2006-01-1026

Design of a Safety-Critical Drive-By-Wire System Using FlexCAN

Manuele Bertoluzzo, Giuseppe Buja
University of Padova, Department of Electrical Engineering

Juan R. Pimentel
Kettering University, Department of Electrical and Computer Engineering

ABSTRACT

This paper describes the design of a drive-by-wire system for a commercial lift truck using the FlexCAN communication architecture. FlexCAN is a recently developed architecture based on the CAN protocol to support deterministic and safety-critical applications. The main features of FlexCAN are its simplicity and ready implementation based on COTS CAN components. The main steer-by-wire design tasks are listed and a description of how each of the tasks was accomplished using the FlexCAN architecture is detailed. A performance evaluation of the design is included.

INTRODUCTION

It is the objective of this paper to address the dependability requirements of the drive-by-wire system of a lift truck and its implementation using the FlexCAN architecture [1]. Drive-by-Wire (DbW) systems are all-electric systems, which are expected to replace the mechanical/hydraulic means that transmit and actuate the driving commands in a car in a few years [2]. The objective is the improvement of the overall automotive performance. As a matter of fact, the DbW systems enhance the safety of the vehicle occupants. This is done by making the conditioning of the driving commands feasible, by allowing more accurate maneuvers thanks to the use of closed-loop controlled electric drives, and by increasing the car's efficiency, since they utilize electrical equipment with much less losses.

Currently there is a thriving activity designing and developing DbW systems. The main issues are the assessment of suitable architectures and the validation of their dependability, especially when the DbW systems are employed for executing safety-critical commands such as steering and braking [3].

The application of DbW systems in the industrial vehicles (e.g., lift trucks) has not been investigated as deeply as in the automobile sector even if they pose less stringent demands in terms of safety and technical specifications. This because the industrial vehicles typically are driven by qualified workers, have a cruising speed much slower than a car and operate in enclosed spaces such as yards or storehouses.

In this paper a DbW system responsible for the steering and acceleration maneuvers of a commercial lift truck is presented. Attention is focused on the control architecture of the DbW system and the design of the communication network. The in-progress activity to validate the design is also illustrated. In detail, the paper is organized as follows. The Drive-By-Wire System section describes the trial lift truck used in the study, discusses the safety requirements for the vehicle and derives the architecture of the DbW system. The section on the FlexCAN architecture analyzes the characteristics of the communication architecture to be used, namely FlexCAN. The FlexCAN based DbW System section explains the implementation of the DbW system using FlexCAN. The next section evaluates the dependability and communication performance of the designed DbW system. The last section concludes the paper.

THE DRIVE-BY-WIRE SYSTEM

A. LIFT TRUCK DESCRIPTION

The lift truck shown in Fig.1 is considered in the paper. The truck, manufactured by Cesab S.p.A., has four wheels, is fed by a 48 Volt battery and is equipped with hydraulic circuitries for steering, hoisting and low-speed braking, and with an electric system for traction. The steering wheels are the rear ones. They are directed by the ram of a cylinder that is inserted in a hydraulic circuit pressurized by an electric motor. The cylinder operates parallel to the floor in the same way as the transmission bar of a car. The mast is also moved hydraulically and the

associated circuit shares the pressurizing motor with the steering circuit. The traction wheels are the front ones and are powered by two induction motor drives, one for each wheel.

Figure 1. The lift truck.

Input devices for the driving commands are the hand wheel, which mounts a distribution valve feeding the steering circuit, two accelerator pedals – one to go forward and the other to go backward – and a brake pedal, which engages its own hydraulic circuit and is employed for the accurate positioning of the truck at low speeds. A potentiometer located on the stub axle of one of the rear wheels detects the steering angle of that wheel. It is entered into and processed by the main control system of the two traction drives to separately adjust the speed of the motors in order that the wheels behave as though they were connected to a driving shaft with a differential gear.

An electronic board, called an IOboard, acquires the speed command from the accelerator pedals, the hoisting commands from three control levers situated in the cockpit and on/off-type commands from push buttons placed on the dashboard. Another electronic board, called a Dboard, switches the dashboard indicators. The IOboard, the Dboard and the traction drives are connected at a bit rate of 125 Kbit/s via a CAN network with a proprietary application protocol. The network transmits i) the IOboard input commands to the traction drives, the hoisting motor and the Dboard, and ii) the actual values of the steering angle and the cruising speed together with the status of the traction drives to the Dboard.

