Implementation of CAN bus in a concept ecocar.

31015 Introductory project - Electrotechnology: GRP_10_MEK1_Ecocar

Peter Skov Bornerup (s153056)
Department of Electrical Engineering,
Technical University of Denmark

Andreas Stybe Petersen (s153610)
Department of Electrical Engineering,
Technical University of Denmark

Søren Møller Christensen (s153571)
Department of Electrical Engineering,
Technical University of Denmark

Tobias Krebs (s153341)
Department of Electrical Engineering,
Technical University of Denmark

Håkon Westh-Hansen (s154313)
Department of Electrical Engineering,
Technical University of Denmark

***Abstract*—**

In this paper, the implementation of Controller Area Network (CAN) bus in a concept eco car is described. A short introduction to the physical layer of the CAN bus is given. This provides the basis for the necessary board design, where it is chosen to use the microcontroller board Teensy 3.6 and the high-speed CAN transceiver MCP2551. Furthermore an innovative protocol using the extended ID is designed. The protocol is designed to lower the CPU load by using the built in filters and making it easily adaptable to expansions of the system, by adding more nodes and sensors. Furthermore a publisher subscriber model of communication is obtained to make all boards relatively independent of the rest of the system.

***Keywords*—CAN bus, CAN Protocol, extended ID, PCB design, Publisher/Subscriber model**

I. Introduction

As a continuation of the DTU Road Runners Electronics Project, the work of the previous generations was inherited. Four PCBs have been used to control the urban concept car. A motor-, front-light-, back-light- and steering PCB - All redesigned for this generation. This year the main priority was to upgrade the communication system from UART 485 serial communication to controller area network bus (CAN bus) in the urban concept eco car.

The serial communication system in the car previously worked as a master slave relationship between the motor PCB (master) and the three other PCBs (slaves). The need for an upgrade arose with the continuous addition of nodes to the network, increasing the workload of the motor PCB, eventually resulting in an unnecessary complexity. Therefore it was chosen to implement CAN bus.

CAN bus is known for its robustness. This is due to its renowned built in error handling, that contains five different ways of detecting errors. CAN bus handles all these automatically, eg. turning off the bus traffic from a node if it transmits errors. CAN bus also manages collisions where multiple devices tries to transmit data at the same time [7]. These attributes secure a stable system, which is needed in an automotive system like the eco car.

CAN bus is also known for its simplicity once it is set up. This is of course the most favourable solution, since this means that the transition to the next generations will be smoother, because of the reduced complexity. The addition of new nodes will also be a simpler process than before, since editing of previous code becomes unnecessary, making it easier for the next generations to add a new PCB if needed.
Narrowing to the essentials of the problem.

A. Problem definition:

The communication between the control units needs to be secured for future expansions, by implementing CAN bus ISO-11898:2003

- The standardised communication protocol, ISO-11898:2003, 'CAN bus' needs to be implemented, to make sure that all PCBs in the car can communicate with each other.

II. Limitations

The project is limited in the way that the electronics have to meet the requirements from the Shell-Eco Marathon 2017 Official Rules. Several rules like a horn, break light, and emergency button etc. has to be implemented in the car

- For easy implementation of everything electrically related to the car the microcontroller Teensy 3.6 will be used throughout the entire redesign of the PCB's from last years design. It also

incorporate an on board CAN controller for CAN bus communication.

- It was predetermined that MCP2551 as line driver 5 V was to be used.
- To simplify programming the Teensyduino add on was heavily used. This simplification is on expense of a slightly slower system, since the built in functions is not necessarily optimised.
- Every PCB will be designed with specific hardware solutions for the single application such that every component will have a custom solution for connection to the Teensy.

III. CAN BUS PHYSICAL LAYER

CAN bus is a network solution that is used to interconnect electric nodes or modules. It was develop by Bosch to simplify the way electronics communicate in cars. Instead of having a large controller which is connected to every sensor, light or cooler etc. resulting in a complex wiring, a bus is introduced where every module is connected leading to a more simple and reliable system.

A two wire bus is terminated with 120 Ω resistors, it is driven by a line driver on each module which is connected to a CAN-controller, whereas the CPU gets filtered information from the CAN-controller. The bus wires consists of two signals one of which is CAN high ”CANH” and CAN low ”CANL”. When a recessive (1) is transmitted neither of the wires is driven high or low and they are biased to around 2.5 V. On the other hand when a dominant (0) is transmitted CANH is driven 1 V higher to 3.5 V and CANL is driven 1 V lower to 1.5 V. This creates a 2 V differential signal between CANH and CANL as seen in Fig. 1. and states that a (0) will win over a (1).

