

Fault Tolerance in Automotive X-by-Wire

Diana Palsetia, Sean Pieper

Electrical & Computer Engineering Department
University of Wisconsin-Madison

4th December 2005

Abstract

Automotive X-by-Wire systems are distinguished from other fault tolerant systems in three major ways. First, they are safety critical, meaning that failure has the potential to result in death, serious injury, or widespread damage to the environment. Secondly, they are complicated electro-mechanical systems, and the failure modes of sensors and actuators must therefore be considered. Finally, automobiles are commodity systems. Cost is therefore an important design consideration. In this paper, we will examine a variety of techniques used to provide safety in automotive X-by-Wire systems.

1 Introduction

X-by-Wire is a budding research area that investigates the incorporation and eventual replacement of mechanical linkages with electronic controls and haptic interfaces. This is desirable for a number of reasons, including cost, weight, and usability. The first X-by-Wire systems were developed for military aircraft to allow for the control of unstable planes, such as the stealth bomber, and these techniques have since spread to commercial aircraft. From the automotive standpoint, X-by-wire offers the potential for significant reductions in weight and cost through the elimination of heavy and expensive mechanical systems. It also makes it possible to add new functionality to improve ride quality[31].

Throttle-by-wire has been implemented since 1986,

and a shift towards brake-by-wire has begun with features such as Antilock Braking System (ABS) [1]. Steer-by-wire is also under investigation. The major challenges in these systems are providing sufficient safety, meeting cost requirements, and providing a convincing user interface (the user should not be able to tell that the brake pedal or steering column are not physically connected to the brakes and wheels). We will only address the first two challenges in this paper.

Because any failure could jeopardize human life, automotive X-by-Wire systems must be considered safety-critical. Intuitively, this means that extreme care should be taken in their design. The primary goal is to avoid mishaps. The United States Department of Defense defines a mishap as "An unplanned event or series of events resulting in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment[15]". To do this, we must first assess which mishaps are possible, as well as under what conditions it is possible for them to occur. Techniques used in this portion of the design process include Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA), which we will describe in more detail later. Once we have isolated potential failure cases resulting in mishaps, we can then apply various redundant design techniques to reduce their probability of occurrence.

An automotive X-by-wire system is comprised of sensors, actuators, processors, communication channels and may have mechanical backup. In order to maintain safety in the event of failure, the system

design should contain no single point of failure. This means that redundancy must be present in all systems components. In the event of a failure, the goal should be either to bring the system into a safe state (fail-safe semantics) or allow the system to continue to operate in a degraded mode so that the user can bring the system into a safe state (fail-operational semantics). Additionally, failure of one component should not affect the operation of others (fail-silent semantics) [25].

Fail-safe semantics are the most desirable, but difficult to achieve in a moving automobile, without driver intervention. Because of this, we additionally require fail-operational semantics that allow the driver to bring the system to a safe state following an alert. Fail-operational semantics require either the presence of backup systems (mechanical or electrical), or the ability of failed components to operate in a degraded mode. Fail-silent behavior is important for reducing the impact of a component failure. While there are many possible ways of achieving these failure semantics and improving the safety of automotive X-by-wire systems, we must consider other factors such as cost, weight, complexity, and other safety implications of our design decision. The remainder of this paper will investigate a number of proposals for safe operation of steer-by-wire and brake-by-wire systems and explore their relative merits. In section 2, we will discuss the use of Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA) in the safety critical design process. Section 3 will look at proposals for fault tolerant actuation and sensing. Section 4 will compare communication protocols for X-by-wire. Section 5 will investigate software based fault tolerance. Finally, we will conclude. Hardware fault tolerance will not be discussed, as the techniques (such as triple modular redundancy) are common across fault tolerant domains.

2 Safety critical design

Fault Tree Analysis and Failure Modes and Effects Analysis are well established techniques for designing reliable systems. FMEA was developed in 1949, and FTA in 1962. FMEA attempts to enumerate

possible malfunctions for each component in the system, and then figure out the potential consequences of such a failure. FTA, on the other hand, starts from a particular negative outcome (e.g. drive-by-wire system fails to respond when wheel is turned), and then determines potential sequences and combinations of events under which this is possible. FMEA is considered to be a bottom-up method of analysis, whereas FTA is top-down. In practice, the two methods are typically used in tandem for hardware design, for example, in the throttle-by-wire system of the Jaguar XK8 [24, 23, 22, 17], with FTA being more common for software [18].

Before continuing, it is necessary to define a few terms, most importantly fault and failure. In fault-tolerant computing, the term fault is typically taken to refer to a basic malfunction which may then manifest itself as an error and eventually result in an incorrect behavior at the system level, also called a failure. In FTMEA and FTA, however, fault and failure are used differently. The Fault Tree Analysis Handbook explains the terms as follows: "Generally speaking, all failures are faults, but not all faults are failures. Failures are basic abnormal occurrences, whereas faults are 'higher order' events[17]." Put differently, a fault is either a failure, or the result of some combination of failures. The consequences of some faults may be more dire than others, which takes us to the idea of severity, and criticality, which considers both the severity and likelihood of a fault[21]. Numeric values are commonly used to indicate severity, with scales assigned in various government and industry standards, for example[15]. A failure mode is a manner in which a failure can occur. For example, a switch might have two separate failure modes-stuck open and stuck closed. Different failure modes for a single component may vary greatly in their criticality, an obvious example being a circuit breaker in the home. If the breaker goes off at too low a current, the worst that is likely to happen is that the inhabitants will be annoyed. If it fails to go off at high current, however, this could result in a house fire and endanger human life. In addition to the result of a failure, we may also be interested in the potential causes. These are termed the failure mechanisms.

