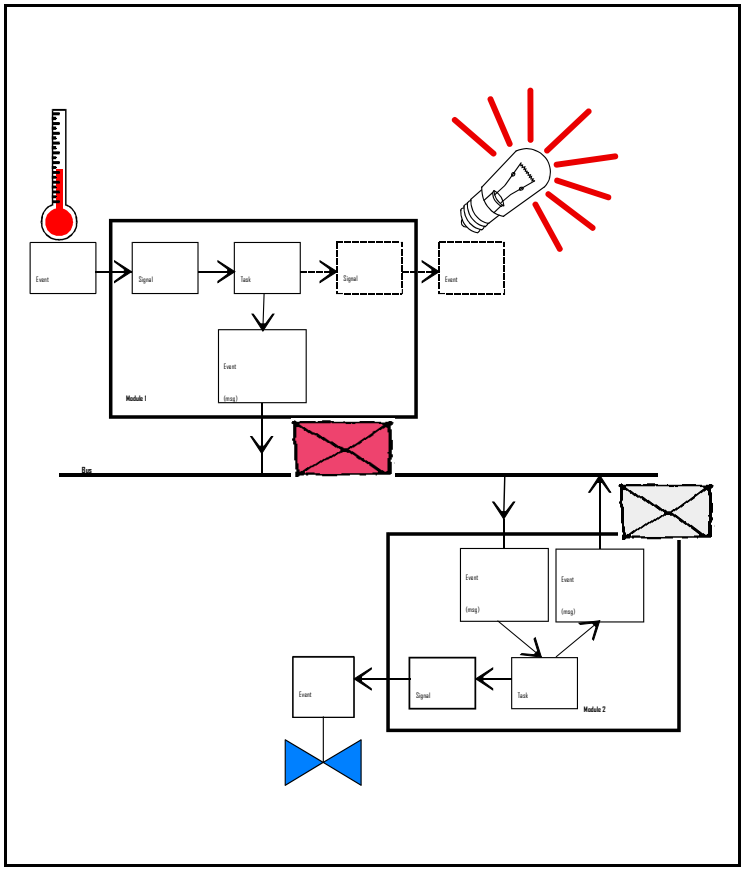

Figure 1 Model of a TSCK system

[1]Chapter 5 System Time
Chapter 6 Bus management
The **King's Page 10.** (Optional) Minimum time elapsing between two consecutive transmissions of the same CAN identifier.
The **King's Page 11**. (Optional) Circular Time Base Setup Page.
The **King's Page 12**. (Optional) Repetition Rate and Open Window Setup Page.

generates two responses: A signal that creates a first response event; turn on the warning lamp, and a second response; transmission of a temperature message (the message transmission is also an event). Module 2 receives the message and the task reacts by increasing the cooling liquid flow. The message reception and the increased flow are events. A counter reaching a certain number is also an event, regardless of whether the pulses are generated by a cogwheel or an oscillator. The goal for TSCK is to help the system designer to relate detected events in one module to reaction events in one or more other modules within defined deadlines by means determined by the various module designers. The process is divided into three steps:

1. Create a virtual schedule for events based on a virtual time for the System.
2. Break down the Virtual System schedule into real Module Schedules based on real time.
3. Download the Module Schedules to the respective module.

This process is very similar to project planning and PERT/CPM or GANTT methods and tools can to advantage be used here In the first step, the system designer creates a System Schedule. He schedules all foreseeable major tasks and events required for the system performance according to a virtual time line, generated by a virtual clock. In the next step, he breaks down the System Schedule into Module Schedules related to a local time. The local notion of time can be different in different modules but it is always sufficiently related to the system time to achieve the system performance (**Figure 2**). The local schedules are “downloaded” to the respective module at system start-up or during runtime, partly as specifications to the module designers and partly as control messages (King’s Letters).

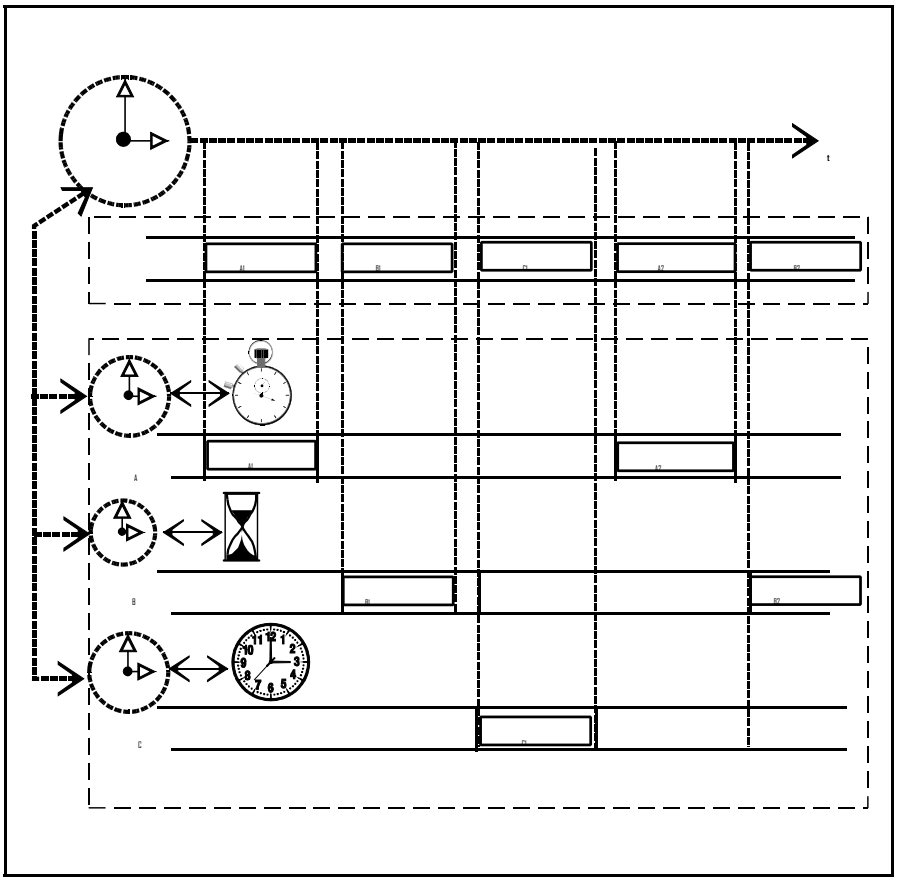

Figure 2 TSCK development process. The system designer schedules the whole system according to a virtual system time. The schedule is broken down to module level. The notion of time can be different in each module but is always related to the virtual system time.

Each module will then have a local schedule where events are related to time in some way. Events includes transmission and reception of messages and thus the communication is an

integrated part of the control algorithms. The sum of all the module schedules will make up a real system schedule based on real time. The notion of time can be different for different tasks. Feedback loops may be based on the physical second, but message delays on the actual bit rate. Control sequences can be described by using a notion of time where the basic time unit varies with speed. Some events are stochastic and cannot be scheduled in time. They have anyhow to be integrated in the system schedule as the responses to a stochastic event have to meet defined deadlines. To schedule all events in a system might require more than one Virtual Schedule based on different notions of time..