B .LIFT TRUCK DBW ARCHITECTURE

The DbW system under design must be in charge of executing the steering and speed commands of the lift truck [4]. Its architecture is strictly related to the safety requirements for the vehicle, which are a fault-tolerant behavior for both acquisition and actuation of the driving commands [5], [6]. This means that, regardless of the fault, the system must maintain the capabilities of developing steering and tractive forces in the actuation chain, and the capabilities of entering and transmitting the command signals in the input chain. The fault-tolerant requirement is met by duplicating the devices of the DbW system. To restrict the overall size of the electric drives, an actuating operation with degraded performance is accepted. To recognize for sure the occurrence and the source of a fault, both the device redundancies and plausibility checks of the variables are exploited. The arranged DbW system has the architecture of Fig.2. Its installation requires some modifications to the original truck, with the addition of sensors, electric drives and electronic units, where the latter ones are microprocessor-based boards with embedded devices for handling the sensors and the electric drives and for carrying out the communication services, including the bus transmission.

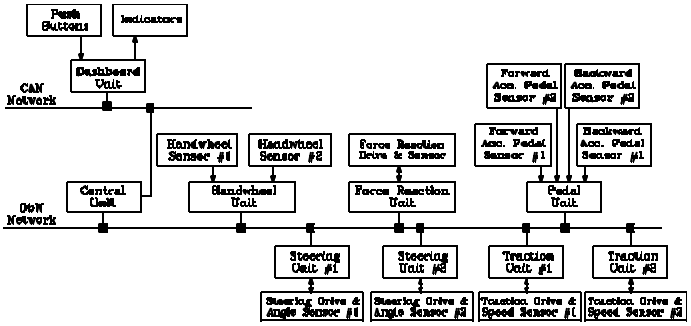

Figure.2. DbW system architecture.

The hand wheel and accelerator pedals each have two sensors for the duplicated sensing of the commands, which are then entered into the Handwheel Unit and the Pedal Unit, respectively. Besides acquisition of the commands, the input units check the sensor integrity and send the acquired data on the bus of the DbW network.

The steering cylinder is replaced by two electric drives, with the motors coupled to the ball screw linking the rear wheels. Each drive is sized to develop half the power needed by the steering maneuver in steady state and full power during transients. Therefore if one drive fails, the other one maintains the steering of the truck for a little while and then moves the truck in a downgraded manner. An equal situation occurs for the traction drives.

Steering Units #1 and #2 control the two steering drives while Traction Units #1 and #2 do the same with the traction drives. The latter units differ from the original ones in the communication interface, which is now interconnected to the DbW network rather than to the CAN network. The actuation units receive the steering angle and speed references via the DbW network and locally close the control loops using the feedbacks provided by the respective sensors. Moreover, the units transmit the feedback and the status of the drives on the bus. A low-power drive, fitted in the hand wheel column and controlled by the Force Reaction Unit, lets the driver feel the effort taken by the steering maneuver [7].

The Central Unit processes all the messages coming from the other units and manages the vehicle. It conditions the

driving commands sent by the input units before generating the references for the actuation units. Conditioning depends on the traveling situations so that, for example, if either a tight swerve is required or the load is lifted up, the speed is reduced to avoid overturning the vehicle. The Central Unit, moreover, brings the vehicle to a safe status whenever a fault is detected. All the units of the DbW system are programmed in such a way that they block any message transmission when faults are detected. This helps the healthy units in recognizing the fault and prepares them to receive the commands scheduled in the presence of a fault from the Central Unit. Furthermore, the Central Unit works as a gateway between the DbW network and the pre-existing CAN network, which is utilized to its extent of connecting the Dashboard Unit with the hoisting system (not shown in Fig.2), the push buttons and the indicators.

C. DBW NETWORK SPECIFICATIONS

The specifications for the DbW network are expressed in terms of the following parameters: information exchanged between the devices, number of bytes transmitting the information, level of reliability of the communication tasks, and update rate of the data. Eight kinds of messages are transmitted on the DbW network: speed command (SpeedCom), speed reference (SpeedRef), actual speed and status of the traction drives (TracStatus), steering angle command (SteerCom), steering angle reference (SteerRef), actual steering angle and status of the steering drives (SteerStatus), force feedback reference (ForceRef), and data exchanged with the CAN network through the gateway (CANMsg).

