1722B – PROPOSAL FOR ENHANCED CONTROL FORMATS – V3
(FOLLOW-ON PRESENTATION FROM "LARGER BUS ID SIZE – V2")

DON PANNELL
21 APR 2020

PUBLIC

SECURE CONNECTIONS
FOR A SMARTER WORLD

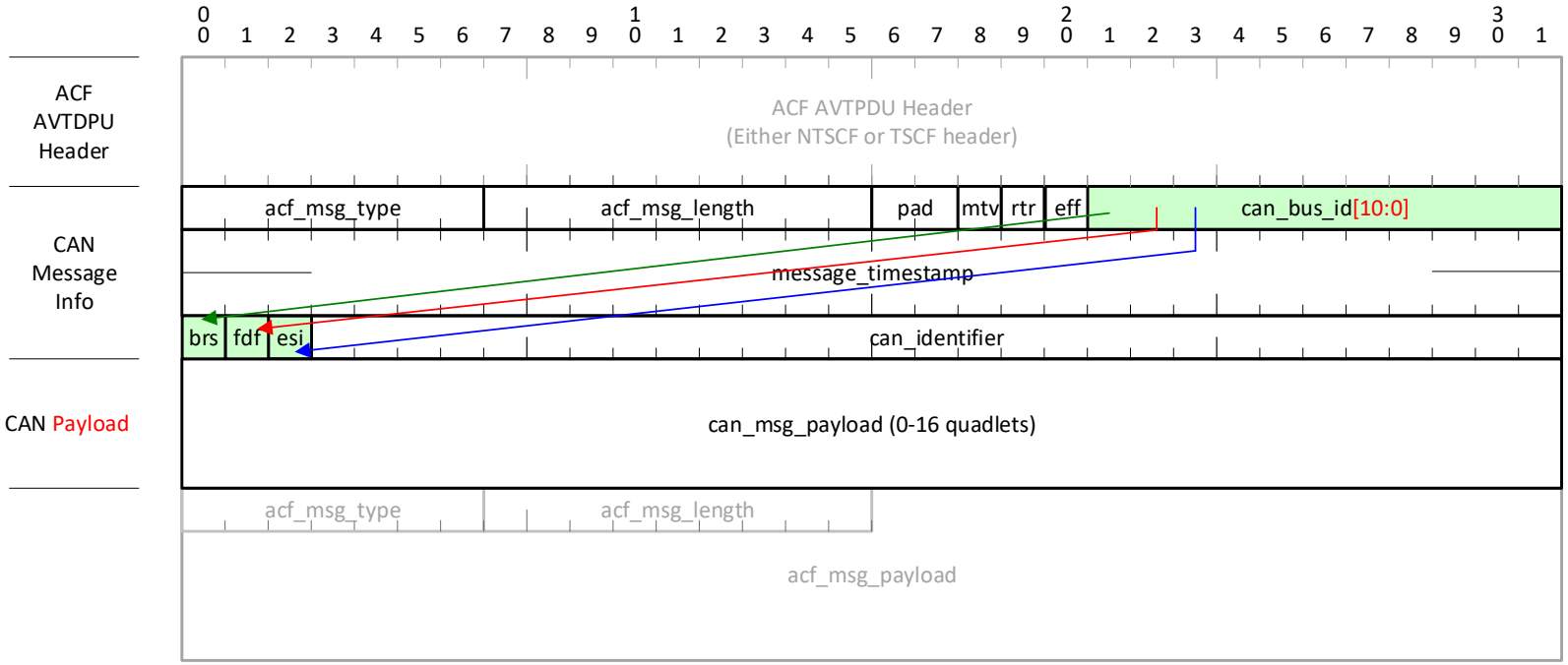
Overview – New Text in Red

- The need for enhancing the control formats in IEEE 1722b was discussed in IEEE1722b-pannell-larger_bus_id_size-2019-12 with examples for CAN & LIN
- Being part of the approved PAR work, this presentation is an updated proposal for a solution
- While the primary goal was to extend the bus_id sizes, a secondary goal emerged to harmonize the various formats since it was clear new formats were needed
  - The consistency of the formats allows for more efficient building & decoding of the frames
  - The current proposal is thus different from the 2019-12 one & the 2020-03 one
- Updated: Optional ACF Validate message (containing validate_data on a preceding message) is proposed for safety systems that can be added after any ACF message
- ACF Message values for the proposed formats are included
- In working on the details, it also became evident that CAN's can_identifier bits appear to be reversed – or at least need clarification

PROPOSAL FOR VERSION 2 CONTROL FORMATS

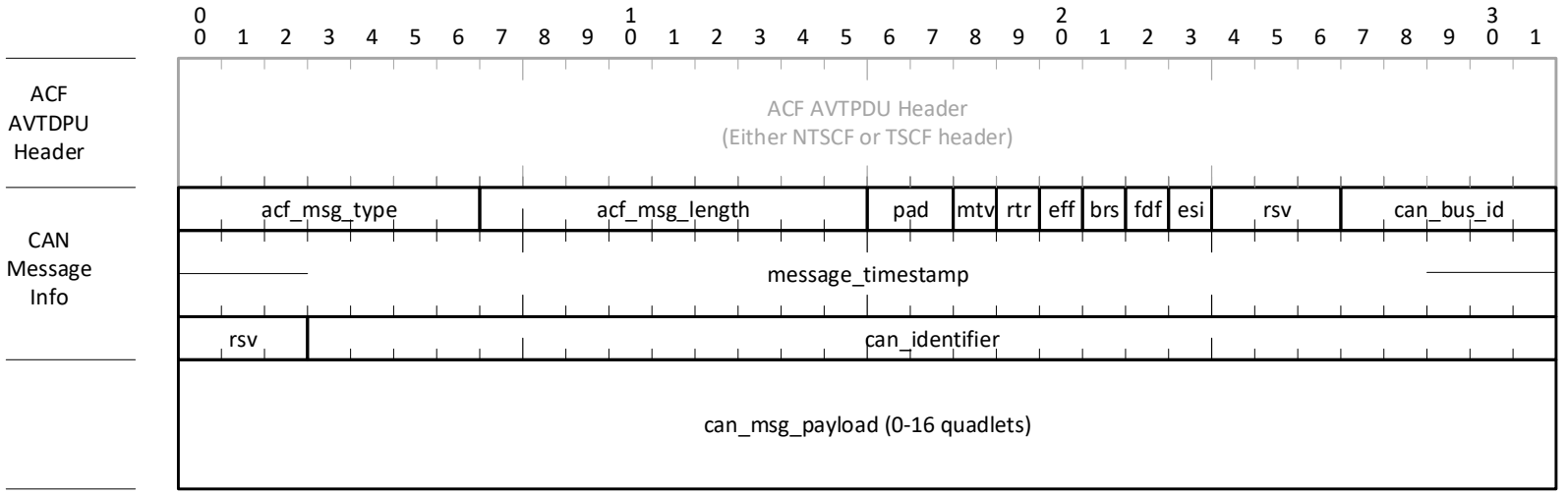
IEEE 1722b – Larger bus_id size – Proposal for CAN

- Move brs (bit rate switch), fdf (flexible data rate), & esi (error state indicator) bits to previously reserved bits ~~& remove the brs (bit rate switch) bit as shown~~
- Extend the can_bus_id to ~~12~~ 11 bits (the lower 5 bits are in the same location)
- The CAN format requires the most bits, thus limiting the bus_id to ~~12~~ 11 bits
- rtr (remote transmission request) & eff (extended frame format) are not moved