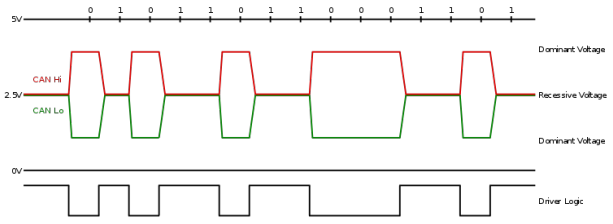

Fig. 1. ISO 11898-2003 [5]

We could choose between two kinds of identifiers: The Standard CAN with 11-bit identifier (2^{11} = 2048 identifiers) or the Extended CAN with 29-bit identifier (2^{29} = 537 million identifiers). They work the same way but the extended version allows the user to implement more complexity, as explained later.

The Can standard is divided in several groups which has different features that adds up to the total message. Each message consist of a identifier (11 or 29 bits), a data block (0-8 bytes), CRC check and some message handling bits this can be seen in Fig. 2.

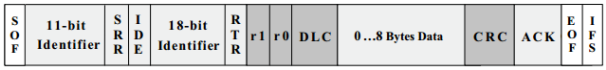

Fig. 2. 29-bit identifier [4] page 4

The identifier contains the specific type of message that is being sent, and it also contains the priority of the message, the lower the id number is, the higher is the priority. This means if two nodes transmits a message at the same time they will collide, but immediately the node with the highest priority will win and gets access to the bus. [4]

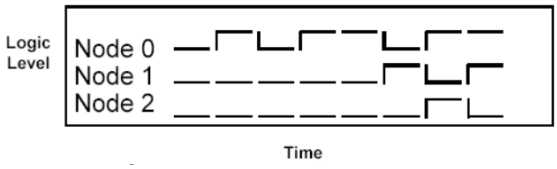

Fig. 3. Bitwise Arbitration [6] slide 26

Fig. 3. shows how three nodes transmits their message. The bus is won by node 2 because of the highest priority. Hereafter node 1 and 0 will receive the message send by node 2. When node 2 has ended its transmission the bus is available again and node 1 and 0 will re-transmit their messages.

For simplicity and not overloading the CPU each CAN-controller has a number of inbuilt masks and filters. If the identifier matches the value stored in one of the filter registers an interrupt is generated by the controller. Hereafter the mask selects which bits is compared with the filter; a ’0’ correspond to don’t care, and a ’1’ matches the bit. Thus, a group of messages can be filtered out relatively easy.

The CAN-controller thereby significantly decreases the work the CPU has to do, if no CAN-controller was implemented. It works independently of the CPU, furthermore it reduces the amount of interrupts generated and as a result lowering the load on the CPU dramatically. The CAN-controller chooses what the CPU gets, so it doesn’t need to react on every message on the bus [8].

The differential bus voltage layout increases the tolerance to electromagnetic noise. In conjunction with the advanced CRC check, the CAN bus provides an extremely robust communication platform. To test the stability, a 24 hour test between two boards was created. To trigger the maximum amount of errors, the bus utilisation was maxed out by constantly filling the internal buffers. The test was carried out without any issues.

In theory a bad node can stop all bus traffic, but it is very common that the CAN controller shuts down at a certain error threshold. The number of receive- and

transmit errors can easily be counted, and based on those numbers the controller can disconnect to maintain bus integrity.

IV. Board design

In regards to the actual implementation of a working CAN bus we only need two things: A line driver and a CAN controller. For a simplified design a microcontroller with an integrated CAN controller would be desirable. Thus the Teensy 3.6 was chosen which contains the 180 MHz ARM Cortex-M4F microcontroller [9] with a built-in CAN controller [1], as well as the high-speed CAN transceiver MCP2551 that implements the physical layer of ISO-11898:2003 [2]. From the datasheet for the line driver we were able to synthesis the schematic as seen on figure 4.

