1998

Networking 8-bit embedded controllers, using the controller area network method

Brett Wilkinson
Edith Cowan University

Edith Cowan University

Copyright Warning

NETWORKING 8-BIT EMBEDDED CONTROLLERS, USING THE CONTROLLER AREA NETWORK METHOD

Investigating the 68HC11 microcontroller, Siemens and Intel CAN devices

A thesis submitted in partial fulfilment of the requirements
for the degree of

Bachelor of Engineering (Electronic Systems)

at

EDITH COWAN UNIVERSITY
FACULTY OF ELECTRONIC ENIGINEERING

September 1998

Brett Wilkinson BEng (Honours)

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

ABSTRACT

The initial aim of this project was to create an Ethernet (IEEE 802.3) Communications System. The system was to connect Embedded controllers to facilitate the operation of a real time operation. After extensive investigation and scheduled meetings with industry, it became apparent that a superior communication system for this type of task lay in the Controller Area Network (CAN) standard.

Communications Networks are extremely susceptible to volatile surroundings. A vital controller network necessitates that its communication network be invulnerable to high noise levels. It is also imperative that critical messages from one controller reach their destination on time. CAN was conceived by BOSCH as a solution to this dilemma.

The project undertaken was to develop a controller area network, whose purpose was to control a solar tracker in a remote fashion. The solar tracker developed needed to be able to work in an autonomous fashion via its own embedded controller, however be able to receive commands from a remote control unit. This remote unit was able to display the status and operation mode of the tracker, yet also be able to issue over ridding commands to the tracker in a real time sense.

A CAN network was created and interfaced to a MC68HC11 embedded controller. A two wire differential (RS 485) system was implemented as the physical CAN bus. Every aspect of interfacing the CAN (Intel 82527) device to the HC11 was investigated. Chip Select problems resulted in the simulation of the of the HC11 Address/Data bus using a MC68HC24 Port Replacement Unit (PRU), even the possibility of Serial Peripheral Interface (SPI) connection was considered before a hardware solution was developed

The driver software and low level communication system developed addresses all aspects of operation, from initialisation of the 82527, to dealing with the reception and transmission of various messages. Software to enable simultaneous network communications and solar tracking operation was completed for the solar tracking device. Every message object type was utilised within the system. The generation of interrupts to deal with the reception of critical messages, and other message prioritisation schemes were incorporated.

The resulting system demonstrated that the controller was able to drive a solar tracking panel and receive additional commands and issue status reports to a remote micro controller, in a real time situation. Such a system as this could have many solar trackers connected to the same bus and result in a cheap but reliable installation. Alternatively virtually any Industrial distributed automated process confined within a relatively close proximity could be developed by using a derivative of such a system described within this report.

DECLARATION

I certify that this thesis does not, to the best of my knowledge and belief:

i) incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education;

ii) contain any material previously published or written by another person except where due reference is made in the text; or

iii) contain any defamatory material.

BRETT WILKINSON 08/09/1998

ACKNOWLEDGEMENT

I would like to acknowledge the assistance of my supervisor Mr. Barry Kauler who has advised me throughout the thesis.

CONTENTS

CHAPTER ONE

SELECTION OF SERIAL COMMUNICATIONS SYSTEM

GENERALLY

The following communications systems were investigated in order to select the appropriate protocol for the system.

- HC11 SERIAL PERIPHERAL INTERFACE (SPI)
- HC11 Asynchronous Serial Communications Interface (SCI)
- Ethernet (IEEE802.3)
- I^2C
- CAN

A brief description and summary of the capabilities of each of these systems follows in this chapter.

The International Standards Organisation (ISO) communications standard is applicable to all the systems and is described hereunder

THE NETWORK LAYERS

The ISO have defined a communications standard called the ISO 7498. This standard is known as the Open Systems Interconnection (OSI) model.

The ISO reference model for a communications system is shown below:

APPLICATION LAYER
PRESENTATION LAYER
SESSION LAYER
TRANSPORT LAYER
NETWORK LAYER
LINK LAYER
PHYSICAL LAYER

APPLICATION LAYER

The function of the application layers is to perform the necessary conversions of programmes designed for a virtual system to be compatible with the real system. This layer usually interacts with application programs and files and therefore it contains a variety of commonly used protocols.

THE PRESENTATION LAYER

Performs the task of converting abstracted data structures from an internal representation to the network standard. This includes data forming and code conversion. Without this layer communications could not be achieved between nodes with different data representations systems.

SESSION LAYER

Once this layer establishes a session, it is responsible for the control, dialogue and synchronisation of the session.

TRANSPORT LAYER

The Transport Layer is responsible for the transportation of the information between nodes and ensuring that the data has been successfully received by the receiving node. As it is in direct communications with the Session Layer, it effectively isolates it from the physical hardware. The Transport layer is responsible for sending the messages in the intended order.

NETWORK LAYER

This layer is concerned with setting up of addresses, the routing of packets, and to control congestion and stop bottlenecks forming due to the presence of too many packets at the subnet.

DATA-LINK LAYER

The function of this layer is to deal with the data at the bit and byte level, in a way as to provide the Network Layer with an error free transmission line. The layer deals with issues such as flow control and duplication elimination.
(A duplication of a message frame can occur if an acknowledgement bit has

been destroyed).

" The physical layer provides the Data Link layer with bits. Now it is time to give these raw bits some meaning. At this point we no longer deal with bits but with *data frames*, packets containing data as well as control information."[1]

"The data-link layer - is the only layer that recognises and understands the format of messages. This layer constructs the messages to be sent to the Physical Layer, and decodes messages received from the physical layer." [1]

PHYSICAL LAYER

This layer deals with the hardware components of a channel and the transmission of the actual bits. The physical layer, by definition, is always the real hardware component of the communications system. All of the physical characteristics of the bus are defined and cover such issues as:

- The Signalling Scheme
  This covers the timing of the data transmissions on the BUS and the hardwiring rules for the establishment of the communications connection.

- Electrical Levels
  All the levels are defined by the voltage levels present on the BUS lines. The physical layer is responsible for the conversion of characters to and from actual electrical levels on the bus.

- Impedance of the BUS
  The physical layer must consider such things as the impedance of the BUS which affects both the receivers and transmitters. A specific impedance or an impedance range is usually specified.

HC11 SERIAL PERIPHERAL INTERFACE

MASTER / SLAVE NETWORK

The HC11 can communicate to other HC11's by a four wire synchronous SPI, which is a standard feature on the chip. The protocol is very basic and does not have all the many features that the CAN system provides.

The main function of the SPI is to allow communications to other peripheral devices or other microcontrollers and since the system consists of four wires, simultaneous transmission and reception of data is possible.

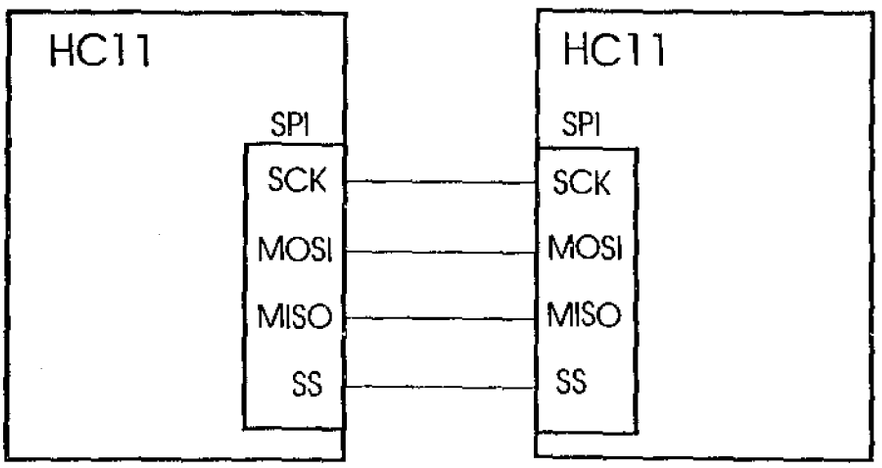
A basic error detection system can detect errors caused by:

- An endeavour to update the SERIAL shift register when a transfer is currently in progress.
- More than one microcontroller attempting to become a master at the same time.

The four lines of the SPI include:

- The Master Out Slave In (MOSI).
- The Master In Slave Out (MISO)
- The Slave Select (SS).
- The Synchronisation Serial Clock (SCK).

The connection of equipment via the SPI is straightforward. The connection of one HC11 to another by this interface is shown below:



HC11'S CONNECTED VIA SPI

The interface can be concisely defined as " a four-wire synchronous communication interface used for high-speed communication with specialised peripheral devices and other microcontrollers. Data is transmitted and received simultaneously: the Baud rate is software programmable."[2]

HC11 ASYNCHRONOUS SERIAL COMMUNICATIONS INTERFACE

The HC11 incorporates a full duplex asynchronous SCI. The following points highlight the characteristics of this communications system:

- It is a Universal Asynchronous Receiver Transmitter (UART).
- It is a full duplex system.
- The Non Return to Zero (NRZ) data format is used.
- Double buffering are used for both the Receiver and the Transmitter.

ETHERNET IEEE 802.3

GENERAL DESCRIPTION

The 802.3 (Ethernet) standard uses the system of Carrier-Sense Multiple Access with Collision Detection (CSMA/CD). If a node wishes to send a message, the protocol works in the following way "The Physical layer of the user's workstation model generates a signal. It listens to detect another carrier signal from another users who is about to send a message. If no other signal is detected, the first users message is sent."[1]

The above description shows how the CSMA portion of the protocol works. Now the Collision Detection is illustrated by the situation where 2 network nodes are located some distance apart :

"It is possible to for their network interface cards to issue a carrier-sense signal, listen and hear nothing, and then send their messages - only to have then collide".[1]

The Collision Detection system has been incorporated into the system to deal with this potentially catastrophic event :

"The two users' network interface cards listen for the other workstation to send a transmission and then transmit the message again."[1]

When a collision does occur a jam signal is sent across the network.

"Each workstation waits a different random amount of time after a data collision before once again transmitting a message."[1]

"After repeated collisions, the network will double its random delays before permitting stations to transmit once again."[1]

The standard specifies that the maximum length of a single cable in the network can be no longer than 1500 feet.

ETHERNET FRAME STRUCTURE

The structure of an Ethernet Frame is shown below:

PREAMBLE	DESTINATION ADDRESS	SOURCE ADDRESS	TYPE	DATA	FRAME CHECK SEQUENCE

The various fields of the Frame are:

- Preamble : This field consists of 8 bytes and is used for synchronisation
- Destination Address : "Can be a single workstation's address, a group of workstations, or even several groups of workstations."[1]
- Source Address : "Is critical so that the workstation receiving the message recognises where it came from"[1]
- Type : " Is important because there must be a way of designating which type of format the data is using. Without this information, it is impossible to decipher the packet when it arrives"[1]
- Data : "Is strictly limited; it can only hold a minimum of 46 bytes and a maximum of 1,500 bytes of information."[1]
- Frame - Check Sequence : This "field ensures that the data in the other fields arrives safely."[1]

NATIONAL SEMICONDUCTOR DP8390

The DP8390 was considered for it's use as an interface adaptor when Ethernet was being considered for use. The following is a brief description of the functionality of this chip:

"The DP 8390 network interface controller provides all the media access control layer functions required for transmission and reception of packets in accordance with the IEEE 802.3 CSMA/CD standard. the controller acts as an advanced peripheral and serves as a complete interface between the system and the network " [3]

"The NIC is used as a standard peripheral device and is controlled through an array of on chip registers."[3]

"The Command Register (CR) is used to initiate transmission and remote DMA operations. Upon packet reception, and of packet transmission, remote DMA completion or error conditions, an interrupt is generated to indicate that an action should be taken. The processors interrupt driven routine then reads the Interrupt Status Register (ISR) to determine what type of interrupt occurred, and performs the appropriate actions.

the NIC transmits packets in accordance with the CSMA/CD protocol, scheduling re-transmission of packets up to 15 times on collisions according to the truncated binary exponential back-off algorithm." [3]

INTER - INTEGRATED CIRCUIT (I2C) BUS

This protocol serves to provide a communication link between Integrated Circuits.