The SpeedCom message originates from the Pedal Unit and is conditioned by the Central Unit to obtain the SpeedRef message sent to the Traction Units. Both the messages use two bytes to encode the sign and magnitude of the speed. Before conditioning, the magnitude is proportional to the position of the accelerator pedal while the sign depends on which of the two pedals is pressed. Two TracStatus messages are sent on the network, one for each Traction Unit. Every message contains the actual values of speed, current and temperature of a traction drive, with the first two variables coded in two bytes and the third one in one byte. The SteerCom message contains the steering command coded in two bytes. It originates from the Handwheel Unit and is conditioned by the Central Unit to obtain the SteerRef message sent to the Steering Units. Each Steering Unit, in turn, sends a SteerStatus message with the same type of data as the TracStatus messages but pertinent to the steering drives. In addition to handling the vehicle, the Central Unit extracts the commanded and actual values of the steering angle from the SteerCom and SteerStatus messages to calculate the force reaction reference and transmits it to the Force Reaction Unit with the ForceRef message.

For the communication tasks to be adequately reliable, all the messages transmit two additional data bytes. The first byte codes the status of the transmitting unit and the occurrence of an alarm, if any. The second byte is a sort of echo computed as a function of the last data received and sent on the bus with the next message to notify the transmitter of the correctness of the received message. The length of the data is hence eight bytes at most, so that protocols with data frames having a short data field are conveniently adopted for the application. The update rate of the data is chosen to be 10 ms, a value which is correlated to the rated speed of the truck and is dictated by the need of not impairing its control.

There are three communication network protocols most likely to be used in DbW systems: FlexRay, TTCAN, and FlexCAN. FlexRay is a time-triggered protocol specifically devised to meet the DbW requirements of determinism, fault tolerance and reliability [8]. It utilizes the Time Division Multiple Access (TDMA) method to access the bus. With TDMA the nodes of a network access the bus at fixed time instants and occupy it for fixed time intervals, commonly termed time slots. This avoids collisions among the messages and the consequent retransmission delays. For non-scheduled messages the protocol accommodates for a fixed-length event-triggered segment within the communication cycle. FlexRay also comes with built-in duplication of the bus. TTCAN is an evolution of CAN, a protocol developed by Bosch at the end of the 80's with the aim of providing a suitable solution for the data exchange between devices on board the vehicle [9]. Like FlexRay, TTCAN utilizes both time-triggered and event-triggered transmissions but, unlike FlexRay, it supports limited fault-tolerance and reliability requirements (e.g., it does not have bus duplication). When redundant transmissions are required, every node must be built up with two or more TTCAN controllers and application software must be arranged for handling replication. Like FlexRay, the FlexCAN [10-12] architecture has been designed to meet requirements of determinism, fault tolerance and reliability, and this has been achieved by adding extra features to CAN. In this paper the FlexCAN architecture is utilized to design the DbW system of the lift truck.

THE FLEXCAN ARCHITECTURE

The goal of the FlexCAN architecture is to provide additional real-time and dependable capabilities to the CAN protocol. This is accomplished by incorporating an additional layer on top of CAN, just as TCP provides reliable-data transfer as an additional layer on top of IP which provides unreliable data transfer. The main features of the FlexCAN architecture are: replicated architecture, support for time-domain composability, replica synchronization, replication management, and enforcement of fail-silent behavior.

