How to use high bit-rates in a CAN-system

Kent Lennartsson and Jonas Olsson, Kvaser AB

The use of higher bit-rates is not difficult to implement into modern microcontrollers. Even the smallest microcontrollers can handle clock-speeds in the 50 MHz range; enough to run CAN-FD bus at 10 Mbit/s. Also, the smallest MCU on the market today offer a 32-bit core with 32 kByte Flash and 8 kByte SRAM, as well as a great number of peripheral circuits. Compared to the overall chip size, the CAN-controller represents just 1% of the total area, with the result that the move from a classic CAN-controller to a CAN-FD version will increase the total chip area by just 0.5%.
More challenging for the industry is how to handle the physical layer and in particular, the cable layout. CAN's strength is its robustness, but this means that it can be forgiving of less than ideal CAN configurations. In most cases a correct bit-length and some resistance between the two wires is enough to make the CN communication work, because as long as the value at the sampling point is correct the CAN-frame is accepted. When the noisy edge comes close to the sample point, CAN is less forgiving. If you need higher bandwidth, the only solution is to increase the bit-rate and learn how to handle the short bits without any problem. The laws of physics are the same for CAN-FD, FlexRay, Ethernet etc., so that if a cable works with one of the above protocols, it will work for all others at the same bit-rate.

The CAN FD Hardware

CAN FD merely represents some additional logic to a standard CAN-controller. The challenge lies in the higher bit-rate, which demands faster clocking from the bit handling logic. Our design is implemented in a Cyclone 4 FPGA from Altera which clocks at 80 MHz. We need at least one clock per Time Quanta to handle the bits. Theoretically, it will be possible to run CAN-FD at 20 Mbit/s, which is far beyond what is possible with the CAN-drives available today.

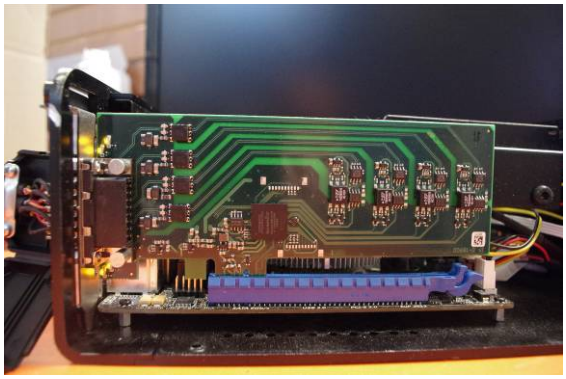


Figure 1: Four channel CAN FD on a PCI-express board

The bus-structure and the cable layout make it very hard to reach such levels, even if CAN-drivers become available that can handle such bit-rates. Whit this knowledge it is clear that the CAN controller logic will never limit the speed of the CAN FD, rather the limiting factor will always be at the physical layer The actual hardware with 4 CAN FD-channels is shown in figure 1 running in a Linux computer.
In figure 2, a block diagram shows one of four CAN-channels. This hardware is optimized for a PC-computer interface where 100% of the messages needs to be received and transferred to the processing application.

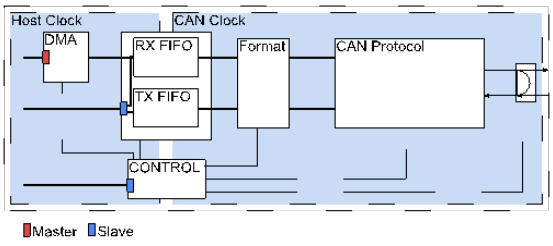


Figure 2: Block view over one of the four CAN FD channels

To reduce the delay over the PCI-express link, a DMA is used to transfer data directly