Failure Modes, Effects, and Analysis then, is a

method for enumerating the failure modes of each component in a system, determining the possible effects of such failures, and assessing the criticality of the failure mode. When the goal of the analysis is to reduce the likelihood and severity of mishaps, FMEA is also called Failure Modes, Effects and Criticality Analysis (FMECA). FMEA is performed using a worksheet which consists of a number of columns and a single row for each failure mode. The exact columns of this worksheet will vary depending upon what standard is being followed, but common ones may include item, failure mode, effects, severity, and detectability. Other items such as number of backup systems may also be included. FMECA allows for the computation of a Risk Priority Number (RPN) based on the product of severity, likelihood of occurrence, and detectability (all qualitative metrics)[22, 16, 21, 23]. Entries in the worksheet can then be sorted by RPN so that the highest priority failure modes can be addressed.

As [17] points out, a failure mechanism for a failure mode of a system is itself a failure mode of a component of that system. This means that FMEA can be carried out at an arbitrary resolution, with finer granularity resulting in a more detailed, and significantly longer, analysis. At higher levels of detail, FMEA may prove unmanageable, in particular because it becomes difficult to determine interactions and relationships between failure modes. It is best suited at higher granularity as a tool for preliminary analysis[22, 17]. Because it focuses on components, FMEA does not map cleanly to software design. When it is used, the basic component can either be a function or a variable, depending upon the detail required [18].

Fault Tree Analysis starts with the selection of a single fault, which corresponds to a particularly important system failure. The choice of top-level fault is important, as it determines the complexity of the analysis, as well as its utility. If this fault can be independently caused by a number of possible conditions, an or-gate is drawn beneath the fault, with the conditions that might cause it, otherwise, if the fault requires multiple simultaneous conditions, an and-gate is drawn beneath. For each of the newly listed faults, the process is repeated. While the inputs to an and-

gate represent a combination of conditions that can cause the parent fault, the inputs to an or-gate represent specific instances of the parent fault. At some point, the analysis will reach conditions that are not further divisible. These may be a component failure where finer granularity is not desired, some abnormal usage condition, or a normal (non-failure) condition.

Because FTA is graphical in nature, an example diagram is of use in explaining the process. Figure 1 shows a sample fault tree detailing the conditions under which the actuators in a brake-by-wire system might fail to engage. This diagram could be part of a larger fault tree describing the conditions under which the braking system as a whole fails. It should be noted that a real fault tree for this scenario would be significantly more detailed, and this example is solely for the purposes of illustration.

The reader will see that the basic symbols are boxes, logic gates, diamonds, and circles. Boxes represent an event. An event should always say both what happens, and under what condition. To explain why this is, consider the two events "brakes do not engage when brake pedal is not depressed" and "brakes do not engage when brake pedal is depressed." The former event represents correct system behavior, whereas the second represents a serious safety hazard. The action in each is identical-brakes do not engage. The condition is what determines whether the action is harmful or not. Circles represent basic events, or failure modes of component systems. In this case, the basic events are non-operational failures. The diamond represents an unexpanded event, and is used for faults whose cause is not important, or cannot be determined due to insufficient information. In our example, it is possible that the detection system has not yet been designed, and therefore, details about its failure causes are impossible to supply.

Once the fault tree is constructed, it is possible to perform both quantitative and qualitative analyses. Qualitative analysis is based upon performing creating a minimum sum of products Boolean expression from the fault tree. Each minterm of the resulting expression is called a "minterm" and indicates a minimal combination of events that will trigger the top-level fault. In the example diagram,

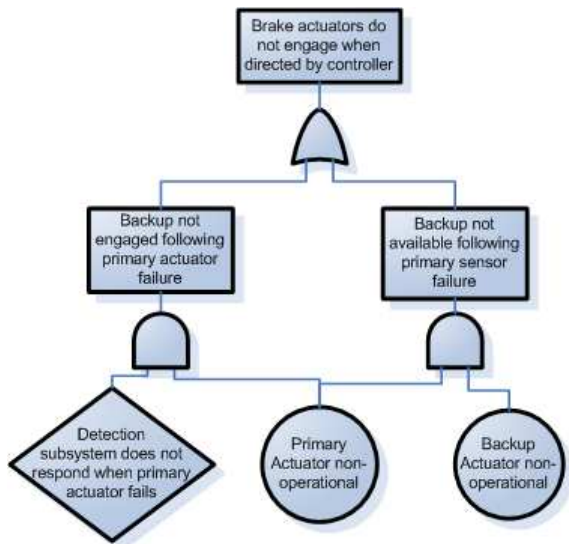


Figure 1: Fault Tree Analysis (FTA)

the two mincuts would be "primary actuator non-operational and detection subsystem does not respond when primary actuator fails" and "primary actuator non-operational and secondary actuator non-operational." Based upon the relative probabilities of the various events, it is possible in this manner to get a quick idea of what paths are most likely to result in the top-level fault. Quantitative analysis using fault-trees is significantly more involved and beyond the scope of this paper. For more detail on all aspects of FTA, [17] is highly recommended.