As depicted in Fig. 3, the FlexCAN architecture can incorporate several replicated channels and several replicated nodes (in addition to normally non-replicated ones) in a flexible fashion. A node and all of its replicas is called an FTU (fault tolerant unit), but not all nodes in a network need to be replicated. Time-domain composability has been a major liability of CAN-based networks for safety-critical applications. There have been several proposals to overcome this limitation such as TTCAN and FTT-CAN, which basically adopt a time-triggered transmission scheme at a high level and still use CAN at the lower level. FlexCAN also adopts the time-triggered paradigm by using reference messages generated by any data source (i.e., from the application). The interval between reference messages is called the basic cycle, and this interval is further divided into a number of sub-cycles or sub-windows. Whereas TTCAN uses a distributed clock synchronization mechanism to implement the time-triggered scheme, FTT-CAN uses a master node to generate the reference messages. FlexCAN on the other hand relies on node-replication to support reference messages on a distributed basis without the need of synchronized clocks. The basic cycles, together with their sub-cycles, not only help with time-domain composability but also help to synchronize replicated nodes and channels, and to enforce fail-silent behavior, particularly using bus guardians. FlexCAN assumes that all messages are periodic, with the period equal to the basic cycle. Periodic messages with different periods can also be handled, but the details are not provided here.

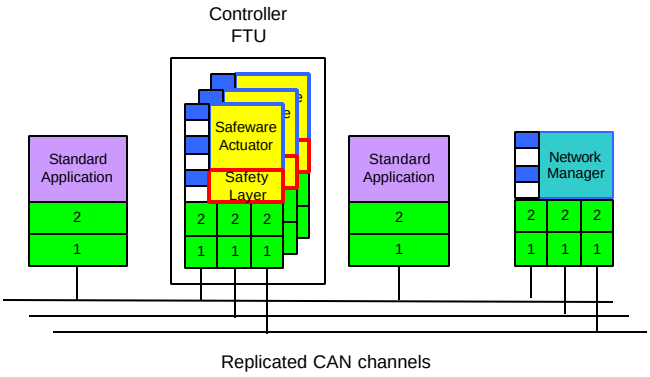

Figure 3. The FlexCAN architecture.

Providing message synchronization on replicated channels is simple, as each node simply sends the same message on all replicated channels in an atomic fashion (i.e., without interruption). However providing message synchronization on replicated nodes is not trivial, and thus a special protocol known as SafeCAN is provided to handle node replication management. There are a number of fault-tolerant features in SafeCAN that are similar to the token bus protocol (IEEE 802.4). The token concept in 802.4 is used to grant a certain node access to the bus on a global and fair basis where token rotation and inconsistencies are handled using a node hardware address. In a similar fashion, SafeCAN grants a certain replica node access to the bus on a local and unfair basis where token rotation and inconsistencies are handled using a node hardware address. SafeCAN is a local and unfair protocol in that its universe of discourse are the nodes in a particular FTU, and it is not important that the token is shared by all nodes in the FTU. In this sense, it is a greedy protocol in that the node with the token (called the primary node) will not release it until it fails. The replacement of the primary node (called the secondary node) is always ready (provided the hardware is available). In terms of the type of redundancy algorithm used, FlexCAN uses a combination of static and dynamic redundancy. The SafeCAN protocol assumes that nodes are fail silent. To enforce such a fault model, FlexCAN uses a similar technique proposed in the FTT-CAN protocol to remove the message from its transmit buffer after a certain interval called the transmission attempt window (TAW) and also by using a special purpose bus guardian. If an additional bus fault tolerant mechanism is needed, the recently developed ReCANcentrate, a replicated CAN star topology, can be used in the FlexCAN architecture.

In the following, additional details of the time-triggered feature are given. The basic cycle T_c is divided into Q sub-cycles each of equal length T_{sc}. Regardless of their node location there can be up to P messages allocated per sub-cycle. Thus the maximum number of messages in the network is PxQ. The goal is to have all messages in a sub-cycle transmitted before the next sub-cycle. This is based on a principle of time independence. Clearly, if this is enforced there will be no message queuing from one sub-cycle to the next and therefore from cycle to cycle. This peculiar message allocation scheme is the one adopted by FlexCAN. All allocations are done in an off-line fashion, just like TTCAN. Unlike TTCAN, this scheme does not require clock synchronization across all nodes, but simply requires management of timers with a minimum resolution of about 0.2 ms.

Just like TTCAN, to implement the time-triggered scheme, FlexCAN requires a synchronization message to explicitly mark the beginning of each cycle. But unlike TTCAN, there is no need for node clocks to be synchronized. Instead, FlexCAN relies on timers to divide the entire cycle into Q (e.g., 4) sub-cycles. TTCAN requires clocks to be synchronized because the *exclusive windows* are allocated to a single message. On the other hand, in FlexCAN the sub-windows are allocated to a group of messages, thus requiring less precision in the definition of the beginning and end of the time windows. In this way, FlexCAN also supports *exclusive windows* but on a group basis. However, the main advantage of FlexCAN over TTCAN in terms of time-triggered and dependable features is that TTCAN has disallowed frame retransmissions, a notable feature of CAN for dependable operation, whereas FlexCAN lets CAN retransmit a frame in error but up to a

certain time limit. Thus FlexCAN enforces strict message deadlines.

