CANbus - hardware description

B.Hallgren - CERN ATLAS DCS

CAN is one of the CERN recommended fieldbuses. It allows implementation of a distributed detector sensor and control system for the LHC experiments. A LHC experiment has for each sub detector has independent control system. The different sensors and control functions are grouped into nodes for each sub-detector. The nodes are connected with each other and with the local control station in a CAN network. The basic principles of CAN network communication are explained with examples in the following sections.

CONTENTS

1. Communication Modes and Data Exchange

When data are transmitted over a CAN network no individual nodes are addressed. Instead the message is assigned an identifier which uniquely identifies its data content. The identifier not only defines the message contents but also the message priority. If a node wishes to transmit information it simply passes the data and the identifier to the CAN controller and set the relevant transmit request. It is then up to the CAN controller to format the message contents and transmit the data in the form of a CAN frame. Once the node has gained access to the bus and is transmitting its message all other nodes become receivers. Having received the message correctly, these nodes then perform an acceptance test to determine if the data is relevant to that particular node.

Therefore, it is not only possible to perform communication on a peer to peer basis where a single node accepts the message but also to perform broadcast and synchronized communication whereby multiple nodes can accept the same message using only a single transmission. The ability to send data on an event basis means that bus load utilization can be kept to a minimal amount.

This concept has become known in the networking world as the producer/consumer mechanism whereby one node produces data on the bus for other nodes to consume. The difference with CAN over other fieldbus solutions is that this mechanism requires no interaction from a bus master or arbiter.

2. Message Format

A CAN message mainly consists of an identifier field and the data field (plus error, acknowledgement and CRC fields). The identifier field consists of 11 or 29 bits (CAN 2.0A and 2.0B respectively) and the data field consists of a maximum of 8 bytes. When a device transmits a CAN message it first transmits the identifier field followed by the data field. The identifier field determines which node gains access to the bus (arbitration) and which nodes receive the data. Individual nodes can be programmed to accept messages with specific

identifiers. A data transfer will occur if the identifier of the transmitted message matches the identifier of message which the node is configured to receive. Nodes that are not programmed with the same identifier as the transmitted CAN message will not receive the data. This is known as acceptance mask filtering and is normally performed by the CAN hardware. The RTR bit is used for a remote transmit request. With this bit set the CAN frame has an identifier with a data field with no data bytes. A node which contains the corresponding identifier will send a message with data bytes as in reply to this request.

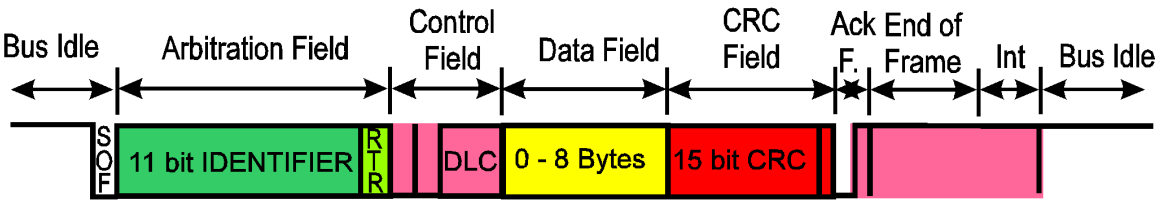


Figure 1 - CAN frame format (Standard frame CAN Spec 2.0A)

3.Arbitration

CAN employs the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) mechanism in order to arbitrate access to the bus. It uses a priority scheme using numerical identifiers in order to resolve collisions between two nodes wishing to transmit at the same time.

On the CAN bus a ‘zero’ (a dominant bit) on the bus is dominant and overwrites a ‘one’ (a recessive bit). Therefore, if a node is transmitting a ‘one’ whilst another transmits a zero will result in a ‘zero’ level on the bus (the one is overwritten). This process is shown in Figure 2.

When two or more nodes wish to transmit they sense the bus and if there is no bus activity they begin to transmit their message identifier most significant bit first. At the same time that they transmit their identifiers they also monitor the bus levels. If one node transmits a recessive bit on the bus and the other transmits a dominant bit the resulting bus level is a dominant bit. Therefore, the node transmitting a recessive bit will see a dominant bit on the bus (situation where B loose in Figure 2) and stop transmitting any further information. This allows the node with the lowest number in its identifier field to gain access to the bus and transmit its message. Any node that has lost during the arbitration process then waits until the bus becomes free before trying to re-transmit its message.