3 Fault Tolerant Actuation and Sensing

While FTA and FMEA can indicate where more design effort is required to reach a minimum level of safety, they do not, in and of themselves, indicate how to accomplish that goal. For example, in a drive-by-wire system might require an angle sensor for the steering wheel, an actuator and second angle sensor to turn the front wheels appropriately, and a processor to control the system. FMEA might indicate that actuator failure modes have an unacceptably high

RPN. The RPN can be reduced in a number of ways: by decreasing the probability of actuator failure, increasing the ability to detect (and address) such a failure, or reducing the severity of failure.

Decreasing failure probability is essentially fault-avoidance, and can be accomplished by selection (or design) of more robust parts, or controlling the environment in which they operate, for example by filtering the air intake and power supply, or providing a better cooling mechanism. Ensuring fail-silent semantics in components that interact with this system will also help.

Sensing is essential for improving the detection of failures as well as actuator control. One particularly important technique that can be used for both purposes is called analytic redundancy. In this case, the system containing the component in question is modeled. Inputs to the real system are monitored and run through the model. Failure can be detected by comparing the outputs of the real system are then compared with the model outputs. This can be viewed as redundancy in the sense that the system is replicated (although not physically), and is comparable to duplication of processors. Additionally, the system model makes it possible to combine information from multiple sensors to get better information about the system state than is available from either. In this sense, analytic redundancy adds another, more reliable sensor to the system. This approach was used by [31] in a prototype drive-by wire system to compensate for the poor reliability of a position sensor on the output of the actuator by taking account of the relationship between position and inductance in their motor. A similar example, with more detail into the calculations involved is presented in [1].

Analytic redundancy has two major advantages. First, interaction between components makes it difficult to compare the behaviors of multiple actuators driving the same output and detect a disagreement. This does not apply to sensors, where duplication and comparison is relatively simple. Second, the analytic model does not add to the per-unit cost of the car. Physical duplication, however, does.

The detail of the models used in analytic redundancy is limited to the extent that any failure must be detected before disaster occurs. A means for co-

ordinating multiple diagnostic models across different processors is described in [27]. Once a failure is detected, some action must be taken to prevent a mishap from resulting. This may involve such steps as informing the driver (fail-safe), disabling the failed system (fail-silent), and enabling a backup (fail-operational).

The severity of a failure can be reduced by enforcing fail-operational and fail-silent semantics. Fail-operational semantics will require that either the device perform in a degraded mode or that there exists a backup system that is capable of performing the same function in a degraded mode. Having fall-back mechanisms reduces the severity of a failure in a single system. Fail-silent semantics reduce the severity by preventing the device from interfering with the backup systems and potentially causing a chain-reaction.

Design considerations for fault tolerant actuators are described in [31], which presents the results of ongoing research into steer-by-wire at Bosch. Initially, the researchers used two DC motors, but found that in the case of a failure, the failed motor would act as a damper, thus interfering with the remaining part. In this case, fail-silent semantics were not achieved. Switching to two different actuators without permanent magnets was considered as a solution, but deemed too bulky. Eventually, they settled on a single motor where each of six sections may fail independently without interfering with the continued operation of the undamaged portions. This example is interesting because it demonstrates fail-silent semantics in an electro-mechanical system, as well as degraded operation of a single component allowing reduction in severity of failure.

Redundant actuation may also be achieved through the use of orthogonal systems. A very interesting idea for accomplishing this in steer-by-wire systems is described in [3]. The idea proposed is to use the ability to independently apply braking force to each wheel in the car in order to create a rotational force about the center of the car. This should result in a compensatory rotation of the front wheels. Through the use of mathematical models, the authors demonstrate that this could achieve performance comparable to the regular steering mechanism. This solution has low added cost, as independent braking is already

present in modern cars as part of the ABS system. As an emergency backup, this is particularly compelling because the braking and steering systems are independent, so failure in the steering system cannot reduce the functionality of the backup.

The steer-by-wire research at Bosch also led to a fault-tolerant design for a sensor to determine steering-wheel angle. In the design, duplicate optical encoders and anisotropic magneto resistive (AMR) sensors are used. If a magnet is placed on a gear, the AMR can act as a precise absolute position sensor. In other words, the current state of the AMR at any single time is sufficient to determine the position of the steering column. The optical encoders, on the other hand, require that state be maintained. Each time the wheel moves, a quadrature signal is generated and the wheel position is determined on the basis of the sum of the distance the wheel has traveled and its initial position. The AMR may also provide higher precision than the optical encoders, as they are discrete. However, the AMR elements are susceptible to strong magnetic fields, at it was determine that the field generated by third rails would be sufficient to induce a temporary failure. As optical encoders are not affected by magnetic field, relying on an entirely separate physical principle, they provide an excellent complement[31, 28].

4 Communication Network Protocols for X-by-Wire

Automotive functions (e.g. braking or steering) are implemented by a number of subsystems, each of which contains a microcontroller and a set of sensors and actuators. Each such subsystem is called an Electronic Control Unit (ECU). Historically, a single ECU was sufficient for the duties of each subsystem, but over time, the functions became distributed over several ECUs, and reliable communication became essential. Until 1990, data was exchanged using point-to-point communication. However, this scheme was not scalable, as the number of communication links grew quadratically with the number of ECUs. This not only caused problems in reliability due to

link wiring but also led to increased cost, weight and complexity. In order to solve this problem, multiplexed networks were adopted[8]. It is claimed that the "wiring LAN harness in the four doors of a BMW reduced the weight by 15 kilograms[9]."