FlexCAN tolerates the following kinds of faults: transient arbitrary faults, permanent hardware faults, and permanent software babbling idiot faults. The semantics of these faults are as follows. Transient arbitrary faults are the kind of faults that are detected by the native CAN protocol and result in error and/or overload frames. Permanent hardware faults are permanent faults in the communication controller, transceiver, or bus, and they are masked by redundant nodes or busses. Permanent software babbling idiot faults are caused by software errors in a host controller that uses the bus with wrong values and at wrong times.

FLEXCAN BASED DRIVE-BY-WIRE SYSTEM

The complete design of a distributed embedded system (DES) including the communication system can be complex. In [13], Mishra, and Naik have outlined a detailed set of design tasks that must be completed, and it is summarized in Table A. On the other hand, the detailed analysis of the drive-by-wire system in Section 2 has resulted in a set of messages and ECUs which are summarized in Table B.

Table A. Main DES design tasks.

Communications	**Control**	**Computing**
Protocol: CAN, FlexCAN, FlexRay (max. frame size)	Estimation algorithms	O.S. ? (none, executive, scheduler)
Network layout: dual bus, number of ECUs	Control algorithms: link controllers, tuning, sturdy Control architecture: hierarchical, supervisory control	Overall software configuration (programming language, O.S., CAN drivers, I/O, timers)
Network Hardware layout: Detailed microcontrollers	Sensors/actuators Sensor fusion	Signal to message mapping
Fault tolerance: Replicated main controllers, Bus guardians	Function to task mapping Task code generation	Frame configuration
Details: Data rate, cycle time	Estimation of task WCETs	Task scheduling (if any)
Message priorities	Determination of task interface variables	Special task programming (not done by code generation)
Global message scheduling	Task precedence constraints	Software and hardware integration

The first important choices involve the communication protocol, the operating system (if any), the microcontroller and associated drivers. For this paper we use the FlexCAN communication architecture based on the CAN protocol. For some applications, particularly if they run on top of backbone type networks (e.g., FlexRay) the use of a real-time operating system (e.g., OSEK) is appropriate. However for safety-critical systems such as a drive-by-wire, the advantages of a real-time operating system have not yet been demonstrated. For simplicity, we do not use an operating system and rely on a real-time executive instead. We have decided to use the Freescale MC56F8367EVM microcontroller.

Table B. Message details of the drive-by-wire system

Message	**Size (bits)**	**ECU**	**Functional Description**
M1	32	Hand wheel (HW)	Steering angle command
M2	32	Pedal (P)	Acceleration command
M3	64	Central (C)	Acceleration Reference (32 bits) Steering angle reference (32 bits)
M4	56	Traction 1 (T1)	Speed and status
M5	56	Traction 2 (T2)	Speed and status
M6	56	Steering 1 (S1)	Speed and status
M7	56	Steering 2 (S2)	Speed and status
M8	32	Force reaction (FR)	Force feedback
M9	64	Central (C)	Gateway message

The control architecture and control algorithms were summarized in Section 2 and described in more detail by Bertoluzzo et al. in [4]. For the drive-by-wire system, the signal-to-message mapping is straightforward, since there is a direct correspondence between signals (steering, acceleration) and the corresponding message. Messages m4 through m7 carry a combination of various signals, speed, motor current, and temperature taking 2 bytes, 2 bytes, and 1 byte respectively. The function-to-software task mapping for this application is simple, since there is no operating system and software tasks are modularized as C functions within a node. Instead of tasks there are monolithic programs (one per ECU) that implement the functions summarized in Table B, including the communications and additional safety considerations described in Section 2.

In terms of dependability, the central ECU in Fig. 2 constitutes a single point of failure. Since it can have safety implications, it will be duplicated. We will also duplicate the CAN bus, resulting in the network layout of Fig. 4. Although not depicted in Fig. 4 but evident from Fig. 2, the sensor inputs to the HW and P control units are also duplicated to improve the fault-tolerant behavior of the overall system.