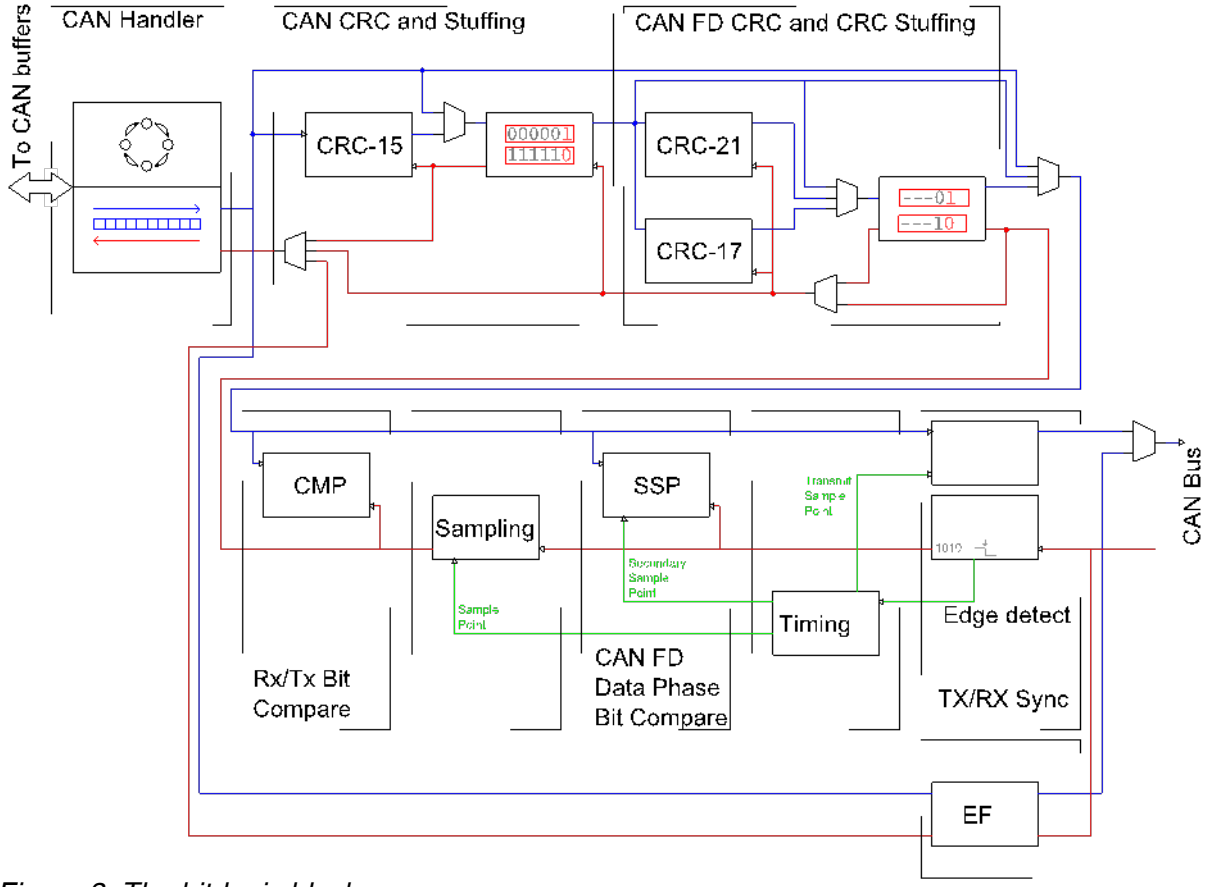


Figure 3: The bit-logic block

from the CAN-controller FIFO to the PC-memory. In most cases, the FIFO is empty and the only reason there is anything in the FIFO is due to a delay caused by Linux performing more important tasks.

The CAN to CAN-FD step

At the block level (see Figure 2) there is almost no difference to be seen between Classic-CAN messages and a CAN FD message. If you restrict CAN FD to handle 0 to 8 byte of data, just two bits need to be added to the CAN message, FDF and BRS, indicating a request to send the package in FD format and if required, at a higher bit-rate. The FDF bit is used to indicate whether the message will be sent as Classic-CAN or CAN FD. The BRS bit is used to indicate a use of higher bit-rate in the CAN FD frame. It is possible to send 0-8 byte data in a CAN FD frame without increasing the bit-rate. The advantage of this compared to Classic-CAN is the use of the more protective CRC-17 (a cyclic redundancy check code) in place of the CRC-15 used in Classic-CAN.

To handle CAN-frames with more than 8 bytes will demand a new layout of the CAN-buffers to fit up to 64 byte of data. The problem of extending the buffer size is not covered in this paper.