An important difference between TSCK and traditional time-triggered approaches is that TSCK scheduling emerges from events and use time as a support for the scheduling. In traditional time scheduled systems, events are handled in defined time slots. When an event has occurred and created a signal, the signal is placed in a queue and treated in its allotted time slot. This is clearly shown in message schedules where the bus access is distributed in time frames to avoid collisions. TSCK starts with the event and uses the time to judge how fast the derived signal has to be handled in relation to other signals in a given situation. The difference in thinking becomes obvious in the message scheduling. An event is handled according to the Module Schedule by the module software and the resulting message is handed over to the CAN Controller for transmission. The collision resolution mechanism in CAN combined with the sum of the Module Schedules will ensure that the message can be delivered in relation to other messages as fast as possible and within its deadline instead of in a specific time slot. TSCK uses a combination of time synchronization and event sequence synchronization for scheduling of tasks. The TSCK concept covers the whole range from a pure event-triggered system to a pure time-triggered system. In most cases, the optimal system performance is achieved somewhere in between these extremes. It is usually not possible to find the right balance until a prototype of the system is up and running. The development process is iterative, starting with rough specifications that are refined stepwise. When the project reaches its final stage, the development of all the modules are completed and the last adjustments are done by means of control messages from a tool or a supervising module (**Figure 3**).

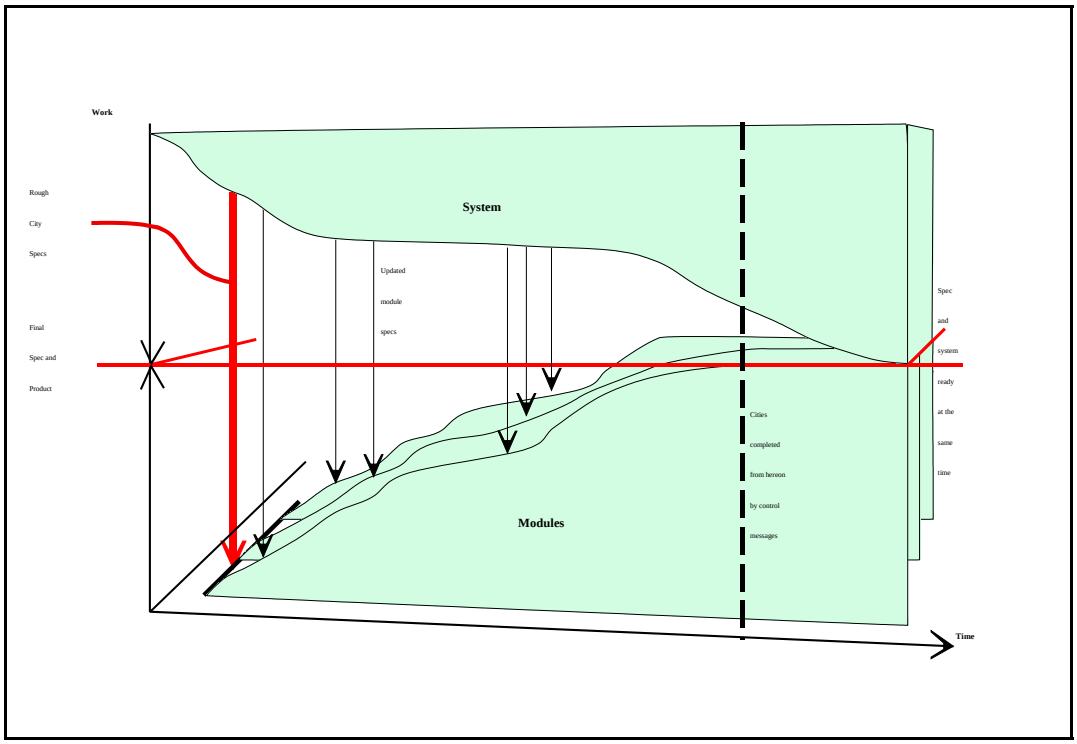

Figure 3 The development of a TSCK system is an iterative process between the system designer and the module designers. The final tuning of the modules is done by means ofcontrol messages from a tool or supervising module.

2.2 Virtual Time and Virtual Clock

The Virtual Clock, VC, is used by the system designer to set up an exact global time reference for his system, but only for specification purposes. The VC is an imaginary system clock, showing a Virtual Time (VT). Each module in a system has its own notion of time, the Local Time (LT) that deviates more or less from the VT. It is the system designer's task to set up relations between the different CTs and the VT. With the help of these relations he can coordinate the different modules to work in concert and, among other things, transmit messages in a timely manner. Using the VC, the system designer creates a system specific notion of time that can be or not be related to any other time standard, e.g., UTM. The VC is just an aid in the design process to make the system behavior predictable. The VC may or may not be replicated by a real clock in a module.

The VC produces either a linear or a circular time. A linear time can either be infinite (continuously growing) or definite (starting at an event and finishing at an event or after a certain time). A circular time runs seamlessly and continuously from a minimum value to a maximum value. GPS time is an example of an infinite linear time. A church clock shows a circular time. The time keeper at a track and field meet uses a definite linear time, starting at the "go" event and ending when the last athlete has crossed the finishing line. The difference between linear and circular time can be subtle. The starting and finishing event can be the same, e.g., when a master clock passes 12 o'clock. This might look like a circular time but it is not. A circular time revolves continuously, independently of any specific events. The difference between the two revolving times might look unimportant but is not. With a revolving linear time, it is natural to synchronize all the clocks in a system at the same event, the starting point. It is also natural to assume that all nodes have the same kind of clock. In a circular time, each clock is synchronized when needed and each node has a clock good enough for its tasks.

The VC issues a Virtual Time Tick (VTT). The length of the VTT can be either constant or variable. In most systems, the VTT is constant and related to the physical second as a decimal or binary fraction, e.g., 1 ms or 1/256 second tick, or to a specific frequency as the 1.25 ms tick based on 13 MHz in Bluetooth and GSM systems. In some cases, it is an advantage to have a variable VTT, e.g., to have a system time related to the angular speed of an engine or to have the time running slower when the system is in sleep mode, but in parallel have a fixed time tick for communication tasks. Many sports like football, hockey, basket ball, etc. make use of a variable time tick. The time tick is usually the physical second but now and then the referee stops the time, i.e., creates a long time tick, making the game time run slower than the standard time. It might be pointed out that a VTT may or may not be related to the bit timing on the CAN bus.

2.3 Local Clock and Local Time

A cornerstone in the CanKingdom concept is that a module designer should need a minimum knowledge about the system into which his module will be integrated. He should provide a variety of options for the system designer to pick the one that fits the system. The description of the VC clearly shows that there are several alternatives for a system clock. By definition, the VC represents the exact time within the system. Any other clock in the system is related to the VT. Some times the VC can be exactly mapped by one master clock in a system but in other cases several clocks in a system can have the role of a master, each of them deviating slightly from the VC. Then the average time of all of them will show the VT. There are also cases when there is no master clock at all but every clock in the system is related to the VC through a chain of event relations. Thus, at the module level it is not necessary to know anything about the VC. The only thing required is a possibility for the system designer to relate the LT to the VC. The commonality of any CAN system is the CAN bus, so the first option to get a relation to the VC is via the bus traffic. In CanKingdom systems, the simplest way to provide such a relation is to support the

King's Page 5, "Action/Reaction.". By means of the KP5 the King can instruct the module to execute a task as soon as a specific message is received. Thus, a module does not even have to have a clock in order to have a relation to the VC and to participate in a time scheduled system.