"The 12C bus was developed in the early 1980's by Philips semiconductors. Its purpose was to provide an easy way to connect a CPU to peripheral chips in a TV-set."[4]

"The I2C serial bus represents a flexible and simple interfacing standard for low distance applications such as on-board peripheral interfacing or inexpensive communications within a single device."[5]

This serial communications system utilises the following two lines:

- The Serial Data Line (SDA) and the
- Serial Clock Line (SCL)

"The protocol provides the following features:

- Master/slave communication
- multiple slaves and masters

- Slaves needs no external clock
- Stateless protocol, addressing and data exchange is done within one frame."[5]

"The I2C bus was originally developed as a multi-master bus. This means that more than one initiator can be active in the system"[6].

An arbitration process has been developed to cater for the situation of more than one Central Processing Unit (CPU) being present on the bus. This system uses Start and Stop conditions. Any CPU on the bus is free to transmit a message as long at the last message on the bus was not a STOP condition. When a CPU wishes to gain control of the bus, it will start by issuing a START, thereby preventing other nodes intervening. A wired-AND configuration of the individual nodes to the bus result in the effect that if one device pulls a line low, it stays low.

The arbitration process uses this fact by ensuring that a transmitting node listens to the bus as it is transmitting. If a master endeavours to change the state of a line to high, but monitors it as low, it is made aware that there has been a possible clash and 'backs-off' until the next stop condition is seen.

THE CAN SYSTEM

The CAN protocol has been specifically developed to run in real time environments. It is a high speed serial communications protocol capable of handling bit rates up to 1Mbit/sec. It is particularly applicable where a system of embedded microcontrollers is to be used in an interrupt driven real-time network.

"Without serial networking, inter-module communication requires dedicated, point-to-point wiring resulting in bulky, expensive, complex, and difficult to install wiring harnesses."[7]

A CAN serial data bus enables data messages to be received by different electronic processing modules and only those which have a process under control needing this data will receive the message.

CAN was originally designed by the automotive company Bosch for car communications. This stemmed from the ever increasing demand for instrumentation at the dash board for control of the greater number of electrical

components and monitoring of the sophisticated technology resulting from the design of the modern car.

The cost of, difficulty in installing, and sheer bulk of the dedicated wiring necessary, led to the development of the serial networking.

CAN APPLICATIONS

Because of CAN's wide flexibility it is beginning to emerge as an effective protocol for use in general applications in industry today and is being widely extended because of the following attributes:

- It is an International Standard
- It is capable of operating at a variety of speeds, lending its use for both high and low speed applications.
- It has proved itself to be extremely reliable and can withstand harsh environments.
- Its digital technology makes it compatible for use with computers and microprocessors.
- The components are physically small
- Its wide acceptance and use has made it readily available at a reasonable cost.

These have seen its application and use extended to the following:

1. Automotive Industry
2. Medical Equipment
3. Elevator and Crane control systems
4. Marine control Navigation Systems
5. Production line control systems
6. Office Equipment

"By 1999 it is projected that over 140 million CAN chips will have been sold"[7]

CAN IMPLEMENTATIONS

The two basic implementations of CAN are BASIC CAN and FULL CAN. These implementations are fully described in the following chapter under the heading of

"CAN TERMINOLOGY"

CAN AND IT'S RELATIONSHIP TO THE OSI MODEL

CAN is only relevant to three of the 7 layers of the OSI model. The function of many of the layers is fairly redundant, such as the Presentation Layer, because in CAN systems the data representations are consistent across the network from node to node. In fact the CAN ISO 11898 only defines the Physical and the Data-Link layers. This is not an unusual situation as few networks conform completely with the OSI model.

The Application Layer, Data Link Layer and the Physical Layer do however have a specific relevance to the CAN protocol and are discussed hereunder:

APPLICATION LAYER

From this level the user is able to interface the other lower levels. Several programs have been designed for use with this layer, those more accepted by the industry being:

- CAN Application Layer (CAL) developed by Philips Medical Systems.
- DeviceNet comes from Rockwell/Allen-Bradley and is in common use in the industrial type of applications
- Smart Distributed System (SDS) by Honeywell is developed for machine control applications

DATA LINK LAYER

The data link Layer is implemented in hardware into the Intel 82527 CAN chip. This layer is usually fairly complex and in CAN systems is divided into two parts.

- The Logical Link Control (LLC). This controls the transmission and reception of data messages between various layers within the (OSI) Model.
- The Media Access Control (MAC). This layer deals with the encoding and serialisation of data messages ready for transmission and the decoding of messages that have been received. The CAN arbitration system and

error detection (reception of Acknoledgement bit) is also handled here.

The Data Link layer however does not provide all of the functions that some CAN applications may require, and for this reason extra functions are often included in some application layer programs. These functions may include the ability to deal with the reception and transmission of data units larger than the eight bytes specified by the CAN ISO11898 standard.

PHYSICAL LAYER

The CAN system does not specify a physical standard, however many other commonly used components of other communication systems are incorporated into a typical CAN implementation. Several methods have been used for the physical layer, including a modified 2-wire 485 type of differential system. The ISO standards however define two systems which the CAN system commonly uses:

- The ISO 11898 is a two wire balanced signalling scheme. This system is defined as one having a nominal cable impedance of around 120 ohms. "The ISO 11898 prescribes that the cable impedance by nominally 120 Ohms; an impedance in the interval of (108 ..132) Ohms is permitted. There are not many cables in the market today that fulfil this requirement. There is a good chance that the allowed impedance interval will be broadened in the future"[8]
- The ISO 11519 two wire balanced signalling scheme for low bus speeds.

As with most serial communication systems the maximum speed with which the system can communicate is dictated by the length of the serial bus. " The maximum speed of a CAN bus according to the standard is 1 M bit / second. At this speed, a maximum cable length of about 40 meters (130-ft) can be used. This is because the arbitration scheme requires that the wave front can propagate to the most remote node and back again before the bit is sampled. In other words, the cable length is restricted by the speed of light"[8]

The following table illustrates the maximum communications speed that can be achieved for the corresponding length of the serial bus.

Length of Cable (meters)	Max Speed (Kbp/s)
40	1000
100	500
200	250
500	125
6000	10

"Depending on the devices, up to 32 or 64 nodes per network is normal, but it is understood that at least one manufacturer is developing devices that will allow networks of 110 nodes, or more."[9]

A CAN node varies in price due to the differing manufactures. Currently a good value for money chip is the Siemens C90, which can be purchased for around about $20.00, whilst the Intel 82527 used in this project costs about $41.00

There is no standard for the connectors used.

CAN COMPARED TO I2C

I2C was originally designed to be used for interfacing integrated circuits within very close proximity. The typical application was for circuits within a maximum distance of 3 to 4 metres apart, dependent upon the speed and the data load required of the bus. This distance barrier has been overcome somewhat by interfacing additional circuitry.

Unlike CAN every component hooked up to the bus has its own unique address. "Each of these chips can act as a receiver and /or transmitter depending on its funcionality"[10].

A similarity of the two systems however is that all nodes are connected to the bus in wired AND configuration.

CAN has a superior arbitration process to I2C due to it's non-destructive process

CAN COMPARED TO THE ETHERNET SYSTEM

The Ethernet system is ideal for shifting large amounts of data quickly, due to the relatively large size of the message packets that can be used, whereas CANs focus is on sending relatively small packets of control information with high error detection.

The main advantage the CAN system has over the ETHERNET system is that when more than one transmitter attempts to transmit on the Bus, the CAN system will allow the immediate transmission of one of these messages according to it's priority, established by the Arbitration Process. (This process has been fully described in chapter two).

On the Ethernet system however, when this situation occurs the Ethernet system will cause all the transmitters to cease transmission. They then attempt once again to transmit their message. This system is called a back off exponential system, and relies on the fact that on the next attempt to transmit, one message will gain access before the other, but there is no guarantee of this and they may continue to clash. The situation can occur where no data is being transmitted, as the individual nodes, attempting to transmit a message continue to clash.

With the Ethernet system there can be no guaranteed response time, regardless of how important the message is to the network and the bus may even Lock Up, whereas with the CAN system the response time of the highest priority messages can be guaranteed.

Conclusion

It can be seen that the CAN system clearly has a place in industry which is not met by these other systems, particularly when small 8-bit microcontrollers or microprocessors are being used.

This system was selected as the most suitable for this project.

CHAPTER TWO

CAN SYSTEM

OVERVIEW

An overview of the CAN system is concisely defined in the following quotation "CAN is a shared broadcast bus which runs at speeds up to one 1 M/bits. It is based around sending messages (or frames) which are of variable length, between 0 and 8 bytes. Each frame has an identifier, which must be unique (i.e. two nodes on the same bus must not send frames with the same identifier). The interface between the can bus and the CPU is usually called the CAN controller; there are now lots of CAN controllers around." [11]

IDENTIFIERS

Data messages on the CAN bus are not sent to a particular node address, rather each message contains a Unique Identifier (ID). (This ID is unique to that network). Every CAN controller node on the network will receive the message, however each controller will look at the message's identifier, carry out an acceptance test, and only if that message is relevant, will it be stored or processed.

"This mode of operation is known as multi-cast"[12].

ARBITRATION

CAN uses Carrier Sense, Multiple Access with Collision Detection (CSMA / CD), but unlike Ethernet Carrier Sense Multiple Access with Collision Detection (CSMA/CD), a collision does not result in the simultaneous transmission resulting in all of the transmissions stopping or aborting. An arbitration method is used so that the message with the lowest identifier gains control of the BUS. This method is refered to as non-destructive bitwise arbitration, rather than simple collision detection as in Ethernet systems. Non-destructive arbitration means that no time is wasted as the message with the highest priority wins the arbitration process and continues being transmitted. In this way the bus can be used to its maximum capacity.

For example:

If a node is wanting to transmit, it will first listen to the bus to ensure it is vacant, and having established this, it begins transmission of its message. Now if another node has done the same at nearly exactly the same time, the Identifier Bits will begin to interact with each other.

This is where the arbitration process occurs. Each node listens to the bus when it is transmitting a message. As a 0 logic is dominant, it will override a logic 1 (recessive). This is due to the wired-and bus connections.

If a node is transmitting an identifier and discovers that its transmitted recessive bit is actually being seen as a dominant logic, it will ceases transmission, thereby losing the arbitration process. However the transmitter which lost will continue to listen to the bus and become a receiver. The other node which was transmitting will continue to transmit undisturbed, and no time is lost. This is why the process is known as a non- destructive arbitration process.

From this it can be seen that:

- Every message should have its own unique identifier in order to avoid the situation of two identical identifiers being transmitted simultaneously by different nodes.
- Each identifier should indicate the approximate priority of the message.
- The node with the numerically lowest identifier (indicating a higher priority) will win the arbitration process.

