A FIXED LENGTH PAYLOAD ENCODING FOR CAN

K.R.Priyanga¹, K.Venkatesan M.E.²
M.E., Communication Systems¹, Department of ECE²,
Sri Shakthi Institute of Engineering and Technology
Coimbatore.

Abstract— The controller area network (CAN) bit stuffing mechanism, it is essential to ensure proper receiver clock synchronization in the network. So it introduces a significant, payload-dependent jitter on message response times, which may worsen the timing accuracy of a networked control system. This paper presents a software payload encoding scheme, which is able to guarantee that no stuff bits will ever be added to the data field by the CAN controller during transmission. As a result jitter is reduced considerably. Practically it is implemented to show how the simplicity and high performance of the encoding scheme make it suitable for low cost and embedded architectures.

Keywords— Controller Area Network (CAN), Clock synchronization, real-time distributed systems.

I. INTRODUCTION

CAN is an important embedded protocol. Since it was introduced about 20 years ago, controller area networks (CANs) [1] have steadily gained popularity and are nowadays adopted in a variety of embedded, networked control systems. One peculiar facet of the CAN protocol is its bit-stuffing (BS) mechanism, which is an efficient way to ensure that a sufficient amount of edges appear in the signal sent over the bus, and thus, guarantee a proper receiver clock synchronization. However, due to BS, the actual length of a message sent over the bus depends not only on the size of its payload but also on its content. As a consequence, the exact duration of frame transmissions cannot be known in advance, leading to a certain degree of jitter on response times. Also for this reason, recent automotive protocols such as FlexRay [2] rely on fixed-length encoding instead. Communication jitter may be, at least in some demanding cases, a limiting factor in the design of a networked control system, both in general [3]–[7], and for CAN in particular [8]–[13]. Even in the case of simple systems that rely on the master–slave approach, the non constant duration of CAN messages leads to annoying jitters on actuation.

The new-generation isochronous protocols e.g., FlexRay and some real-time Ethernet solutions are the best option for a new design of a high-performance dependable control system [15], CAN is still a viable option in many other cases [16]. In fact, CAN technology is undoubtedly really stable, well known, and widespread. For these reasons, most microcontroller families from all major vendors include at least one member that embeds, at no extra cost, a CAN controller. As a consequence, when design constraints are not so tight and the focus is instead on reducing system cost, complexity, and time-to-market, many designers still prefer to stick to CAN. Recently, some researchers started considering the use of conventional CAN also for applications with tight timing constraints, as witnessed by some proposals made for software-based synchronization techniques in CAN.

II. BIT STUFFING MECHANISM IN CAN

The physical layer of CAN relies on the non-return to zero encoding scheme with bit stuffing. Every time the CAN controller in the transmitting node detects that five consecutive bits with the same value have been sent, it inserts a stuff bit at the opposite level.

This paper is structured as follows. Section II explains the problems of CAN by bit stuffing and the encoding scheme is explained. Section III explains the implementation and the development of portable fixed length encoding scheme. Section IV shows the experimental results on two dissimilar microcontrollers. Section V explains the conclusion.
For example, if the original sequence is 01111101 000000001…, the sequence of bits sent on the bus will be 01111101 000001001…, (from now on, underlining is used to denote stuff bits). Bit stuffing applies only to those fields in a CAN frame from the start of frame bit (SOF) up to the cyclic redundancy check (CRC) included. The remaining fields, from the CRC delimiter up to the end of the frame (EOF), are of fixed form and are not affected.

Where EXOR was applied selectively to either the payload as a whole (SXB) or on single bytes separately (SXB). The encoding efficiency decreases because of the need to include information about the EXOR process while encoding time increases. Moreover, some stuff bits may still be added to the data field nevertheless. the Software Bit Stuffing approach performs a software bit stuffing in advance on the original payload, so that one software stuff bit is added every time 4 consecutive bits are found at the same value. Processing time increases consequently with respect to SXP and SXB. In a similar way, EEM is a fixed-length 8-to-11 modulation scheme [20], which encodes every byte in the original payload as an 11-bit pattern and proved to be faster than SBS.

B. 8B9B Encoding

The 8B9B encoding scheme is straightforward. Basically, every single byte of the original payload is translated separately to a suitable pattern made up of 9 b. The encoded bit stream is then obtained by concatenating all these patterns in the original order. Clearly, this does not apply to frames with an empty payload (i.e., when DLC is 0). Encoded 9-b patterns must satisfy two properties, given here.

1) None of the patterns is allowed to include primer sequences. For instance, the pattern 010000011 is unsuitable for our purposes.
2) Primer sequences must never appear in the encoded bit stream, not even across pattern boundaries. In order to meet this additional constraint, all patterns that include 3b at the same level, at either the beginning or the end, have to be discarded as well. For instance, the pattern 010101000 is not suitable because, when followed by the pattern 001010101, it would give rise to the sequence 010101000 001010101, which includes a primer sequence.