The logic behind CAN FD

Figure 3 shows the logic that takes the selected message buffer and places it on the bus during a transmit request or receives the CAN-messages from the bus and hands it over for storage in a buffer. This logic includes the functions necessary to handle both Classic CAN and CAN FD frames. Two dedicated sub blocks have been added to the bit-logic to handle CAN FD text. The rest of the logic is common to both Classic CAN and CAN FD. This logic needs to be active all the time, according to the CAN-protocol i.e. from the ‘bus-on’ command until the ‘bus-off’ command, which turns communication off. Figure 3 shows a simplified data flow. There are a great number of control signals to handle the logic under different conditions.

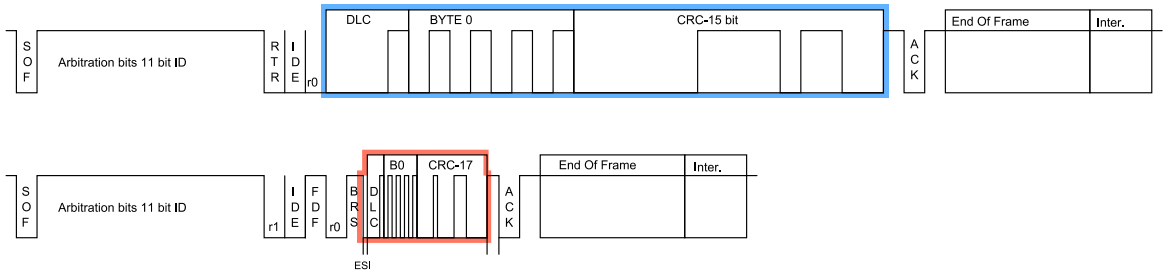


Figure 4: Classic-CAN compared with CAN-FD with one byte and 5 time faster bit-rate

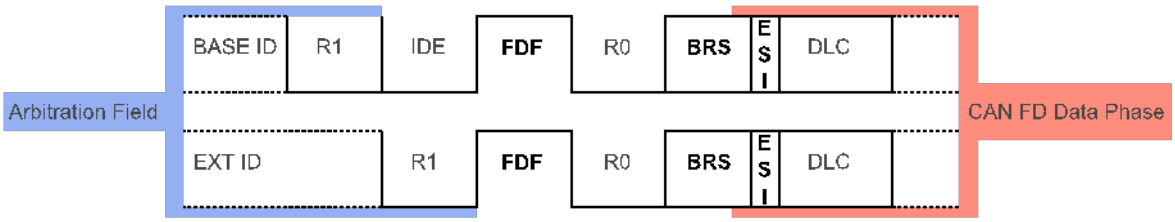
Also, the Error handling logic is left out of the picture to simplify the description.
The receiving path is not as complex as the transmit path. The receive path is the red line (the lines with arrows pointing to the left). The input from the CAN-driver is in the right lower corner of the figure "TX/RX Sync". This interface will detect if there is a recessive to dominant edge that indicates the start of a CAN-Frame, SOF-bit. This sets the bit-timing logic in motion, which samples the bit-stream according to the programmed configuration of the bit from the bus.
The received bit is passed directly to CRC-17 and CRC-21. However, the received bit passes through de-stuffing before it reaches the CRC-15, which is one of the major differences between Classic-CAN and CAN FD. By including the stuff-bits in the CRC-calculation you will get better protection against certain combinations of bit-errors in data-bits and any stuff-bits not included in the CRC-calculation.
The de-stuffed bits are sent to the "CAN Handler" where the bits are collected and assembled into a complete message. The CAN handler will also check some control bits in the data-stream, which indicate the different types of CAN-frames to process. Classic-CAN has 4 types of messages all with CRC-15 and with nominal bit-rate.

1. 11-bit ID with 0-8 byte of data.
2. 11-bit ID with Remote Request.
3. 29-bit ID with 0-8 byte of data.
4. 29-bit ID with Remote Request.

Depending of the detected message type, it is necessary to adjust the assembly of the bits in the CAN handler before it is transferred into the CAN-buffer.
The picture is made a little more complex when CAN FD is introduced into the control logic. For CAN FD, it is not enough to handle the bits differently in the CAN Handler, because the bit-handling shown in figure 1 also needs to be modified. CAN FD does not support Remote request but there are other CAN frame types that need to be handled. The following CAN FD message formats have to be handled in different ways in the CAN Handler and in bit-logic.