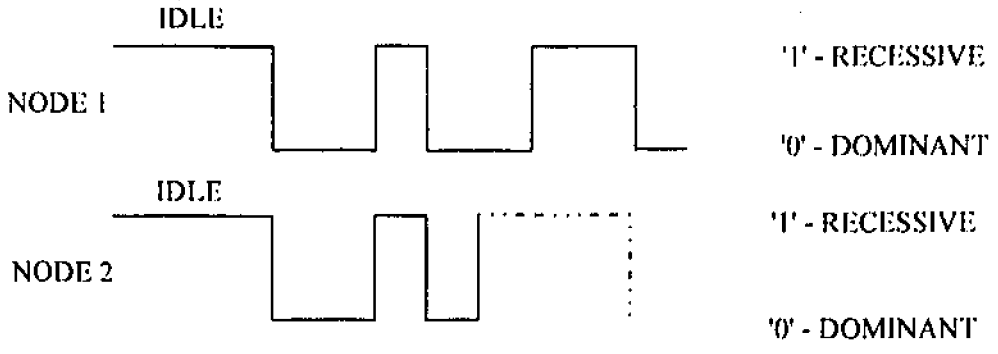
Any transmitter which lost the previous arbitration process will try to gain access once the current message has been transmitted. If more than one transmitter lost this arbitration process then there may be a situation where different nodes begin transmitting simultaneously again.

Some of the benefits of this system are:

- Messages are transmitted in order of priority.
- Unlike Ethernet the bus system will not lock up or collapse.
- No time wastage in application process, whence maximum use of the bus.

An example of two nodes fighting to gain access to the bus, to transmit a frame can be seen below:

ARBITRATION PROCESS



ARBITRATION IS LOST BY
NODE 2 AT THIS POINT

CAN TERMINOLOGY

As the CAN system is relatively new, there are still confusing terminologies in use due to the system being reviewed and consequently updateded and extended. The following two categories assigned to CAN are very different and should not be confused with each other.

- Standard Vs Extended
- Basic CAN V's Full CAN.

STANDARD Vs EXTENDED

The original identifier as defined in the CAN 1.0 standard was 11 bits in length, however due to the demand for more message identifiers, giving the system the ability to extend the number of different messages from 2048 to 536 870 911, the 29 bit Extended Identifier was introduced in the CAN 2.0 Standard.

The format of a Data Frame that utilises a 29 bit identifier, as well as the identifier extension bit wich designates a frame as a Standard or Extended Frame, is described hereafter under the heading of FRAME TYPES.

FULL CAN Vs BASIC CAN

As CAN grows these terms are beginning to become defunct, as most CAN devices can deal with both situations. However a description is included here for completeness.

FULL CAN

The Intel 82527 type of CAN is considered to be FULL CAN. This means that the microcontroller has the ability to update any of the 16 messages which can be stored in the Message Object (MO) locations. The microcontroller has the option of either dealing with the message or leaving the CAN device to deal with the transmission or reception of data frames autonomously. This means that because the messages can be dealt with, without the interaction of the microcontroller, the burden on the microcontroller is greatly reduced.

These devices usually have a large message buffer for incoming and outgoing messages.

The attributes of Full CAN can be summed up as

" Full CAN devices contain additional hardware to provide a message "server" that automatically receives and transmits CAN messages without interrupting the associated microcontroller. Full CAN devices carry out extensive acceptance filtering on incoming messages, service simulataneous requests, and generally reduce the load on the microcontroller."[9]

BASIC CAN

The Philips 82C200 on the otherhand is considered a basic CAN device. The microcontroller communicates to the device via a queue system where individual messages are dealt with one at a time. It is interrupted with the reception of every message it receives and it has to refer to the identifer of that particular message before it can decide whether or how it cares to deal with it.

Because of this the system is very microcontroller intensive. The dependence of the CAN controller and the associated microcontroller on each other is summed up as follows:

"In Basic CAN configurations there is a tight link between the CAN controller and the associated microcontroller. The microcontroller, which will have other

system related funstions to administer, will be interrupted to deal with every CAN message. For example, an interrupt is generated to check the identifier of every received message to determine if the message is relevant to the node."[9]

CAN VARIETIES

These various functionalities come together to form two fundamental versions of the CAN protocol, each having their own variations as outlined below:

- CAN1.0 : This is the original version which was confined to using a standard identifier of 11 bits.
- CAN 2.0 : This version is further broken down into two variations:
  - CAN 2.0A : A CAN chip of this type can operate solely with an identifier of 11 bits. It is directly compatible with CAN1.0
  - CAN 2.0B : The CAN system implementing CAN2.0B will function, without causing an Error Frame, using an Identifer of 11 or the extended 29 bits. Any CAN2.0B system can be either:
    - CAN 2.0B Passive: These controllers are capable of sending 'standard' size frames only. They are also capable of receiving only standard size frames. Any Extended Frames will simply be ignored, and the device will not issue Error Frames, which would destroy all bus traffic.
    - CAN 2.0B Active: These controllers are capable of receiving and sending both standard and extended frames. These CAN controllers are also backwards compatible with the other 'simpler' versions.

CAN controllers now are usually CAN2.0 as this version has rendered the CAN1.0 redundant.

A general rule simplifying the whole of the above is :

"- Controllers implementing 2.0B and 2.0A are compatible as long as the controllers implementing 2.0B refrain from sending extended frames! "[8]

ERROR HANDLING

Every node on the bus has a role in the detection of errors and in notifying all other nodes on the bus of the presence of an error, in case they did not discover it; If any node on the bus detects an error it will send an error frame. (The actual structure of the Error Frame is discussed in the FRAME TYPES section, under the heading of ERROR FRAMES). This error frame will destroy the current bus traffic and all the nodes will detect this and discard the current message.

Each node operates two separate error counters:

- RECEIVE ERROR COUNTER. This counter is incremented every time an error or an Error Frame has been monitored.
- TRANSMIT ERROR COUNTER. This counter is implemented whenever an error occurs while that node is transmitting a message.

In order to prevent a malfunctioning node which is detecting more errors than other nodes, continually destroying Bus traffic, these counters are used in the following way to produce a type of error confinement.

- When the receive counter reaches a certain value, the node will stop sending error frames upon the detection of any further errors, and thereby become error passive. If the node continues to monitor errors and the error counter reaches the next preset value, the node will completely stop all participation on the bus and go 'bus off'.
- In all probability, when an error occurs, the node which is currently transmitting is most likely to be the node at fault. Hence to remove the faulty node from the Bus, the Transmit Error counter is incremented faster than the Receive Error Counter.

FRAME TYPES

The CAN system uses four different frame types to transport data across, and maintain the correct functioning, of the network. These four frame types are:

- DATA FRAME
- REMOTE FRAME
- ERROR FRAME
- OVERLOAD FRAME

Before discussing the Data and Remote Frames in detail and because these frames are very similar in construction, the main fields which are common to both are described hereunder:

- START OF FRAME (SOF)

  This field, as the name implies indicates that a frame is beginning. The SOF only consists of 1 dominant bit.

- ARBITRATION FIELD

  This can be either 11 or 29 bits long depending whether it pertains to CAN A or CAN B. This field of the MO is used to determine the Arbitration and is also called the identifier as it is unique to the type of message. The REMOTE TRANSMISSION REQUEST (RTR) bit is included in this field. It is set to a dominant value to indicate the frame is a data frame or it is set to a recessive value to indicate the frame is a remote frame.

- CONTROL FIELD

  The Control Field is similar for both Data Frames and Remote Frames, however the field changes depending on whether the message is an extended or a standard length frame.

  The layout can be seen in the diagram under the following section headed "DATA FRAME". Both the Standard and the Extended frames include the Data Length Code (DLC). However the Standard frame consists of :

  - The IDENTIFIER EXTENSION BIT (IDE)
  - The r0 bit
  - DLC field

  While the Extended frame consists of :

  - The r1 bit
  - the r0 bit
  - DLC field

The r0&r1 bits in both cases are redundant with the present CAN system and should always be set as a dominant value. The DLC indicates the number of bytes each frame will contain.

- IDENTIFIER EXTENSION BIT (IDE)

  This bit is located in the Control Field in the Standard Frame format, and located in the Arbitration Field of the Extended frame format. A dominant value is used to indicate that the MO is a standard type message, while a recessive bit indicates that it is an extended format message.

- CYCLIC REDUNDANCY CHECK (CRC) FIELD

  This contains a 16-bit checksum which is an essential tool in detecting errors on the bus. This checksum is derived by evaluating most of the message.

- ACKNOWLEDGEMENT (ACK) SLOT

  When any node on the bus receives a message, be it a Data or a Remote Frame, it will respond by asserting an acknowledgement bit at the end of each message. If the transmitter does not detect an acknowledgement bit at the end of the message, it will retransmit the message.

- END of FRAME

  "Seven recessive bits ending the frame"[13]

DATA FRAME

The Data Frame is the one which is generally employed in CAN systems.

- DATA FIELD. This consists of a maximum of eight bytes of data.

THE FOLLOWING IS A REPRESENTATION OF THE DATA FRAME:

DATA FRAME						
Start of Frame	Arbitration Field	Control Field	Data Field	CRC Field	ACK Field	End of Frame

STANDARD FORMAT OF THE FRAME:

DATA FRAME - STANDARD FORMAT					
	ARBITRATION FIELD		CONTROL FIELD		
SOF	11 bit IDENTIFIER	RTR	IDE	R0	DLC

EXTENDED FORMAT OF THE FRAME:

DATA FRAME - EXTENDED FORMAT								
	ARBITRATION FIELD					CONTROL FIELD		
SOF	11 bit IDENTIFIER	SRE	IDE	18 bit IDENTIFIER	RTR	R1	R0	DLC

REMOTE FRAME

The purpose of the Remote Frame is to enable the CAN system to have the ability to request data from any one of the other nodes on the network. When a particular node receives a Remote Frame request for a MO it contains, it will respond by transmitting the message. The Remote Frame is similar to the data frame, however there are certain differences between the two and these are:

- The Remote Frame contains no data field.
- The Remote Frame is designated as such by the setting of the RTR bit. A dominant value indicates the message is a Data Frame while a recessive value indicates it is a Remote Frame.

The Remote Frame is not used very often in normal CAN Bus communications, and as a consequence the exact behaviour of the Remote Frame is not described in great detail in the CAN specification. Upon reception of the frame, the 82527

can be programmed to either automatically send the required data frame or to issue an interrupt request to the host CPU. This process can be summed up as "a type of request-response type of bus traffic management."[8]

THE STRUCTURE OF A REMOTE FRAME:

REMOTE FRAME					
Start of Frame	Arbitration Field	Control Field	CRC Field	ACK Field	End of Frame

ERROR FRAME

When any node on the network detects an error in a data message it will send an error frame. An error frame is one which does not obey the CAN protocol and manages this by contradicting the bit stuffing rules. Six consecutive dominant bits are sent in a row, thereby affecting the start of the frame, the ACK field or, end of line field. The transmitter upon detection of this error will then retransmit the message that it was transmitting when the error occurred.

There are two types of error flags:

- The Active Error Flag. This is only sent by a node if it has been designated an error active node.
- The Passive Error Flag.

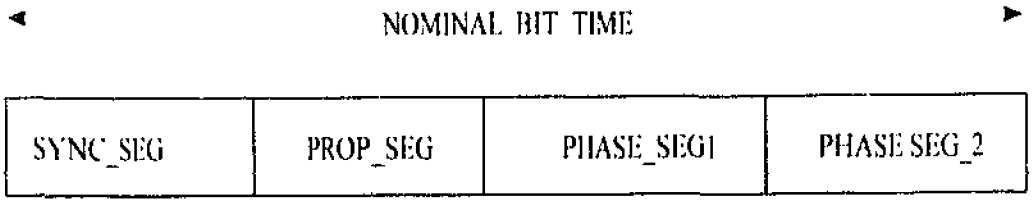
CAN includes a system which prevents the entire bus traffic being destroyed by continually retransmitting the same message

OVERLOAD FRAME

The Overload Frame is similar to the Error Frame and is transmitted by a node that becomes too busy, CAN controllers are now more sophisticated and very rarely use this frame.