A forward lookup table (FLT) for the direct replacement of bytes in the original payload with corresponding 9-b patterns. Let be the original data byte, Y the corresponding 8B9B-encoded value, and F(.) the encoding function carried out through the FLT. The forward translation process can be synthetically expressed as Y = F(X).

C. Break Bit and Padding Field

The reasoning above takes into account the data field alone. Unfortunately, the preceding parts of the frame may contribute to the creation of a primer sequence together with the beginning of the data field. The DLC field is not in complete control of the user, since it specifies the size in bytes of the payload represented on 4 b as a binary value. For this reason, unlike the payload, it cannot be encoded. For example, when the DLC value is 8(1000) and the first encoded pattern is 0010101, the resulting (partial) bit sequence is …1000 0010101, which includes a primer sequence.

A simple remedy is inserting a single bit, called break bit (BB), in the very first position of the data field. The value of BB is opposite to the last DLC bit, i.e., it is either when the DLC value is even or on the contrary. Thanks to BB, bit strings with two or more bits at the same value cannot appear immediately before the first 9-b pattern in the encoded payload, for any value of DLC between 1 and 8, included. Only when DLC = 15, the BB could be preceded by one stuff bit at its same level, appended to the DLC by the CAN controller. In any case, the occurrence of a primer sequence affecting the data field is completely precluded.
TABLE I
8B9B FORWARD LOOKUP TABLE (ENCODING PROCESS)

<table>
<thead>
<tr>
<th>X16</th>
<th>Y2</th>
<th>X16</th>
<th>Y2</th>
<th>X16</th>
<th>Y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
</tr>
<tr>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
</tr>
<tr>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
</tr>
<tr>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
</tr>
<tr>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
</tr>
<tr>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0A</td>
</tr>
<tr>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0A</td>
<td>0B</td>
</tr>
<tr>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0A</td>
<td>0B</td>
<td>0C</td>
</tr>
<tr>
<td>08</td>
<td>09</td>
<td>0A</td>
<td>0B</td>
<td>0C</td>
<td>0D</td>
</tr>
<tr>
<td>09</td>
<td>0A</td>
<td>0B</td>
<td>0C</td>
<td>0D</td>
<td>0E</td>
</tr>
</tbody>
</table>

D. Other Fields of the Frame

Only the data field is covered by 8B9B. There are, however, other parts of the CAN frame that are subject to BS too, namely the frame header and CRC. The header in CAN frames is basically made up of the message identifier, DLC and few additional bits whose value is fixed (e.g., SOF and res). Stuff bits are typically not a problem here.

TABLE II
8B9B-ENCODED DATA FIELD VERSUS ORIGINAL PAYLOAD

<table>
<thead>
<tr>
<th>Original payload size (byte)</th>
<th>Data field in the 8B9B-encoded frame</th>
<th>DLC val. size (byte)</th>
<th>BB size (byte)</th>
<th>9-bit pattern size (byte)</th>
<th>PAD size (byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>010</td>
<td>1</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>001</td>
<td>0</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>010</td>
<td>1</td>
<td>27</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>010</td>
<td>1</td>
<td>36</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>010</td>
<td>1</td>
<td>45</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>011</td>
<td>0</td>
<td>54</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>100</td>
<td>1</td>
<td>63</td>
<td>0</td>
</tr>
</tbody>
</table>

since in real-world applications both the message identifier and the payload size for any given message stream are usually not allowed to change over time. As a consequence, the number of stuff bits that are added by the CAN controller to the frame header is fixed. The CRC field follows immediately the data field and is the last part of the frame to which BS applies. Unlike the preceding fields, for which counter measures exist to prevent jitters, there is no simple remedy to avoid the insertion of stuff bits in the CRC, because it is computed in hardware by the CAN controller at runtime. No more than four stuff bits can be added in the CRC when using 8B9B.

III. IMPLEMENTATION

The encoding and decoding algorithms have been implemented in the C language and then cross-compiled using two embedded hardwares. In that hardware CAN is an inbuilt device and is known as a peripheral.

The MPC5604B family is one of a series of next generation automotive microcontrollers based on the Power Architecture Book E architecture and designed specifically for embedded applications. The MPC5604B/C microcontrollers operate at speeds up to 64 MHz and offer high performance processing with low power consumption. They are compatible with the existing development infrastructure of current Power Architecture devices and are supported with software drivers, operating systems and configuration code to assist with application development.

