Examining Mono-Polarity Bit Sequences in the Controller Area Network Specification

Steve C. Talbot
Illinois Institute of Technology
Chicago, Illinois 60616, USA
talbste@iit.edu

Abstract

The Controller Area Network (CAN) architecture was developed for use in automobiles in the 1980's. A key "Error Detection" tactic employed in the CAN specification is the so-called "bit stuffing" technique. This paper considers the effects of the "mono-polarity bit sequence" rule found in the CAN specification. This rule states that mono-polarity bit sequences of 5 bits or more are considered errors, unless this bit sequence is followed by a stuff bit of opposite polarity. Without bit stuffing, a certain number of messages that contain these "illegal" mono-polarity bit sequences would be excluded from the range of the total number of messages possible for transmission on the CAN bus. This paper investigates how many of these sequences would be lost for a "hypothetical specification", one which would not implement bit stuffing. The intention of this paper is to illustrate why "bit stuffing" is justified.

1 Bit Stuffing

Detection of an error message ("Error Detection") on the bus occurs because CAN specifies that no message on the bus may consist of 5 or more bits which have the same polarity (i.e.: ..00000.., and ..11111.. are considered errors). It happens that the first 6 bits of an error message are all dominant bits (000000). This serves as an intentional flag to the rest of the network that an error has occurred, with each node employing bit monitoring to count sequences of mono-polarity bits. Whenever a node identifies a 5-bit sequence of mono-polarity bits (or 6-bit sequence from an error message), it halts whatever it is doing and issues an error message. [1,2]

"Bit stuffing" is the technique by which CAN allows mono-polarity bit sequences to be included in messages. The nature of CAN messages ("frames") is such that the "start of frame" (SOF) field, the arbitration" field (with 11-bit / 29-bit message id), the "control" field (6 bits), the "data" (8 to 64 bit) and the CRC (15 bits) fields (see Figure 1) are capable of having or contributing to mono-polarity bit sequences greater than the 5-bit limit imposed by CAN (each of these message fields, except SOF, consist of greater than 5 bits).

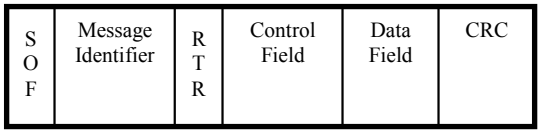

Figure 1: CAN "data" frame (CAN 2.0A) fields which employ "bit stuffing" as a means of detecting false-positive bit-errors

If a method for allowing mono-polarity bit sequences were not employed, CAN messages would be severely constrained to an unintuitive sub-set of message id, data and CRC values. "Bit stuffing" is the process of inserting a reverse polarity bit immediately following the sequence of mono-polarity bits. This "stuff bit" serves as a flag to receiving nodes that the prior mono-polarity bit sequence should not be an indication of an error state. Receiving nodes remove the identified "stuff bit" from the incoming message, and the remainder of the message is parsed. The receiving node restarts its count of subsequent mono-polarity bit sequences following the presence of the "stuff bit", so that it can continue checking for bit errors. [1,2]

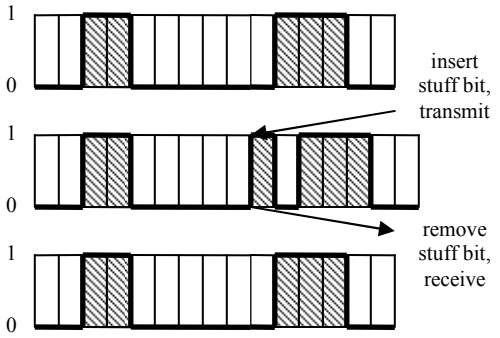

Figure 2: Transmitting inserts a stuff bit, receiving node removes the stuff bit

2 Investigation

We start be examining some basic properties of the CAN frame fields. The total number of illegal messages due to mono-polarity bit sequences is "I". The total number of non-illegal messages is "NI". The total number of possible messages (both I and NI) is "T". Therefore,

$$NI = (T - I).$$

However, mono-polarity bit sequences occur within specific fields. Each field has a different number of bits, making the number of combinations possible for each field unique. Also, a single field may have an illegal combination with all other fields having non-illegal combinations, or multiple fields may have illegal combinations at the same time. Therefore, this formula can be represented more precisely as follows.

$$\prod_{f=1}^{F} ni_f = T - \prod_{f=1}^{F} i_f$$

Here, "F" is the total number of fields within the message subject to bit stuffing. "f" corresponds to the specific field. "ni_f" corresponds to the number of non-illegal messages for the specific field. "i_f" corresponds to the number of illegal messages for the specific field. This formula assumes that the total number of combinations of the composite message depends on the multiplication of the combinations of the component fields. Based on examination of this formula which assumes bit stuffing, it becomes clear that a method of determining the number of illegal field combinations is important in order to determine the total number of messages lost due to the hypothetical situation when bit stuffing is not implemented in a CAN frame.