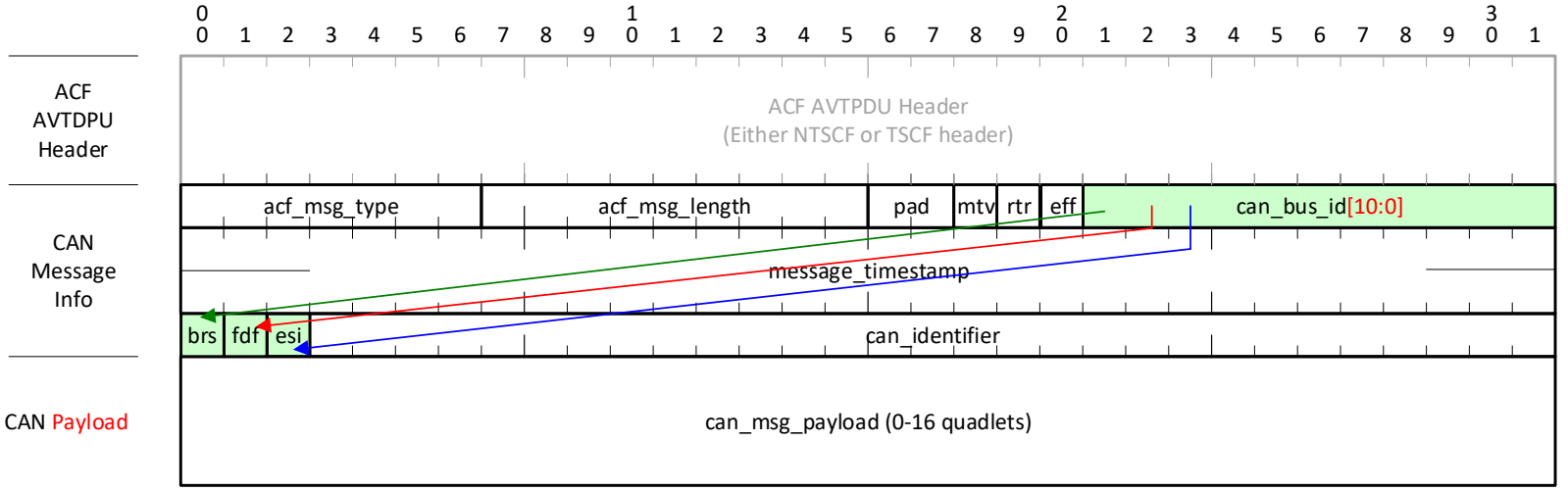


Larger bus_id size – Proposal for CAN – old vs. new

1722-2016

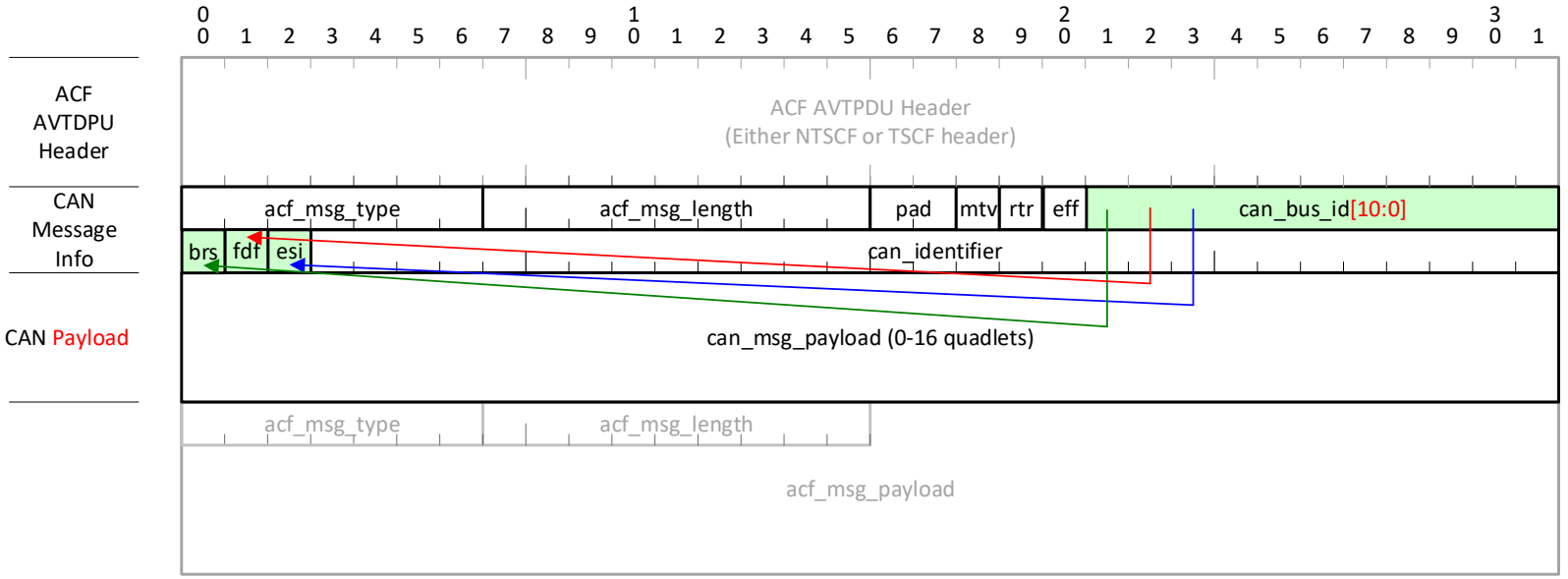


Proposal



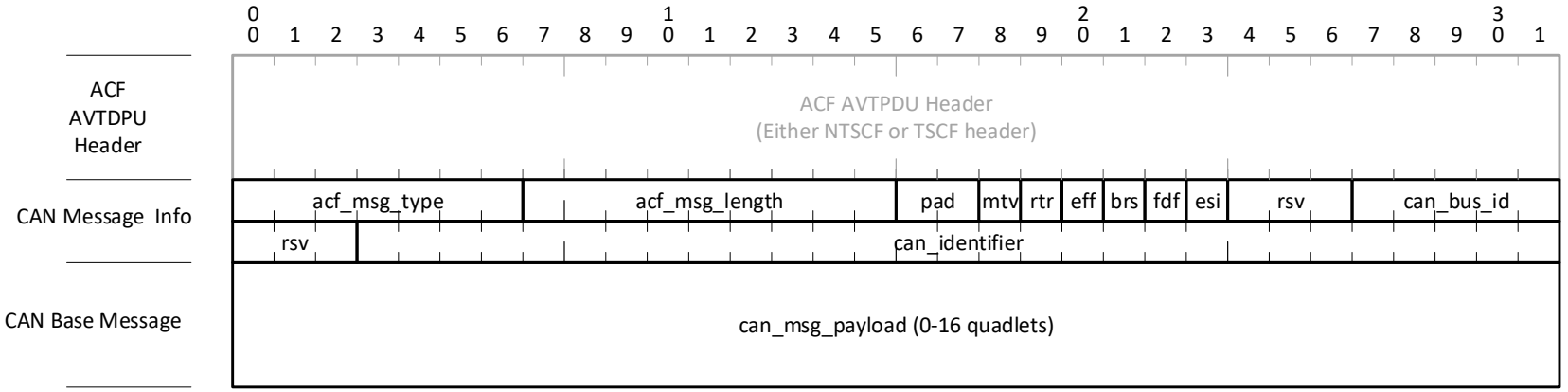
IEEE 1722b – Larger bus_id size – Proposal for CAN BRIEF

- CAN BRIEF gets the same changes as were done with CAN
- Move brs (bit rate switch), fdf (flexible data rate), & esi (error state indicator) bits to previously reserved bits ~~& remove the brs (bit rate switch) bit as shown~~
- Extend the can_bus_id to ~~12~~ 11 bits (the lower 5 bits are in the same location)
- rtr (remote transmission request) & eff (extended frame format) are not moved