Besides providing a medium for communication, the networks also must ensure real-time execution i.e. tasks must be completed within a set period of time. For example, if a driver pushes the brake pedal, then the effects must be seen almost immediately. The Society of Automotive Engineers (SAE) classifies communication networks based on bandwidth and diverse functionality requirements[8]. For X-by-wire we consider only those networks that meet the real-time operation requirement i.e. class B and class C. Class A networks are used for low data rate transfer (< 10 kb/s) and non-real-time communication such as seat control, door lock, lighting etc., and are not relevant to the design of X-by-wire systems.

4.1 Class B networks

This class is intended for applications that need medium to high-speed (25kb/s - 1Mb/s) real-time communication. For example, this could be appropriate for an all wheel drive control system. The main representation of this class is the Control Area Network (CAN) protocol and is preferred over other protocols such as J1850 and Vehicle Area Network (VAN) [8].

4.1.1 Control Area Network (CAN)

CAN was designed by Bosch in mid 1980's and is the most widely used in-vehicle network[10]. It is an event triggered communication paradigm where the messages are transmitted to indicate the occurrence of significant events. The data is transmitted periodically or aperiodically or on demand and hence there is no specific time slot for any communication node to a send message.

CAN Overview CAN consists of a simple bus, and a number of nodes, each of which consists of a microcontroller that interfaces with the bus via a CAN controller. CAN's physical layer and Data Link Layer

(DLL) are specified in the International Standards Organization's (ISO) standards 11519-2 and 11898. The physical layer specification describes the implementation of the bus, signaling, timing, and voltages used. The DLL is the next layer and specifies the message formats.

CAN Physical Layer CAN's physical layer contains a two-wire differential bus that can transmit two values called "recessive" (logic 1) and "dominant" (logic 0). The dominant bit always overwrites the recessive bit. To achieve higher transmission rates, CAN uses a Non-Return-to-Zero (NRZ) bit representation. NRZ uses only two voltage values for bit transmission. This is different from Return-to-Zero (RZ) policy, which uses high (logic 1), low (logic 0) and reference. In RZ, the reference value is sent between each data bit, which cuts the available bandwidth in half. By eliminating this overhead, NRZ is able to more efficiently use the available bandwidth. The elimination of the reference voltage in NRZ, however, is not free. Problems arise when we have long runs of consecutive bits with the same value. The lack of transitions prevents the receiver from reliably regenerating the clock, making it impossible to detect the boundaries of the received bits at the receiver. The end result is that it is no longer possible to reliably determine the number of bits received. To address this problem, a technique called bit stuffing is used to insert artificial edges, thus enabling resynchronization. For example 000000 will be stuffed, as 0000010 and the receiver will apply the reverse procedure to unstuffed bits. Can specifies that every fifth bit is stuffed, which means that one fifth the available bandwidth is wasted. This is still a major savings over an RZ policy[10].

CAN Data Link Layer CAN frames do not contain the addresses of either the transmitting node or intended recipient. Instead, the message contains an identifier field that determines the priority of the message. All other nodes on the bus receive this message and examine the identifier to determine if the content is relevant. The bus arbitration process is determined by Carrier Sense Multiple Access with Collision De-

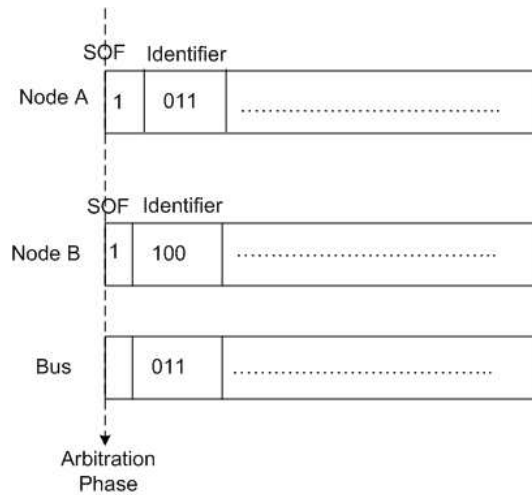


Figure 2: Collision resolution in CAN

tection (CSMA/CD) but with enhanced capability of non-destructive bitwise arbitration to provide collision resolution. This property is guaranteed by statically assigning a unique priority to each node. At the start of each communication, this priority is transmitted from most significant to least significant bit. In the case of a conflict, the node with the lower priority will eventually dominate the other node. Upon detecting that its communication has been clobbered, the node with the higher priority will back off (see Figure 2). At no point in this communication is any data destroyed.

In meeting class B requirements, CAN works very well as it efficiently utilizes network bandwidth. It also provides some fault tolerance mechanisms such as information redundancy using error detection mechanisms such as acknowledgment error check, CRC and bit stuffing. However, CAN is unable to detect the situation where a faulty node continuously transmits the dominant bit. This situation is known as the babbling idiot problem, and results in one node monopolizing the bus[33]. Because of this limitation, CAN is unsuitable for safety-critical applications such as suspension, braking and steering. Class C networks provide the predictability and high fault tolerance required for x-by-wire applications.

