C *omputing*
S *urface*

Overview of the Control Area Network (CAN)

S1002–10M140.00

meiko

Meiko's address in the US is:

Meiko
130 Baker Avenue
Concord MA01742

508 371 0088
Fax: 508 371 7516

Meiko's address in the UK is:

Meiko Limited
650 Aztec West
Bristol
BS12 4SD

Tel: 01454 616171
Fax: 01454 618188

Issue Status:

Draft	
Preliminary	
Release	x
Obsolete	

Circulation Control: *External*

Contents

1. The Control Area Network 1

Introduction 1
Network Hierarchy 1
CAN Messages 2
Network Protocol 3
Prioritisation 4
Network Error Detection and Recovery 4
Example — Snooping the CAN 5
Appendix A — Packet Format 7
Header Data 7
Address Data 8

The Control Area Network 1

Introduction

The Control Area Network (CAN) is a low bandwidth serial network. It is used by the CS-2 to carry control, diagnostic, and remote console traffic between processors. The CAN is independent of the CS-2 data network and does not therefore impact on its performance.

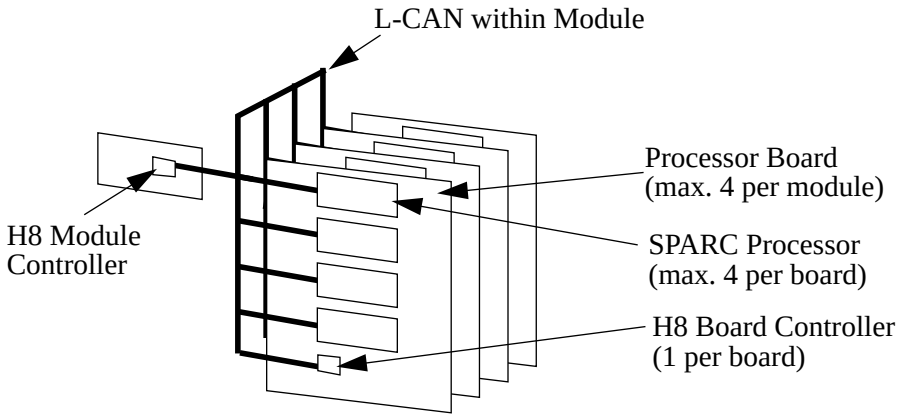
An understanding of the CAN is not required for normal operation of the CS-2. It is typically used by the resource management system to maintain the machine database, and by Pandora to create remote console connections and to gather component operating status. The information in this document is therefore provided for information only.

Network Hierarchy

The CAN is hierarchical, with the number of nodes on each network limited (by the electrical characteristics of the CAN transceivers) to around 30 nodes.

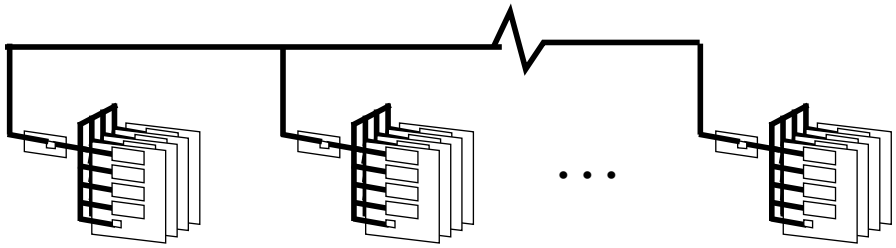
At the lowest level is the L-CAN, a network interconnecting the processors in each module. This connects up to 16 SPARC processors (allowing for four boards each with four SPARC processors), 4 board controller processors (H8 processors), and 1 module controller (also an H8 processor). The interface between the processor and the CAN is handled in each case by dedicated CAN transceiver devices.

Figure 1-1 L-CAN Connections within a Processor Module



The modules within a Cluster (3 bays, up to 24 modules) are interconnected by the X-CAN, and the interconnection of Clusters is via the G-CAN. The transfer of network traffic from the L-CAN to the X-CAN is handled by each module's controller, whereas the transfer from X-CAN to G-CAN is via nominated routers (selectable from Pandora).

