**CAN BUS PROTOCOL**

1. CAN Protocol Basic Properties
2. Bus Termination
3. Wired AND Logic
4. Bit stuffing
5. Arbitration prosses
6. Physical and Data Link Layer
7. Standard Frame Format
8. Extended Frame Format
9. Frame Types
10. ERROR Frame
11. Overload and Interspace Frame
12. Bit/Stuff/CRC/ Form/ ACK error
13. Synchronization
14. Resynchronization
15. Bit Time and Bit Construction
16. CAN Bit segments (SYNC, PROP, PH\_SEG1, PH\_SEG2)
17. Edge phase error adjustment
18. Bit Lengthening
19. Bit Shortening
20. CAN Controllers
21. Fault confinement, TEC & REC, need of fault confinement
22. CAN Controller State (Error Active/Passive, Buss off)

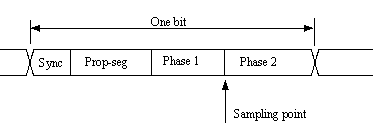
**CAN Bit Timing**

The Layout of a Bit

Each bit on the CAN bus is, for timing purposes, divided into at least 4 quanta. The quanta are logically divided into four groups or segments –

* the Synchronization Segment
* the Propagation Segment
* the Phase Segment 1
* the Phase Segment 2

Here is a picture of a CAN data bit:

[](https://www.kvaser.com/wp-content/uploads/2014/01/bit-timing-1.gif)

### Bit Timing Calculator

Use the calculator to calculate all possible sets of CAN bus parameters for a given input frequency and a given bus speed.

The Synchronization Segment, which always is one quantum long, is used for synchronization of the clocks. A bit edge is expected to take place here when the data changes on the bus.

The Propagation Segment is needed to compensate for the delay in the bus lines.

The Phase Segments may be shortened (Phase Segment 1) or lengthened (Phase Segment 2) if necessary to keep the clocks in sync.

The bus levels are sampled at the border between Phase Segment 1 and Phase Segment 2.

Most CAN controllers also provide an option to sample three times during a bit. In this case, the sampling occurs on the borders of the two quanta that precedes the sampling point, and the result is subject to majority decoding (at least this is the case for the 82527.)

Clock Synchronization

In order to adjust the on-chip bus clock, the CAN controller may shorten or prolong the length of a bit by an integral number of quanta. The maximum value of these bit time adjustments are termed the Synchronization Jump Width, SJW.

Hard synchronizationoccurs on the recessive-to-dominant transition of the start bit. The bit time is restarted from that edge.

Resynchronizationoccurs when a bit edge doesn’t occur within the Synchronization Segment in a message. One of the Phase Segments are shortened or lengthened with an amount that depends on the phase error in the signal; the maximum amount that may be used is determined by the Synchronization Jump Width parameter.

Bit Timing Register Calculation

Most CAN controllers allows the programmer to set the bit timing using the following parameters:

* A clock prescaler value
* The number of quanta before the sampling point
* The number of quanta after the sampling point
* The number of quanta in the *Synchronization Jump Width*, SJW

Usually two registers are provided for this purpose: btr0 and btr1. Things tend to vary slightly between different controllers, however, so read your data sheets carefully.

On the 82c200 and SJA1000, both from NXP (nee Philips), the register layout is like this:

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| btr0 | SJW1 | SJW0 | BRP5 | BRP4 | BRP3 | BRP2 | BRP1 | BRP0 |
| btr1 | SAM | TSEG22 | TSEG21 | TSEG20 | TSEG13 | TSEG12 | TSEG11 | TSEG10 |

* BRP0..BRP5 sets the clock prescaler value
* SJW0..SJW1 sets the length of the SJW
* TSEG10..TSEG13 sets the number of quanta before the sampling point (the start bit is not included)
* TSEG20..TSEG22 sets the number of quanta after the sampling point.
* SAM is set to 1 if three samples is to be obtained and to 0 if one sample is enough.