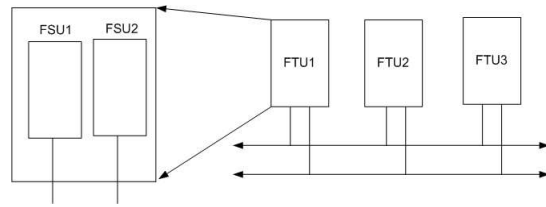


Figure 3: Time Triggered Architecture

4.2 Class C networks

According to the SAE, Class C is intended for safety-critical applications that require high dependability and guaranteed timeliness (i.e. deterministic behavior). This class provides data speed over 1Mb/s. The Time Triggered Protocol (TTP) meets these requirements[4, 30].

4.2.1 Time Triggered Protocol

TTP provides deterministic behavior as well as all services necessary for fault tolerance in a real time environment. The services include membership service, fault tolerant clock synchronization, mode change support, error detection with short latency and distributed redundancy management[4].

Time Triggered Architecture (TTA) Overview

The TTP architecture specifies both a system hardware architecture and a node architecture. The system architecture contains a set of nodes connected by a duplicated bus. Each node is required to possess fail-silent behavior.

As shown in figure 3, a node may be formed by the combination of a host microcontroller and TTP-controller. In this case, the node is called a Fail Silent Unit (FSU). FSUs may be replicated and grouped into a more complex node called a Fault Tolerant Unit (FTU). An FSU may achieve fail-silent behavior through the use of special hardware or software fault detection mechanisms, or alternatively, it may depend on other nodes within an FTU for error detection. In a duplex system, failure by one FSU will lead to the failure of the entire FTU. In a TMR FTU,

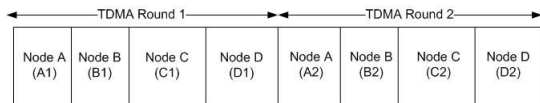


Figure 4: TDMA cycle with four nodes in TTP

a single failure may be tolerated, after which the behavior will be identical with the duplex system[2].

TTP-controllers decide when to transmit a message independently of the host microcontroller. To do this, each controller maintains its own data and message schedule. Each controller also has an independent component called a Bus Guardian (BG). The BG enforces that the controller never transmit outside of its allotted timeslot. If the controller violates this law, the BG shuts it down, thus preventing the node from interfering with the ability of other nodes to communicate. At a system wide level, outfitting each controller with a bus guardian prevents any node from monopolizing the bus. It can be seen, then, that TTP is not susceptible to the babbling idiot problem[5, 2].

TTP Protocol Description TTP uses a Time Division Multiple Access (TDMA) scheme based off the global time, which TTP maintains through clock-synchronization (as opposed to a dedicated clock signal). As shown in figure 4, each node is provided a single time slice within a communication period or round. During this time, the node has exclusive rights to transmit data. This time slice is called a membership point. The advantage of this technique is that each node has the same priority for its bus access and latency can easily be calculated. Correct behavior is enforced by the bus guardian, so collisions are only possible in the event of multiple independent failures (e.g. a single node sees both its bus guardian fail, and its TTP controller suffer a timing failure).

The TTP standard specifies the following services that are required for fault tolerant operation in a real time application, which are described in more detail in [4]:

1) State Agreement: Message frames are only accepted if both sender and receiver agree on the controller state (C-state). In TTP, the C-state consists

of three fields: the mode, the time and the membership. The mode field contains current operation mode of the system. The time field maintains the global internal time. The membership field in C-state lists which nodes were active and inactive at the last round. In order to enforce the agreement of the C-state and to reduce the message overhead, a CRC (Cyclic Redundancy Check) calculation is performed on the C-state's contents and concatenated to the data field. If the CRC of the receiver's C-state does not match that received, the message is ignored, and the sender is not added to the recipient's membership list. This will result in the sender and receiver ignoring each-other on future rounds. It is worth noting that there will typically be many nodes simultaneously receiving, so a node that writes a bad C-state will be ignored by all the recipients in the future.

2) Mode change support: In TTP, the nodes can operate under different modes such as start-up, normal operation, emergency operation etc. Each mode has its own (statically assigned) TDMA sequence, message format and static task schedule. If a node wants to change its mode, then it alerts all nodes about this change by specify the successor mode in the mode subfield of the control field at its next membership point.

3) Fault tolerant clock synchronization: A global time base of known precision is generated by fault tolerant internal synchronization of local clocks. Each node in the system knows the membership point of every other node. When a transmission is received, the actual time is compared against the correct time for the membership point. The difference in these two times is the clock skew between the sender and the recipient. To even out the skew and robustly perform global synchronization, the following calculation is performed: the greatest and smallest of the last four observed clock skews are discarded. The two remaining measurements are averaged and some additional correction term is added. The result of this computation is then used to adjust the local time[6, 20].

4) Membership service: Nodes must determine when they are out of line, and remain inactive for at least two rounds. A node is not behaving correctly if it detects an error in itself (e.g. through some software or hardware mechanism), if it has re-

jected the majority of the messages it has received in the last round (if it thinks most nodes are wrong, odds are good, it's wrong), or if it sent a message in the last round and no other node acknowledged the transmission (all other nodes think it's bad, or it is having problems sending or receiving). A node may successfully rejoin by adjusting its C-state so that it conforms with the expectations of the majority. These rules prevent a divergence in opinion that would result in some of the nodes refusing to communicate with others and potentially colliding due to disagreement on the global time.

5) Error detection with short latency: The receiving end can perform the error detection, as the receiver knows the point in time when a message should arrive. If a message is not received during a node's membership point, all receivers assume it has failed.