Figure 1-2 X-CAN Connections within a Cluster



CAN Messages

A node requests status information from, or sends control requests to, another CAN node by sending a message to it. The information or function that is required by the sender is specified by addressing an *object* at the recipient. Objects are either hardware devices or software functions, the mapping from an object id

to device or function being performed at the recipient. The operation that is to be performed on the object, such as a read or write, and any associated object-specific data is also included in the CAN message.

The CAN packet consists of up to 10 bytes, of which 2 bytes are the packet header, 4 bytes are the object address, and 4 bytes are the optional object-specific data. The header defines the source and destination nodes for the packet; the source and destination will always be on the same L-CAN. The object address define the absolute address (in terms of cluster, module, node, and object) of the target object, and also define the operation that is to be performed on that object (e.g. a read, write, or a signal). Absolute addressing of the object is required by the routing software when the source and destination nodes are on different L-CAN networks; in this case the source node sends a message to its module controller, and the software running on the module controller determines from the full object address the routing of the message, via the network hierarchy, to the remote node.

Network Protocol

The CAN uses a master/slave protocol in which most SPARC processors are configured as slaves, and most board and module controllers (i.e the H8 processors) are configured as masters.

Masters have the capability to read from and write to objects over the CAN, and thus have the capability to control and reconfigure the machine. Slaves can signal to masters that they have information for their attention by sending a signal message. A slave can be given the capability of a master by another master for specific purposes, for example to enable a SPARC host to read/write the console object of another SPARC. By assigning the role of masters to the H8 processors, which are themselves controlled by firmware programme code, interference of the CAN by user processes is prevented.

By default the masters (the H8 controllers) receive all messages from the CAN and use filters on the CAN controllers to extract the messages appropriate to each processor; this allows a broadcast capability to be defined for the module and board controllers. Slave processors (the SPARCs) are configured only to receive messages that are explicitly targeted at them, which reduces CAN workload on the SPARC processors.

Prioritisation

Message priorities are used at two stages; during arbitration between CAN devices on the L-CAN, and during routing between network levels by a routing processor. The two stages of prioritisation are represented by the two priority bits in the CAN message packet; the header defines the priority between CAN devices (either 0 for high priority, or 1 for low), and address data specifies the prioritisation for routed messages. A high priority message might indicate a power supply failure, whereas less urgent messages (switch errors) are recorded by low priority messages. During routing the priority in the message's address field is used to reduce congestion at the routers and on the X-CAN/G-CAN networks, and may be modified by the router processors to give highest priority to message responses and those making their way down the CAN hierarchy. Note that a low priority message sent to a node can be overwritten by a high priority message, causing the low priority message to be scrapped.

Network Error Detection and Recovery

Packets are acknowledged (ACK'ed) if they reach their destination and are interpreted correctly. A not-acknowledge (NACK) is sent if they fail to be correctly interpreted at their destination.

Reasons for failure are:

- Bad message. Perhaps the sender attempted to write to a read-only object, or an object that doesn't exist.
- Hardware errors. Either the message or the acknowledgement failed.
- Hardware overruns. No spare input buffer at the transceiver.

Bad messages, or messages to non-existent objects, are signalled to the sender by the return of a not-acknowledge packet.

Hardware errors are detected using a timeout at the message sender. The expiry of the timeout period indicates that the message and/or its acknowledgement failed. The behaviour of the sender in this case is function specific, but may be to resend the message or to give-up.

Hardware overruns occur when a transceiver's two receive buffers are full and a message on the network is targeted at the device. The incoming message is ignored, and an interrupt is issued to the CAN device's controlling processor. The sender of the message detects the failed transmission by the absence of an acknowledgement within its timeout period.

The CAN transceivers maintain a count of input and output errors using two counters. The transceivers are initialised with two threshold values called the *warning* limit and the *bus-off* limit at which the number of errors becomes serious and requires attention. When a count reaches the warning limit an error flag is set and the attached processor is interrupted, but the transceiver continues to operate. On reaching the bus-off limit the transceiver goes into a permanent reset state and is no longer operational. The accumulation of not-acknowledge errors cannot force the transceiver into a bus-off state, to accommodate the case at system start-up where a transceiver may become operational in advance of its peers.