1. 11-bit ID with 0-8,12,16 byte of data CRC-17 with nominal bit-rate
2. 11-bit ID with 0-8,12,16 byte of data CRC-17 with data bit-rate.
3. 11-bit ID with 20,24,32,48,64 byte of data CRC-21 with nominal bit-rate
4. 11-bit ID with 20,24,32,48,64 byte of data CRC-21 with data bit-rate.
5. 29-bit ID with 0-8,12,16 byte of data CRC-17 with nominal bit-rate
6. 29-bit ID with 0-8,12,16 byte of data CRC-17 with data bit-rate.
7. 29-bit ID with 20,24,32,48,64 byte of data CRC-21 with nominal bit-rate
8. 29-bit ID with 20,24,32,48,64 byte of data CRC-21 with data bit-rate.

Even if the handling of the 8 different formats is very similar, it is necessary to modify the logic (shown in Figure 4) to correctly receive CAN FD frames.



Figur 5: The bits that control the different CAN-FD functions

Figure 4 compares a Classic-CAN and CAN FD message with one byte payload where the data-rate is 5 times the nominal bit-rate. As can be seen in the figure, the CAN FD frame is much shorter because data as well as DLC and CRC are sent at the higher bit-rate. The CAN Handler will process the bits one by one in the same way independent of the used bit-rate. The only demand is that the logic is fast enough to process one bit before the next is received.

Reception of a CAN FD frame

Figure 5 is a view of the bits that changes the interpretation of the received bits when CAN FD is included in the CAN-standard. The first bit that involves CAN FD is the FDF-bit "Flexible Data Format" and at this point we already know if the CAN-message is 11 or 29 bit ID. Also the RTR-bit has been resolved in Classic-CAN. At this point of the message it could either be a Classic-CAN frame with 0-8 byte or one of 4 CAN FD formats. The Classic-CAN format is indicated by a dominant FDF-bit and in that case there will be no change needed in the logic in figure 3.
If the FDF-bit is recessive the CAN format will be one of four possible types. In the FD format, CRC-15 is not used and can be disabled. In Classic-CAN, DLC follows the FDF-bit but in CAN FD, three more bits are included before DLC is reached. The first bit is R0-bit, which is reserved for future use to indicate the protocol that will come after CAN FD. For the CAN FD protocol this bit will always be dominant.
The next bit is the BRS-bit, "Bit Rate Switch". If this bit is dominant, the communication will retain the nominal bit-rate through the rest of the message. This condition will not change anything in the figure 3 logic. If this bit is sampled as recessive it indicates that the rest of the bits will be sent at the higher data bit-rate. To handle the higher bit-rate it is necessary to change the sample clock to sample at this higher rate. Also, the edge detector need to switch to a clock that matches the shorter TQ "Time Quanta" used at the higher data bit-rate.
The rest of the logic will continue to work in the same way, except at a higher clock rate.
The next bit is the ESI-bit, "Error Status Indicator", which is normally dominant but will be sent recessive when the sender is Error-Passive (the module has a major communication problem). This bit does not change anything in the control logic, but could be used for higher layer fault handling or diagnostic functions. The Error-Passive status is an internal condition in the CAN-controller and the sending CAN-controller can send this information without any support from host software.
The ESI-bit is the first high speed bit and is followed by the 4-bit DLC pattern. The table below shows the possible number of data bytes supported by Classic-CAN and CAN-FD. The CAN handler has to convert the DLC into a number of bytes that need to be received in this particular message. It is also necessary to select which CRC will be used - CRC-17 or CRC-21 - at the end of the message, so that it can be compared with the CRC received from the sender of the CAN-message. When the last bit in the DLC is received, it is 100% clear how to handle every bit in this particular CAN message.

Table 1: DLC coding

DLC	**CAN Bytes**	**CAN-FD Bytes**	**CAN-FD CRC-**
0-8	0-8	0-8	17
9	8	12	17
10	8	16	17
11	8	20	21
12	8	24	21
13	8	32	21
14	8	48	21
15	8	64	21

The last task for the CAN-handler when all data-bits have been received is to start the CRC-checking. This task is performed differently in Classic-CAN and CAN FD. To secure resynchronization, CAN FD uses fixed stuffing in the CRC-sequence where every 5th bit is followed by a bit value inverted by the previous bit value.