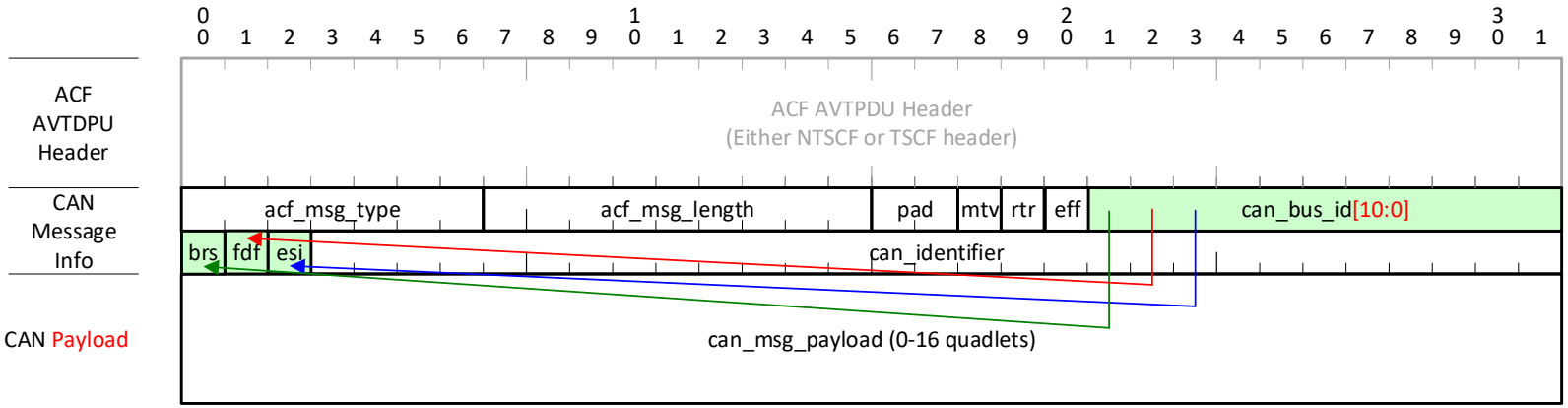


Larger bus_id size – Proposal for CAN BRIEF – old vs new

1722-2016

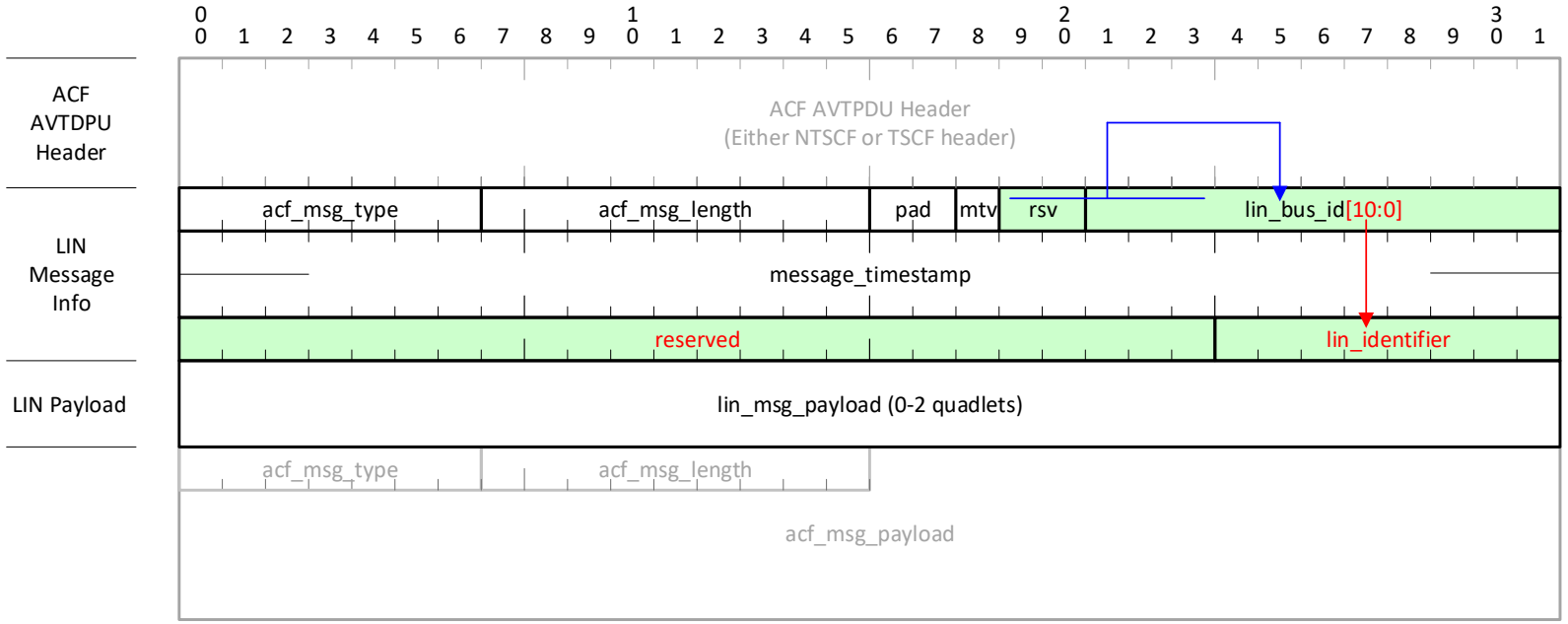


Proposal



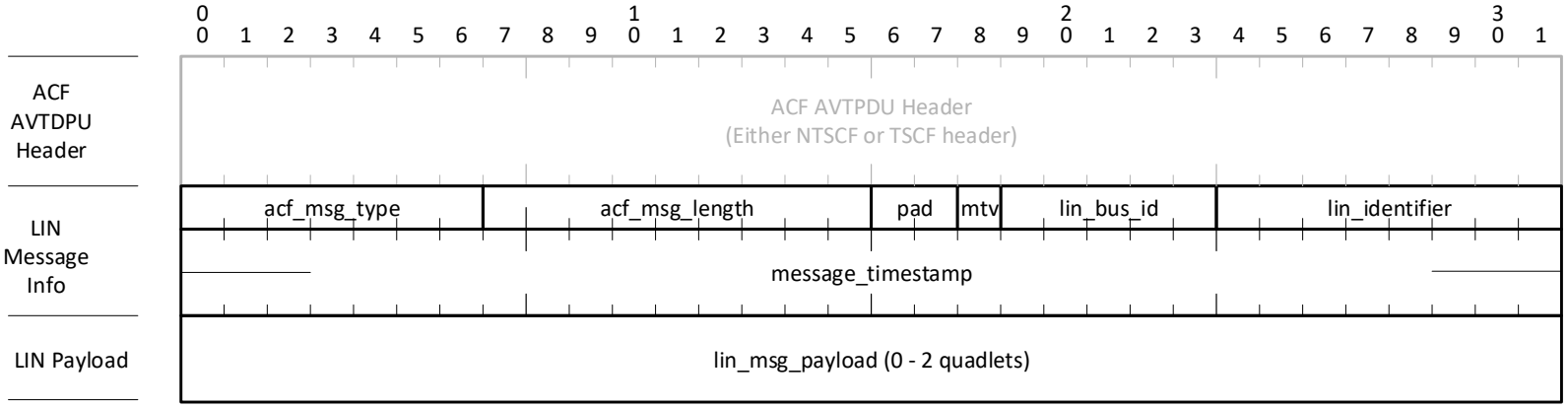
IEEE 1722b – Larger bus_id size – Proposal for LIN

- LIN was fully packed, so a full quadlet needs to be added (added green quadlet)
- The lin_identifier is moved to the new quadlet in the same area as the can_identifier (although its not as large as CAN's)
- The lin_bus_id is shifted right by 8 bits & expanded to ~~12~~ 11 bits leaving 2 ~~a~~ rsv bits