BIT LAYOUT

Every message that is sent over the can bus is made up of a series of bits each of which consist of a synchronisation segment propagation segment and two phase segments. The bit can therefore be represented as follows:



The individual fields of each bit, as shown above are:

Synchronisation Segment (SYNC_SEG) During this period of time the individual nodes on the bus are synchronised. "An edge is expected to lie within this segment" [N]

Propagation Segment (PROP_SEG)"This part of the bit is used to compensate for the physical delay times within the Network."[N]

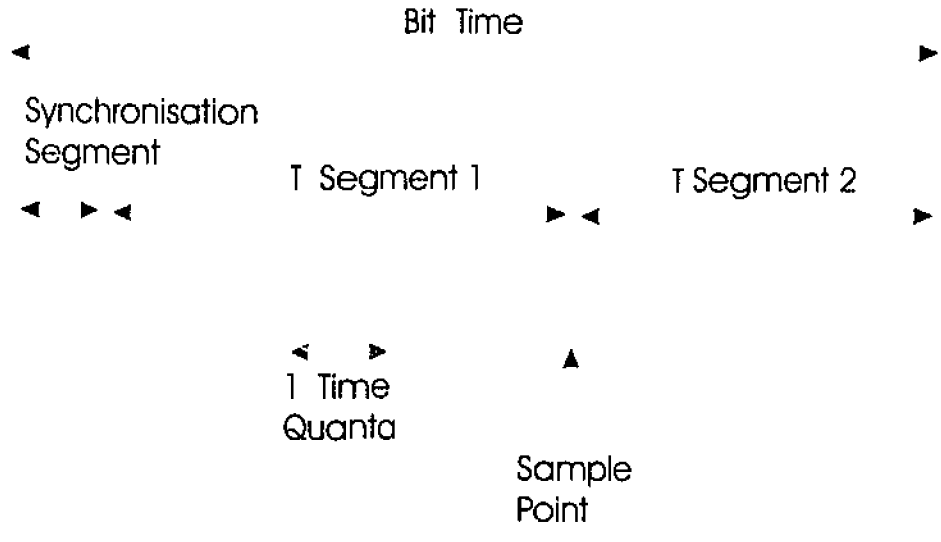
Phase Buffer Segment 1 (PHASE_SEG1)" used to compensate for edge phase errors.". This "segment can be lengthened or shortened by resynchronisation."[N]

Phase Buffer Segment 2 (PHASE_SEG2) Serves the same function as the Phase Buffer Segment 1

The 82527 controller sees each bit as made up of time quanta, the number of quanta per bit determining its length. This time quanta is determined by the baud rate prescaler which is contained in the bit timing register 3F h. The value of each bit is determined at the sample point which lies between the two phase segments.

The diagram overpage is a representation of one of these bits:

BIT REPRESENTATION



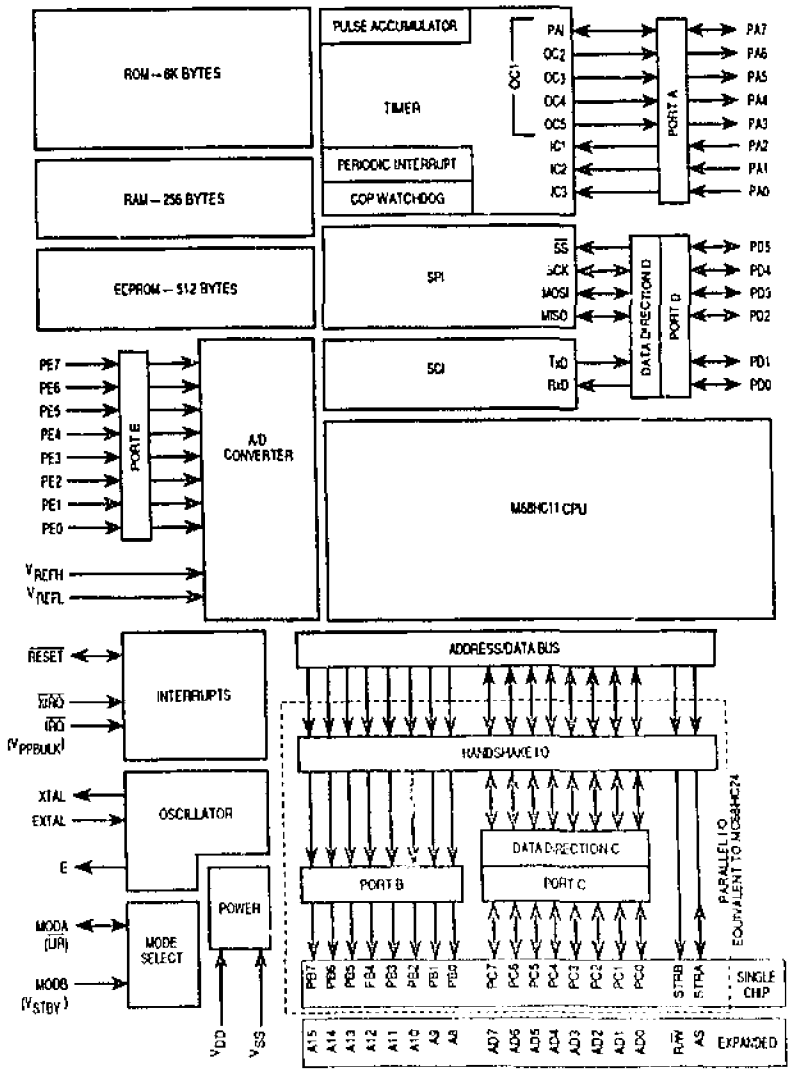
CHAPTER THREE

THE MC68HC11 AND THE EVALUATION BOARD

HC11 OVERVIEW AND DESCRIPTION

The HC11 is a versatile microcontroller with many onboard features such as an A/D converter and latchable Input / Output (I/O) lines. It contains an Electrically Erasable Read Only Memory (EEPROM), small volatile Random Access Memory (RAM) and the ability to run in two modes, single chip or expanded. There are five I/O ports, these are A, B, C, D and E. In expanded mode however general purpose I/O ports B& C are lost as they become the lines for the Multiplex Address / Data bus. A Block Diagram of the HC11 follows.

BLOCK DIAGRAM OF THE HC11 - [15]



MC68HC11 EVALUATION BOARD (HC11 EVB)

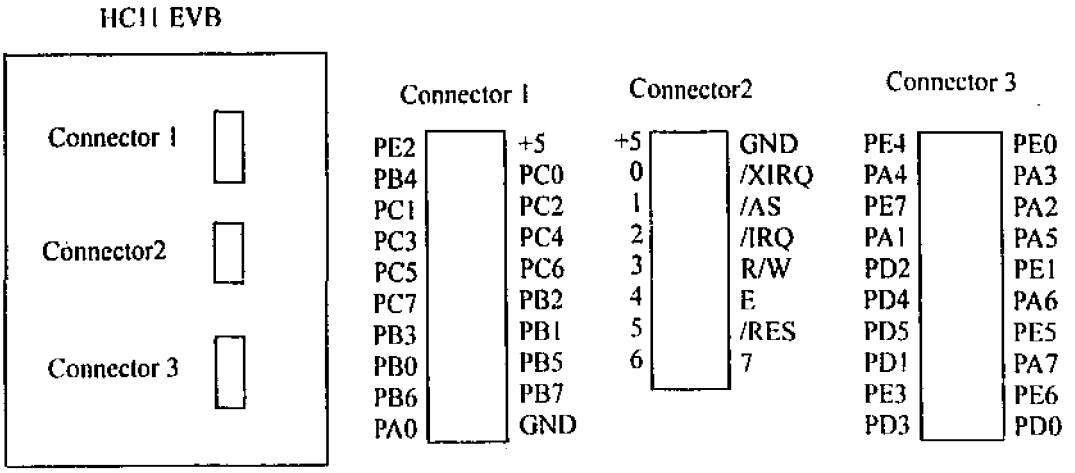
The MC68HC11 used in this project has been incorporated into an evaluation board which was purchased for this project as a complete unit. This device, frequently referred to throughout this project, is a simple, cost effective and robust unit, the main components of which are an EPROM, RAM, MC68HC11, 5 Volt Regulator, Multiplexor chip and a RS232 driver chip.
The Circuit diagram of this unit is included in the Appendices.

EVALUATION BOARD CONNECTORS

There is a group of three major connectors on the board. The pins on these connectors give access to:

- Every port of the HC11 including the control lines
- 5 Volt regulated supply and Ground line
- 8 different Chip select lines

These connectors have been used in this project and a block diagram showing the location of the connectors on the HC11EVB and the details of the pins on each connector follows:



HC11 EVB CONNECTORS

CHAPTER FOUR

CAN DEVICES

CAN PRODUCERS

The following manufacturers are some of those who produce CAN devices: Bosh, Hittachi, IAM, Inicore, Intel, Inermetall, Mitsubishi, Motorola, National Semiconductor, NEC, Philips, SGS-Thompson, Siemens, Temic, Texas Instruments, Unitrode - from [16]

From this product range the Intel chip was selected, however a study was carried out on the Siemens chip and this chapter contains a brief description of that chip.

SIEMENS SAE 81C90 - Stand Alone Full CAN Controller

The SAE 81C90 is a large scale integrated peripheral device which is responsible for the protocol of the network. Its main features and distinctive characteristics are outlined below:

- It is a 44 pin PLCC chip which includes two 8 bit I/O ports.
- Each pin on each port can be configured as Input or Output. This is determined by the port-direction register.
- It can be hooked up to different implementations of the physical layer.
- It can be linked to a host controller via a multiplexed 8-bit address/data bus or by SERIAL synchronous interface.

MESSAGE MEMORY

The main features of the message memory are:

- Incoming messages are filtered in a Content Addressable Memory (CAM).
- Each incoming message has its identifier compared with the identifiers stored in the CAM.
- When a match occurs, the received data bytes are written into the matching message's buffer. Receive interrupt is then issued to the host controller.
- When reading a byte of a message, that entire message is transferred to the shadow register.

TIME STAMP

This a superior feature of the Siemens chip and is the main difference between it and the Intel Chip.

Because it is impossible to know when a message stored in the message memory was received by looking at the data, a time stamp is appended to the end of each of the messages 0 through to 7.

The timer is a 16-bit on chip timer which starts at 0 after reset and cannot be stopped. The counter is incremented depending on a certain number of pulses of the system clock. This is determined by the state of the TSP0 and TSP1 bits in the CTRL register.