This secures a recessive to dominant edge within at least 10 bits. When a CAN FD frame reaches the CRC, the CRC-bits are redirected to CRC de-stuffing before the received CRC-bits are compared to the internal calculated CRC-sequence. Whether CRC-17 or CRC-21 is used depends on the number of data-bytes received. If the number of bytes is 16 or less, CRC-17 is used and if the number of bytes is more than 16 then CRC-21 is used.

Transmission of a CAN-FD frame

The main difference during transmission is the bit-error checking, when the sender must check that the transmitted bit is reflected with the correct value on the bus-line.
When a bit is transmitted it will take some time before the next bit has passed through the electronics out to the bus line and back again. This delay for Classic-CAN devices are in the range 50 to 300 ns "nano seconds" and this could fit into the typical nominal bits that is not shorter than 1000 ns. This make the logic for the nominal bit very simple because the transmitted bit is still valid when the bus is sampled and a direct compare can be used (shown in Figure 3 with the box CMP).

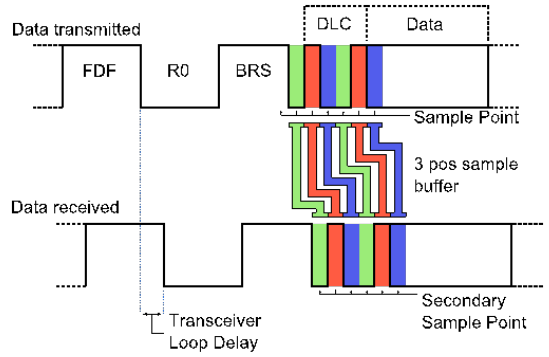


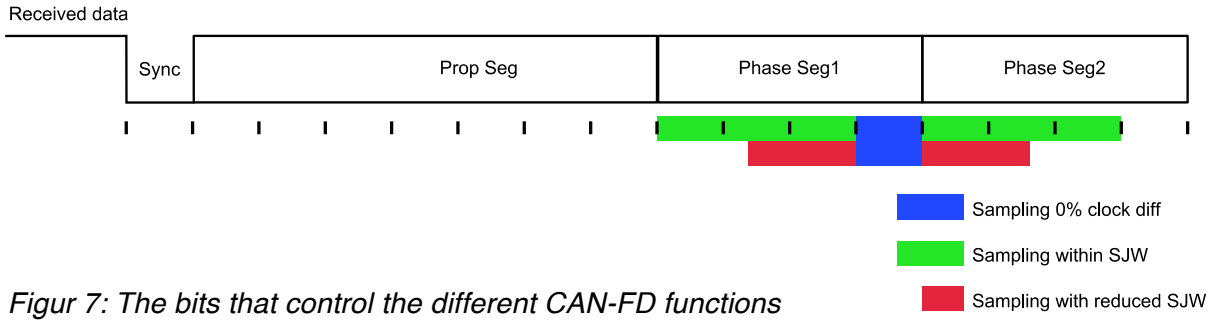
Figure 6: at the higher bit-rate it is necessary to include a delayed compare

As shown in figure 6 can in CAN FD this delay be longer than the length of high speed bits and a direct compare is not possible.

The logic has to store the sent value and compare this value when this bit have been received back from the CAN-line. The more complex solution that's necessary for CAN FD is shown in Figure 6. The delay between the bit sent and the bit value received is measured at the EDL to R0 edge. This measured time is used as a delay in the SSP-block "Secondary Sample Point" in figure 3. The bit-value sent is saved and after the delay is compared with the value sampled from the CAN-bus. This delay is normally 1 or 2 bits but our design can handle up to 8 bits.
The CAN handler also needs to select correct stuffing, CRC and bit-rate depending on the configuration of the CAN-buffer.