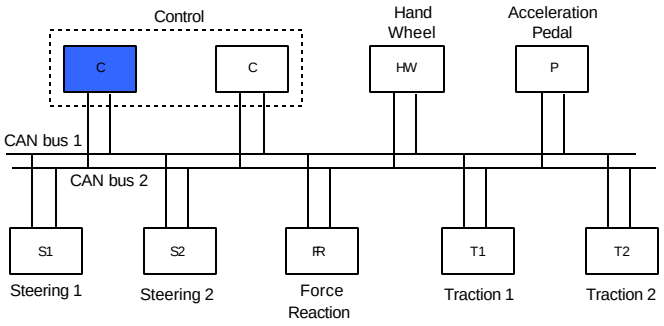

Figure 4. Drive-by-wire network layout.

Since FlexCAN requires a data source to generate reference messages, we chose *the steering angle command* message m_1 as the reference message with a basic cycle of 10 ms and a sub-cycle of 2.5 ms (i.e. Q = 4). We then generate the global message schedule as shown in Fig. 5, which also shows the write times for the bus guardians. Table C shows the worst-case message latencies for a 1 Mbps network for various error situations (0, 1, 2, or 3 errors per sub-cycle). For example, assuming 3 errors per sub-cycle (an extremely high error rate) the data transmission, including the errors takes 3.314 ms, resulting in a 33.14 % bus utilization.

SOFTWARE ISSUES

The global message schedule depicted in Fig. 5 provides the synchronization among the software running on the various nodes. Because of its role as the source of the reference message, the hand wheel node is where everything begins in terms of a data chain. This node has an interrupt set to the basic cycle (10 ms). When the pedal ECU sees the message ID for m1 on the bus, it samples the accelerator pedal position and queues message m2 for transmission. Meanwhile, ECUs S1, S2, and FR, upon seeing message m1 on the bus, start timers programmed to interrupt after 2.5 ms, after which they sample their corresponding signals and send messages m_6, m_7, and m_8. Likewise, ECUs T1 and T2, upon seeing message m_1 on the bus start timers set to interrupt after 5 ms, after which they sample their corresponding signals and send messages m_4 and m_5. In the meantime, when the central ECU reads messages m1 and m2 it begins processing the commands contained in these messages to generate the corresponding references contained in message m_3. The central ECU will queue messages m_3 and m_9 for transmission 7.5 ms after seeing the ID for message m_1 on the bus by means of an interrupting timer. The software in the central ECU and its backup (i.e., replica) are identical. The only difference is that the hardware addresses in the central ECU and its replica are set to 00 and 01 (binary) respectively. The SafeCAN protocol will take care of initialization and the fault-management details transparently from the user.

DEPENDABILITY AND PERFORMANCE EVALUATION OF THE DESIGN

In terms of payload, protocol efficiency (*?*), and bus utilization (U_b) calculations, *?* is defined as the ratio of the payload bits *A* to the total number of bits (*F*) of the corresponding data frame. For CAN, assuming 11-bit identifiers,

$$F = \lceil (34 + A)/5 \rceil + 47 + A \qquad [1]$$

Thus,

$$h = \frac{A}{F} \qquad [2]$$

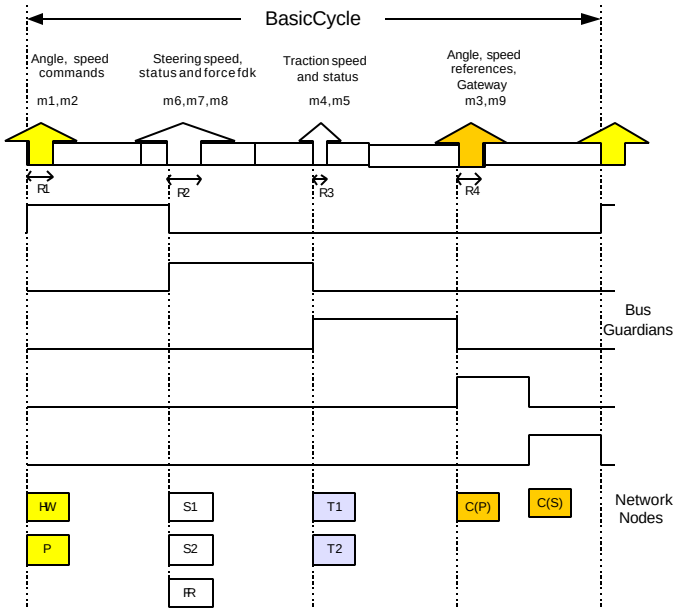

Figure. 5. Global message schedule including network nodes and bus guardians.