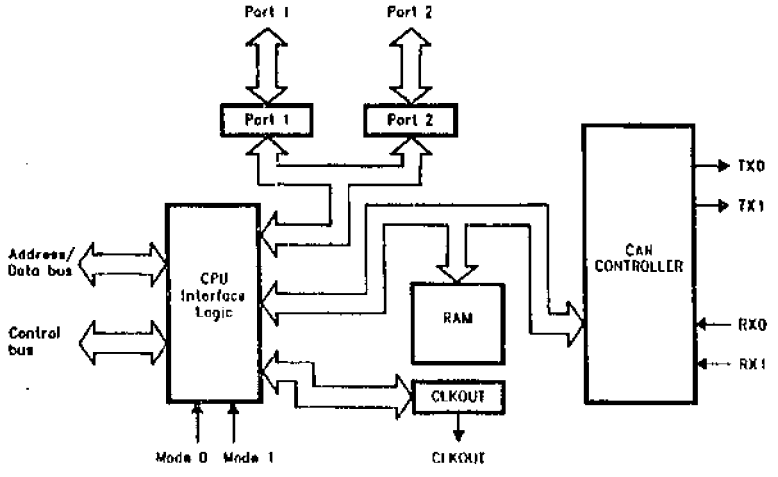
THE INTEL 82527

This Intel chip which was chosen for this project is one which has been widely used and accepted in the industry and was readily available.

The functionality and the characteristics of this chip are described throughout the relevant sections of the report, including;

- Memory Map
- Register Description
- Interfacing
- Programming

BLOCK DIAGRAM OF THE 82527 - [13]



CHAPTER FIVE

PHASE ONE - Attempted Utilisation of the 74HC138 on the EVB

This chapter describes the various initial design steps that were taken in an endeavour to implement the utilisation of the 82527.

PARALLEL I/O CONNECTION OF INTEL CAN CHIP

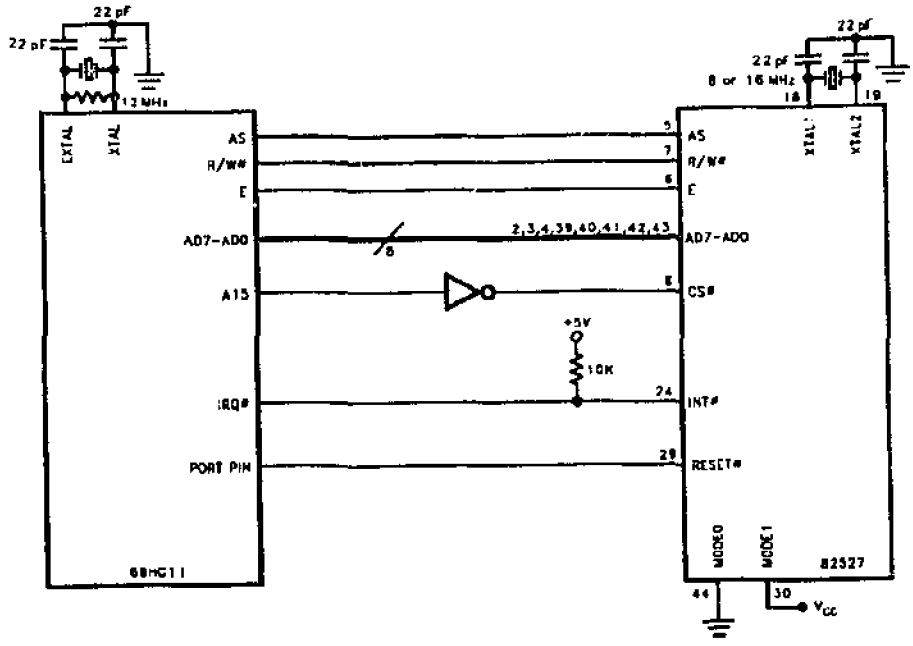
There are 2 main methods of connecting the 82527 to a host CPU, one of which has a number of variations. The classifications are outlined below:

- Connection via Address/Data Bus. This is to interface the 82527 via the multiplexed 16 Address/8 Data control bus to the host CPU. The most common types of bus are catered for, due to the ability of the 82527 to be able to select one of the 3 addressing modes. These are:
  - Mode 0: "8-bit Intel multiplexed address data bus"[13]
  - Mode 1: "16-bit Intel multiplexed address data bus"[13]
  - Mode 2: "selects an 8 bit non Intel multiplexed address data bus".[13]
  - Mode 3: "selects an 8-bit non-multiplexed address data bus for either synchronous or asynchronous communications"
- SPI (accessed by Mode 0 & "RD# & WR# pins being low at reset") with the 82527 configured as a Slave.

From this range of host - 82527 connection alternatives, the parallel Address/Data Bus was chosen, even though it had no main advantage over the SPI. Due to the nature of the Motorola parallel multiplexed Address / Data bus, the multiplexed, non-Intel mode (mode 2) was needed for correct communications.

ADDRESS DECODING AND CHIP SELECT

The application notes that accompany the Intel chip (AP-772) specify in the circuit diagram, a partial address decoding technique, and in particular, that used when selecting the Intel chip when address line 15 is active. This means any address on the bus greater than (8000h) 32768 will activate this line. This is not an efficient use of the memory mapped I/O because only the Intel chip can be used for the remaining 32K of memory mapped locations. This design is shown below :



INTEL 82527 TO MOTOROLA HC11 INTERFACE SCHEMATIC - [17]

This partial address decoding is not an adequate solution for the HC11 EVB. The EVB uses memory locations (E000h) for the Read Only Memory (ROM) containing the Buffalo program and (C000h) for the Static RAM. Hence every access to these external memories would clash with the Intel chip which would be active due to the active chip select. While the RAM chip could be removed and not accessed by any programs, Buffalo is always needed by the EVB hence the ROM chip cannot be removed.

For these reasons it was decided not to use the connection diagram specified in the Intel Application notes. Instead it was proposed to connect the Chip Select line - Active Low (CS#) direct to the existing 74HC138 chip on the EVB. This chip is a 3-to-8 decoder.

A summary of relevant details of the chip are:

- The address decoder splits locations into 8 - 8K slices.
- 3 select signals are used, but there are still five slices still available.
- The output lines of this chip corresponds to an active low configuration. Therefore the Intel CAN chip can be directly interfaced.

For this technique of CS line interfacing, using the existing 74HC138 on the HC11 EVB and choosing memory locations 2000 - 3FFF for access to the CAN chip, the following EVB memory map results :

MEMORY MAP OF HC11 - EVB AS HOST OF CAN CHIP		
CS Line Number	Memory Location	Memory Location Function
CS1	2000 - 3FFF	CAN chip
CS3	6000 - 7FFF	INTEL CAN - SERIAL INTERFACE
CS4	8000 - 9FFF	LIQUID CRYSTAL DISPLAY (LCD) UNIT
CS6	C000 - DFFF	6264 SRAM (MEMORY CHIP)
CS7	E000 - FFFF	27C64 EPROM - BUFFALO PROGRAM

DESIGN OF THE CONTROLLER BOARD (INTCAN7)

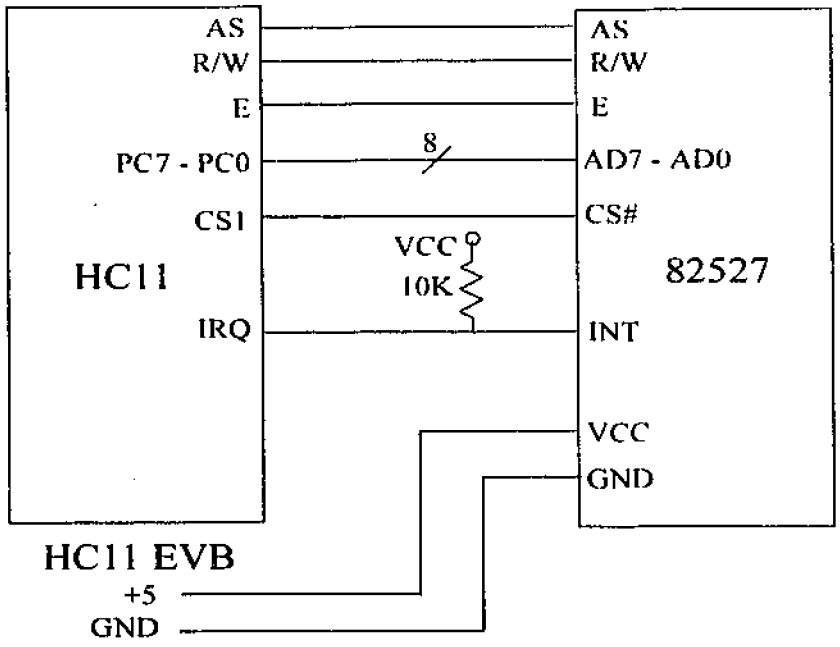
Once the previous items under consideration and design characteristics had been decided upon, the design of the Controller Board could then undertaken.

The simplified Schematic shown overpage does not show the connections of the RESET line, nor any of the other supporting connections to the 82527 (including connection to the CAN bus), only the connections directly from the HC11 and the HC11 EVB, to the new 82527 CAN board, are shown. For the ease of reference the new CAN board is now referred to as INTCAN7.

Once the final decision on the design of the circuit was made, the board artwork was produced using the EDWIN circuit design software package. The artwork files were laser printed onto a plastic transparency and using the positive photoresist procedure (Outlined in the Appendices), the image was set onto the double sided circuit board.

After the image was developed the Board was etched in Ferric Chloride acid and the holes drilled and components installed.

SIMPLIFIED SCHEMATIC OF HC11 TO 82527



After the board was produced the next step was to test the Host - Board communications

THE TESTING OF INTCAN VERSION 7

Once the board was finaliy etched, the various components and Integrated Circuit (IC) sockets were soldered in place. All the Power lines and signal lines were tested with a multimeter for continuity and as a check for short circuits. The IC sockets were finally checked with a continuity meter to ensure that the pins were traced to the correct connection pins. This was then referenced against the pin details enclosed in the INTEL 82527 data sheets.

Then, while a static strap was being used the 485 driver and the 82527 were inserted. This board was then connected to the HC11 EVB. Several accesses to the registers were tried, including the high speed registers, however all the locations were stored with FFh. This is the value that is expected to be registered from a location, if that location does not have any memory or peripheral devices attached to it. Various other methods were tried in an endeavour to obtain data from the device, but none were successful. Next an attempt to write to one of the registers was made, but as the response from the BUFFALO program was "ROM", this clearly indicating no writes to the device were working.
It was clear that the design did not work.

No valid communication was established between the 82527 and the HC11 and the digital oscilloscope was connected to the unit to try to discover the source of the problem:

DIGITAL OSCILLOSCOPE

To view the CS line the digital oscilloscope was employed. The data on the E line was captured and then analysed. From this waveform it indicated that the bus was running at 2 Mhz as the period of one E cycle was 500 nS. Then the CS line was looked at and a low transition was captured. This had a good clear waveform containing a low period of 250nS. This indicated that the Bus frequency was as expected and was not the source of the problem.

Conclusion

The 74HC138 on the EVB was not able to be used due to the fact that it was not giving an acceptable output for use with the 82527.
This matter is dealt with in more detail further on in this report under the heading of HC11 EVB 74HC138 CHIP SELECT PROBLEM

CHAPTER SIX

PHASE TWO - Use of the Port Replacement Unit (PRU) to simulate the HC11 Address/Data BUS

ANALYSIS OF THE 82527 - HC11EVB TIMING RELATIONSHIP DIAGRAMS

Clearly further investigation of the timing relationship of the bus signals became necessary to enable two possible outcomes to be evaluated.

- To directly discover the problem with the hardware connections.
- To develop the necessary sequence needed so as to use a software method of simulating the HC11 Address/Data bus in order to communicate with the 82527 in a slower, observable manner.

SIMULATION OF THE HC11 ADDRESS / DATA BUS

In order to simulate the HC11 Address / Data bus and all of the necessary control lines the following set of I/O lines are required :

HC11 Lines That Need to Be Simulated	Resources Needed by HC11 for Simulation
The 8 line multiplexed Address / Data bus	8 Bidirectional Input / Output lines, capable changing data direction easily.
Address Strobe (AS)	Output Line
Chip Select (CS#)	Output Line
E System Clock	Output Line
Read / Write line (R/W#)	Output Line

NOTE - The # symbol indicates that these lines are of acitve low configuration.