CAN-FD with Classic-CAN

One major problem with CAN FD is the redefinition of the Classic-CAN R1-bit into the FDF-bit. No Classic-CAN controller will accept CAN FD frames until the CAN controller has been modified according to the next ISO-11898-1 standard. In Classic-CAN, the FDF-bit is received as either dominant or recessive and both values should be accepted. This is problematic for Classic-CAN because in it, the DLC is defined in the following four bits, rather than the new bits defined in CAN FD. Sooner or later, Classic-CAN will consider that the CAN FD frame has violated the CAN protocol. To solve this problem it is suggested that a redefinition of the FDF-bit is implemented in Classic-CAN where Classic-CAN controllers will ignore all bits following the recessive FDF-bit and reestablish communication when the CAN-FD frame has ended. The end of any CAN message is a sequence of 11 recessive bits. However, just sampling the bus at a nominal rate is not enough in order to find this sequence correctly. With such a simple solution there is a risk that this sequence could be found when sampling within the CAN FD frame before the EOF "End Of Frame" section. As of September 2013, no final solution has been specified as to how this will be implemented.



Figur 7: The bits that control the different CAN-FD functions

Enhanced CAN-format

During the development of the CAN FD controller, we had to analyze the CAN-bits in detail to secure our behavior on the bus. During this work we found that Classic-CAN is very insensitive to noise within the bit and that it will accept noise in the bit, as long as you receive the correct value at the sampling point. To understand how this works, we have to study the definition of a nominal bit, as shown in figure 7. In Classic-CAN, the CAN-controller looks for recessive to dominant edges that are used to adjust the sampling point in relation to this edge. When the CAN-bus is idle, this edge causes a hard synchronization of the sampling and the sampling is performed at 11 TQ from this edge, with a bit-configuration as shown in figure 7.

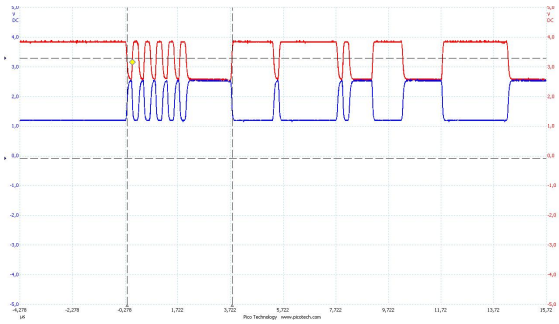


Figure 8: the nominal bit definition

In the rest of the message, the Stuff-bits ensure that there will be at least one such recessive to dominant edge every 10-bit to readjust the sampling point. If the receiver clock is running too slow or too fast, it must be ensured that the sampling point hasn't moved by more than 3 TQ (shown in green, in figure 7). When the dominant edge is detected, the bit is adjusted by the number of TQ, as configured in the SJW "Synch Jump Width" register.

If the edges are 2 TQ too early (in Phase segment 2), this segment will be 2 TQ shorter in this bit. If the edges are 2 TQ late (in the propagation segment) then Phase segment 1 will be 2 TQ longer in the following bit time. All other edges will be ignored and considered as noise.
With this knowledge, it was possible to put high-speed data in the propagation segment that would be treated as noise by Classic-CAN controllers but could be received and collected by other controllers with the means to interpret the additional information. This solution is just an extension of Classic-CAN that effectively increases data-rate without making any changes to the Classic CAN-controller hardware. It is even forward compatible with CAN FD because the CAN FD controller will interpret this as a Classic-CAN frame with a noise problem that will be filtered away by the robust CAN-protocol. The name given to this solution is CAN-EF "Enhanced Format"

CAN-EF performance

It is clear that CAN-EF will not be as efficient as CAN-FD because the CAN-EF protocol needs to use a large number of TQ to protect the Classic-CAN sampling point. The simplest solution is to just add one extra bit in the propagation segment, which shouldn't be problematic when running below 1 Mbit/s. One bit may seem marginal, but it still represents an increase of almost 100% in the CAN-EF communication – effectively, you double the performance at no cost. In this case, the performance is not too far removed from CAN-FD, with a data-rate twice the nominal-rate, except with a CRC that is twice as long in CAN-EF as that of CAN FD.