The maximum efficiency of CAN occurs when *A* = 64 bits (8 data bytes) yielding *?* = 48.85%. For a 29-bit identifier such as the option used in FlexCAN or the SAE J1939 standard, the total number of bits in the frame is

$$F = \lceil (53 + A)/5 \rceil + 66 + A \qquad [3]$$

Table D lists the payload bits A, the total number of bits F per message frame, the protocol utilization per message,

and the CAN identifiers used for all messages of the DbW system.

For a cyclic communication pattern such as the one used in FlexCAN, bus utilization is a more appropriate performance measure than protocol efficiency. There are three types of intervals in a sub-cyclic interval T_{sc}: message transmission interval T_{tx}, the bus inactivity interval T_{ina}, and the free interval T_{free}. All error-free messages are transmitted during the message transmission interval. Likewise all error and overload frames are transmitted during the inactivity interval. The interval when the bus is idle and could be used to transmit additional messages not currently scheduled is called the free interval. These intervals need not be continuous, and can be spread out over the entire sub-cycle. Thus,

$$T_{tx} + T_{ina} + T_{free} = T_{sc} \qquad [4]$$

The bus utilization is defined as the ratio of the time period when there is bus activity (i.e., data and error transmission) over the sub-cyclic interval. That is,

$$U_b = \frac{T_{tx} + T_{ina}}{T_{sc}} \qquad [5]$$

Table C. Maximum message latencies (in microseconds) for global message schedule with errors.

University of Padova Lift Truck Project					
Message Schedule	1 Mbps	0 Errors	1 error	2 errors	3 errors
M1+m2	R1	232	382	532	682
M6+m7+m8	R2	404	582	760	938
m4+m5	R3	288	466	644	822
m3+m9	R4	308	496	684	872
	Total	1232	1926	2620	3314

The control delay is an important performance measure for network-based control systems such as the DbW system. It is defined as the time interval from when a sensor value is taken until the corresponding reference signal is delivered to the control system. For the steering control system, the control delay involves the time to acquire a steering angle signal and generate the steering command message (very small), the queuing and transmission of this message on the bus, the time for the central ECU to calculate the steering reference message, the queuing and transmission of this message on the bus, and the generation of the reference signal for the steering control system. Because of the message schedule shown in Fig. 5, the control delay is between 7.5 and 10 ms. As noted; the FlexCAN architecture tolerates several errors per sub-cycle. In Table C we have calculated the sum of message transmission interval T_{tx} and the bus inactivity interval T_{ina} and labeled it Ri for 1, 2, and 3 errors. The worst-case calculation is 3 errors per sub-cycle (2.5 ms), which is equivalent to a bit error rate (BER) of 3/2500 or 1.2×10^{-3}. Table E shows the performance details of the FlexCAN-based design of the DbW system

Table D. Details of FlexCAN frames with message IDs.

Message	**Applica-tion Size (bits) A**	**Frame Size with stuff bits F**	**Protocol efficiency per message**	**Message ID (Priority) In Hex**
M1	32	116	27.58 %	100
M2	32	116	27.58 %	300
M3	64	154	41.55 %	200
M4	56	144	38.88 %	400
M5	56	144	38.88 %	480
M6	56	144	38.88 %	500
M7	56	144	38.88 %	580
M8	32	116	27.58 %	600
M9	64	154	41.55 %	280

Table E. Performance details of FlexCAN-based DbW system.