6) Distributed redundancy management: This service provides removal of failed nodes and reintegration of spare nodes and repaired nodes. This also helps the network property called composability[7], which means that new nodes or subsystem designed, developed and tested can be added to existing network without affecting the performance or needing adjustments to the whole network. This service directly follows from the combination of the previous services.

4.3 Hybrid Networks

While TTP meets (and exceeds) the requirements of class C networks, it makes inefficient use of available bandwidth and introduces unnecessary latency when aperiodic communication is necessary. The solution to this problem is to use networks that provide features from both (hybrid) network classes. TTCAN and FlexRay are the two protocols that fall into this category.

4.3.1 TTCAN

Time-Triggered CAN (TTCAN) protocol is an extension to CAN protocol that adds an additional protocol on top of the Data Link Layer of the CAN. It provides messages to be transmitted in event-triggered and time-triggered mode without any additional over-

head to the existing CAN frames. Time-triggered mode is indicated by a reference message. Once the reference message is recognized from the identifier in the CAN frame, the local time unit is synchronized and individual nodes are then configured to know when they can send their frames. The time between two reference frames is called basic cycle. Within a basic cycle there are exclusive windows during which only one node can send a frame. Apart from that there are free windows and arbitrating windows in which the nodes compete for the bus access just as in the regular CAN communication[13].

TTCAN is beneficial as it allows efficient use of network bandwidth by allowing event and time triggered messages. It is also low-cost as the higher layer on top of CAN is built in software. However it does not provide important dependability service such as bus guardian, clock synchronization, and membership service. FlexRay is a solution to the problems faced by TTP and TTCAN.

4.3.2 FlexRay

FlexRay is currently under development by consortium of major companies like BMW, Bosch, Daimler-Chrysler, General Motors, Motorola, Philips and Volkswagen[11]. Industry experts expect FlexRay to become the decommunicationfacto standard for high-speed control applications within vehicles.

According to the FlexRay protocol overview[12], the network is very flexible with regard topology and redundant transmission support at the physical layer. The transmission can be a bus, star, or multi-star topology. FlexRay also provides fault tolerance by distributed time-triggered synchronization (clock synchronization) and error containment on the physical layer through an independent bus guardian.

Like TTCAN, FlexRay also allows both time-triggered and event triggered communication. In FlexRay, this is accomplished by means of a communication cycle, where a time-triggered (static) window and event triggered (dynamic) window are concatenated (see figure 5). Each window has a fixed duration with some dead-time separating the two. The time-triggered window uses TDMA like in TTP, but unlike TTP, a given node may be able to access the

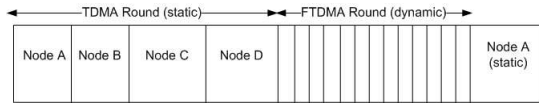


Figure 5: FlexRay Communication Cycle

bus multiple times before for all remaining nodes access it. This behavior is still statically determined, perhaps on the basis of priority. The event-triggered window uses a technique called Flexible TDMA (FTDMA) to provide event-triggered behavior without collisions. When no nodes are using a channel, time is divided into very short, fixed time periods called minislots. Each node has a unique ID and tracks the number of minislots that have passed since system startup. When that number (modulo the total number of nodes in the system) matches a node's ID, that node is free to begin transmission. Once a node grabs the bus, no more minislots are counted until the communication is finished. Termination of communication is known by the declared message size in bytes, as specified in the frame header. Because the dynamic window is of fixed length, some nodes may not get access to the bus during the dynamic window of a given communication cycle. In this case, they must wait however many communication cycles are necessary before it is their turn to access the bus. If a node does not wish to use the bus, the only overhead induced is from the time required for the minislots to pass[12].

In addition to providing high performance while meeting the class C requirements, FlexRay is expected to provide the ability to combine both critical and non-critical systems in a single network, thereby reducing design complexity. This may also result in monetary savings by reducing the number of controller types.

5 Software fault tolerance

Because software is very flexible and relatively cheap, it is a very desirable medium for implementing fault tolerant mechanisms. These may be for the purpose of verifying the correct behavior of sensors and actu-

ators as discussed previously, for detecting processor and software design failures, as well as maintaining service in the event of faults. The techniques employed can be broken into the broad categories of those that increase reliability through added redundancy, and those that are only capable of detection. It is not possible in the available space to describe in detail every fault-tolerant software technique. Instead, we will briefly explain some of the more common ones.

Redundancy in software is accomplished through redundant computation or redundant storage. Each of these can increase system cost, either by adding to the required memory or computational performance. If the increase is sufficiently small, it is possible that no additional cost will be introduced.

Perhaps the most common computational redundancy technique involves the use of a software controlled watchdog timer. Most microcontrollers have a built-in watchdog timer with a programmable timeout period. As the program executes, the timer is periodically reset by writing a new value into the timer register (commonly zero, but potentially a larger number in order to adjust the timeout period, as reset is triggered on overflow). In the event of a failure, the watchdog will typically restart the processor from a re-entry point in the code (usually somewhere past the initialization code, as memory contents will not be affected by the reset). This technique is particularly useful for avoiding deadlock conditions due to communication failures (e.g. blocking while waiting for a message that gets lost). Because re-execution only occurs in the case of a severe failure, and the overhead of inserting watchdog timer resets into the code is relatively small, there is almost no additional cost for using a software controlled watchdog timer.