CAN-EF will also need some overhead bits that will limit the number of extra bytes to less than double.
Another example is a bus running at 250 kBit/s with a bit as described in figure 7. In this case, the propagation segment is 7 TQ (62,5 ns) 437,5 ns long and if we put 3 bits in this part, it will result in a CAN-EF bit-rate of less than 2 Mbit/s. With this solution you can increase the CAN-EF data-rate by 4 times, but still running the Classic-CAN unmodified at 250 kBit/s. The arbitration is a pretty complicated condition and if the system can handle that event at 250 kBit/s, it will most probably be possible to send data at 1 Mbit/s. To increase the nominal bit-rate beyond 250 kBit/s may be impossible due to cable length. If you can run CAN-EF at 2 Mbit/s, it is also possible to run CAN FD at this bit-rate. The propagation segment does not limit CAN FD and in this case, the data-rate is almost twice CAN-EF's performance. The justification for using CAN-EF is its ability to use already available modules that are not compatible with CAN FD. With CAN-EF it will be possible to increase the data-rate at least 4 times for every module that is replaced with CAN-EF. When all modules are replaced it will be possible to make the next performance step by enabling CAN FD and make a last doubling of performance. Of course, is it not necessary to make one big step from 250 kBit/s to 2 Mbit/s. It could be done in smaller steps where you slowly adopt a higher bit-rate and possibly adjust filters and cable layout to handle an increased bit-rate, gradually. By this point, you will be well prepared for the final step to CAN FD when all modules are replaced with CAN FD functionality.
The rule of thumb is that the space for CAN-EF bits is between 40 and 80% of the nominal bit. The lower value, 40%, is when there is a big variation in clock speed and a very short propagation segment. How many CAN-EF bits fit into the propagation segment depends on how high a bit-rate your CAN-driver and cable layout can handle.

The CAN-EF limitation

The major challenge to using CAN-EF is CAN's inherent robustness. Its robustness makes it possible to get a working system with almost any setting of the sample point and SJW, as long as the bit-length is correct. This has resulted in a system with modules that can have almost any bit configuration. All will work well as long as there is not too much noise or other disturbance in the system. However, as soon as you have poor cabling or electrical noise, you can expect to encounter problems that will be hard to find because they are hidden in the bit-handling, internal to the different CAN-controllers connected to the CAN-bus. For Classic-CAN controllers, the CAN-EF communication is just noise even if the noise is generated by the CAN-EF protocol. If Classic-CAN does not have the correct bit or clock settings outside the specification, problems will occur when too much of the propagation segment is used with CAN-EF bits. In other words CAN-EF communication is a perfect tool to check how robust your Classic-CAN network is against disturbance.
To use CAN-EF efficiently, it is necessary to ensure that there are optimal settings in all Classic-CAN modules. This ensures a more robust Classic-CAN system and a maximum number of bits in the CAN-EF frames at the same time.
It will of course be possible to measure how much of the propagation segment is available for CAN-EF and only utilize that for that purpose, but that will leave the rest of the CAN-system as weak as it was without CAN-EF.
As with all CAN-systems, one module with a bad bit-timing setting or bad clock is enough to compromise the performance of the system. It is the weakest link in the chain that sets the limitation of the entire system.

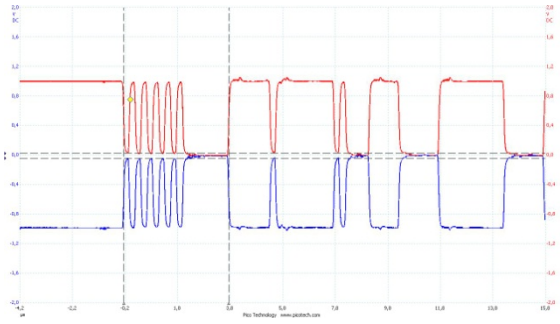


Figure 9: 250 kBit/s nominal bits with 10 EF bits at 4 MBit/s in the propagation segment

Figure 8 is an oscilloscope snapshot where the CAN-EF frame has been sent over 42 meter twisted pair with an impedance of 100 Ohm and with a 120 Ohm resistor at each end of the cable.