Data Rate (Mbps)	*No. of buses*	*Basic Cycle (ms)*	*No. of ECUs*	*No. of messages*
1	2	10	8	9
Replicated ECUs	*BER Tolerated*	*Control Delay (ms)*	*Bus Utilization*	
Central	1.2×10^{-3}	[7.5-10]	33.14%	

SUMMARY AND CONCLUSIONS

The detailed design of a FlexCAN-based safety-critical DbW system for a lift truck has been presented. The FlexCAN design is currently being implemented at Kettering University with the final overall implementation and testing to be performed at the Laboratory of Industrial Automation, University of Padova, Italy. The number of messages of the actual safety-critical DbW system is in the order of magnitude of 10, well within the capabilities of FlexCAN. FlexCAN reduces jitter thanks to its TDMA feature. Global message scheduling is simple, even when one takes communication errors into account. The assumption of 3 errors per sub-cycle (2.5 ms) is equivalent to a BER of 1.2×10^{-3}, well below that found in actual environments. The DbW system is a safe design with some fault-tolerant features and meets the safety-critical requirements specified at the beginning of the project.

REFERENCES

1. J.R.Pimentel, “An Architecture for a safety-critical steer-by-wire system,” Proc. of the *SAE* World Congress, Detroit, Michigan, 2004.

2. E.A.Bretz, "By-wire cars turn the corner", *IEEE Spectrum magazine*, vol.38, no.4, Apr. 2001, 60-73.

3. M.Bertoluzzo, P.Bolognesi, O.Bruno, G.Buja, A.Landi and A.Zuccollo, “Drive-by-wire systems for ground vehicles”, in Proc. of IEEE International Symposium on Industrial Electronics, pp.711-716, 2004.

4. M.Bertoluzzo, G.Buja and A.Zuccollo, “Design of drive-by-wire communication network for an industrial vehicle”, Proc. of IEEE International Conference on Industrial Informatics (INDIN), pp.155-160, 2004.

5. E.Dilger T.Führer, B.Müller and S.Poledna, "The X-By-Wire Concept: Time-triggered information exchange and fail silence support by new system services", SAE, Paper 8-PC124, 1998. Available: http:// www.vmars.tuwien.ac.at /projects /xbywire /projects /new-bosch.htm

6. R.Isermann, R.Schwartz and S.Stolzl, "Fault-tolerant drive-by-wire systems," IEEE Control Systems Magazine, vol.22, no.5, Oct. 2002, 64-81.

7. A.T.Zaremba, M.K.Liubakka and R.M.Stuntz, "Control and steering feel issues in the design of an electric power steering system," in Proc. of American Control Conference, 1998, vol.1, 36-40.

8. C.Temple. (2004, June). Protocol Overview, Presented at FlexRay International Seminar. [Online]. Available: www.flexray.com/publications.php.

9. T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel and M. Walther, “Time-triggered Communication on CAN (Time-triggered CAN-TTCAN)”, Proc. of the 7th International CAN Conference, 2000. http://www.can-cia.de/can/ttcan/fuehrer.pdf.

10. J.R.Pimentel and J.Kaniarz, “A CAN-Based Application Level Error-Detection and Fault-Containment Protocol”, Proc. of 11th IFAC Symp. on Information Control Problems in Manufacturing (INCOM), Salvador, Brazil, 2004.

11. J.R.Pimentel and J.A. Fonseca, “FlexCAN: A Flexible Architecture for highly dependable embedded applications,”, RTN 2004 – 3rd Int. Workshop on Real-Time Networks, held in conjunction with the 16th Euromicro Intl. Conference on Real-Time Systems, Catania, Italy, June 2004.

12. G.Buja, J.R.Pimentel and A.Zuccollo, “Overcoming Babbling-Idiot Failures in the FlexCAN Architecture: A Simple Bus-Guardian,”, Proc. of the 10th IEEE Int. conference on Emerging Technologies on Factory Automation (ETFA 2005), pp. 461-468, Catania, Italy, Sept. 2005.

13. P.K.Mishra, and S.M.Naik, Distributed Control System Development for FlexRay Based Systems, SAE paper 2005-05AE-329.

CONTACT

Dr. Juan R. Pimentel, Professor
Department of Electrical and Computer Engineering
Kettering University
Flint, MI 48504
USA
Email: jpimente@kettering.edu

Manuele Bertoluzzo and Giuseppe Buja
Department of Electrical Engineering
University of Padova
Padova, Italy
E-mail: giuseppe.buja@unipd.it
bertoluzzo@die.unipd.it