Begin by examining the 11-bit message id, starting with a 5-bit mono-polarity bit-sequence (MPBS) of all recessive bits. (i.e.: "11111nnnnnn", where "n" stands for a digit that can be either "0" or "1"). The number of combinations for a sub-sequence of 6 binary digits (aka, the 6 LSB digits on the right, "nnnnnn") would be the $2^6 = 64$. Continuing, we could shift the block of "1"s over to the right by 1 bit-space (i.e.: "n11111nnnnn"), and calculate the number of possible combinations to arrive at $2^6 = 64$. Continuing this process, we arrive at the following.

11111nnnnnn	$2^6 = 64$
n11111nnnnn	$2^6 = 64$
nn11111nnnn	$2^6 = 64$
nnn11111nnn	$2^6 = 64$
nnnn11111nn	$2^6 = 64$
nnnnn11111n	$2^6 = 64$
nnnnnn11111	$2^6 = 64$

Adding these results, we arrive at the value 64 + 64 + 64 + 64 + 64 + 64 + 64 = 448. We would arrive at the same value for a mono-polarity bit sequence of "0"s, raising the total to 2 * 448 = 896. In addition, the top 2 and bottom 2 sequences shown above could have an additional MPBS in each (for a total of 2 MPBSs in each field) that should also be counted.

11111n11111	11111011111, **11111111111**
1111111111n	11111111110, **11111111111**
n1111111111	01111111111, **11111111111**
00000n00000	00000100000, **00000000000**
0000000000n	00000000001, **00000000000**
n0000000000	10000000000, **00000000000**

The results above indicate that there are 12 extra possible combinations. However, as can be seen above, some of these combinations are duplicates (shown in bold). This is also true in the general case above. This approach identifies an upper limit on the number of combinations and identifies the issue that duplicates must be accounted for. Therefore, in order to get a count of all combinations without duplicates, we need to abandon the approach above in order to employ a stricter approach below.

In order to identify a general pattern by which we may remove duplicates from our count of possible combinations, we start by enumerating the combinations, counting the combinations and removing the duplicates from the count. These enumerations follow, with duplicate combinations shown in bold and counted only 1 time. The count that is shown indicates the number of unique combinations for the size of the field and the number of "constrained" bits (a bit that is held static as either "0" or "1", but is not free to be either "0" or "1"; by contrast we said "n" is unconstrained).

2-bit field, 1 bit constrained (count = 4)
XX
1n: 10, **11**
0n: **00**, **01**
n1: 01, **11**
n0: **00**, **10**

2-bit field, 2 bits constrained (count = 2)
XX
11
00

3-bit field, 1 bit constrained (count = 8)
XXX
1nn: 100, **101**, **110**, **111**
0nn: **000**, **001**, **010**, **011**
n1n: 010, **011**, **110**, **111**
n0n: **000**, **001**, **100**, **101**
nn1: 001, **011**, **101**, **111**
nn0: **000**, **010**, **100**, **110**

3-bit field, 2 bits constrained (count = 6)
XXX
11n: 110, **111**
00n: **000**, 001
n11: 011, **111**
n00: **000**, 100

3-bit field, 3 bits constrained (count = 2)
XXX
111
000

At this stage we can observe the first pattern, which is that for the 2-bit field, 1 bit constrained case, count = 4 is identical to $2^t = 4$, where t = 2 is the size of the field. For the 3-bit field, 1 bit constrained case, count = 8 corresponds to $2^t = 8$, where t = 3 is the size of the field. Therefore, for the 1 bit constrained case, for any field length, in general it is assumed that count = 2^t.

4-bit field, 1 bit constrained (count = $2^4 = 16$)
(pattern developed - 1 bit constrained ~ 2^n)

4-bit field, 2 bits constrained (count = 14)
XXXX
11nn: **1100**, 1101, **1110**, **1111**
00nn: **0000**, **0001**, 0010, **0011**
n11n: 0110, **0111**, **1110**, **1111**
n00n: **0000**, **0001**, **1000**, 1001
nn11: **0011**, **0111**, 1011, **1111**
nn00: **0000**, 0100, **1000**, **1100**

4-bit field, 3 bits constrained (count = 6)
XXXX
111n: 1110, **1111**
000n: **0000**, 0001
n111: 0111, **1111**
n000: **0000**, 1000