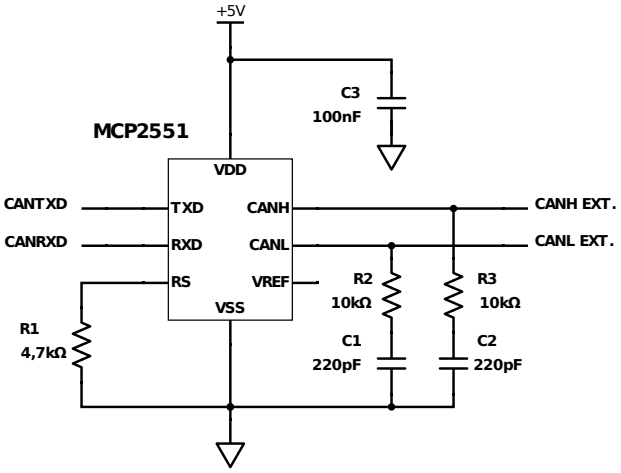

Fig. 4. Circuit diagram for the MCP2551

A couple of capacitors with current-limiting resistors have been added to each line as a simple precaution for signal integrity. Moreover a rather small resistor has been placed at the RS-input. This slope control resistor sets the threshold for the slew rate of the CAN signals. Thus choosing a $4.7\,\mathrm{k}\Omega$ resistor sets the threshold at about $30 - 35\,\mathrm{V}/\mu\mathrm{s}$ which in turn reduces electromagnetic interference by limiting the rise and fall time of the CAN lines [2]. As for power integrity a small capacitor has been added to the $5\,\mathrm{V}$ supply line close to the MCP2551.

Since the MCP2551 is a $5\,\mathrm{V}$ transceiver and the Teensy 3.6 operates at $3.3\,\mathrm{V}$ we are forced to make a bilateral conversion. For this purpose two level converters has been implemented on each line of the bus. In theory the CANH line is the only one that could cause a problem since it goes as high as $4.5\,\mathrm{V}$, but if we convert one the other has to be converted as well. The bi-directional level shifters are implemented as indicated on the schematic below:

As indicated on the schematic the level converter only consists of an enhancement type N-MOSFET and two pull-up resistors. By using this design we are able to

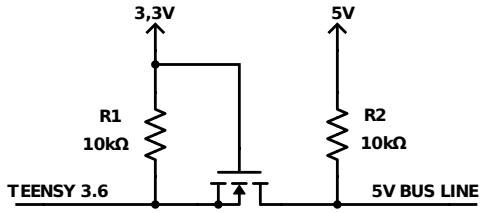

Fig. 5. Circuit diagram for the level converter

have our two devices operating at different voltages work seamlessly together. This is evident after considering the three cases in regards to pull-down of each device:

1) Neither the MCU or line driver is pulling its end down: This results in $V_\mathrm{G} = V_\mathrm{S} \Rightarrow V_\mathrm{GS} < V_\mathrm{TH}$ therefore each side of the MOSFET is pulled up to their respective voltages.
2) The MCU is pulling its end down: This results in $V_\mathrm{G} > V_\mathrm{S} \Rightarrow V_\mathrm{GS} > V_\mathrm{TH}$ and as a result the N-MOSFET is conducting and the line driver end is pulled down.
3) The line driver is pulling its end down: This results in the drain-subtrate diode pulling the MCU side down and consequently causing $V_\mathrm{G} > V_\mathrm{S} \Rightarrow V_\mathrm{GS} > V_\mathrm{TH}$ and as a result the N-MOSFET is conducting and the MCU side is pulled down further [3].

Additionally an unfortunate consequence of switching microcontroller manufacturer was that they did not share register standards or code platform, hence all code needed to be rewritten, turning out to be a comprehensive part of the project.

V. Protocol

The previous protocol was designed to overcome the limitations of a single-master bus and would not be suitable to implement on a CAN bus, since CAN bus has fewer hardware limitations. The new protocol was designed to take advantage of the CAN multi-master capabilities and hardware level message filtering. Early in the process we settled on the extended 29-bit identifier, as opposed to the smaller 11-bit identifier. By choosing the extended ID, the communication speed of the whole system is reduced slightly, compared to the speed if the normal ID was chosen, due to the fact that more bits has to be send for each message. For the worst-case scenario, that is sending an empty message, the message length is increased by 45%, for a more realistic scenario (4 data bytes) this is reduced to 26%, and utilising every data field the total message length is only increased by 19 % (128 bits and 108 bits). The slightly slower message transmission opens up for a far more advanced message-filtering, which can be used to decrease the CPU overhead. The protocol was designed from a number of requirements:

1) All PCBs should be more or less independent of the rest of the system.
2) A maximum delivery time can be defined and guarantied. Several messages must be passed within a deadline for safety reasons.
3) CPU load should not change as more nodes are added.
4) More nodes can be added without changing code on existing nodes.