A module may provide a Local Clock (LC). In its simplest form, this is just a free running counter with a specified accuracy. This is enough to make it possible to set up the means of a local schedule based on this local time that in turn can be correlated to the VC by one or more messages through KP5. If better accuracy is needed, system wide synchronized module clocks can be implemented. By supporting KP11 "Circular Time Base setup Page" and KP12 "Repetition Rate and Open Window setup Page"and the "Time Herald"in the module, the King can synchronize a LC to other clocks in the system.

2.4 The Virtual Schedule

Like the VT, the Virtual Schedule (VS) is just an aid for the system designer to specify the system requirement for the relation between events in time (including message transmissions and receptions). The VS shows the ideal timing, and acceptable deviations from this are then described with the VS as reference. The Virtual Clock generates the Virtual Time line and each message is assigned a time slot when it is allowed to occupy the bus. The minimum and maximum time a message will occupy the bus, how much it will jitter within the slot, and how much the slot itself will jitter in reality, depends on the relation between the VT and the LT. There are several ways to relate the VT to the local LT and the relation can be different between different modules as well as different for different events and tasks within a module. The relations can also shift between modes such as set-up, diagnostic, runtime, etc.. Some examples on relations:

1. Some or all modules are connected to the same external time reference, e.g., GPS
2. One module clock is regarded as system clock and other module clocks are synchronized to that clock.
3. The LT is set to specific values at reception of certain messages.
4. The LT is set to specific values at certain events.
5. The module lacks a LC and events are related to reception of certain messages.

2.5 Module Schedule

The CanKingdom concept does not require a module to have any knowledge about how the messages are scheduled within the system. Only the system designer needs to know. The module has only to provide the possibility for the King to schedule messages according to the LC and/or to certain events. The combination of all local schedules will create a real system schedule. As mentioned in section 2.2, a module can participate in a Time Scheduled System even if it has not got a clock and schedule. In the next version of CanKingdom, there will be support for setting up and identifying module schedules.

3 Examples of message scheduling

The first approach is a traditional time schedule and this is then compared with other approaches. They will show that traditional systems are resource hungry because they rely on time-triggered transmissions and that a more than three fold gain can be achieved by applying time scheduling on event-triggered transmissions.

To illustrate the different solutions, we use a small and simple system with only three modules, A, B and C (**Figure 4**). They can transmit two messages each: A1 & A2, B1 & B2, C1 & C2

respectively, with the same length of 100 bits, the same update rate, 600 ms, and a maximum latency time of 200 ms. Each module can also transmit an alarm message (indicating local failures), Aa, Ba and Ca respectively with a maximum latency time of 600 ms.

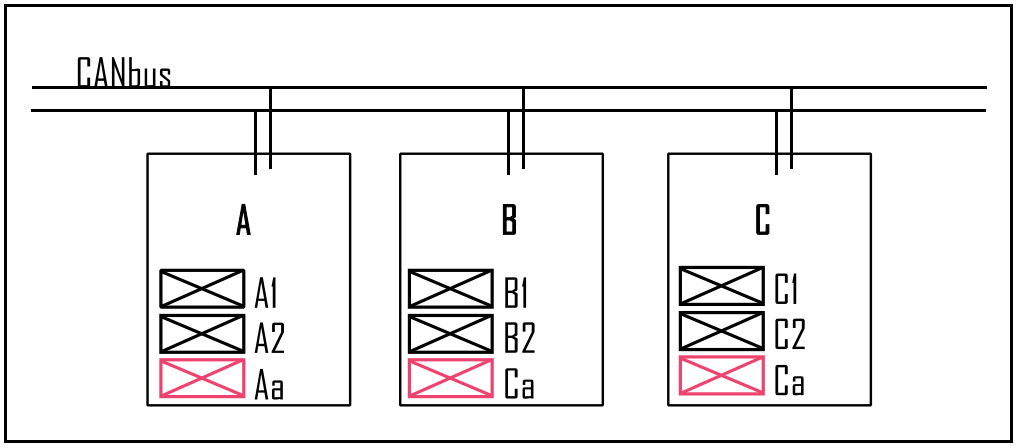

Figure 4 Example system with the modules A, B and C.

We will begin the examples by writing a short VS for the regular messages. This is the ideal time schedule for our messages and the outcome of the different methods will be compared with this. The most efficient schedule during normal conditions we can come up with is to transmit the messages back to back as this would require the lowest bit rate. We can set the virtual time tick to 1 ms, equal to one bit, the virtual time slots to 100 VTT, which gives us the virtual time schedule shown in to figure **Figure 5**.

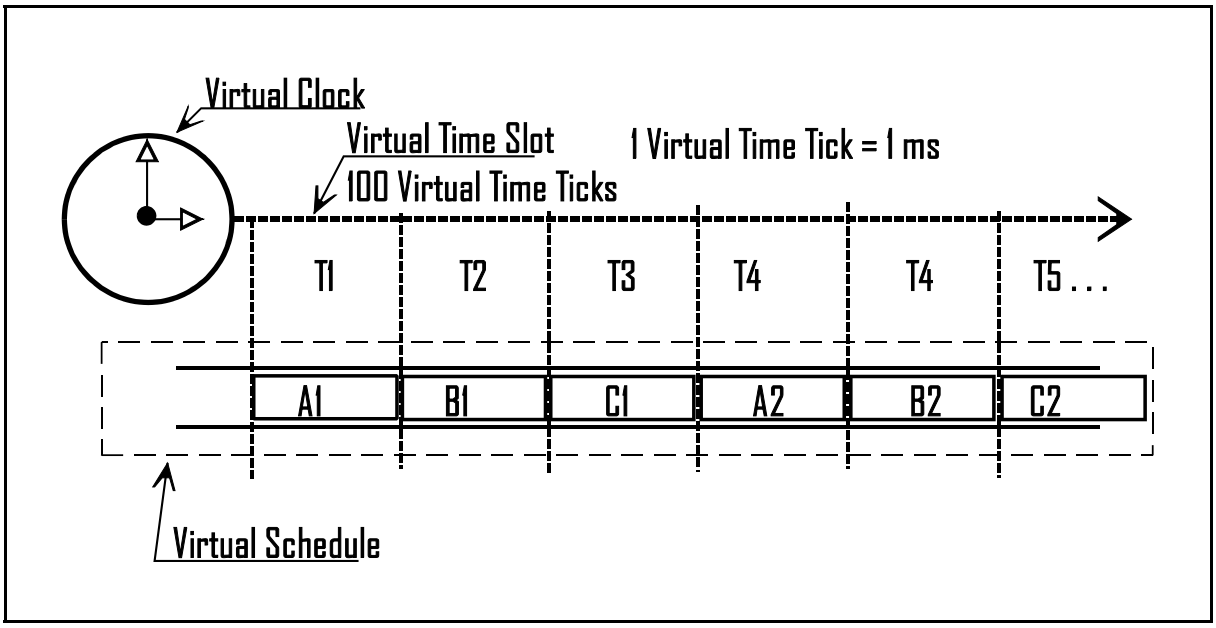

Figure 5 Basic Virtual Schedule

3.1 Traditional time-scheduling

Beside being time-triggered, the basic rules for a traditional time-scheduled system are:

1. Each message is assigned specific time slots
2. Each module has a clock that is synchronized to a master clock with a certain accuracy
3. Message collisions on the bus are not allowed
4. Corrupted messages are not retransmitted

Rule 1, “Each message is assigned specific time slots,” is fulfilled with the basic virtual schedule for repeatedly transmitted messages. Each message has its own time slot, A1in T1, B1 in T2, C1 in T3, A2 in T4 and so forth and the schedule repeats itself. The schedule can be shown in a table:

T1 100 ms	T2 200 ms	T3 300 ms	T4 400 ms	T5 500 ms	T6 600 ms
A1	B1	C1	A2	B2	C2
A1	B1	C!	A2	B2	C2

But the alarm messages have to be scheduled as well although they will rarely appear during normal conditions To make space for them, the time slots have to be made smaller, i.e., the VTT has to be shortened to .67 ms and the bit rate increased to 1.5 kbit/s to meet the maximum delay allowed for alarms, 600ms. The adjusted schedule is shown below.

67 ms	133 ms	200 ms	267 ms	333 ms	400 ms	467 ms	533 ms	600 ms
A1	B1	C1	A2	B2	C2	Aa	Ba	Ca

Rule 2, “ Each module has a clock that is synchronized to a master clock with a certain accuracy” sets some physical requirements. Each module needs its own local schedule containing at least those messages it should transmit or receive. The reference for this schedule is the local clock and this deviates more or less from the master clock. We will not discuss the master clock here, only assume that it communicates time by time messages on the bus. The ISO 11898-4 standard gives examples of how it can be done. **Figure 6** shows that module A has a positive offset error (dA) and a correct clock frequency, that module B has a negative offset (dB) and a correct clock frequency and that module C has a slightly faster clock resulting in an increasing offset over time (dC - dC’). There are three ways to cure this problem :

1. Increase the frequency of clock synchronization messages
2. Widen the time slots
3. Use more accurate clocks in the modules

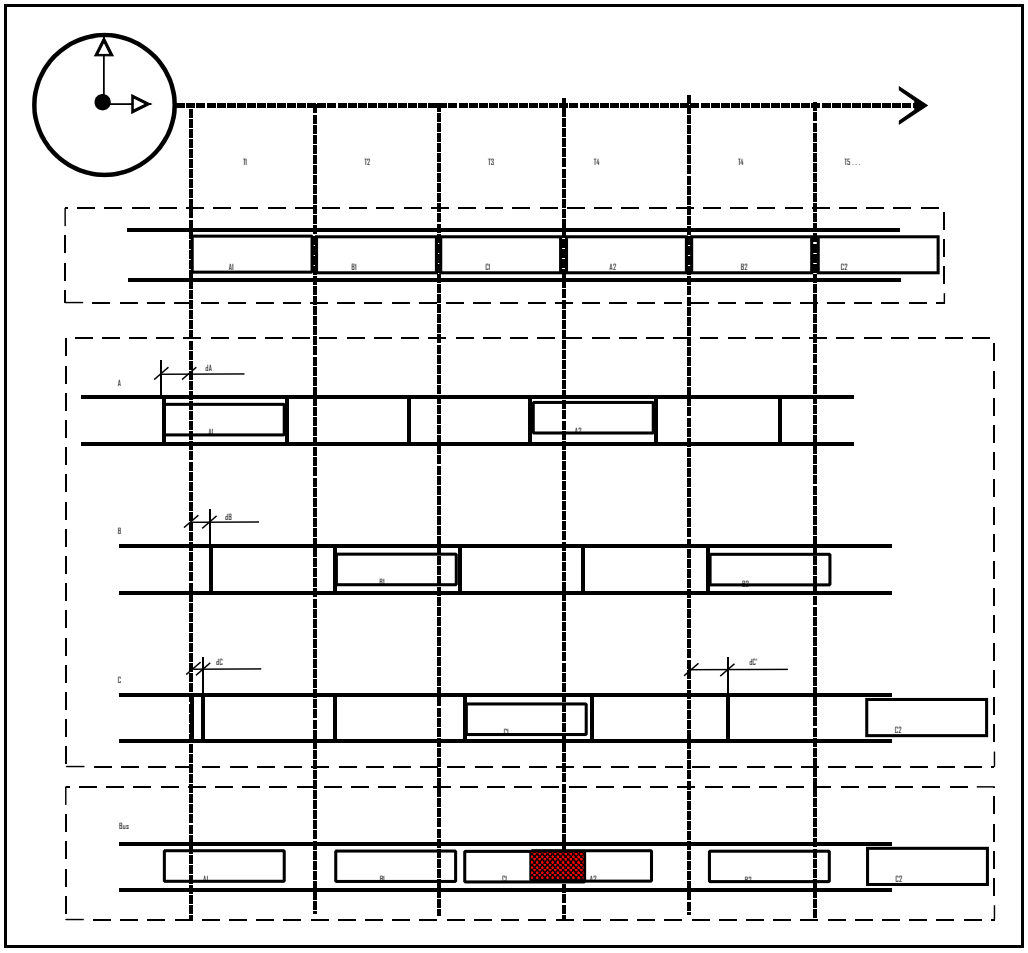

Figure 6 Impact of the deviation of module clocks to the master clock.

To keep the local clocks synchronized, we have to schedule a message from the Time Master to announce the correct system time. How often this has to be sent depends, among other things, on the quality of the least accurate clock within the system which determines how often the clocks in the system should be synchronized.

There are many different ways to synchronize clocks and to keep them in shape using the bus communication. This paper merely discusses principles, so we use just one additional message, TM, to cover the issue. We have then to add a time slot for TM. To fulfil the timing requirements, we decrease the VTT to .6 ms and increase the bit rate to 1.67 kbit/s:

60	120	180	240	300	360	420	480	540	600
TM	A1	B1	C1	A2	B2	C2	Aa	Ba	Ca

To meet Rule 3 “Message collisions on the bus are not allowed and Rule 4 “Corrupted messages are not retransmitted” the message rate has to be at least doubled to meet the update rate for the ordinary messages and the maximum latency for alarm messages. The reason for this is that disturbances on the bus are stochastic and thus cannot be scheduled. If a message is corrupted on the bus, it has to wait for its next time slot and still meet its maximum latency time:

n \ t	30	60	90	120	150	180	210	240	270	300
1	TM	A1	B1	C1	A2	B2	C2	Aa	Ba	Ca
2	TM	A1	B1	C1	A2	B2	C2	Aa	Ba	Ca

The VTT has now to be 0.3 ms and the bit rate 3.33 kbit/s. As the messages are transmitted back-to-back, all clocks have to be perfectly synchronized. Assuming the clocks can be accurate to within +/- .5%, the bit rate has to be further increased by 1% to avoid collisions due to jitter.

In this example we started with a bandwidth requirement of 1 kbit/s for the basic message transmission and ended up with 3.4 times more to cope with alarm signals, corrupted messages and clock accuracies. All modules have to support clock synchronization and to be equipped with clocks good enough to keep time within half a percent of the master clock. On top of what is already discussed, the system relies upon that each local clock is working correctly. To enssure that this is true, additional resources are required. It can be concluded that time-triggered systems are resource hungry. A better approach is needed and TSCK is a candidate as will be shown below.