This table specifies that to simulate the HC11 Address/Data Bus the following resources will need to be available from the simulating microcontroller:
8 Bidirectional lines and 4 output lines.

The following table lists the available HC11 I/O resources :

Port	General Purpose I/O Lines	Number Available
A	Input	3
	Output	4
	Bidirectional	1
B	Output	None (All Used)
C	Bidirectional	None (All Used)
D	Bidirectional	6
E	Input	8

Note Ports B & C cannot be used as they are being used for the real bus lines. (HC11 EVB is running in Expanded Mode).

Analysis of the available resources indicated that while the 4 output ports are available only 7 Bidirectional Ports are available.

One possible solution to this problem was to investigate the use of an input and output line connected together to the same Address / Data line on the 82527. The other was to use the PRU.

This Port Replacement unit would be used for programming ease and to ensure no damage occurred to the CPU. Damage may occurred if one of the port lines which was tied to the other was fighting for differing logic levels.

MC68HC24 PORT REPLACEMENT UNIT

The function of the MC68HC24 chip is to replace the general purpose I/O Ports B & C. which are lost by the HC11 when its is running in the expanded mode of operation.

DESIGN OF THE PORT REPLACEMENT BOARD

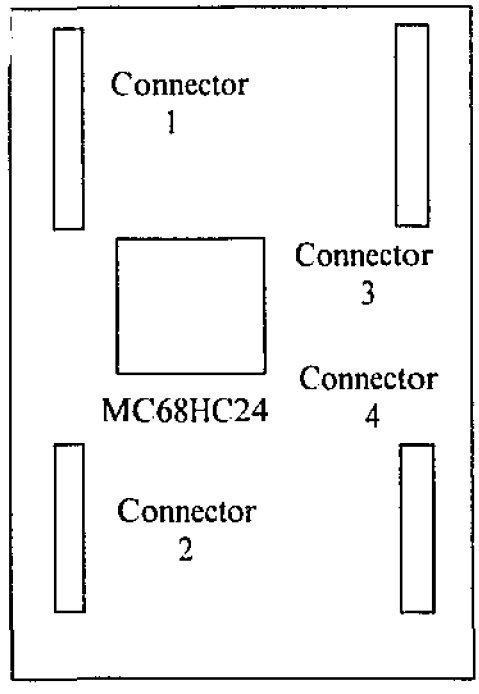
This board was designed to use the pin placements of the HC11 EVB and give back the B & C Ports with a similar pin placement on the connections. as they are on the HC11EVB. The board obtains all its power from the regulated 5 volt

power supply, on the HC11 EVB.

This board was produced using the positive photoresist procedure, and then etched in the aerated acid bath. The HC24 was placed in the circuit while a static strap was being worn.

The Block Diagram of the board is represented below:

BLOCK DIAGRAM OF PORT REPLACEMENT BOARD



Connector 1 and Connector 3 are set up with the same pin allocations as the HC11 EVB Connector 1. Connector 2 and Connector 4 are set up with the same pin allocations as the HC11 EVB Connector 2.

See the appendices for the Circuit Board Design

TESTING OF THE PORT REPLACEMENT BOARD

Once the board was constructed, the PRU was attached to the HC11 EVB. Then the following test was carried out to determine if the outputs at the HC24 PRU were functioning correctly.

When using MM commands the output data stays on the Port lines. The following is a listing of the code used.

```
LDAA #$FF   ;DDRC is directed to make every Port C line an output
STAA $1007  ;
LDAA #$00   ;A low logic state on every Port C line
STAA $1003  ;
```

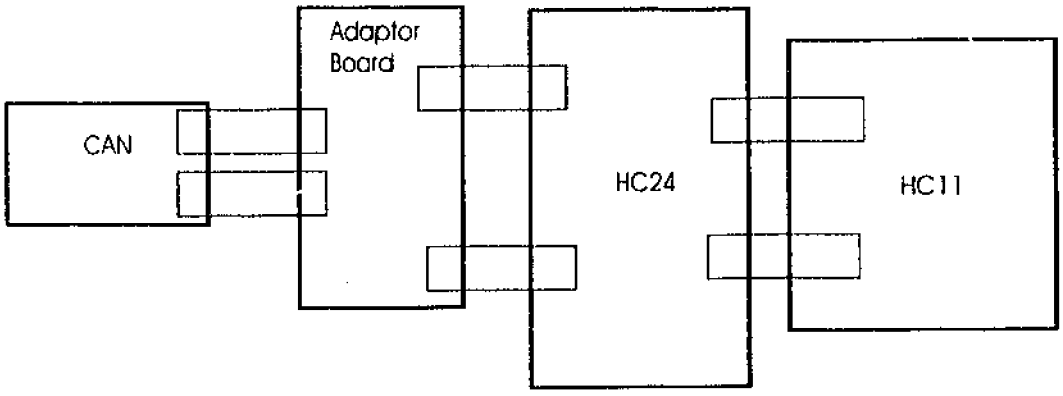
A normal write to Port C does not produce any handshake signals and a normal read returns the current state of the Port C pins.
However the other Port C is Port CL (Port C Latch). It's address is at $1005. This Port works in association with the Strobe A (STRA) and Strobe B (STRB) pins. When a write is made to Port CL, the data is driven out through the Port C lines and a handshake signal is made using the STRB pin.
Similarly the STRA pin is used to signal the processor to store the current data at Port C into the Port CL register. In this way the value that was at Port C can be analysed or processed later.

The use of this Port Replacement Unit posed another problem in that it required an adaptor board to ensure that the pin arrangements that the INTCAN board required for it's interconnection plugs would remain the same. The following section deals with the development of this board.

ADAPTOR BOARD DEVELOPED FOR PRU TO INTCAN7 CONNECTIONS

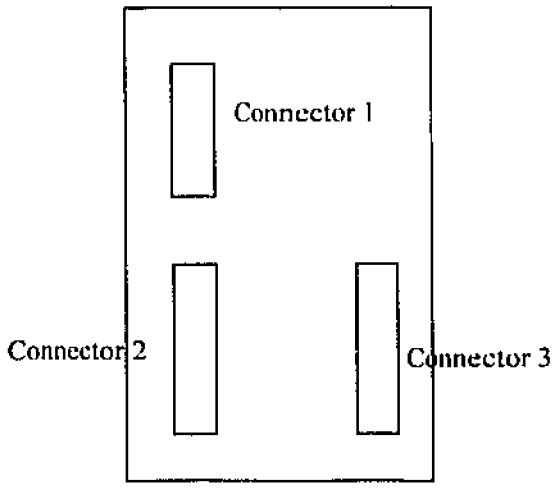
The INTCAN7 board which had already been developed had pin connectors developed for the HC11, not the PRU. For this reason an adaptor board needed to be developed to enable the PRU to connect to the HC11. The purpose of this adaptor was to route the various pins on the PRU which would be used for the simulation to the correct Address / Data bus and control lines on the INTCAN7 board. The proposed system would be as follows:

SETUP FOR SOFTWARE BUS SIMULATION



The Port C lines were simply routed as per the original configuration. However lines PB4 through to PB7 needed to be routed from the 20 pin connector to the 16 pin connector which was used for the Bus control lines. See the following diagram of the PRU to INTCAN7 adaptor board.

HC24 SIM ADAPTOR



SOFTWARE ASPECTS OF THE BUS SIMULATION

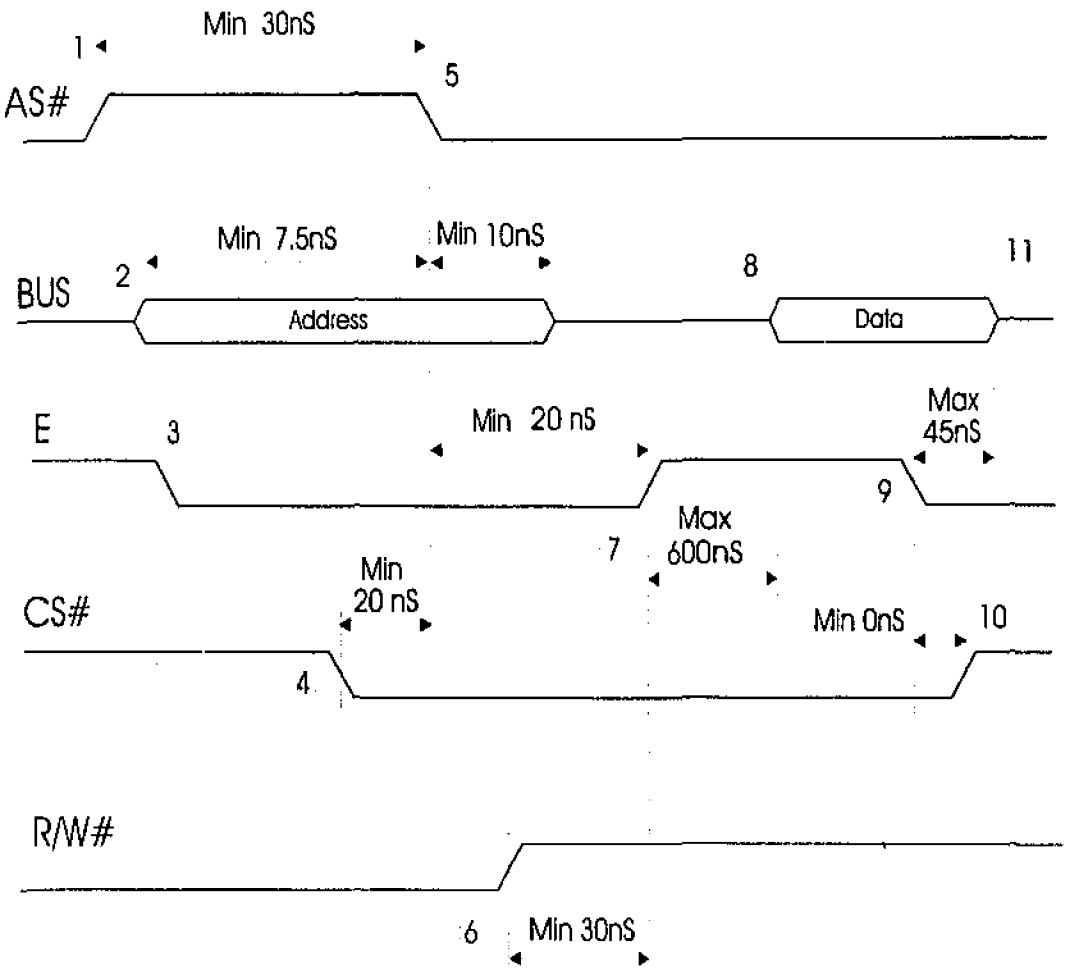
Firstly the exact timing relationships of the HC11 and INTEL CAN chip busses were investigated. It was to be expected that there would be many minimum time values associated with these timing sequences, but few maximums. By explanation, the INTEL chip specification list a set of minimum time delays for each change of state in each of it's parallel communications lines. In most

instances the 82527 copes with long periods before another, change of state, but does not necessarily cope with rapid changes with respect to time.

This makes the simulation of the HC11 bus possible. Because there is no minimum time specification between the change of one state to another the Minimum time in which the HC11 would be able to effect these various changes does not need to be a limiting factor.

The following diagram is a graphical representation of the sequence of operations that are required in order that communication can be made by the HC11 to the 82527. This is then broken down into a series of programming steps for the algorithm.