Note that this scheme means that no bandwidth is wasted during the arbitration process. Ethernet (for example) also uses CSMA/CD but if there is a collision between two nodes, one node will transmit a jamming signal causing both nodes to back off the bus. Both nodes will then wait a random period of time before trying to re-transmit. The bus arbitration process used by CAN means that the node with the highest priority (lowest value in the identifier field) will continue to transmit without having to back off the bus. This gives CAN very predictable behaviour (no random waiting) and very efficient bandwidth use of the bus. In fact it is possible to have CAN networks operating at near 100% bus bandwidth.

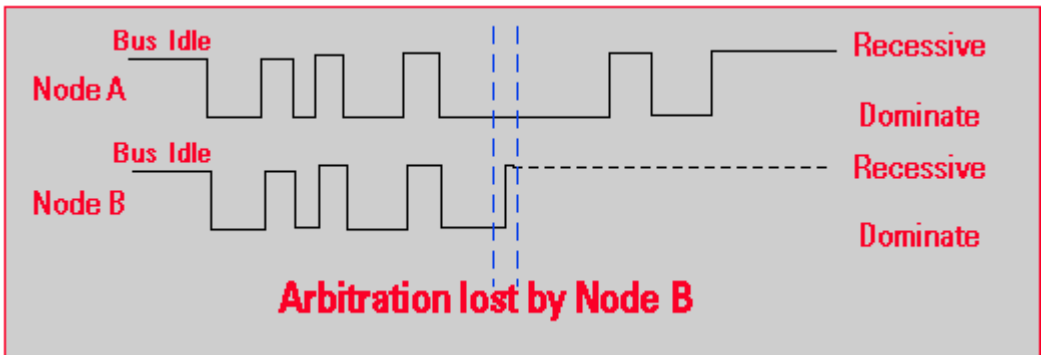


Figure 2 - CAN Bus Arbitration (non-destructive Bitwise Arbitration)

4. Message Acknowledgement and Error Checking/Signaling Mechanisms

Unlike other bus systems CAN does not use acknowledgement messages which waste bandwidth on the bus. As mentioned in section 2.3, each receiver that receives the message correctly acknowledges the message by transmitting a dominant bit in the ACK slot thus notifying the transmitter that the message was received correctly by at least one node. All nodes check each frame for errors and any node in the system that detects an error actively signals this to the transmitter. This means that CAN has network wide data security as a transmitted frame is checked for errors by ALL nodes irregardless of any filtering of the CAN telegrams.

The error checking mechanisms are:

Bit Errors - When a transmitter transmits a bit on the bus it also monitors the bus to determine whether the actual bit level on the bus matches the intended one.

Bit Stuffing Errors - Bit stuffing occurs when five bits are transmitted on the bus with the same polarity a bit of the opposite polarity is inserted into the bit stream. CAN uses bit stuffing for two purposes. The first is to provide edges to allow receivers to re-synchronise and adjust internal timing accordingly. The second is as an error checking mechanism whereby a violation of the bit stuffing rule is deemed an error, for example, six consecutive recessive bits.

Cyclic Redundancy Check (CRC) - An incoming telegram is checked using a 15-bit CRC check. A 15-bit CRC check is calculated by both the transmitter and the receiver. The transmitter transmits its CRC as part of the frame and this is compared with the receivers own independent CRC calculation. If the two calculations do not agree an error has occurred during transmission of the frame.

Form Errors - The incoming CAN frame is checked by the receiver to make sure that the size in bits of individual parts of the frame are as expected i.e. there are no extra illegal bits in a field of the frame.

Acknowledgement Errors - As mentioned earlier frames are acknowledged by inserting an ACK bit into the ACK slot of the frame. If no acknowledgement is received by the transmitter of the message this may mean that there is a transmission error which has been detected by the recipients, the ACK slot has been corrupted or that there are no receivers.

If an error is detected by ANY of the other nodes (irregardless of whether the message was

meant for it or not) the current transmission is aborted by transmission of an active error frame from at least one node. An active error frame consists of six consecutive dominant bits and prevents other nodes from accepting the erroneous message. The active error frame violates bit stuffing and may also corrupt the fixed form of the frame causing other nodes to transmit their own active error frames. After an active error frame, the transmitting node begins re-ransmission of the frame automatically within 23 bit periods or in some cases 31 bit periods.