The ID is made global, so it can be used for all CAN controllers. The 29-bits in the ID are divided into 4 sections, as shown in figure 6. The 5 most significant bits are used for the dynamic priority. The priority of each message can be set in real-time, and this structure guaranties faster delivery of important messages. A lower number in the priority field, increases the priority. The dynamic priority must be used carefully, and the higher priorities should be reserved for critical messages. As an example, our CAN bus library has a built in *emergency* function, that triggers the highest priority. Assuming a correct use of high priorities, the maximum delivery time can be calculated (@1Mbs). The bit-period is 1μs, assume a transmission has just started, having a total length of 108 bits, the high priority message will begin transmission approximately 108μs after, and will be delivered (worst case) 216μs after the initial transmission request.

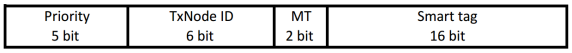

Fig. 6. Extended ID

The next 6 bits are for the specific ID of each CAN-controller. This is done to guarantee a unique identifier for all different message. If two nodes were to transmit to identical ID's, the messages would collide and trigger a transmit error. The unique *tx*-ID solves this problem, and can furthermore be used to sort messages. The *Message Type* (*MT*) is 2 bits and the last 16 bits are used for the *Smart Tag*. The *Message Type* is used to divide each ID into 4 general types. At the moment only two of these are used: *Status* and *Measurement*.

For a *Status* message, the *Smart Tag* is divided into two 8-bit blocks, which are used to send the number of receive errors and transmit errors. This information can be used to detect a faulty node, and react accordingly, thus increasing the stability and integrity of the system. It is then optional to send more data on the status of the CAN controller. See figure 7.

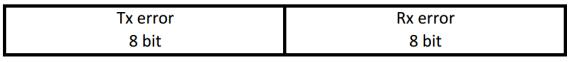

Fig. 7. *Smart Tag* for *MT: Status*

The *Smart Tag* in the *Measurement* is divided into two blocks of 5 bits and 11 bits. The first block is for the measurement type and the second is a specific sensor ID for the sensor which data that are about to be transmitted. See figure 8. Since our hardware has a dedicated single precision floating point unit, it is chosen that all measurement data following a *Measurement* is passed around as floats. The obvious advantage is that all received measurements can be handled uniformly, and the format for each sensor does not have to be known. This may seem overly complicated when transmitting integers, but is easily handled in software.

Fig. 8. *Smart Tag* for *MT: Measurement*

By choosing this division of the ID, a publisher subscriber pattern is obtained. Each node can easily be set up to only listen to the exact information that is necessary for correct operation, thus ignoring all other messages to reduce CPU load. In this way, each node is its own master, and operates independently based on the information published on the bus.

The standard specified by Bosch is handled in hardware, including the CRC check. Each correctly received message triggers an interrupt, and the content and ID is copied to RAM. The message is then easily sorted in accordance with the previously defined protocol. The CPU load is extremely low, and the sorting is therefore done in interrupt. To simplify the project, all measurements are copied to an array, sorted by the unique sensor ID. Sending is also handled in hardware, only the ID must be set, and message copied to a hardware buffer.

To increase security in the system, a time stamp for each transmitting node is saved. This is done to ensure that outdated information doesn't corrupt the system performance. This function is based on the *tx*-ID included in the identifier. Using the standard 11 bit identifier, functions as this one could not be implemented as naturally, since the receiving node not necessarily know the origin of the message.

VI. Application

To illustrate the effectiveness of the publisher subscriber pattern we can mention the ease at which the brake light functionality was added to the existing electronic system. This was done by setting up a pressure sensor close to the motor board, due to convenience. The motor board then acts as the publisher and sends out the measured voltage. The back light board then simply subscribed to this value to turn on the brake light at a given threshold. As a whole this only took a couple of lines of code to implement.