4 Violating the rules

In modern CAN Controllers the automatic retransmission of corrupted messages can be turned off, thus removing the biggest obstacle to use CAN in time-triggered systems. The nondestructive message collision resolution feature of CAN opens up a safe way to violate the third rule “Message collisions on the bus are not allowed” in time-triggered systems. Breaking this rule results in more efficient use of the bandwidth and allows for using less accurate clocks in modules. It will also be shown that the fourth rule “Corrupted messages are not retransmitted” can also be violated safely and successfully. The communcation is then not time-triggered but can still be time scheduled.

4.1 Allow message collisions

The key to better use of the bandwidth is to allow message collisions. The very bandwidth thieves in the time-triggered example are the alarm messages. As they should not be needed during normal conditions, they do not lend themselves to be time scheduled. They should be transmitted when needed. The non-destructive collision mechanism in CAN makes the communication fully predictable also when collisions occur. The first step is to use this feature for alarm messages but it will also be shown that scheduling messages to deliberately collide will result in a more efficient use of the bandwidth and less demands on clock accuracy.

4.1.1 Alarm messages

In traditional time-scheduled systems, alarm messages take a great portion of the bandwidth. The reason for this is that they do not really fit in the time scheduling concept as they are genuinely triggered by unpredictable events. The maximum latency time allowed for alarm messages is usually short and thus they have to be allotted frequent time slots. In our example a long latency was allowed, but thirty percent of the bandwidth still had to be allocated for the alarm messages. By allowing message collisions, no bandwidth has to be allotted to alarm messages. They can be transmitted immediately whenever alarming conditions are detected. The alarm messages are then unscheduled:

n \ t	43	86	129	171	214	257	300
1	TM	A1	B1	C1	A2	B2	C2
2	TM	A1	B1	C1	A2	B2	C2

The VTT is now .43 ms and the bit rate 2.32 kbit/s. The bandwidth utilization is increased by 30% and, on top of that, the maximum latency time of an alarm message is reduced seven times, now less than 86 ms compared with 600 ms for the traditional schedule!

4.1.2 Reduced clock accuracy requirement

Less accurate clocks can be used by deliberately scheduling messages to collide with the respectively preceding ones on the bus. If a message B collides with the previous one A after the first bit of A is sent (the Start Of Frame) on the bus, according to the CAN rules, B waits until A is fully transmitted and then accesses the bus immediately . As the next time slot is reserved for B, no other message is competing for the bus and B will be transmitted regardless of its CAN priority. If pending messages are “pre-fired” 49 bits into the preceding messages, each clock could deviate +/-49 bit times from the LC and the timing on the bus would still be correct (**Figure 7**). If a module knows how much its schedule is “pre-fired”, it can use the actual message release time to adjust its clock offset.

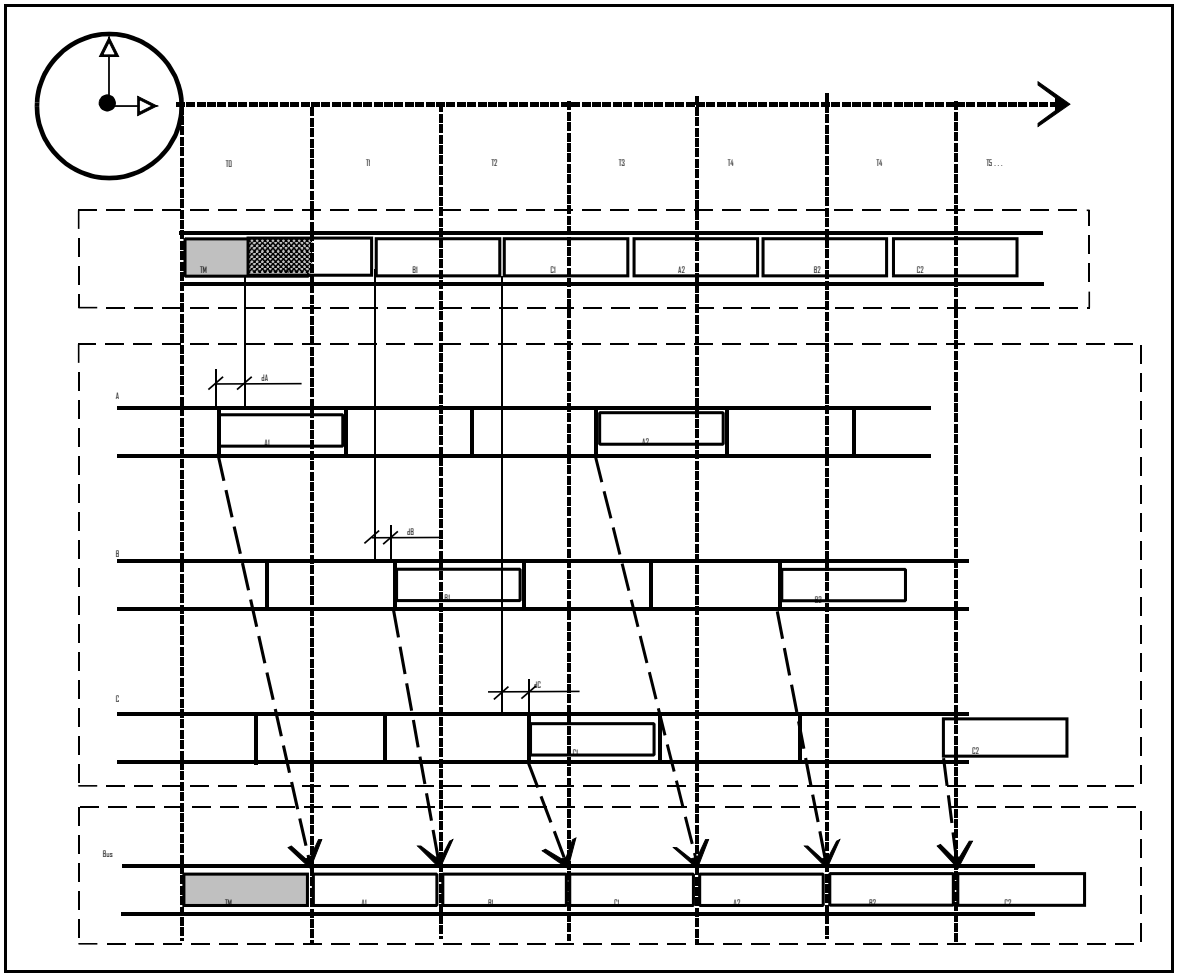

Figure 7 Local schedules deliberately skewed into the previous time slot. The CAN collision resolution mechanism puts the messages into the correct slot.