4-bit field, 4 bits constrained (count = 2)
XXXX
1111
0000

5-bit field, 1 bit constrained (count = $2^5 = 32$)
(pattern developed - 1 bit constrained ~ 2^t)

5-bit field, 2 bits constrained (count = 30)
XXXXX
11nnn **11000**, **11001**, 11010, **11011**, **11100**, **11101**, **11110**, **11111**
n11nn **01100**, 01101, **01110**, **01111**, **11100**, **11101**, **11110**, **11111**
nn11n **00110**, **00111**, **01110**, **01111**, 10110, **10111**, **11110**, **11111**
nnn11 **00011**, **00111**, 01011, **01111**, **10011**, **10111**, **11011**, **11111**

00nnn **00000**, **00001**, **00010**, **00011**, **00100**, 00101, **00110**, **00111**
n00nn **00000**, **00001**, **00010**, **00011**, **10000**, **10001**, 10010, **10011**
nn00n **00000**, **00001**, **01000**, 01001, **10000**, **10001**, **11000**, **11001**
nnn00 **00000**, **00100**, **01000**, **01100**, **10000**, 10100, **11000**, **11100**

5-bit field, 3 bits constrained (count = $2^4 = 16$)
XXXXX
111nn 11100, 11101, **11110**, **11111**
000nn **00000**, **00001**, 00010, 00011
n111n 01110, **01111**, **11110**, **11111**
n000n **00000**, **00001**, **10000**, 10001
nn111 00111, **01111**, 10111, **11111**
nn000 **00000**, 01000, **10000**, 11000

5-bit field, 4 bits constrained (count = 6)
XXXXX
1111n: 11110, **11111**
0000n: **00000**, 00001
n1111: 01111, **11111**
n0000: **00000**, 10000

5-bit field, 5 bits constrained (count = 2)
XXXXX
11111
00000

At this point, we can see another pattern developing. Whenever all of the bits in the field are constrained, the count = 2. Whenever all but 1 of the bits in the field are constrained, the count = 6. Additional data is useful in order to continue with the identification of patterns. However, enumerating the combinations of fields for larger fields is becoming unnecessary. With only a little more information we can deduce the nature of larger fields without further enumeration.

6-bit field, 1 bit constrained (count = $2^6 = 64$)
(pattern developed - 1 bit constrained ~ 2^t)

… (skipping, 2/3/4 bit constrained, as we will be deducing the counts for these)

6-bit field, 4 bits constrained (count = $2^4 = 16$)
XXXXXX
1111nn: 111100, 111101, **111110**, **111111**
000nn: **000000**, **000001**, 000010, 000011
n1111n: 011110, **011111**, **111110**, **111111**
n0000n: **000000**, **000001**, **100000**, 100001
nn1111: 001111, **011111**, 101111, **111111**
nn0000: **000000**, 010000, **100000**, 110000

6-bit field, 5 bits constrained (count = 6)
XXXXXX
11111n: 111110, **111111**
00000n: **000000**, 000001
n11111: 011111, **111111**
n00000: **000000**, 100000

6-bit field, 6 bits constrained (count = 2)
XXXXXX
111111
000000

It is now time to specify how we may determine any combination count for any field length and any number of constrained bits. We deduce our formulation by re-stating the results from above in a more compact format. "x/y" shown below indicates "x" constrained bits within a "y"-bit length field. For instance, "3/5" indicates 3 constrained bits in a 5-bit field ("5-bit field, 3 bits constrained"). The value shown to the right of "x/y" below indicates the count of combinations (without duplicates) associated with this case. For instance "$1/2 \rightarrow 4 \rightarrow 2^2$" indicates that it considers a "2-bit field, 1 bit constrained", that the count of unique combinations is "4", and that the count can be represented as "2^2".

$1/1 \rightarrow 2 \rightarrow 2^1$

$1/2 \rightarrow 4 \rightarrow 2^2$
$2/2 \rightarrow 2 \rightarrow 2^1$

$1/3 \rightarrow 8 \rightarrow 2^3$
$2/3 \rightarrow 6 \rightarrow 2^3 - 2$ Rule C1
$3/3 \rightarrow 2 \rightarrow 2^2 - 2$ Rule C2

$1/4 \rightarrow 16 \rightarrow 2^4$ Rule A1
$2/4 \rightarrow 14 \rightarrow 2^4 - 2$ Rule A2
$3/4 \rightarrow 6 \rightarrow 2^3 - 2$ Rule **A3**
$4/4 \rightarrow 2 \rightarrow 2^2 - 2$ Rule **A4**