*Note: the actual value of these parameters is one more than the value written into the register.*

Example: if the oscillator signal fed to the SJA1000 is 16 MHz, and we want a bit rate of 250 kbit/s, with a sampling point close to 62% of the whole bit, and a SJW of 2 quanta, we can set –

BRP = 4, which gives a quantum length of 2 \* 4 / 16000000 s = 500 ns (2 is the external clock divider, see SJA1000 datasheet for more information)  
TSEG1 = 5, which gives 5 quanta before the sampling point, and  
TSEG2 = 3, which gives 3 quanta after the sampling point.

Each bit will then comprise 5 + 3 = 8 quanta, which results in the desired bit rate of 1 / (8 \* 500 ns) = 250 kbit/s. The register values should then be

|  |  |
| --- | --- |
| btr0 = | (SJW – 1) \* 64 + (BRP -1) = (2-1)\*64 + (4-1) = 67 = 0x43 |
| btr1 = | SAM \* 128 + (TSEG2 – 1)\* 16 + (TSEG1 – 1) = 0\*128 + (3-1)\*16 + (4-1) = *(“4” because the start bit isn’t included)* 35 = 0x23 |

The sampling point is at 5/8 = 62.5% of a bit.

**CAN Bus Error Handling**

How CAN Handles Errors

Error handling is built into in the CAN protocol and is of great importance for the performance of a CAN system. The error handling aims at detecting errors in messages appearing on the CAN bus, so that the transmitter can retransmit an erroneous message. Every CAN controller along a bus will try to detect errors within a message. If an error is found, the discovering node will transmit an Error Flag, thus destroying the bus traffic. The other nodes will detect the error caused by the Error Flag (if they haven’t already detected the original error) and take appropriate action, i.e. discard the current message.

Each node maintains two error counters: the Transmit Error Counter and the Receive Error Counter. There are several rules governing how these counters are incremented and/or decremented. In essence, a transmitter detecting a fault increments its Transmit Error Counter faster than the listening nodes will increment their Receive Error Counter. This is because there is a good chance that it is the transmitter who is at fault! When any Error Counter raises over a certain value, the node will first become “error passive”, that is, it will not actively destroy the bus traffic when it detects an error, and then “bus off”, which means that the node doesn’t participate in the bus traffic at all.

Using the error counters, a CAN node can not only detect faults but also perform error confinement.

Error Detection Mechanisms

The CAN protocol defines no less than five different ways of detecting errors. Two of these works at the bit level, and the other three at the message level.

1. Bit Monitoring.
2. Bit Stuffing.
3. Frame Check.
4. Acknowledgement Check.
5. Cyclic Redundancy Check.

Bit Monitoring

Each transmitter on the CAN bus monitors (i.e. reads back) the transmitted signal level. If the bit level actually read differs from the one transmitted, a Bit Error is signaled. (No bit error is raised during the arbitration process.)

Bit Stuffing

When five consecutive bits of the same level have been transmitted by a node, it will add a sixth bit of the opposite level to the outgoing bit stream. The receivers will remove this extra bit. This is done to avoid excessive DC components on the bus, but it also gives the receivers an extra opportunity to detect errors: if more than five consecutive bits of the same level occurs on the bus, a Stuff Error is signaled.

Frame check

Some parts of the CAN message have a fixed format, i.e. the standard defines exactly what levels must occur and when. (Those parts are the CRC Delimiter, ACK Delimiter, End of Frame, and also the Intermission, but there are some extra special error checking rules for that.) If a CAN controller detects an invalid value in one of these fixed fields, a Form Error is signaled.

Acknowledgement Check

All nodes on the bus that correctly receives a message (regardless of their being “interested” of its contents or not) are expected to send a dominant level in the so-called Acknowledgement Slot in the message. The transmitter will transmit a recessive level here. If the transmitter can’t detect a dominant level in the ACK slot, an Acknowledgement Error is signaled.