The software used is CODEWARRIOR IDE 5.9 and it is compatible with C language. There is a version of Freescale’s CodeWarrior development environment specifically made for MPC5604B. It has a practical debug environment and some basic library for setting the device, setting software INTC, mapping registers to the memory etc. The CodeWarrior Integrated Development Environment (IDE) provides an efficient and flexible software-development tool suite. The debugger used to debug the code in codewarrior is In-Circuit Debugger for Power PC (ICDPPC) and it is connected to the JTAG port which is a Nexus application development.

The goal of the optimization was to improve code performance as much as possible to minimize the additional end-to-end response time introduced by the encoding and decoding process. Three kinds of optimization have been identified and implemented.

1) Code/Data Placement: Microcontrollers usually support different kinds of memory such as Flash memory, dynamic RAM (DRAM), and static RAM (SRAM). Each kind of memory has its own peculiar behavior in terms of access speed and determinism. The goal of this optimization step was to force the tool chain to place the code, data, and stack segments of the encoding and decoding routines in the most appropriate place. Even if this is a relatively straightforward decision, most tool chains are unable to take it autonomously.

2) Computed Masks: In the base implementation shown in [20], several auxiliary arrays hold the bit masks used on every iteration of the encoding and decoding loop, to split the 9-bit value obtained from the forward lookup table into two output bytes during encoding, and join them again during decoding. In this optimization, the masks have been computed directly, as a function of the loop index. Since this computation can be carried out in the processor registers, the extra memory accesses to the arrays are avoided. The arrays themselves can be deleted to save memory, too.
3) Load/Store Reduction: The base version of the encoding algorithm, on each iteration, loads one input byte and stores some data into two adjacent output bytes. One of the output bytes overlaps with those of the previous iteration, and hence, each output byte except the first one is accessed twice. With this optimization, a local buffer has been introduced to carry forward the common output byte from one iteration to the next, in order to store it only once. The decoding algorithm has been optimized in the same way, too.

IV. EXPERIMENTAL ANALYSIS

The C program based on CAN 8B9B encoding scheme was programmed in the codewarrior. It is interfaced with the MPC5604B through CAN. The Joint Test Action Group (JTAG) is responsible for the debugging operation. When the code is completed, it is debugged and then connected with the MPC5604B.

After execution of the program the ICDPPC is opened and then the required registers and variables is added in the variable window and then again it is debugged. After debugging, in the ICDPPC window click EXECUTE → GO and then the program is executed. In the memory window check the required memory address and then the transmitted data is viewed. Figure 3 shows the hardware interfacing wuth the PC using the USB cable. Another hardware is connected with the previous one using the DB9 male connector through Controller Area Network. The received data is viewed in the PC. Figure 4 shows the output of the received data along with the base address on which it is received.

V. CONCLUSION

The datas have been transmitted between the MPC5604B and has been viewed in debugger window. The performance optimization of code/data placement, computed mask and load store reduction is carried out. This paper presented 8B9B, an encoding scheme designed to prevent bit stuffing within the data field of CAN messages, a property especially useful in tightly synchronized systems. Although several other methods with the same purpose have been proposed in the recent past, we believe that 8B9B can outperform them when the typical requirements of small embedded systems are taken into account. Namely, it provides a balanced blend of determinism, encoding efficiency, codec speed, and footprint.

The encoding scheme is completely transparent to both the sending and receiving applications, as well as other mechanisms aimed at reducing or removing frame-level jitter. Since no assumptions are made on how the payload is used and formatted by the upper protocol layers, the proposed approach is completely general-purpose. Moreover, it is possible to apply 8B9B only to jitter-sensitive messages, because encoded messages can coexist with plain ones. In this way, full backward compatibility is achieved. As is typical in CAN, the distinction between the two kinds of message can easily be done through the message identifier. The only limitation is that the payload to be encoded cannot be larger than 7 bytes.

ACKNOWLEDGMENT

First and foremost, we wish to express our deep gratitude and indebtedness to our institution and our department for providing us a chance to fulfill our long cherished of becoming Electronics and Communication engineers. We wish to acknowledge with thanks the excellent encouragement given by the management of our college. We wish to express our hearty thanks to the Principal of our college and HOD. We are committed to place our heartfelt
thanks to all teaching and supporting staff members, lab technicians and friends, and all the noble hearts that gave us immense encouragement towards the completion of our project. Finally we thank almighty for bestowing the gifts of life on us and also for providing us the necessary help through his lovely creations in this endeavor of us.

REFERENCES


BIOGRAPHY

K.R.Priyanga did her Bachelor of Engineering in Electronics and Communication Engineering at Maharaja Prithvi Engineering College, Avinashi and doing Master of Engineering in Communication Systems at Sri Shakti Institute of Engineering and Technology, Coimbatore, India. Her research interests include embedded systems, Communication System Design. She has presented two papers in International Conference, three paper in National Conference and presented a project and paper presentation in Kumaraguru college of technology, Coimbatore.