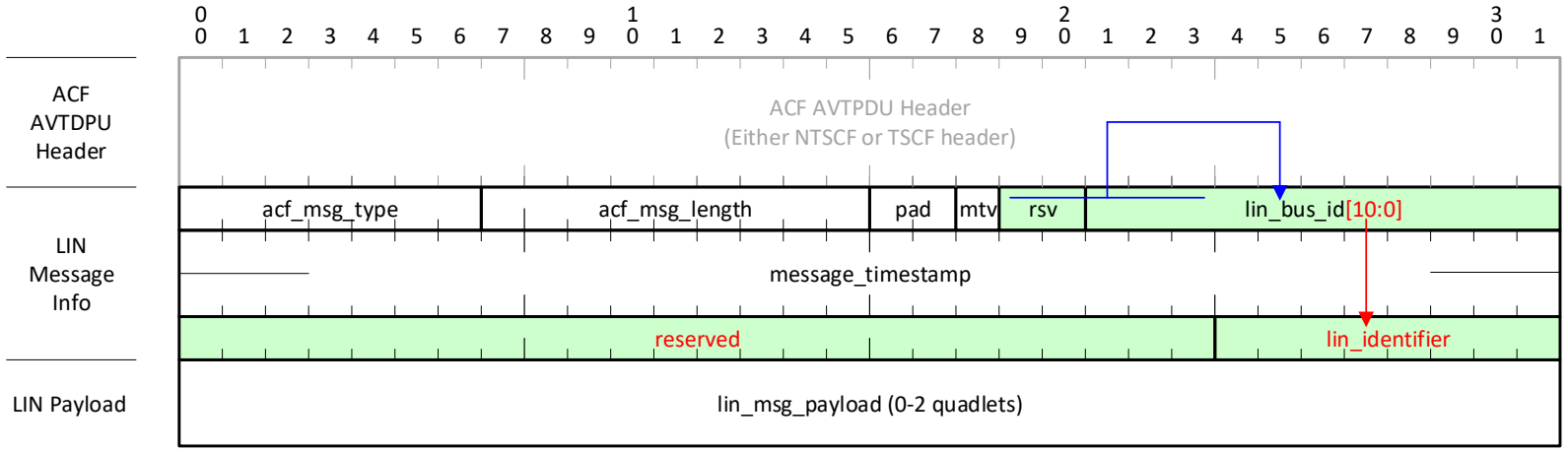


Larger bus_id size – Proposal for LIN – old vs. new

1722-2016

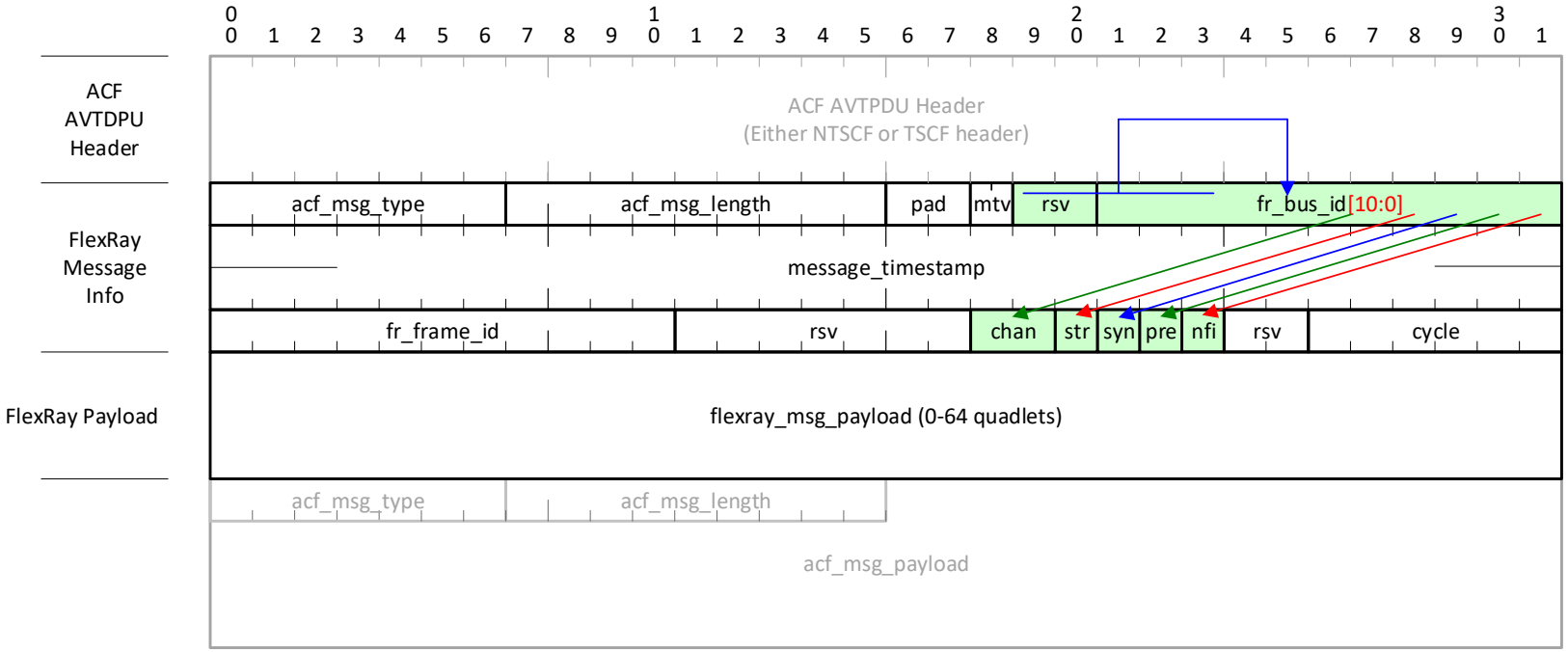


Proposal



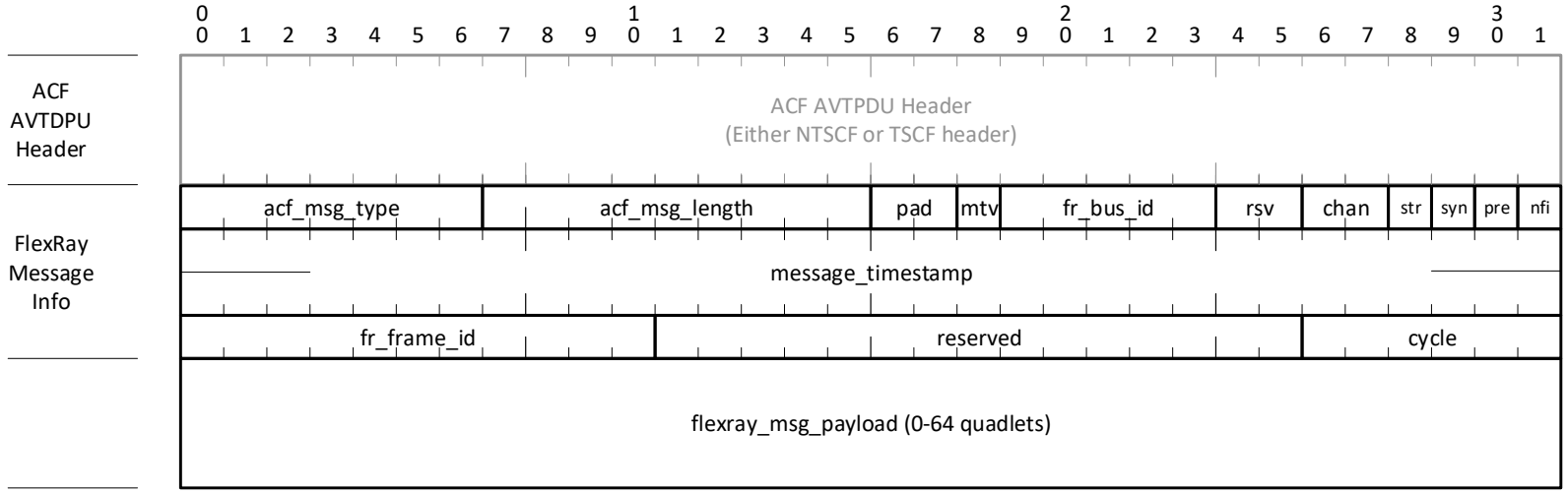
IEEE 1722b – Larger bus_id size – Proposal for FlexRay

- Move the chan (source channel), str (startup), syn (sync), pre (payload preamble), & nfi (null frame indicator) bits to previously reserved bits as shown to free up room for the expanded fr_bus_id
- The fr_bus_id is shifted right by 8 bits & expanded to ~~12~~ 11 bits leaving 2 ~~a~~ rsv bits