4.1.3 Allow retransmissions

As a rule of thumb, more than one error frame in one thousand messages indicates a severe problem in a CAN network. As errors are unpredictable events, the bandwidth was cut to half to cope with the problem in the previous examples. By allowing retransmission of a corrupted message, assigning a proper priority to each message, the extra bandwidth needed for alarm messages can be cut down to almost zero and the maximum latency time for any message is decreased by a factor 2. In **Figure 8** a small gap between the messages has been introduced in the virtual schedule to allow for retransmissions. The “pre-firing” is applied in the local schedules. Message A is corrupted just before it is completed and immediately retransmitted (A’). A’ now

occupies a part of the next slot intended for B1 and B1 has to wait until A' is completed. The following messages will be sent back-to-back until the sum of the gaps has compensated for the additional message. In this case, with inaccurate clocks and a long "pre-firing," the priority of the messages has to be in decending order. Otherwise some messages may shift place in the back-to-back sequence. This is of importance only for message timing as each message has a unique identifier and is not dependent on the slot for identification.

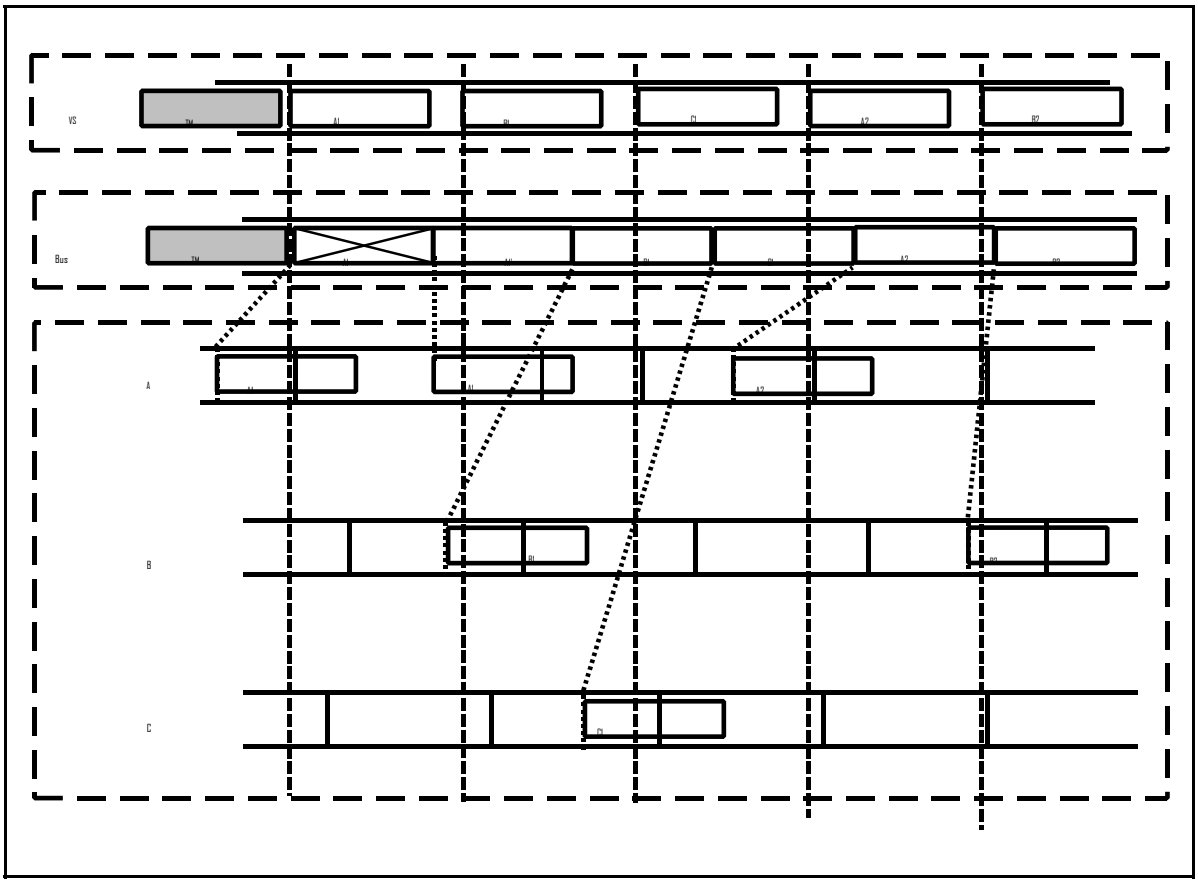

Figure 8 Retransmission of a corrupted message

The latency due to a transmission can be fully controlled by the selection of CAN identifiers, gap between messages, clock accuracies and their "pre-firing." It would be a nice feature if the maximum number of attempted retransmissions could be set in the CAN Controller. This is currently not the case but the number of attempts can be controlled by the module CPU if the module has a CAN Controller with error flag indication (which most CAN Controllers have).

4.1.4 Time scheduled systems without time master

It is often taken as a prerequisite that a time scheduled system needs a physical time master. This is not the case. Many applications need a specific and accurate sequence of messages only for a short period of time, e.g., during one revolution of a shaft. Examples of such systems are air-jet weaving machines and combustion engines. In an air-jet weaving machine, the opening of the main air-jet nozzle and the relay nozzles have to be accurately controlled in time when the shed is open to achieve a high quality fabric and low energy cost. In this case, the VTT is constant. A combustion engine needs a precise control of the closing and opening of valves and the point of ignition, all related to the shaft position. This can be obtained by a virtual time where the VTT changes with the rpm. Common to both examples are that some tasks in some modules have to be coordinated with the turning angle of a shaft in the machine. There are several control systems that show the same or similar characteristics. Such systems can be seen as time-triggered during the development phase, i.e., when making the Virtual Schedule, but be implemented as event-

triggered in each module, the events being certain messages appearing on the bus. The local schedules are then implemented as delays after reception of certain messages before generating one or more events. The delays are measured by the local clock and how this is done is entirely a local problem.

Figure 9 illustrates the principle. A combustion engine is controlled by a CAN network One or more sensors detects the top dead center (TDC) position of the shaft.