Example — Snooping the CAN

The `cansnoop`(1m) command can be used by the root user to monitor the activity on the local CAN. The following example output shows some of the CAN traffic that is generated when a console connection is created via the CAN with either Pandora or `cancon`(1m). The following example has been edited (for clarity) to remove the numerous heartbeat signals which are sent between nodes to signal their continuing operation.

In the following output the initial 2 digit field is a feature of `cansnoop`(1m) and represents a microsecond time stamp; subsequent fields represent the contents of the CAN packets. The second field identifies the source and destination CAN nodes (the packet header), the third field identifies the message type (WO= write, ACK = acknowledge, D = data), the next group of 4 fields is the full object address (with key object id's replaced by an ASCII mnemonic), and the remaining fields being the optional 4 bytes of data and their ASCII representation.

The output shows node 008 sending a packet to the console connection object at node 004; the data field identifies the initiator of the connection and the object that is to be used for subsequent communications. The connection request is acknowledged by node 004.

Following the initial handshaking characters typed at the keyboard are sent via the CAN to the remote console port. Each typed character is acknowledged by node 004 and echoed back to node 008. The connection is dropped by a write from node 008 to node 004's console disconnect object.

```
111.870 008->004  WO  00,02,04 CONSOLECONN 30 02 23 82 '0.#.'
111.897 004->008  ACK 00,02,04 03f0        ff ff ff ff '....'
122.335 008->004  D   00,02,04 0000        0d          '.'
122.335 004->008  ACK 00,02,04 0000
122.336 004->008  D   00,02,08 0382        0d 0a       '..'
122.336 008->004  ACK 00,02,08 0382
122.340 004->008  D   00,02,08 0382        6e 6f 76 61 'nova'
122.341 008->004  ACK 00,02,08 0382
122.341 004->008  D   00,02,08 0382        31 20 63 6f '1 co'
122.341 008->004  ACK 00,02,08 0382
122.341 004->008  D   00,02,08 0382        6e 73 6f 6c 'nsol'
122.341 008->004  ACK 00,02,08 0382
122.342 004->008  D   00,02,08 0382        65 20 6c 6f 'e lo'
122.342 008->004  ACK 00,02,08 0382
122.342 004->008  D   00,02,08 0382        67 69 6e 3a 'gin:'
122.342 008->004  ACK 00,02,08 0382


138.993 004->008  D   00,02,08 0382        36 5d 23 20 '6]# '
138.993 008->004  ACK 00,02,08 0382
138.993 004->008  D   00,02,08 0382        65 78 69 74 'exit'
138.993 008->004  ACK 00,02,08 0382
138.994 004->008  D   00,02,08 0382        0d 0a       '..'
138.994 008->004  ACK 00,02,08 0382
144.698 004->008  D   00,02,08 0382        0d 0d 0a    '...'
144.698 008->004  ACK 00,02,08 0382
144.702 004->008  D   00,02,08 0382        6e 6f 76 61 'nova'
144.702 008->004  ACK 00,02,08 0382
144.702 004->008  D   00,02,08 0382        31 20 63 6f '1 co'
144.702 008->004  ACK 00,02,08 0382
144.703 004->008  D   00,02,08 0382        6e 73 6f 6c 'nsol'
144.703 008->004  ACK 00,02,08 0382
144.703 004->008  D   00,02,08 0382        65 20 6c 6f 'e lo'
144.703 008->004  ACK 00,02,08 0382
144.704 004->008  D   00,02,08 0382        67 69 6e 3a 'gin:'
144.704 008->004  ACK 00,02,08 0382
144.704 004->008  D   00,02,08 0382        20          ' '
144.704 008->004  ACK 00,02,08 0382
149.177 008->004  WO  00,02,04 CONSOLEDISC 30 02 23 82 '0.#.'
149.203 004->008  ACK 00,02,04 03f1        30 02 23 82 '0.#.'
```