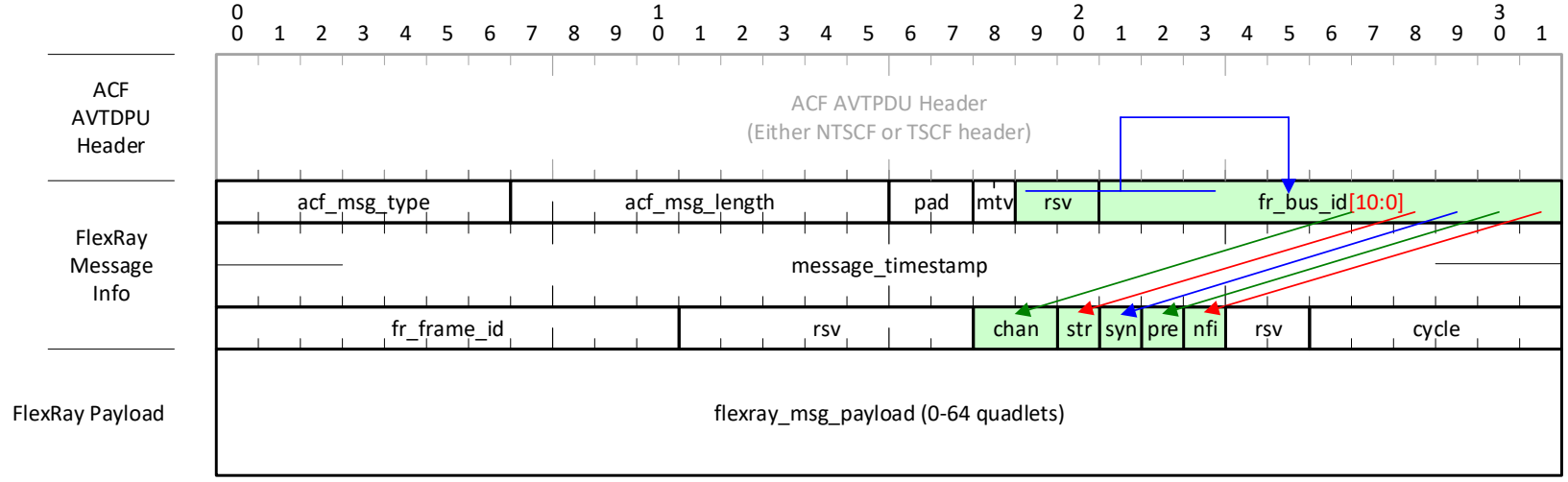


Larger bus_id size – Proposal for FlexRay – old vs. new

1722-2016

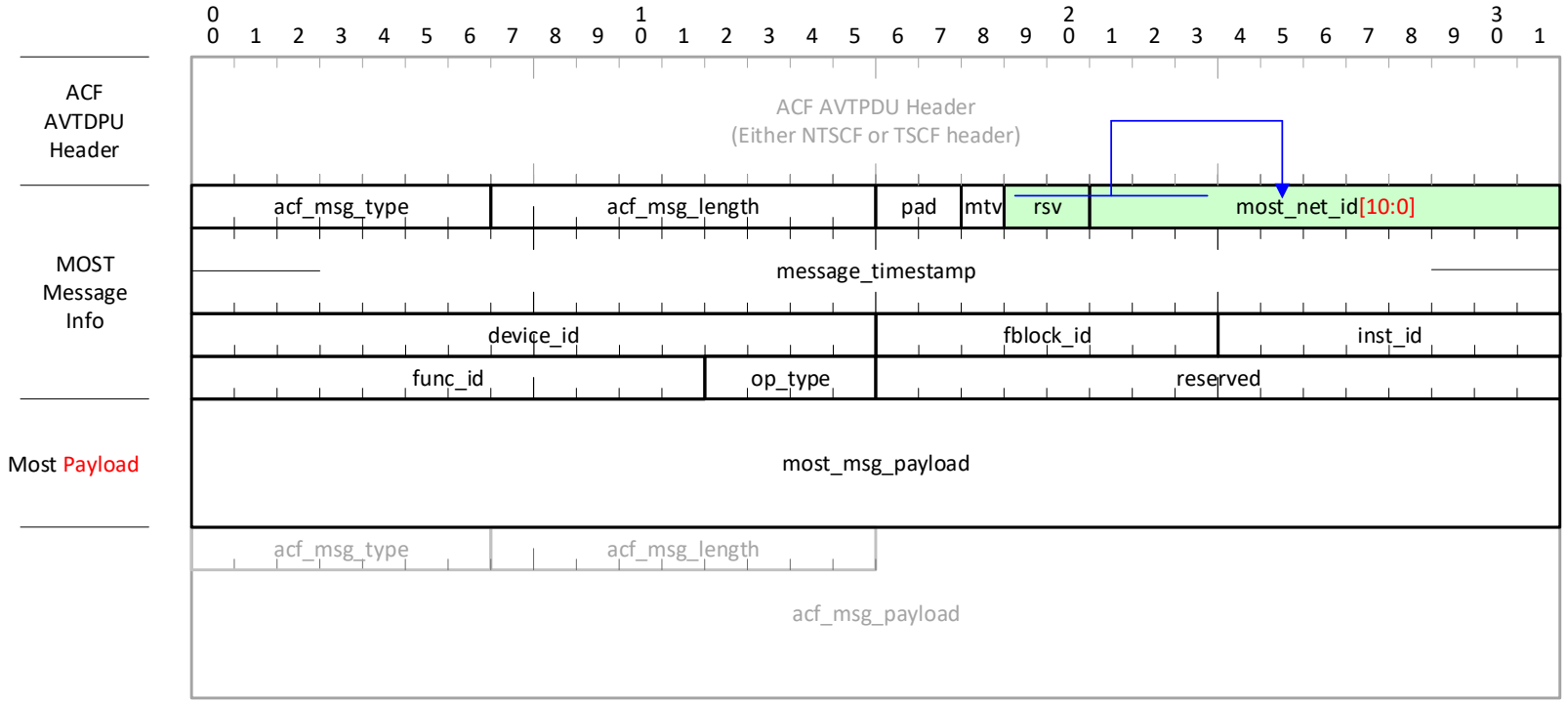


Proposal



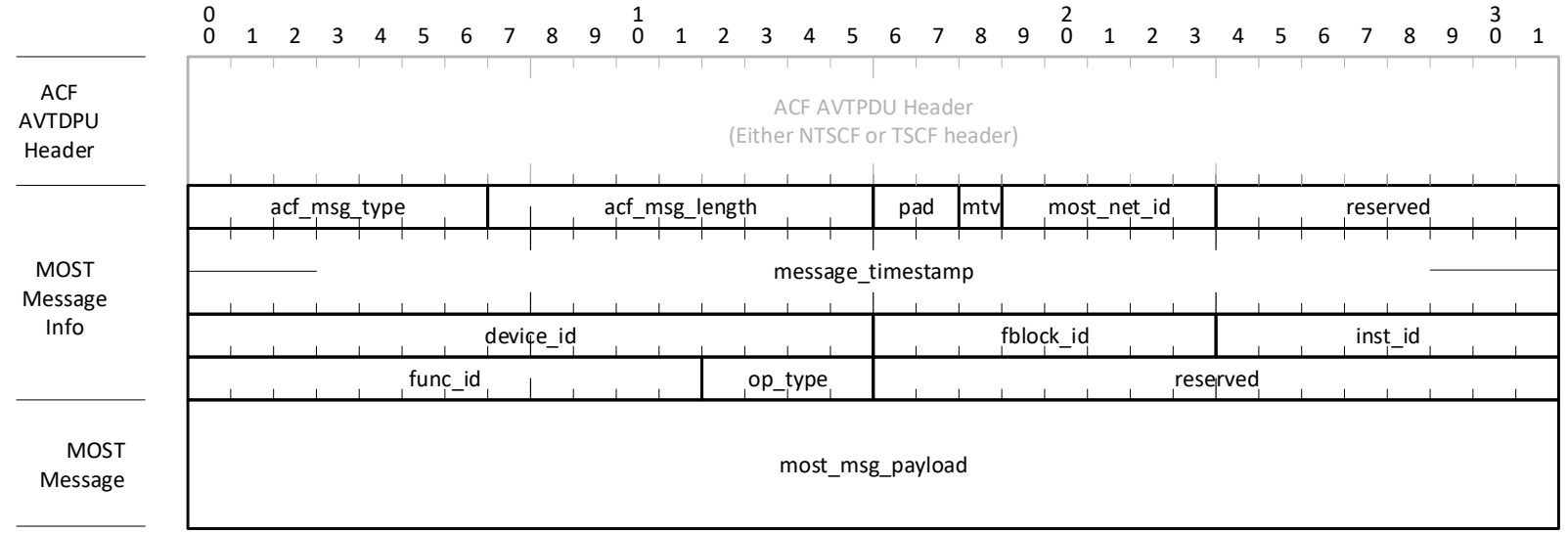
IEEE 1722b – Larger bus_id size – Proposal for MOST