The physical layer

When increasing the bit-rate the mayor problem will be the cable layout. To make a high speed transmission line is basically very simple because the only demand is to keep the same impedance from the source of the signal to the receiver. The actual impedance can have any value and if you take two typical wires used in a car and twist them you will get 120 Ohm and if the wires are thinner like in an EtherNet CAT5 cable you will get 100 Ohm. Any of them can be used for the CAN-bus, but you will get problem if you mix them in the same bus-line. Even if you manage to use the very same cable in the CAN-bus it will be necessary to end the cable in a connector that fits to the CAN-module. Even if the RJ45 connector, used for a standard EtherNet cables at 10/100/1000 Mbit/s, looks simple it is carefully designed to keep the 100 Ohm impedance when connected to your computer or hub. In CAN and CAN-FD it is little simpler, because there is a relation between the bit-rate and how much impedance miss-match that can be accepted without disturbing the signal.
The right most column, in table 2, is a rough estimation how much of your CAN-bus cable that can be of different type with different impedance. A drop line could be compared with replacing a segment of the cable with cable segment with the same length and with 50% of the impedance value.

Table 2: Relation between

Bit-rate (Mbit/s)	Bit-length (ns)	rise-fall time (ns)	Rise length (m)	Acceptable impedance miss-match length (m)
1000	1	0.1	0.02	0.01
100	10	1	0.2	0.1
10	100	10	2	1
1	1000	50	10	5
0.5	2000	100	20	10
0.125	8000	400	80	40

The physics is the same as sound waves passing through an iron bar connected to an aluminum bar where some of the wave will bounce back at the change in material. If you have a cable with the same impedance it will continue forever, but for every miss match will you get some reflections and the amount of reflections will depend on the scale of the miss-match. Even if the miss-match is small they will all add up and make the signal to disturbed to be useful.

As can be seen from the table it is necessary to keep the impedance miss-match in EtherNet within centimeters compared to meters for a typical Classic-CAN. When we increase the bit-rate we will be more and more sensitive to impedance miss-match in the bus layout. If you have several drop lines or cable segments with different impedance, you have to add them all together and check that they are less than the value in the rightmost column to know that they will not give problem in the communication.
Classic-CAN is very insensitive and it will be possible to have much larger numbers compared to the table above before there are Error-frames in the communication. This can be observed by connecting an oscilloscope to the CAN-bus and if you see a signal with a lot of ringing and disturbed edges then you have impedance miss-match in the bus-layout.

Twisted wires

To get constant impedance in a CAN-bus it is not necessary to twist the two wires CANH and CANL, but there are three reasons to twist the two wires. The first two are to protect the communication wire for electromagnetic radiation. If the two wires are kept together within say 1 mm. It will be necessary to have an electrical field at 1000 Volt/meter to get a 1 Volt difference between CANH and CAN-low and that is not enough to make any problem in a CAN-communication. The second reason is for magnetic field protection. A magnetic field will induce a current in the CAN-wires and the amount of induced current depends on the strength of the magnetic field multiplied with the area between the two wires. First of all will the twist reduce the exposed area and secondly will the twist change the polarity of the magnetic field and by that reverse the induced current in every pair of twist. This is the two reasons why differential CAN-communication is so much more robust compared to K-line, LIN and SWC (Single Wire CAN) even at low bit-rates.

At higher bit-rate there is a third reason to have a twisted pair and that is to get constant impedance in the communication wire. The Impedance is given by the cross section, the electrical parameters in the conductor and the isolation around the conductor. If you keep those parameters fixed you will also get a fixed impedance. To keep the cross section you could glue the two wires together or keep them together by twisting them. The gluing is more difficult and will not protect the wires from the magnetic disturbance so the twisting is the best solution. To put the twisted pair in a shroud will protect the wires from being untwisted and make them less sensitive to changes in the surrounding.

This text is too limited to describe all the details around impedance. I have found that the book Signal integrity-simplified describes the details in such ways that most engineers would grasp the important part. The book “High Speed Signal Propagation” describes better the cable problem, but is little more difficult to understand.

Kent Lennartsson
Kvaser AB
Aminogatan 25 A
SE-43153 Mölndal
Tel.: +46 31 886344
Fax: +46 31 886343
kent@kvaser.com
www.kvaser.com

Jonas Olsson
Kvaser AB
Aminogatan 25 A
SE-43153 Mölndal
Tel.: +46 31 886344
Fax: +46 31 886343
kent@kvaser.com
www.kvaser.com

References

[1] ISO-11898-1:2006
[2] ISO-11898-1:201x working draft
[3] “Signal integrity-simplified” by Eric Bogatin ISBM 0-13-066946-6
[4] “High Speed Signal Propagation, advanced black magic”, by Howard Johnsson and Martin Green.