CAN controllers have their own transmit and receive error counters which register errors during transmission and reception respectively. These counters are implemented in hardware and incremented by eight every time a frame is found to be erroneous and decremented by one every time a message is transmitted or received correctly. Over a period of time the error count may increase even if there are fewer corrupted frames than uncorrupted ones.

During normal operation the CAN controller is in its ***error-active state***. In this state, the node is able to transmit an active error frame every time a CAN frame found to be corrupt. If one of the error counters reaches a warning limit of 96 error counts (indicating significant accumulation of errors) this is signalled by the controller usually using an interrupt. The controller operates in its error active mode until a limit of 127 error counts has been exceeded.

Once 128 error counts has been reached, the CAN controller enters an ***error-passive state***. In this state, an error-passive controller is still able to transmit and receive messages but signals errors by transmitting a passive error frame. A passive error frame consists of six recessive bits and this frame may be ignored or overwritten by other CAN controllers. If the error count drops below 128 again the controller then becomes error-active again transmitting active error frames as required.

If the error count reaches or exceeds a limit of 256, the controller enters its ***Bus-OFF*** state. In this state the controller can no longer transmit or receive messages until it has been reset by the host processor resetting its hardware counters back to zero.

5. CANbus speed and lengths

A CAN node is capable of transmitting at a maximum speed of 1 Mbits/s. This speed is given by the method of processing of the bit pattern by the CAN controller. All the nodes on the CAN bus must have sufficient time to sample and decode the bus signal at the same bit time. Therefor allowance has to be given for the propagation delays of the cable and transmitter. The Table 1 below shows the relation between the bitrate and the cable length. The number of nodes per cable is limited by the transceiver hardware to 110.

Bitrate	Cable length
10 kbits/s	6.7 km
20 kbits/s	3.3 km
50 kbits/s	1.3 km
125 kbits/s	530 m
250 kbits/s	270 m
500 kbits/s	130 m
1 Mbits/s	40 m

Table 1 Recommended bitrates and cable lengths

The length of a CAN message varies depending on the number of data bytes transmitted but also on the contents of the message. This is due to the bit stuffing method described above. Table 2 shows the effective length of CAN message as a function of the data field length.

Number of data bytes	0	1	2	3	4	5	6	7	8
Minimum message length	44	52	60	68	76	84	92	100	108
Maximum message length	51	60	70	80	89	99	108	118	128

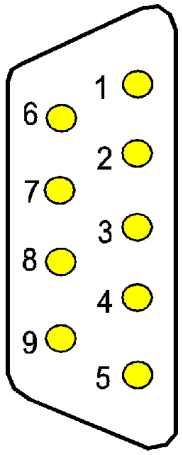
Table 2 Effective message lengths (in bits)

5.1 Example of CAN message transfer times

The effective speed of a data transfer of the CAN bus can be illustrated by the following example: 64 kbytes of a flash memory of a microcontroller is to be down loaded in remote CAN node. The distance from the control room to the microcontroller is estimated to 150 m. From Table 1 is obtained the recommended CAN bus bitrate of 250 kbits/s. A simple CAN message format is used with the data field divided into 2 bytes for the address of the flash memory and 1 byte for the data. This gives in total 3 bytes per message. The worst case message time is therefore 80 time 4 usec or 320 usec. The total transfer time is 21 seconds assuming neglible other data traffic. If a more compact data field format is used, with 2 data bytes as address and 6 bytes for data, the Table 2 gives 128 times 4 usec or 512 usec time per 6 flash memory bytes. In total the transfer time is then 5.6 seconds. It should be mentioned that the above example is not the typical use for the CAN bus. It is more suitable to send data on an event basis eg. on error conditions.

6 Can bus connector

The recommended CAN bus connector is shown in Figure 3. The CERN ATLAS DCS Local Monitor Box can be powered via the CAN bus connector. It needs two power supplies one analogue and and digital as shown in Figure 3.



pin	CiA	ATD-DCS
1	reserved	RESET
2	CAN_L	CAN_L
3	CAN_GND	CAN_GND
4	reserved	AGND
5	CAN_SHLD	CAN_SHLD
6	GND	DGND
7	CAN_H	CAN_H
8	reserved	+VA
9	CAN_V+	+VD

Figure 3 The CAN bus connector

7 CERN Can bus implementation

Operation in the radiation environment at the LHC detector cavern requires radiation tolerant CAN controller. The magnetic field requires special considerations and low power and cost are very demanding specifications which have been considered in the ATLAS DCS Local Monitoring Box.