The number of steps to append the existing system with a new sensor or device has thus been reduced in contrast to the system that the CAN bus superseded (RS-485). This is apparent if we consider the case where only

one board needs the new information from the new sensor/device. We are able to do so by only reprogramming the board that needs to receive the new message with the publisher subscriber pattern, whereas with the old communication one would have to ensure syncing with the main node and more.

VII. Future development

The ecocar is a dynamic process where a lot can change for each year. The changes can be for many different reasons like optimisation, new rules or new categories to contest in. Therefore it is important to make new implementations easily adaptable to future developments. This is one of the main reasons for why the extended ID for the CAN bus protocol was chosen. It makes it a lot easier to add extra nodes, sensors etc. without having to change a lot of code for each node. An increased number of nodes or sensors will not result in increased CPU overhead due to the filters and the smart division of the ID.

A clear example of this protocol being easily adaptable was already made this year, where another group worked on making the car autonomous for a competition in 2020. Their system was seamlessly integrated in the existing system, and could be unplugged just before the competition in London without changing anything anywhere else on the car. Their system is an excellent example of an independent system, reacting to information published on the bus.

Furthermore it is also considered to add more PCBs to the car for next year, which are going to take over the control of the motor injection and much more, that is controlled by a RIO in the current version of the car. However this is not certain yet, although the CAN bus protocol is ready for expansions to the system.

VIII. conclusion

As a part of the ongoing DTU Road Runners Electronics project, an upgrade of the communication system was established. An implementation of the communication standard CAN bus (ISO-11898:2003) was installed, by redesigning the PCBs of the previous generations, and designing the protocol. The primary aspect of redesigning the PCBs was to replace the AT mega 2560 microcontroller with Teensy 3.6, which has a CAN controller included, and thereby simplifying the project.

The Teensy 3.6 runs at 3.3 V which proved to be a challenging part of the design process, since our CAN bus line driver (MCP2551) is a 5 V transceiver. This was however solved by using level conversion implemented with a N-mosfet, which made the two components operating at different voltages work seamlessly together. The shift in microcontroller manufacturer meant that all previous code had to be rewritten, turning out to be a comprehensive part of the project.

For the CAN bus protocol, the extended 29 bit ID was chosen to lower the CPU load by using the built in filters. The protocol makes it easy to expand the system, by adding more nodes and sensors. With the designed protocol, the communication becomes a publisher subscriber model, which makes all the boards relatively independent of the rest of the system.

The two main purposes of the project was to secure the stability and reduced complexity of the communication protocol. The stability of the system was tested with a 24 hour test of heavy bus traffic, where the bus worked with no problems. Another electronics group worked on making the car autonomous, and they could add their nodes to the network by plugging in a single cable, transmitting and receiving data from the bus without any problems - proving the simplicity of the implemented system. In conclusion we have implemented a robust communication system that is easy to expand by adding new nodes and sensors, making the transition to new generations smoother.

References

[1] "K66 Sub-Family Reference Manual," *nxp.com*, Apr-2017. [Online]. Available: http://www.nxp.com/assets/documents/data/en/reference-manuals/K66P144M180SF5RMV2.pdf. [Accessed: 17-Jun-2017].

[2] "MCP2551," *microchip.com*, 2010. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf. [Accessed: 17-Jun-2017].

[3] "Level shifting techniques in I2C-bus design," *nxp.com*, 18-Jun-2007. [Online]. Available: http://www.nxp.com/documents/application_note/AN10441.pdf. [Accessed: 17-Jun-2017].

[4] S. Corrigan, *Introduction to the Controller Area Network (CAN)*, Texas Instruments. Available: http://www.ti.com/lit/an/sloa101b/sloa101b.pdf. [Accessed: 19-Jun-2017].

[5] Illustration from Wikipedia. Available: https://en.wikipedia.org/wiki/File:ISO11898-2.svg. [Accessed: 19-Jun-2017].

[6] Illustration from pdf slide 26 Availble: http://kurser.iha.dk/eit/tidrts/powerpoint/canbus-introduction.pdf. [Accessed: 19-Jun-2017].

[7] Kvaser. (2017). CAN Bus Error Handling. [online] Available: https://www.kvaser.com/about-can/the-can-protocol/can-error-handling/ [Accessed 19 Jun. 2017].

[8] J. Kaiser and M. Mock
Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN)

[9] "Teensy USB Development Board," *PJRC*. [Online]. Available: https://www.pjrc.com/store/teensy36.html. [Accessed: 19-Jun-2017].