- The most_net_id is shifted right by 8 bits & expanded to ~~12~~ 11 bits leaving 2 ~~a~~ rsv bits
- The lower 8-bits of the new most_net_id previously was reserved bits

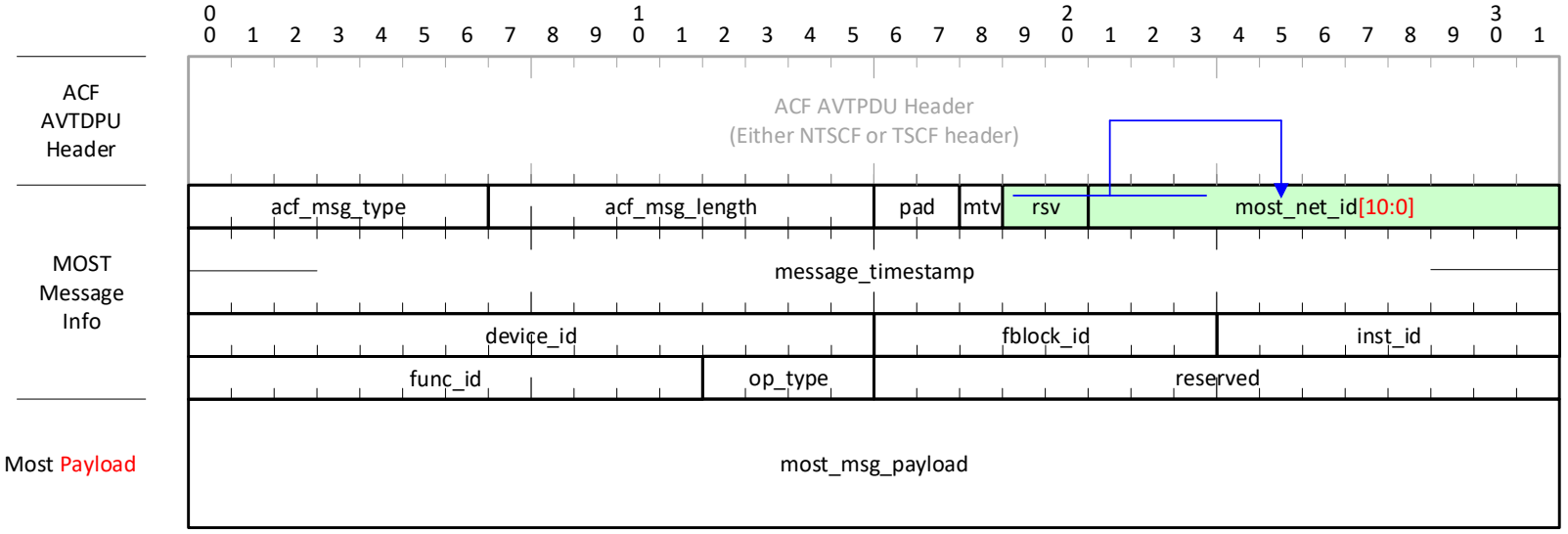


Larger bus_id size – Proposal for MOST– old vs. new

1722-2016



Proposal



PROPOSED NEW ACF MESSAGE TYPES FOR NEW VERSION 2 CONTROL FORMATS

Proposed New Table 22 – ACF Message Types (9.4.1.2)

Value	Name	Description	Subclause
00_{16}	ACF_FLEXRAY	FlexRay™ message	9.4.2
01_{16}	ACF_CAN	Controller Area Network (CAN)/CAN with Flexible Data-Rate (CAN FD) message	9.4.3
02_{16}	ACF_CAN_BRIEF	Abbreviated CAN/CAN FD message	9.4.4
03_{16}	ACF_LIN	LIN® message	9.4.5
04_{16}	ACF_MOST	MOST® message	9.4.6
05_{16}	ACF_GPC	General purpose control message	9.4.7
06_{16}	ACF_SERIAL	Serial port message	9.4.8
07_{16}	ACF_PARALLEL	Parallel port message	9.4.9
08_{16}	ACF_SENSOR	Analog sensor message	9.4.10
09_{16}	ACF_SENSOR_BRIEF	Abbreviated sensor message	9.4.11
$0A_{16}$	ACF_AECP	IEEE Std 1722.1 AECP message	9.4.12
$0B_{16}$	ACF_ANCILLARY	Video ancillary data message	9.4.13
$0C_{16}$ to $1F_{16}$	Reserved	Reserved	—
20_{16}	ACF_FLEXRAY_V2	FlexRay™ message v2	9.4.2
21_{16}	ACF_CAN_V2	Controller Area Network (CAN)/CAN with Flexible Data-Rate (CAN FD) message v2	9.4.3
22_{16}	ACF_CAN_BRIEF_V2	Abbreviated CAN/CAN FD message v2	9.4.4
23_{16}	ACF_LIN_V2	LIN® message v2	9.4.5
24_{16}	ACF_MOST_V2	MOST® message v2	9.4.6
25_{16} to 77_{16}	Reserved	Reserved	—
78_{16} to $7F_{16}$	ACF_USER	User-defined ACF message	—

Since the V2 formats are very similar to the originals, the proposal is to document them in the same Subclause as the originals

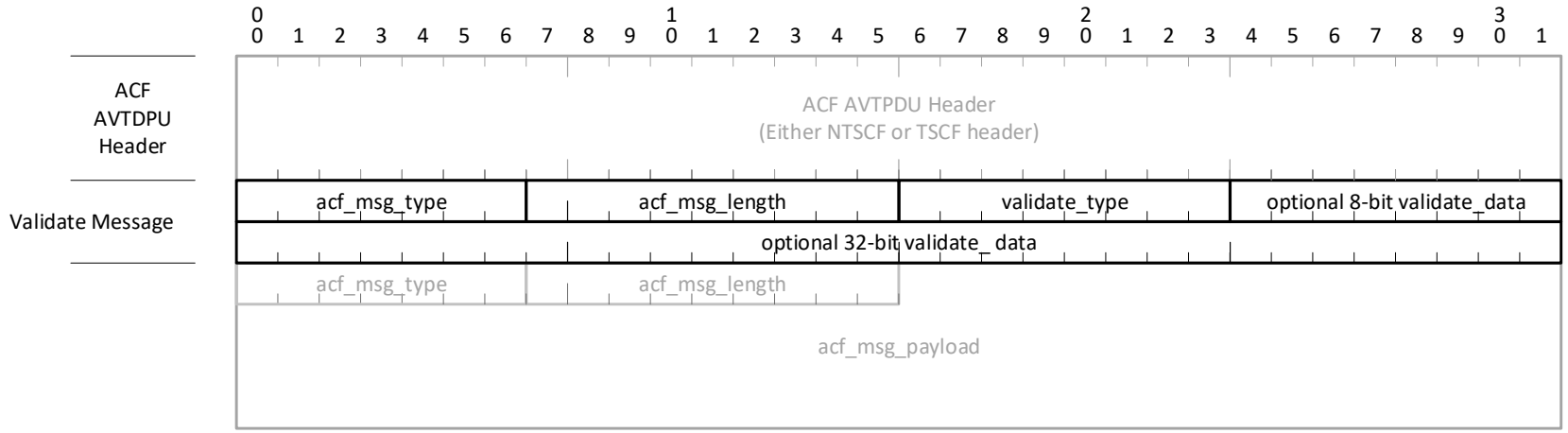
PROPOSED NEW ACF MESSAGE TYPES FOR NEW VERSION 2 CONTROL FORMATS

IEEE 1722b – Proposal for New ~~Checksum~~ Validate Message

- Some safety systems want to verify that a message has not been corrupted end-to-end & this needs to be done after the Ethernet CRC has been removed
- This proposal is to support a new ~~single quadlet Checksum~~ Validate ACF Message type that when used, carries ~~the checksum~~ validation data for the immediately previous acf_msg in this same frame, for the previous message's acf_msg_length
- This is mainly needed as an enhancement for CAN & LIN, but when done this way it works for all acf_msg_types
- Rule of use: Before acting on a current message, a check needs to be made to see if there is a subsequent Checksum Message or not – this is not hard to do