$1/5 \rightarrow 32 \rightarrow 2^5$ Rule A1
$2/5 \rightarrow 30 \rightarrow 2^5 - 2$ Rule A2
$3/5 \rightarrow 16 \rightarrow 2^4$ Rule B1
$4/5 \rightarrow 6 \rightarrow 2^3 - 2$ Rule **A3**
$5/5 \rightarrow 2 \rightarrow 2^2 - 2$ Rule **A4**

$1/6 \rightarrow 64 \rightarrow 2^6$ Rule A1
$2/6 \rightarrow 62 \rightarrow 2^6 - 2$ Rule A2
$3/6 \rightarrow 38 \rightarrow 2^5 + 6$
$4/6 \rightarrow 16 \rightarrow 2^4$ Rule B1
$5/6 \rightarrow 6 \rightarrow 2^3 - 2$ Rule **A3**
$6/6 \rightarrow 2 \rightarrow 2^2 - 2$ Rule **A4**

$1/7 \rightarrow 128 \rightarrow 2^7$ Rule A1
$2/7 \rightarrow 126 \rightarrow 2^7 - 2$ Rule A2
$3/7 \rightarrow 86 \rightarrow 2^6 + 2^4 + 6$
$4/7 \rightarrow \mathbf{40} \rightarrow 2^5 + 2^3$
$5/7 \rightarrow 16 \rightarrow 2^4$ Rule B1
$6/7 \rightarrow 6 \rightarrow 2^3 - 2$ Rule **A3**
$7/7 \rightarrow 2 \rightarrow 2^2 - 2$ Rule **A4**

$1/8 \rightarrow 256 \rightarrow 2^8$ Rule A1
$2/8 \rightarrow 254 \rightarrow 2^8 - 2$ Rule A2
$3/8 \rightarrow 188 \rightarrow 2^7 + 2^6 - 4$
$4/8 \rightarrow 94 \rightarrow 2^6 + 2^5 - 2$
$5/8 \rightarrow \mathbf{40} \rightarrow 2^5 + 2^3$
$6/8 \rightarrow 16 \rightarrow 2^4$ Rule B1
$7/8 \rightarrow 6 \rightarrow 2^3 - 2$ Rule **A3**
$8/8 \rightarrow 2 \rightarrow 2^2 - 2$ Rule **A4**

We can see from the results shown here that a new pattern emerges when you grow from considering the 1-bit field up to the 8-bit field. This pattern can best be seen in the case of the 8-bit field, which is re-stated below with each line labeled with a preceding letter.

a) $1/8 \rightarrow 256 \rightarrow 2^8$
b) $2/8 \rightarrow 254 \rightarrow 2^8 - 2$

c) $3/8 \rightarrow 188 \rightarrow 2^7 + 2^6 + 4$
d) $4/8 \rightarrow 94 \rightarrow 2^6 + 2^5 - 2$
e) $5/8 \rightarrow 40 \rightarrow 2^5 + 2^3$
f) $6/8 \rightarrow 16 \rightarrow 2^4$
g) $7/8 \rightarrow 6 \rightarrow 2^3 - 2$
h) $8/8 \rightarrow 2 \rightarrow 2^2 - 2$

A. These rules only apply to 4-bit fields and larger:

1. If "1" bit is constrained (line a), the unique count is 2^8.
    a. "t" is the field length, count = 2^t.
2. If "2" bits are constrained (line b), the unique count is $2^t - 2$.
3. If "(t-1)" bits are constrained (line g), the unique count is $2^3 - 2$.
4. If "t" bits are constrained (line h), the unique count is $2^2 - 2$.

B. This rule only applies to 3-bit fields and larger:

1. If "(t-2)" bits are constrained (line f), the unique count is 2^4.

C. This rule only applies to the 3-bit field:

1. If "(t-1) = 2" bits are constrained, the unique count is $(2^t - 2) = (2^3 - 2)$, the same as "A3".
2. If "t" bits are constrained, the unique count is $2^2 - 2$, the same as "A4".

The 1-bit and 2-bit cases are "special cases" which serve as the basis for the remainder of the cases, but do not necessarily abide by the rules which apply to the other cases.

The rules above would seem to suggest that there is an orderly transition from "all bits constrained" (8/8) to "1 bit constrained" (1/8), such that we might be able to formulate a straightforward analytical equation to predict the counts for each case. However, the counts "in the middle" (lines c, d and e) do not seem to have any pattern which can be easily captured to serve our purpose of creating a formula for prediction. The counts re-stated below indicate that as the field length increases by 1, the number of "middle" lines increases by 1. Also, it is seen that the count for "$4/7 \rightarrow \underline{\mathbf{40}} \rightarrow 2^5 + 2^3$" is identical to the count for "$5/8 \rightarrow \underline{\mathbf{40}} \rightarrow 2^5 + 2^3$", suggesting that a pattern is continuing to emerge, but that it requires a more complicated formulation than that which is provided here.