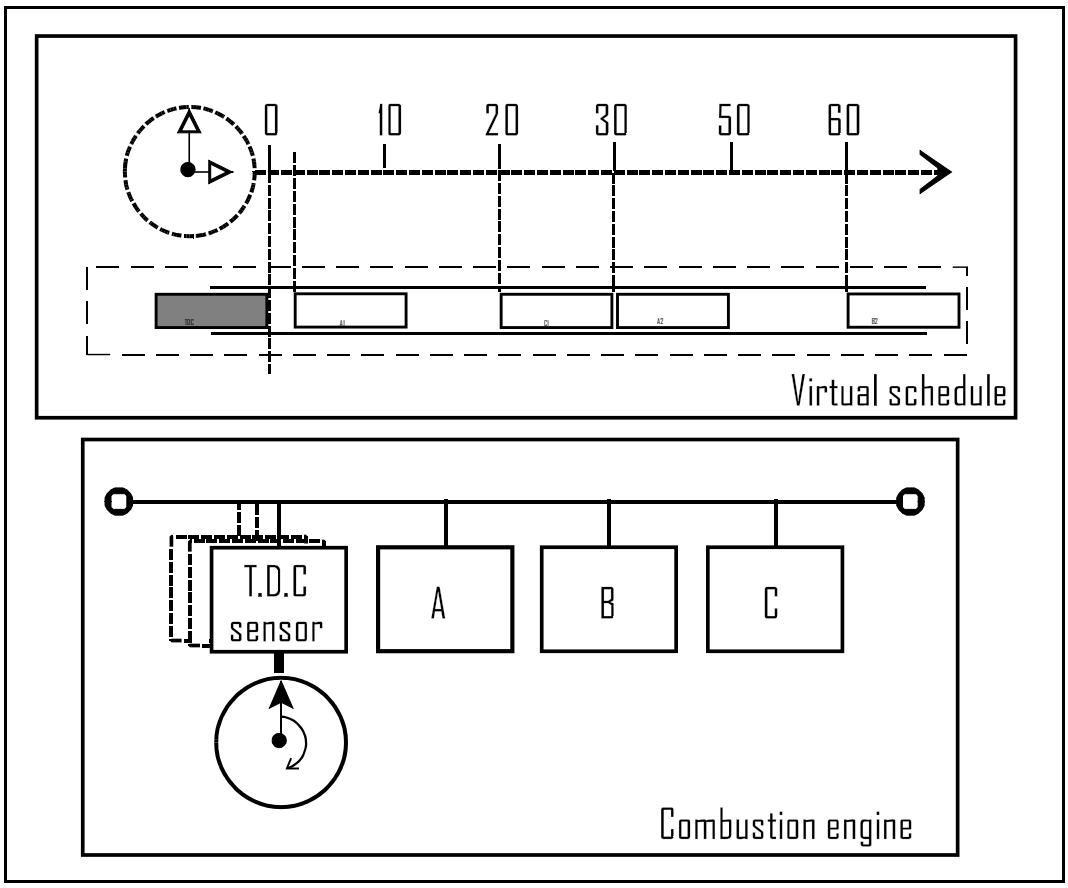

Figure 9 Sequential message scheduling with a VTT related to shaft speed in a combustion engine.

When the TDC, is detected, the TDC sensor module or modules immediately transmit a message without data. If there are more then one detector module, they may all use the same identifier. Messages that hit the bus within the SOF bit will all appear as one and the same. A timeout that aborts a transmission attempt after a number of bit times would avoid those missing SOF duplicating the message or retransmitting at too a late time if the TDC message is corrupted. The reception of the TDC message starts the virtual clock and the virtual schedule would look as follows:

VT	Message
0	TDC
2	A1
20	C1
30	A2
60	B2

The virtual schedule is then converted to local schedules for each module:

Module A			Alternative schedule		
Trigger message	Delay Time Ticks	Transmit message	Trigger message	Delay Time Ticks	Transmit message
TDC	2	A1	TDC	2	A1
TDC	30	A2	C1	0	A2
Module B			**Alternative schedule**		
Trigger message	Delay Time Ticks	Transmit message	Trigger message	Delay Time Ticks	Transmit message
TDC	60	B2	A2	10	B2
Module C			**Alternative schedule**		
Trigger message	Delay Time Ticks	Transmit message	Trigger message	Delay Time Ticks	Transmit message
TDC	20	C1	A1	8	C1

One way to schedule messages is to let all of them use the TDC message as triggering event, but as shown in the alternatives, only one has to use the TDC message and the others can be related to the virtual clock via a chain of messages. In the highly simplified example above, the local Time Ticks happened to be the same as the VTT. This might not always be the case and is not necessary. A module has only to know its own schedule and it is the task of the system designer to see to it that all local schedules fit into the virtual schedule.

Only messages are scheduled in the example but the same process can be applied for other purposes. The process can be described as “Input event” ! “task” ! ”Output event” and illustrated in a table:

Input event	Task	Output event
Shaft reaches TDC	Sense TDC	Transmit TDC message
Receive TDC message	Wait 20 Time Ticks	Transmit C1
Shaft reaches BDC	Sense BDC	Transmit BDC message
Receive BDC message	Wait 140 Time Ticks	Fire spark plug

Such relations can be set up with the King's Page 5, "Action/Reaction" in CanKingdom.

The sequences in a combustion engine are related to the shaft angle. The VTT and the local LT tick (LTT) should then change linearly with the shaft speed to map the position in time:

VTT= const*rpm , LTT_A =$const_A$*rpm, LTT_B =$const_B$*rpm, LTT_C =$const_C$*rpm

The fact that VTT and LTT are not related to the CAN bit timing has to be considered when calculating the scheme. From a module point of view, the time a message transmission will take is variable and increases with the rpm.

5 Robust communication

A safety critical system has only one safe state and that is "power off". As soon as the system is started, it is only a matter of statistics before it gets into a dangerous state. It is the system designer's task to minimize the probability for the system to enter a dangerous state. A first step in reducing the probability for failures is to make the communication predictable and reliable. One reason for choosing a time scheduled approach is its predictability. Traditional time- scheduled communication relies on a global clock and each module being synchronized to this clock. If the synchronization is lost, the reliability is also lost. There are two main causes for lost synchronization, either the global time generation is corrupted or one or more modules get it wrong. There are several ways to generate the global time. One method is to have a time master that transmits the right time. This method is used in ISO 11898-4, "Time-triggered CAN." Here it is obvious that the synchronization is lost if the time master fails. Therefore one or more redundant time masters are introduced and an arbitration procedure assures that only one will appear at a time. It can always be questioned whether a redundant system is safer than a non-redundant one. The more complex a system is, the more things there are that can go wrong. Even if the global time is correctly generated, there is still the risk that the communication will be corrupted by an unsynchronized module. The cure for this is often a bus guardian at each node but this adds additional components that can fail.

Another approach is to keep the system as simple as possible and make it failsafe. As shown earlier in this article, some traditional rules for time scheduled communication can be violated safely if the CAN protocol is used. The communication can be outlined as a traditional time scheduled one. If the time master fails, it will slowly degrade into an unsynchronized communication, relying on message priority. An unsynchronized node would only cause a jitter in the time slots. This can be accepted as the message identification is not dependent on the time slot. Each message has its own identifier. Timing problems can easily be detected. On the system level, an unsynchronized module is revealed by the CAN identifiers. By providing a module a schedule for reception of messages from two or more other modules, it can easily detect if it is out of synch: If all received messages arrive at the wrong time, the module's own clock is incorrect. In such a case, it may shift into a backup mode and use other messages on the bus as transmit triggers.

5.1 Double triggers

An expected but missing message on the bus can be considered as a single failure. If this message is used as a trigger for other messages to be sent and no other steps are taken for their transmission, the failure will propagate. One way to prevent this situation is to use two trigger messages from different modules. If the delay for the message transmission can be set individually for each trigger message, the triggered message will hit the right time slot regardless of whether one of the triggering messages appears or not. A missing message creates an empty slot on the bus. If the delay is set to the same value, a missing message will result in a shift of the slots to the left and leave the bandwidth to be used later on for an unscheduled message. Both cases are covered in the examples below. Let us start with the empty slot case. We assume that the virtual schedule looks as below:

VT	0	12	24	36	48	60
Msg	A1	B1	C1	A2	B2	C2

The modules do not support a global clock so the local schedules will be based purely on trigger messages with delays according to their local clocks. The local schedules for keeping the time slot in case of a missing trigger may look as follows:

Module	Trigger message	Delay	Transmit message
A	C2	2	A1
	B2	12	
	C1	2	A2
	B1	12	
B	A1	2	B1
	C2	12	
	A2	2	B2
	C1	12	
C	B1	2	C1
	A1	12	
	B2	2	C2
	A2	12	

The result is shown in the upper part of **Figure 10**.