IEEE 1722b – Proposal for New Validate Message

- Some safety systems applications only need a 8-bit Checksum while others need a 32-bit CRC, etc.
- This proposal defines the octet following the message's acf_msg_length as a validate_type
  - This allows 256 different types where 0x00 is proposed to be an 8-bit Checksum and 0x80 is a 32-bit CRC (using the upper bits to define types and the lower bits as a size indicator)
  - If only 8-bits of validate_data are needed this is a single quadlet message
  - If the 32-bit validate_data is used, the 8-bit validate_data is 0x00 & ignored on read



PROPOSED NEW ACF MESSAGE TYPE FOR NEW VALIDATE CONTROL FORMATS

Proposed New Table 22 – ACF **Validate** Message Type (9.4.1.2)

Value	Name	Description	Subclause
00_{16}	ACF_FLEXRAY	FlexRay™ message	9.4.2
01_{16}	ACF_CAN	Controller Area Network (CAN)/CAN with Flexible Data-Rate (CAN FD) message	9.4.3
02_{16}	ACF_CAN_BRIEF	Abbreviated CAN/CAN FD message	9.4.4
03_{16}	ACF_LIN	LIN® message	9.4.5
04_{16}	ACF_MOST	MOST® message	9.4.6
05_{16}	ACF_GPC	General purpose control message	9.4.7
06_{16}	ACF_SERIAL	Serial port message	9.4.8
07_{16}	ACF_PARALLEL	Parallel port message	9.4.9
08_{16}	ACF_SENSOR	Analog sensor message	9.4.10
09_{16}	ACF_SENSOR_BRIEF	Abbreviated sensor message	9.4.11
$0A_{16}$	ACF_AECP	IEEE Std 1722.1 AECP message	9.4.12
$0B_{16}$	ACF_ANCILLARY	Video ancillary data message	9.4.13
$0C_{16}$ to $1F_{16}$	Reserved	Reserved	—
20_{16}	ACF_FLEXRAY_V2	FlexRay™ message v2	9.4.2
21_{16}	ACF_CAN_V2	Controller Area Network (CAN)/CAN with Flexible Data-Rate (CAN FD) message v2	9.4.3
22_{16}	ACF_CAN_BRIEF_V2	Abbreviated CAN/CAN FD message v2	9.4.4
23_{16}	ACF_LIN_V2	LIN® message v2	9.4.5
24_{16}	ACF_MOST_V2	MOST® message v2	9.4.6
25_{16} to 76_{16}	Reserved	Reserved	—
77_{16}	ACF_VALIDATE	Optional validation data for the immediately preceding ACF message in the same frame	???
78_{16} to $7F_{16}$	ACF_USER	User-defined ACF message	—

ACF_VALIDATE is placed at the end of the table as it can work for all ACF_Message types & added at the end of the Subclause

CAN_IDENTIFIER PROPOSED DOC CLARIFICATIONS

Clarification of the can_identifier bits in CAN Messages

- IEEE 1722-2016 makes it clear the most significant bits are on the left as seen:

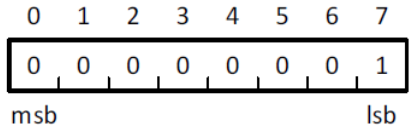


Figure 1—Bit ordering within an octet

Represented as 0x01

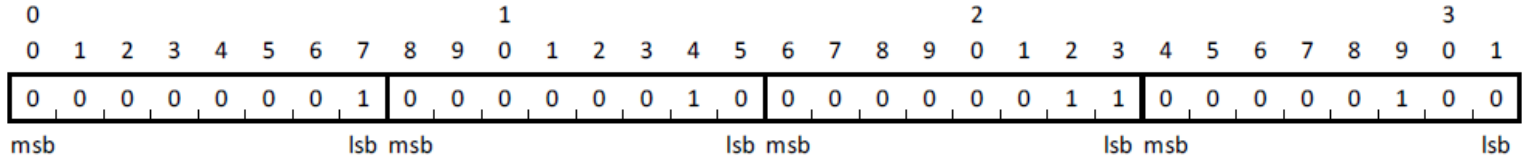
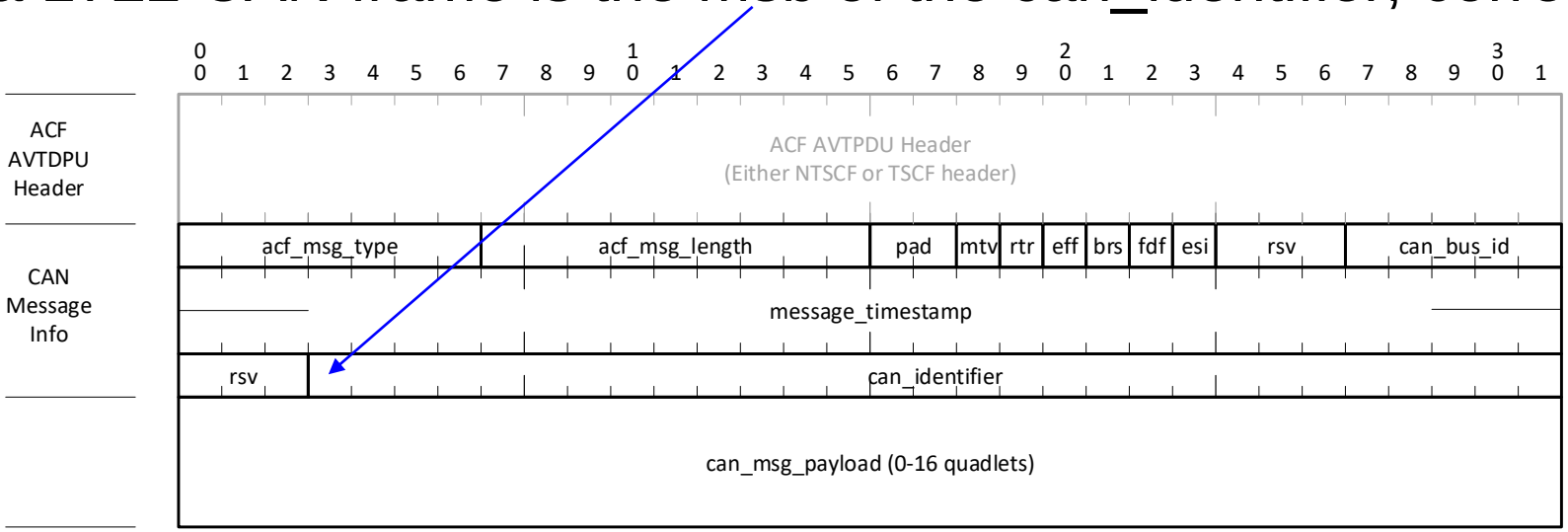


Figure 2—Octet ordering within a quadlet

Represented as 0x0102 0304

- Bit 03 in a 1722 CAN frame is the msb of the can_identifier, correct? It should be!



can_identifier's usage from a CAN community point of view

- CAN & CAN-FD both support two can_identifier sizes:
  - An 11-bit base ID identified as bits 28 to 18 of the ID where bits 17 to 0 do not exist
  - 29-bit extended ID identified as bits 28 to 0 of the ID
    - Bit 28 is the most significant bit in both sizes & it's the 1st bit transmitted down the wire
- The ID is used for bus arbitration using a bit-by-bit comparison of what I transmitted vs. what I see on the wire where a 0 is dominate (wins)
  - Whenever I see a 0 when I transmitted a 1 during the ID phase, I have to stop transmitting until the next transmit opportunity
  - An 11-bit ID, written as: 0b000 0000 1111 or (by industry convention) 0x00F
  - Wins over an ID of: 0b111 0000 0000 or (by industry convention) 0x700
    - As the 1st 0b0 bit is the msb & 1st bit transmitted down the wire (& identified as bit 28 of the ID)
    - In other words, the lowest ID number always wins the bus

can_identifier's problem

- IEEE 1722-2016 supports a 29-bit field for the can_identifier & the eff bit (extended frame format) to indicate its size (0 = 11-bit, 1 = 29 bit)
- The standards states which of the 29-bits should be used for an 11-bit ID as:

9.4.3.11 can_identifier field

The 29-bit **can_identifier** field contains the CAN message identifier. CAN message identifiers are either 11 or 29 bits in length. The length of the **can_identifier** field is communicated by the value of the **eff** bit (see 9.4.3.5). 11-bit CAN message identifiers are stored in bits 21 through 31 of the quadlet holding the **can_identifier** field and the remaining bits of the field are set to zero (0). 29-bit CAN message identifiers occupy the entire 29-bit field. Storage of the 11-bit CAN identifier is shown in Figure 55. The letter *v* indicates a valid identifier bit.