A simple technique for data redundancy is complement data write. Rather than simply duplicating the data, it is first complemented. This adds in some additional fault tolerance by providing the means to detect stuck-at faults. Duplicating every piece of data may be prohibitively expensive, as it doubles the total memory requirements. Instead, it may make more sense to duplicate only the safety-critical data[19].

More complicated techniques may combine both computational redundancy and data redundancy.

For example, executing a program twice can detect and correct transient faults that result in data corruption[32], but will not be able to detect permanent faults or transient faults that affect the instruction memory. Corruption of the instruction memory can be detected and tolerated by duplicating the program in memory and executing each copy in sequence (this will still not detect and correct permanent faults in the functional units). For further reliability, it may be desired to use "redundant orthogonal coding"[19]. This technique is similar to n-version programming in that different programs that should produce the same or similar outputs are written. The difference is that in redundant orthogonal coding, different algorithms are intentionally used to perform the same calculation. This addresses the problem of n-version programming where different people tend to design programs for a given task in very similar ways and also to make similar mistakes in the design by explicitly ensuring dissimilarity. This technique may also catch programming errors. Again, extra computation and memory are required for this technique, but it is possible that the amount required is significantly less than double if one technique is simpler than the other. One instance where this is frequently used is avionic control systems that may use primary/backup control laws. The simpler control law will be more reliable and easier to compute, but provide degraded service. In normal operation, the primary control law will be invoked, and its output compared to that of the backup. In the event of a disagreement, the system will switch to the backup. For an example of this, see[14]. As the behaviors performed by X-by-wire become increasingly complex, the use of backup control laws should become prevalent in automobiles as well.

Most of the error detection techniques are focused on memory integrity. One common technique is to use checksums to verify the contents of memory or of data communications. Frequently, the checksum is computed as a summation of each byte, but more complicated checksums, particularly CRCs may be calculated at the cost of additional computation. Another technique to determine that system memory is still working correctly is to test the RAM by writing various patterns of ones and zeros into every bit

in memory. A variety of RAM tests have been developed by the test community, and there is a wide variation in the sorts of faults that each test is capable of detecting as well as the amount of time required. Another common error detection technique is to use assertion tests inside a program. The idea here is that the programmer knows ranges that values should fall within as well as logical relationships between values. Using assertion checks can catch programming errors as well as errors arising from unusual conditions (data corruption, etc.), and has little associated overhead.

The limitation of assertion checking is that it only detects errors in code and data as they are used. This can be a problem, because the least used execution paths tend to be for error handling. This could allow a hazardous condition to result from a fault when the error handling code is initiated. The solution proposed in [29] is to periodically use more intensive tests (such as computing checksums or verifying the functionality of infrequently used functional units) during operation. The authors show that testing may only be required every ten to one hundred hours. Given that automobiles are rarely driven for ten hours straight, it may then be sufficient to perform these tests only at startup, which would not require additional computational resources. The authors, however, do not consider this possibility.

6 Conclusion

In this paper, we have provided a survey of the components and methodologies used in the design of X-by-wire systems. Because the consequences of system failure can be catastrophic, extreme care must be taken throughout the design process. FTA and FMECA are important techniques that can be used to indicate where more effort is necessary to guarantee a sufficient level of safety. This effort may be required anywhere in the system, including actuator and sensor design as well as hardware and software fault tolerance mechanisms.

Because of their complexity, communication schemes are typically not designed independently for each system. Instead, one of a number of proven protocols can be chosen to meet the system's needs.

While CAN is the most common protocol in automotive applications, it does not conform to the requirements of class C networks as described by the SAE. Thus, it cannot be used for X-by-wire. TTCAN is an inexpensive solution to this problem that adds a software layer to an underlying CAN network and qualifies as a class C network. TTP is also a class C network, but is more reliable and recommended over TTCAN for X-by-wire applications[8]. While still undeployed, FlexRay is expected to be the network of choice in future X-by-wire designs, as it provides high-bandwidth, meets the class C requirements, and can meet the needs of both critical and non-critical systems, allowing it to be the only protocol in an automobile.

The widespread deployment of X-by-wire will enhance the fuel-efficiency, ride quality, safety, and affordability of all automobiles. To achieve this goal, new fault tolerant techniques to improve the reliability and safety of X-by-wire systems while keeping down cost must be developed. This represents an opportunity for research for engineers of all backgrounds.