$3/6 \rightarrow 38 \rightarrow 2^5 + 6$
$3/7 \rightarrow 86 \rightarrow 2^6 + 2^4 + 6$
$4/7 \rightarrow \underline{\mathbf{40}} \rightarrow 2^5 + 2^3$

$3/8 \rightarrow 188 \rightarrow 2^7 + 2^6 - 4$
$4/8 \rightarrow 94 \rightarrow 2^6 + 2^5 - 2$
$5/8 \rightarrow \underline{\mathbf{40}} \rightarrow 2^5 + 2^3$

In addition, it can be said that "$3/6 \rightarrow \underline{\mathbf{38}} \rightarrow 2^5 + 6$" prepares the way for "$4/7 \rightarrow \underline{\mathbf{40}} \rightarrow 2^5 + 2^3$", such that subsequent cases following the 4-bit case all include a "$\underline{\mathbf{40}} \rightarrow 2^5 + 2^3$" value. This appears to be true also for "$2/4 \rightarrow \underline{\mathbf{14}} \rightarrow 2^4 - 2$" preparing the way for "$3/5 \rightarrow \underline{\mathbf{16}} \rightarrow 2^4$", and "$1/2 \rightarrow \underline{\mathbf{4}} \rightarrow 2^2$" preparing the way for "$2/3 \rightarrow \underline{\mathbf{6}} \rightarrow 2^3 - 2$". This supports the view that "middle" lines are not simply "noise", but rather values that require a more complicated formulation than is provided here.

However, although a formula has not been provided here, a computer program has been written which calculates the unique count for each case. The user specifies the field length ("fieldLength") and the number of bits of the MPBS ("bitsConstrained"). The program creates all of the enumerations, and then either outputs the enumerations or outputs the counts to standard output. The only restrictions on "fieldLength" and "bitsConstrained" are that "bitsConstrained <= fieldLength" and "bitsConstrained > 0". Only 1 MPBS per field is considered, but "fieldLength" and "bitsConstrained" may be any arbitrary length (this computer program was used to produce the counts in the 6, 7 and 8 bit cases above).

Using the computer program, it was found that for an 11-bit field, 5 bits constrained ("5/11"), there are 502 non-duplicate combinations. These combinations represent message ids in a CAN 2.0A 11-bit data frame which would be "illegal" if "bit stuffing" were not employed. The total number of combinations for an 11-bit field is $2^{11} = 2048$. Therefore, it is seen that without bit stuffing, the CAN specification would lose nearly 1/4 of the possible message ids, severely constraining the scalability of a CAN network.

One further issue remains to be addressed before concluding discussion of this topic. If a MPBS overlaps both one field (say the message id) and an adjacent field (say the control field), then the analysis presented here still applies. The reason is that in order for an overlap to occur, a portion of the MPBS has to exceed a field boundary. In calculating the total possible number of combinations of bit sequences having the

constrained MPBS embedded in them, we "translate" the MPBS bitwise from one boundary to the other in order to constrain the possible combinations of the bits on either side of the MPBS. As soon as the MPBS moves away from the field boundary, the overlap includes 1 or more "n" bits instead of 1 or more MPBS bits, as seen below.

Step 1: "nnnnnnn1111|1"

Step 2: "nnnnnn11111|n"

…

The analysis up until now has implicitly assumed that the bits on either side of the field boundary were "n" bits, in that we did not constrain these bits nor did we consider them in our calculation. Therefore, in order to account for the overlap, the effective "field length" is increased to include the entire bit sequence, including the "overlap bits". For instance, the bit sequence message id "nnnnnnn1111|1" (where "|" is meant to indicate the field boundary) would necessitate that we increase the field length in our calculations from the standard 11-bit length to 12 bits, to include the overlapping bit on the right-most side.

Conclusions

Using the computer program, it was found that for an 11-bit field, 5 bits constrained ("5/11"), there are 502 non-duplicate combinations. It is seen that without bit stuffing, the CAN specification would lose nearly 1/4 of the possible message ids, severely constraining the scalability of a CAN network. In addition, although no analytic formula was produced as the result of this paper, the groundwork for future work was established (should someone deem this necessary), and a computer program which calculates the number of non-duplicate MPBS combinations was constructed.

References

[1] D. Paret, Multiplexed Networks for Embedded Systems, Wiley, 2007
[2] W. Voss, A Comprehensible Guide to Controller Area Network, Copperhill Technologies Corporation, 2005