Cyclic Redundancy Check

Each message features a 15-bit Cyclic Redundancy Checksum (CRC), and any node that detects a different CRC in the message than what it has calculated itself will signal an *CRC Error*.

Error Confinement Mechanisms

Every CAN controller along a bus will try to detect the errors outlined above within each message. If an error is found, the discovering node will transmit an Error Flag, thus destroying the bus traffic. The other nodes will detect the error caused by the Error Flag (if they haven’t already detected the original error) and take appropriate action, i.e. discard the current message.

Each node maintains two error counters: the Transmit Error Counter and the Receive Error Counter. There are several rules governing how these counters are incremented and/or decremented. In essence, a transmitter detecting a fault increments its Transmit Error Counter faster than the listening nodes will increment their Receive Error Counter. This is because there is a good chance that it is the transmitter who is at fault!

A node starts out in Error Active mode. When any one of the two Error Counters raises above 127, the node will enter a state known as Error Passive and when the Transmit Error Counter raises above 255, the node will enter the Bus Off state.

* An Error Active node will transmit Active Error Flags when it detects errors.
* An Error Passive node will transmit Passive Error Flags when it detects errors.
* A node which is Bus Off will not transmit anything on the bus at all.

The rules for increasing and decreasing the error counters are somewhat complex, but the principle is simple: transmit errors give 8 error points, and receive errors give 1 error point. Correctly transmitted and/or received messages causes the counter(s) to decrease.  
Example (slightly simplified): Let’s assume that node A on a bus has a bad day. Whenever A tries to transmit a message, it fails (for whatever reason). Each time this happens, it increases its Transmit Error Counter by 8 and transmits an Active Error Flag. Then it will attempt to retransmit the message.. and the same thing happens.

When the Transmit Error Counter raises above 127 (i.e. after 16 attempts), node A goes Error Passive. The difference is that it will now transmit Passive Error Flags on the bus. A Passive Error Flag comprises 6 recessive bits, and will not destroy other bus traffic – so the other nodes will not hear A complaining about bus errors. However, A continues to increase its Transmit Error Counter. When it raises above 255, node A finally gives in and goes Bus Off.

What does the other nodes think about node A? – For every active error flag that A transmitted, the other nodes will increase their Receive Error Counters by 1. By the time that A goes Bus Off, the other nodes will have a count in their Receive Error Counters that is well below the limit for Error Passive, i.e. 127. This count will decrease by one for every correctly received message. However, node A will stay bus off.

Most CAN controllers will provide status bits (and corresponding interrupts) for two states:

* “Error Warning” – one or both error counters are above 96
* Bus Off, as described above.

Some – but not all! – controllers also provide a bit for the Error Passive state. A few controllers also provide direct access to the error counters.

The CAN controller’s habit of automatically retransmitting messages when errors have occurred can be annoying at times. There is at least one controller on the market (the SJA1000 from Philips) that allows for full manual control of the error handling.

Bus Failure Modes

The ISO 11898 standard enumerates several failure modes of the CAN bus cable:

1. CAN\_H interrupted
2. CAN\_L interrupted
3. CAN\_H shorted to battery voltage
4. CAN\_L shorted to ground
5. CAN\_H shorted to ground
6. CAN\_L shorted to battery voltage
7. CAN\_L shorted to CAN\_H wire
8. CAN\_H and CAN\_L interrupted at the same location
9. Loss of connection to termination network

For failures 1-6 and 9, it is “recommended” that the bus survives with a reduced S/N ratio, and in case of failure 8, that the resulting subsystem survives. For failure 7, it is “optional” to survive with a reduced S/N ratio.

In practice, a CAN system using 82C250-type transceivers will not survive failures 1-7, and may or may not survive failures 8-9.

There are “fault-tolerant” drivers, like the TJA1053, that can handle all failures though. Normally you pay for this fault tolerance with a restricted maximum speed; for the TJA1053 it is 125 kbit/s.