TIMING DIAGRAM AND ALGORITHIM SEQUENCE



The simulation carried out was one to confirm whether or not the 82527 could communicate to the HC11. This could be performed by attempting to read any memory location on the 82527. In order for the Read operation to be known to

have been successfully accomplished, the contents of the memory sell read must be known in the first place. This was not a problem however due to the fact that several registers on the 82527 have a set default value upon a hardware reset. One

such register is the CPU Interface Register 02 H. The default reset value of this register is 61H (after the RESET# has been deactivated) . If a read of this memory location returned such a value instead of a FFH then this demonstrated that a successful read operation had in likelihood occurred.

The bus cycle for a read operation of this nature can be broken down into the following steps:

STEP 1

Initialise the Bus so all lines start at a set pattern. These initialised values being :

Address / Data - Off AS - Low E - High CS# - High

R / W# - Low

STEP 2

Alter the state of these lines as per the following sequence, which can be referenced back to the previous diagram.

1 AS line to go High; Indicating that an address is about be asserted on the bus.

2 HC11 to assert address onto the Address / Data Bus

3 E line to go Low; clock cycle continuing

4 CS# line to go low; tell the 82527 that it is to become active on the parallel bus

5 AS line to go low; the address is no longer being asserted

6 R/W# line to go High; the 82527 is to put the data corresponding to the appropriate memory location onto the bus

7 E line to go High; clock cycle continuing

8 HC11 can no read the data on the Address / Data Bus put there by the 82527

9 E line to go Low; clock cycle continuing

10 CS# line to go High; the 82527 is now to become inactive with its parallel bus communications

11 Data on Bus no longer Valid

This algorithm was then converted into a functioning Motorola assembly language program. As previously indicated, the PRU will be used for the hardware connection of the simulation.

The PRU gives us back the use of 8 Bidirectional I/O lines (Port C), and 8 output only lines (Port B). The lines of Port C will simulate the Address / Data bus, which need to be bidirectional, while Port B will be used for the AS, E, CS# and R/W# control lines (as these only need to be output lines).
This is clarified by the following table :

HC11 Address / Data Bus Lines	PRU Resources Used
Address / Data Bus	Port C (Bidirectional)
AS line	Port B, line 7 (PB7)
E clock	Port B, line 6 (PB6)
CS# line	Port B, line 5 (PB5)
R/W# line	Port B, line 4 (PB4)

The program developed strobes the appropriate lines as per the sequence described above. For further testing, another set of timing sequences for the analysis and simulation of the Write cycle could be developed.

TESTING THE INTCAN BOARD USING THE SIMULATION SYSTEM

The simulation system developed as described above proved that the INTCAN7 board worked as it should. It obviously obeyed all the timing sequence restrictions, and the chip is not damaged. This indicated the problem with the parallel communications to the board was with the HC11 EVB.
The full source code of the program called CANTEST.ASM follows:

SOURCE CODE FOR CANTEST.ASM

```
        org   $C000            ;Starting address of program
*************************************************
*   Initialise all lines                        *
*   A/D - off, A/S - low, E - high, CS# - high, R/W# - low *
*************************************************
        jsr   slower            ;Wait
        ldaa  #$00              ;DDRC set all lines of Port C
        staa  $1007             ;for input
        ldaa  #%01100000        ;Set Port B
        staa  $1004
*************************************************
*   Set lines for step 1                        *
*************************************************
        jsr   slower
        ldaa  #%11100000
        staa  $1004
*************************************************
*   Set lines for step 2  Address to look at here *
*************************************************
        jsr   slow
        ldaa  #$ff              ;DDRC set all lines of Port C
        staa  $1007             ;for output
        ldaa  #$02              ;Address to look at here
        staa  $1003             ;and put on 'Bus' Port C
*************************************************
*   Set lines for step 3                        *
*************************************************
        jsr   small
        ldaa  #%10100000
        staa  $1004
*************************************************
*   Set lines for step 4                        *
*************************************************
        jsr   slow
        ldaa  #%10000000
        staa  $1004
*************************************************
*   Set lines for step 5                        *
*************************************************
        jsr   slower
        ldaa  #%00000000
        staa  $1004
*************************************************
*   Set lines for step 6                        *
*************************************************
```

```
        jsr   slow
        ldaa  #%00010000
        staa  $1004
*************************************************
*    Set lines for step 7                       *
*************************************************
        jsr   slower
        ldaa  #%01010000
        staa  $1004
*************************************************
*    Set lines for step 8  Get Data Here        *
*************************************************
        jsr   slower
        ldaa  #$00              ;DDRC set all lines of Port C
        staa  $1007             ;for input
        jsr   slower            ;Wait
        ldaa  $1003             ;Get Data From 'Bus' Port C
        staa  $c500             ;and store at location C500
*************************************************
*    Set lines for step 9                       *
*************************************************
        jsr   small
        ldaa  #%00010000
        staa  $1004
*************************************************
*    Set lines for step 10                      *
*************************************************
        jsr   small
        ldaa  #%00110000
        staa  $1004
        jmp   finish            ;End program
*************************************************
**    Timer Routes                               *
*************************************************
small:    ldx   #$00ff          ;Small Time Delay Routine
tilop:    dex
          bne   tilop
          rts
slower:   ldx   #$ffff          ;Longest Time Delay Routine
timeloop: dex
          bne   timeloop
          rts
slow:     ldx   #$f000          ;Medium Time Delay Routine
loopy:    dex
          bne   loopy
          rts
finish:   end
          end
```

Upon the execution of this programme the correct value of the CPU Interface Register (being 61 H) was successfully retrieved from the simulated HC11 parallel bus.

Conclusion

From this simulation the following points were confirmed:

- The 82527 CAN chip was functional
- The Board communicated and functioned correctly with the test hardware configuration and the software program

The problem with the HC11 to 82527 communication was therefore likely to reside with the HC11 EVB

CHAPTER SEVEN

PHASE THREE - Use of an external 74HC138 for Chip Selection

This chapter deals with the investigations that were carried out in an endeavour to discover the External Address/Data bus communication fault with the HC11 EVB.

HC11 EVB 74HC138 CHIP SELECT PROBLEM

Further analysis of the schematic diagram of the HC11 EVB highlighted a problem not previously detected. The area of concern is shown below:

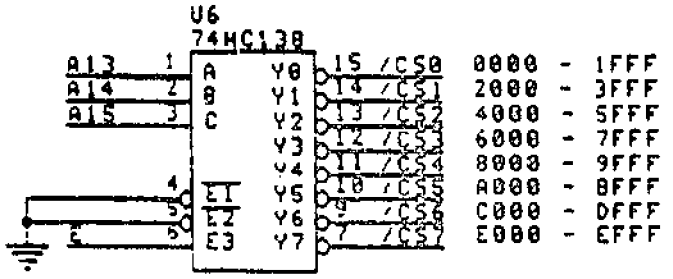


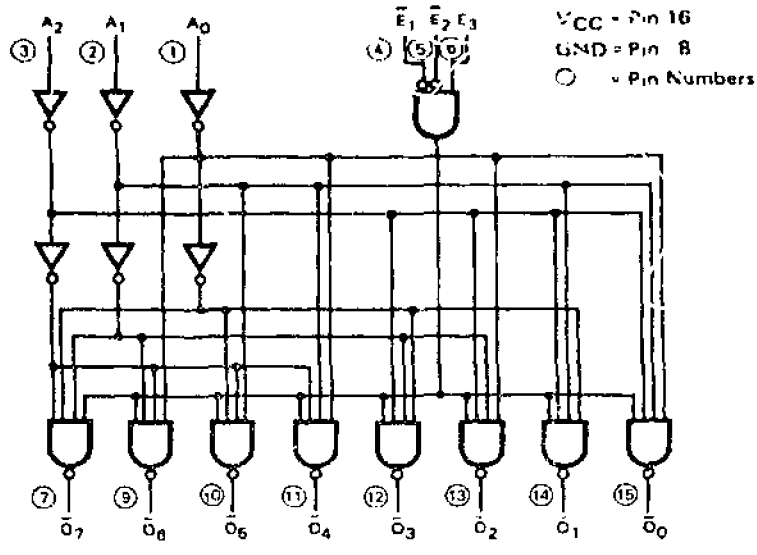
DIAGRAM OF THE 74HC138 - RELATIVE TO THE EVB CIRCUIT DIAGRAM -[18]

The 74HC138 3 to 8 decoder has an enable system where a 3 input and gate enables the output of the decoder. The three EVB connections to this chip are:

- E1 to GND
- E2 to GND
- E clock to the non inverted input

This implies that the decoder will only have a valid output when the E clock is high. This appears to be the problem with the timing sequence. The 82527 requires the CS# line to be active low until the data has been retrieved from the Address / Data bus. However with this connection of the 74HC138, the CS# line will become inactive at the time the E clock drops low, this being prior to the Data being placed on the Address / Data bus. This is due to the design of the 74HC138, which follows:

74HC138 LOGIC DIAGRAM - [12]



The 74HC138 will only assert the necessary active low output when the E clock input connected to E3, (pin 6), is high.

The solution of this problem required the use of another method of acquiring the CS line. The best way of achieving this was to use another 74HC138 chip and enable it's CS output lines continuously. Then the 3 Address input lines could be connected directly to the highest 3 bits of the Address / Data bus.
Hence when a memory address corresponded to a memory location of the 82527 chip, the 74HC138 activated the appropriate CS# line to which the 82527 was connected. In this way the CS# line was active as long as this address remained on the bus. The upper 8 bits of the Address data is conveyed on the Port B lines. The lower 8 bits of the Address are multiplexed with the data on Port C (the upper 8). The upper 8 bits of the bus remain with the valid address location even while the lower 8 bits are in the data mode of the 'address / data cycle'. This means that the CS# line will be giving the appropriate active low signal that the 82527 chip expects while the data is valid on the bus.

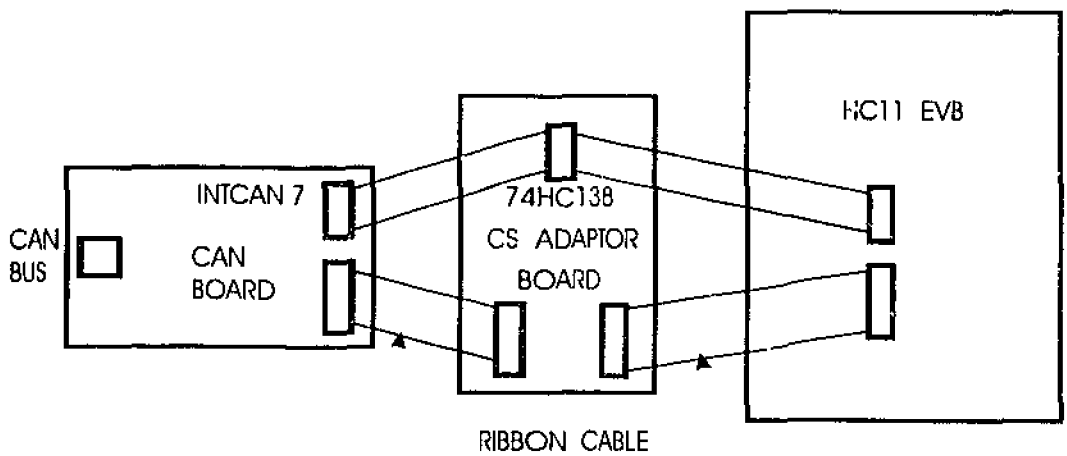
74HC138 ADAPTOR BOARD

To avoid designing a new INTCAN board simply to incorporate the 74HC138 chip, an adaptor board was created. The board was designed to interface between the INTCAN7 board and the HC11 EVB in such a way that it would not alter the

Address / Data bus address lines, but it would open circuit the current CS# line coming from the HC11 EVB and then create a new CS# line which will be fed into the INTCAN7 board.