The alternative Virtual Schedule for getting a slot shift in case of a missing message will look as below:

VT	0	12	24	36	48	60
Msg	A1 B1	B1 C1	C1 A2	A2 B2	B2 C2	C2 A1

The schedule allows two adjacent slots for each message transmission. During normal conditions, the upper schedule is valid, but if one of the messages is missing, the system will automatically switch to the lower schedule. One way of interpreting this solution is that each message is deliberately delayed one slot during normal conditions.

The local schedules can look as shown below:

Module	Trigger message	Delay	Transmit message
A	C2	2	A1
	B2	2	
	C1	2	A2
	B1	2	
B	A1	2	B1
	C2	2	
	A2	2	B2
	C1	2	
C	B1	2	C1
	A1	2	
	B2	2	C2
	A2	2	

Figure 10 shows the difference between the two approaches. The first one leaves a gap and the timing is maintained. The second one shifts the timing one message ahead, allowing a message to be inserted later for restoring the timing.

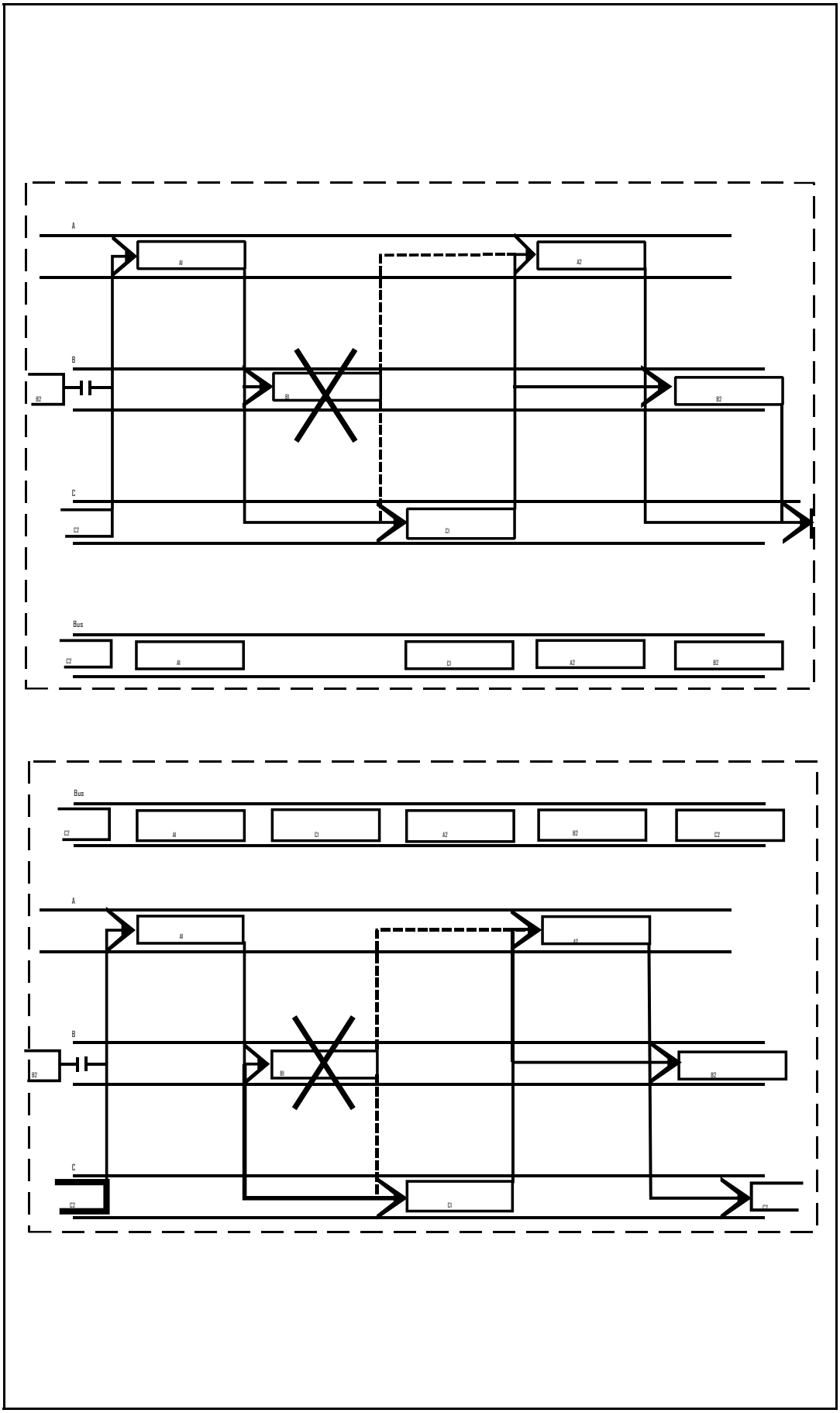

Figure 10 Double triggered messages and a missing message. A missing message results in a void slot when using delay compensating triggers (top). Same delay for both triggers result in a left shift of the slots (bottom)

5.2 Priority controlled schedules

The collision resolution mechanism in CAN is a key feature for an efficient use of the bandwidth. It has earlier been shown that messages will appear back-to back on the bus if they are deliberately scheduled to collide by the “pre-firing” method and that the latency is predictable. This method can be further developed. Messages can be time scheduled to act as heart beat or life guarding messages at a low frequency and shift, to a higher transmission rate or transmission on events according to predefined conditions. This method lends itself to “ imprecise programming”, i..e., that the bandwidth is designed to meet the needs for control messages during extreme conditions to maintain safety but that the same bandwidth can be used for obtaining better or smoother control during normal conditions. The full bandwidth can then be safely saturated during a critical condition if the messages not needed for safety are assigned lower priority than those needed for maintaining safety.

6 Summary

A basis for designing distributed embedded control systems in general and safety critical systems in particular is that the latency of each message is finite and predictable. Time-triggered communication is generally regarded more predicable than event-triggered ones. Traditional time-triggered scheduling requires a substantial part of the bandwidth to be allocated to messages originating from rare events, e.g., alarm messages for guaranteeing a short latency. Significant measures have to be taken to assure that a correct notion of the global time is prevailing within the system. Time-triggered CAN systems require substantial resources in the shape of bandwidth, software and hardware to reach the goal of dependability.

Time Scheduled CanKingdom is an event-triggered but time scheduled approach to solving the problems of dependable CAN systems. The design process is characterized by two separate levels, a system level with a system designer and a module level with module designers. The modules are designed in a way that they can relate input events to output events according to commands from the system level. They may have a local clock and support a global time, but that is not mandatory. The timing analysis of the system is performed off line during the development phase. This is condensed to one or more virtual schedules which in turn are broken down to individual schedules for each module. In its simplest form, timing between sequential events in different modules is maintained by message transmissions and reception. Message collisions on the CAN bus are allowed, using the predictable and non-destructive resolution mechanism in CAN. By deliberately schedule messages to collide, the available bandwidth can be used to its maximum.

1. Tindell, K., Burns, A.; Guaranteeing Message Latencies on Controller Area Network (CAN), *Proceedings 1st International CAN Conference, Mainz 1994*.

2. (1996) “CAN Kingdom. Ver. 3.01" www.cankingdom.org