References

- [1] R. Isermann, R. Schwarz, and S. Stölzl, "Fault-Tolerant Drive-by-Wire Systems," *IEEE Control Systems Magazine*, vol. 22, no. 5, pp. 64-81, October 2002.
- [2] B. Hedenetz and R. Belschner, "Brake-by-Wire Without Mechanical Backup by Using a TTP-Communication Network", 981109, SAE World Congress, 1998.
- [3] A.D. Dominguez-Garcia, J.G. Kassakian, J.E. Schindall, "A Backup System for Automotive Steer-by-Wire, Actuated by Selective Braking", *IEEE 35th Annual Power Electronics Specialists Conference*, vol. 1, June 2004.
- [4] H. Kopetz and G. Grunsteidle, "TTP - A Time Triggered Protocol for Fault-Tolerant Real-Time Systems", *IEEE Computer*, pp. 14-23, January 1994.
- [5] Ch. Temple, "Bus Guardian Principle of Operation TTP/C IP", Technical Report", Institut für Technische Informatik, Vienna University of Technology, Vienna, Austria, 1996.
- [6] H. Kopetz, & W. Ochsenreiter, "Clock Synchronisation in Distributed Real-Time Systems", *IEEE Trans. Computers*, vol. 36, no. 8, pp. 933-940, 1987.
- [7] M. Bertoluzzo, G. Buja, A. Zuccollo "CAN Upgrade Toward Determinism and Composability", *Industrial Electronics Society, IECON '03. The 29th Annual Conference of the IEEE*, vol. 2, pp. 1894 - 1898, Nov. 2003.
- [8] N Navet, Y. Song, F. Simonot-Lion and C. Wilwert, "Trends in Automotive Communication Systems", *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1204 - 1223, June 2005.
- [9] G. Leen and D. Hefferman, "Expanding Automotive Electronic Systems", *IEEE Comput.*, vol. 35, no. 1, pp. 88-93, Jan. 2002.
- [10] N. Navet, "Controller Area Network: CANs use within automobile", *IEEE Potentials*, vol. 17, no. 4, pp. 12-14, October 1998.
- [11] FlexRay Consortium, <http://www.flexray.com> (Dec. 2005)
- [12] "Protocol Overview", FlexRay Consortium, FlexRay International Workshop, March 2003. <http://www.flexray.com/products/protocol/overview.pdf> (Dec. 2005).
- [13] "Time Triggered Communication on CAN (TTCAN)", *CAN in Automation*. <http://www.can-cia.org/can/ttcan/> (December 2005).
- [14] B. Frisberg, "Usage of Ada in the Gripen Flight Control System", *Proceedings of the 1998 annual ACM SIGAda international conference on Ada*, pp. 140-141, November 1998.
- [15] "Standard Practice for System Safety", MIL-STD-882D, US Dept. of Defense, 2000.

- [16] "Procedures for Performing a Failure Modes, Effects, and Criticality Analysis", MIL-STD-1629A -FMECA, US Dept. of Defense, 1980.
- [17] "Fault Tree Handbook", NUREG-0492, US Nuclear Regulatory Commission, 1981.
- [18] B.J. Czerny, J.G. D'Ambrosio, B.T. Murray, P. Sundaram, "Effective Application of Software Safety Techniques for Automotive Embedded Control Systems", SAE 2005-01-0785, SAE World Congress, April 2005
- [19] E.G. Leaphart, B.J. Czerny, J.G. D'Ambrosio, B.T. Murray, C.L. Denlinger, D. Littlejohn, "Survey of Software Failsafe Techniques for Safety-Critical Automotive Applications", SAE 2005-01-0779, SAE World Congress, April, 2005.
- [20] E Dilger, T Fuehrer, B Mueller and S Poledna, "The X-by-Wire Concept: Time-Trigged Information Exchange and Fail Silence Support by new System Services", SAE 98-PC 124, 1997.
- [21] H. Pentti, H. Atte, M. Jarvinen , "Failure Mode and Effects Analysis of Software-based Automation Systems", STUK-YTO-TR-190, August 2002.
- [22] M.S. Feather, "Towards a Unified Approach to the Representation of, and Reasoning with, Probabilistic Risk Informaton about Software and its System Interface", ISSRE, 2004.
- [23] R. Bono, R. Alexander, A. Dorman, Y. Kim, J. Reisdorf, "Analyzing Reliability-A Simple Yet Rigorous Approach", IEEE Transactions on Industry Applications, vol. 40, no. 4, July 2004.
- [24] I. Kendall, "The Safety Assurance of the AJV8 Electronic Throttle", IEE Colloquium on The Electrical System of the Jaguar XK8, pp. 2/1 - 2/8, Oct. 1996.
- [25] H. Edler, J. Erikson, J. Hedberg, H. Sjostrom, "Definitions Version 2.0", PALBUS Task 10.1, April 2001.
- [26] W. Dunn, "Designing Safety-Critical Computer Systems", Computer vol. 36, no. 11, pp. 40 - 46, Nov. 2003.
- [27] N. Kandasamy, J.P. Hayes, B.T. Murray, "Time-Constrained Failure Diagnosis in Distributed Embedded Systems: Application to Actuator Diagnosis", IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 3, pp. 258 - 270, March 2005.
- [28] E. Dilger, M. Gulbins, T. Ohnesorge, and B. Straube, "On a Redundant Diversified Steering Angle Sensor", On-Line Testing Symposium, pp. 191 - 196, July 2003.
- [29] C. Scherrer, and A. Steininger, "Dealing with Dormant Faults in an Embedded Fault-Tolerant Computer System", IEEE Transactions on Reliability, vol. 52, no. 4, pp. 512 - 522, Dec. 2003.
- [30] F. Ataide, M. Santos, and F. Vasques, "A comparison of the communication impact in CAN and TTP/C networks when supporting steer-by-wire systems", Industrial Technology, vol. 2, pp 1078 - 1083, Dec. 8-10 2004.
- [31] E. Dilger, R. Karrelmeyer, and B. Straube, "Fault tolerant Mechatronics, On-Line Testing Symposium, pp. 214 - 218, July 2004.
- [32] V. Claesson, S. Poledna, J. Soederberg, "The XBW Model for Dependable Real-Time Systems", Proceedings of the International Conference on Parallel and Distributed Systems, pp. 130-138, Dec. 1998.
- [33] C. Temple, "Avoiding the Babbling-Idiot Failure in a Time-Triggered Communication System", Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, pp. 218 - 227, 23-25 June 1998. e IEEE, vol. 93, no. 6, pp. 1204 - 1223, June 2005.