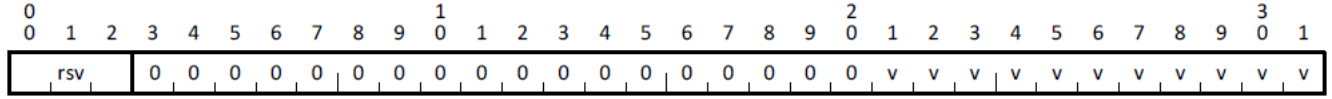


Figure 55—Storage of an 11-bit CAN identifier

- Which is backwards as bit 3 is the msb!
- Or is bit 31 is the msb and all 29 bits are swizzled compared to convention,
- Or is bit 11 the msb for 11-bit IDs & bit 3 is the msb for 29-bit IDs

can_identifier's solution defining bit 3 as msb for both sizes

- This at least needs to be clarified in IEEE 1722b
  - The correct solution to me is define bit 3 in Fig 55 as the msb for both ID sizes, move the v's to bits 3:13 and update the text and figures accordingly:

9.4.3.11 can_identifier field

The 29-bit **can_identifier** field contains the CAN message identifier. CAN message identifiers are either 11 or 29 bits in length. The length of the **can_identifier** field is communicated by the value of the **eff** bit (see 9.4.3.5). 11-bit CAN message identifiers are stored in bits ~~21~~ 3 through ~~31~~ 13 of the quadlet holding the **can_identifier** field and the remaining bits of the field are set to zero (0). 29-bit CAN message identifiers occupy the entire 29-bit field. Storage of the 11-bit CAN identifier is shown in Figure 55. The letter *v* indicates a valid identifier bit.

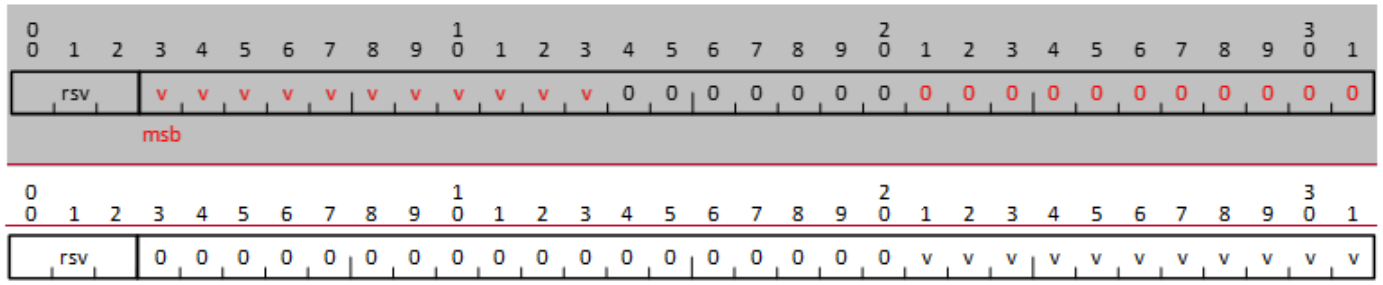


Figure 55—Storage of an 11-bit CAN identifier

- This could be done for all CAN formats on just the Proposed V2 Formats (above)

can_identifier's solution defining bit 31 as msb for both sizes

- Alternatively, define bit 31 in Fig 55 as the msb for both ID sizes:

9.4.3.11 can_identifier field

The 29-bit **can_identifier** field contains the CAN message identifier. CAN message identifiers are either 11 or 29 bits in length. The length of the **can_identifier** field is communicated by the value of the **eff** bit (see 9.4.3.5). 11-bit CAN message identifiers are stored in bits 21 through 31 of the quadlet holding the **can_identifier** field and the remaining bits of the field are set to zero (0). 29-bit CAN message identifiers occupy the entire 29-bit field. Storage of the 11-bit CAN identifier is shown in Figure 55. The letter *v* indicates a valid identifier bit.

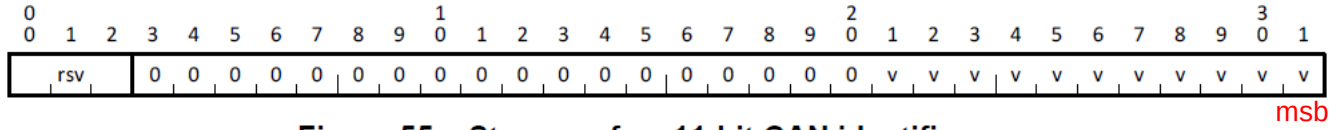


Figure 55—Storage of an 11-bit CAN identifier

- If the previous solution is not chosen for the existing CAN formats (assuming people implemented it as shown in Fig 55) the msb must at least be labeled!
- But if the msb is bit 31 this goes against IEEE 1722's convention & there is no guarantee that implementers assumed this!

can_identifier's solution defining bit 31 as msb for <u>both</u> sizes

- Alternatively, define bit 21 in Fig 55 as the msb for 11-bit ID sizes & bit 3 as the msb for 29-bit ID sizes:

9.4.3.11 can_identifier field

The 29-bit **can_identifier** field contains the CAN message identifier. CAN message identifiers are either 11 or 29 bits in length. The length of the **can_identifier** field is communicated by the value of the **eff** bit (see 9.4.3.5). 11-bit CAN message identifiers are stored in bits 21 through 31 of the quadlet holding the **can_identifier** field and the remaining bits of the field are set to zero (0). 29-bit CAN message identifiers occupy the entire 29-bit field. Storage of the 11-bit CAN identifier is shown in Figure 55. The letter *v* indicates a valid identifier bit.

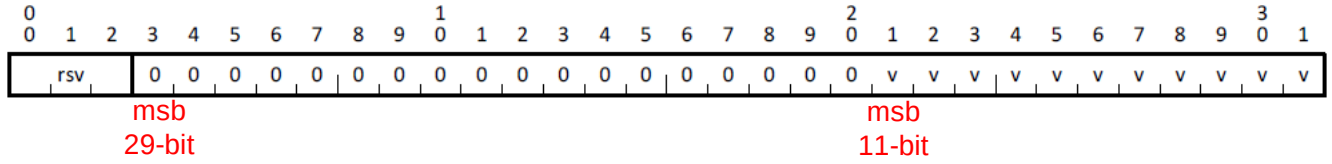


Figure 55—Storage of an 11-bit CAN identifier

- This keeps the msb to the left, but is weird to anyone used to CAN & how it is specified & operates (i.e., the msb of the ID never moves & is always the 29th bit)
  - An 11-bit ID of 0x3FF, seen as 0x0000 03FF, is higher priority than a 29-bit ID of 0x0FFF FFFF!
  - The ID are priority bits and it is hard to tell the size without looking at the message's eff bit.
- Because of these problems, some clarification is needed in 1722b!
  - But which one for the current formats & should be different for the V2 formats?

SECURE CONNECTIONS
FOR A SMARTER WORLD

NXP and the NXP logo are trademarks of NXP B.V. All other product or service names are the property of their respective owners. © 2019 NXP B.V.