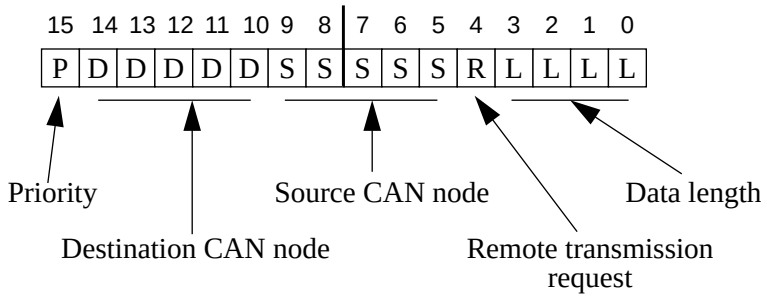
Appendix A — Packet Format

CAN packets are 10 bytes in length consisting of 2 bytes of header information, 4 bytes address data, and an optional 4 bytes of message data.

Header Data

The 2 byte message header identifies the source node, destination node, and message priority.

Figure 1-3 Message Header (2 bytes)



The fields in the packet header have the following meanings:

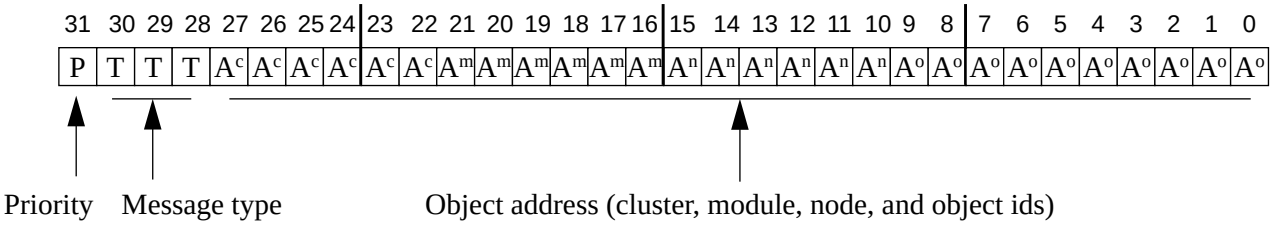
Bit(s)	Meaning
15	Message priority (for arbitration on the L-CAN). 0 is high, 1 is low.
14-10	Destination CAN node id. (in the range 0–29).
9-5	Source CAN node id. (in the range 0–29).
4	Remote transmission request (always 1 for the CS-2).
3-0	Length of following data. This will be either 4 or 8 for the CS-2; 4 bytes are required for the address data, and an optional 4 bytes for object-specific data.

Within a module CAN node ids are allocated as shown in Figure 1-5.

Address Data

The action performed at the destination node is specified by the address data. This defines the message type (Write to object, read from object etc.), and the address of the object that is to be targeted.

Figure 1-4 Object Addressing (4 bytes)



The fields in the address data have the following meanings:

Bit(s)	Meaning
31	Message priority for arbitration by the X-CAN/G-CAN routers. 0 is high, 1 is low.
30-28	Message type (see below).
27-0	The address of the object that the transaction is to apply to. This if a full machine address consisting of a 6 bit cluster id, 6 bit module id, a 6 bit node id, and a 10 bit object id. A broadcast is specified by 111110; 111111 means never route to this level. Object id's are listed in the header files `/usr/include/sys/cankobj.h` and `/opt/MEIKOcs2/etc/include/canio/canobj.h`.

Message types are:

Id	Meaning
000	Read request; the address identifies an object to be read. For use by master CAN nodes only (typically the H8 processors).
001	Write request; the address identifies the object to be written to. For use by master CAN nodes only (typically the H8 processors).
010	Write request without acknowledge from destination; the address identifies the object to be written to. For use by master CAN nodes only (typically the H8 processors).
011	Data.
100	Write acknowledge.
101	Unused.
110	Write a not-acknowledgement.
111	Send a signal. The data field defines the object that has changed (using the least significant 10bits of the 4byte data field); the address field includes the broadcast code (111110) in the node, and/or module and/or cluster fields. This is used by CAN slave nodes to notify CAN masters that an object has changed. At least one master should read the changed object. The signal is repeated at regular intervals (the timeout period) until at least one master queries the object. Signals are not directly acknowledged.

Figure 1-5 CAN Addresses within a Module