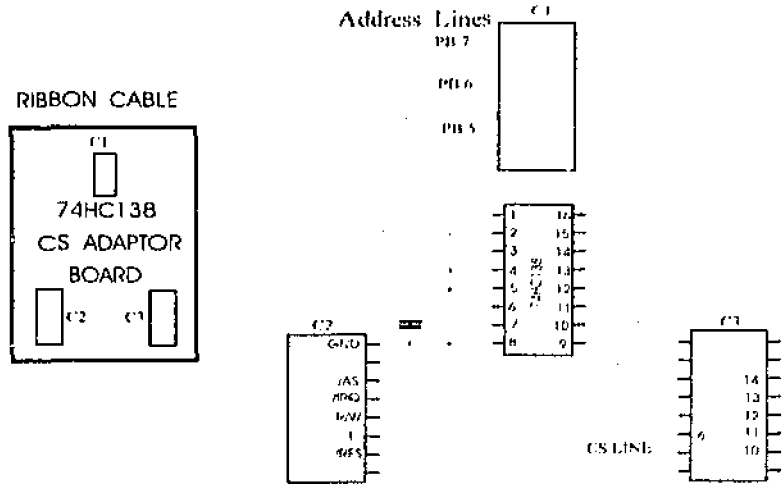
The proposal was to use the existing controller board which was known to be functioning correctly, this confirmed by the testing program described in the previous chapter, and thus avoid the errors which may be generated with the design and production of an entirely new board.
The problem, if any would then be confined to that generated by the addition of a new CS line.
The ideal solution would be one which required no modification or alteration to the HC11 EVB or the INTCAN7 board.
This could be achieved by producing an adaptor board to interface between these two boards in the manner shown in the following diagram:

CONNECTION OF THE SYSTEM USING THE 74HC138 ADAPTOR BOARD INTERFACE



As initially the boards only requirement was to enable a test to be made to discover whether the problem lay with the CS, and as it only contained a small circuit, a quick method of its production was chosen rather than using the time consuming Photoresist / Etchant process. It was produced using the Vero Board printed circuit prototyping process.
The design of this board is shown overpage:

CIRCUIT DIAGRAM OF ADAPTOR BOARD



Following the production of the board the system was connected together, and was tested by attempting to retrieve a known value of a register, directly, without the simulation software as employed in the system previously used (described in the previous chapter).

The data from register 61h was retrieved successfully.

Conclusion

The successful writing and then retrieval of various data to numerous registers was effected., thereby demonstrating that the write operation worked correctly. The success of these simple tests, indicated that the 82527-HC11 Address/Data Bus communication system problems had finally been resolved.

The next step was to investigate the INTEL chip more fully with respect to the software aspect of its operation. The following chapter deals with its registers.

CHAPTER EIGHT

INTEL CAN REGISTERS

This chapter gives a brief description of the most important registers frequently referred to and which needed to be investigated for use in this project. The specific values which these registers are assigned are discussed under the topic "PROGRAM CONSIDERATIONS FOR REGISTERS."
Other more detailed information on the registers required for programming, including the description of all the bits can be found in the INTEL Architectural Overview paper 272410-003.

CONTROL REGISTER 00H

7	6	5	4	3	2	1	0
0	CCE	0	0	EIE	SIE	IE	Init

This registers main feature among others is to take the 82527 in or out of initialisation mode. Only when the 82527 is in initialisation mode can other control registers be modified. It is important to note that while the 82527 is in initialisation mode no serial transmission or reception of data bits can occur.
The function of the various flags in this register are:

- CCE (Change Configuration Enable). Unless this is set, the Configuration Registers cannot be written to.
- EIE (Error Interrupt Enable). When this is set, a change in the error status will cause an interrupt to be generated.
- SIE (Status Change Interrupt Enable). When set, an Interrupt will be generated under certain CAN bus errors, or on the successful transmission or reception of messages.
- IE (Interrupt Enable). Unless this is set the 82527 cannot generate any interrupts.
- Init (Initialisation). This is set when the 82527 is to be configured, but must be reset to 0 for bus communications to work.

STATUS REGISTER 01

7	6	5	4	3	2	1	0
BOff	Warn	Wake	RXOK	TXOK	LEC2	LEC1	LEC0

This register is accessed by the Host CPU to determine the state of the 82527.

The flags contained in this register are:

- Boff (Buss Off Status)
- Warn (Warning Status)
- Wake (Wake up Status)
- RXOK (Receive Message Successfully)
- TXOK (Transmit Message Successfully)
- LEC2 - LEC0 (Last Error Code)

CPU INTERFACE REGISTER 02H

7	6	5	4	3	2	1	0
RstST	DSC	DMC	PwD	Sleep	MUX	0	CEn

The flags contained in this register are:

- RstST (Hardware Reset Status)
- DSC (Divide System Clock (SCLK))
- DMC (Divide Memory Clock)
- PwD (Power Down Mode Enable)
- Sleep (Sleep Mode Enable)
- MUX (Multiplex)
- CEn (Clockout enable)

This is a very important register, as it dictates the form of communications that will occur between the UC and the 82527. Due to this fact the contents of this register must be able to be altered by the Host Microcontroller (Uc) to establish a correct connection between the two. For these reasons it is a high speed register which is capable of being accessed via the Double Read operation. This would be one of the first registers to check if the Host to 82527 connection is not functioning correctly.

The CPU Interface Register is used to determine the Frequency/Period of the system clocks.

This important aspect is discussed at the end of this chapter under the heading of "CLOCK AND FREQUENCY OF CAN BUS"

HIGH SPEED READ REGISTERS 04 - 05

These registers are used to serve CPU's who's Address/Data bus runs at a frequency which is too fast for the 82527. For example a read request may be placed on the bus, along with the address, but the 82527 is not able to process the request and place the correct data on the bus before the CPU is read to receive it. If the CPU is unable to alter it's read/write cycle or extend it, then these registers must be used in conjunction with the Double Read operation.

GLOBAL MASK REGISTERS

STANDARD 06 - 07 AND EXTENDED 08 - 0B

As the name implies a global mask will act as a type of message acceptance filtering. One set of these registers is to deal with standard message objects, while the other mask will deal with extended message objects.

The Global Masks can be set to either accept or reject a range of messages. The acceptance method is straight forward but the implications can be rather complicated to deal with, particularly when Remote Frames are being used across a network.

Each bit location of a mask register will act as a mask for that same bit location of the incoming messages. If a bit location of the mask is set to a '0', it means that the controller will accept an incoming message with either a '0' or a '1' for the same corresponding bit location. Similarly if a bit location of the mask is set to a '1', it means that for any message to be accepted for further filtering, the same bit location for any incoming message must match exactly with the same bit locations of one of the 15 MO identifers.

This can be a useful feature when messages are grouped in order of their function and the global masks on each node are set to either accept or reject these nodes depending on their function.

Careful consideration must be made however when using Remote Frames. For

example a node may accept a Remote Frame request due to the programming of the mask bits. The Remote Frame however may not exactly match the Identifer Bits of the message object that will deal with the frame. The corresponding data frame that is sent by this node will have the identifer bits of the MO it has been accepted by, not the exact identifer bits of the remote frame.

MESSAGE 15 MASK REGISTER 0C - 0F

The mask was not used by this project, but the following is an explanation of it as it is a useful register.

Like the other MO locations in the CAM, this location will not accept any message, unless it has passed through the Global Mask filter first. The main difference with this register however is that it can be set in a similar fashion to the Global Masks, that is a '0' in one of the Mask register bit locations will accept any messages who's bit is either a '0' or a '1', providing the other identifer locations match. A '1' means that the corresponding bit in the incoming message must match the identifer bits of MO 15 exactly.

CLOCKOUT REGISTER 1F

This was not relevant to this project. This register is used to control an output clock from the 82527, which may be used to control some external device.

BUS CONFIGURATION REGISTER 2FH

7	6	5	4	3	2	1	0
0	CoBy	Pol	0	DcT1	0	DcR1	DcR0

While register 02h is concerned with the Host/CAN connection this register is concerned with the CAN/CAN connection. This register has more of a bearing on the data link layer aspect of the Serial Bus connections, and determines the following Bus characteristics:

- Dominant and Recessive bit polarities.
- Single wire or differential bus, determined by the operation of one or two of the input and output pins and by the use of the input comparator.

The full names of the various flags in this register are:

- CoBY (Comparator Bypass)
- Pol (Polarity) - of Bit of Serial Bus
- DcT1 (Disconnect TX1 output)
- DcR1 (Disconnect RX1 input)
- DcR0 (Disconnect RX0 input)

BIT TIMING REGISTERS 0 AND 1 (3FH & 4FH)

REGISTER (0) 3F

7	6	5	4	3	2	1	0
SJW		BRP					

REGISTER (1) 4F

7	6	5	4	3	2	1	0
Spl	TSEG2			TSEG1			

These registers are also primarily concerned with the data link layer. Register 0 dictates the following :

- The Synchronisation Jump Width (SJW), which deals with the number of time quanta a re-synchronisation period may take.
- The Baud Rate Prescalar (BRP), which determines the length of one Time Quanta.

In essence, Register 1 determines the width of each bit and whether it should be sampled once or three times. It also dictates the point at which each incoming bit is to be sampled to determine its value. This is done via the use of two time segment values TSEG1 and TSEG2. Each bit is made up of the total of these value of these two bits plus the Synchronisation Segment. Each of these periods is determined by a number of time quanta and the sample point of the bit occurs between these two. To concur with the CAN specification however the total bit length needs to be of 8 time quanta in total.

INTERRUPT REGISTER 5F

This is a read only register, whose value indicates which MO issued the interrupt to the Uc. If more than one MO was capable of issuing an interrupt this register would need to be looked at by the interrupt routine before the interrupt was cleared otherwise there would be no clear indication as to which message issued the interrupt. In the case of the Tracker however, only one MO is capable of issuing an interrupt. Therefore the interrupt routine knows which MO to look at and as a result this register is not needed by the respective program.

CONTROL REGISTERS 0 AND 1

These registers are actually a group of registers, there is a Control Register 0 and a Control Register 1 for each MO location in the CAM. These registers should not be confused with the Control Register 00 and Control Register 01, which affect the characteristics of the chip as a whole.

As the values are frequently altered, a system has been used so that one of the group of values can be altered without affecting the rest of the values in the register. This system is time saving, in that it avoids the necessity for the register to be read in order to find out what the existing values assigned to each field are, prior to the whole register being written to with the specific field altered.

Each field can be seen as representing either a '0' or a '1' just like a normal flag in a register. Writing a specific value to the fields will either change the state of the 'flag', or leave it unaltered, as shown below:

11 The field will not be altered; the old data will be left in the field.

10 The data in the field will be seen as set '1'.

01 The data in the field will be seen as reset '0'.

00 This value is not allowed to be given to one of the fields.

CONTROL REGISTER 0

7	6	5	4	3	2	1	0
BASE ADDRESS + 0							
MsgVal		TXIE		RXIE		IntPnd	

The four values (flags), and their functions which are set by this register are :

- MsgVal (Message Valid)
- TXIE (Transmit Interrupt Enable)
- RXIE (Receive Interrupt Enable)
- Intpnd (Interrupt Pending)

CONTROL REGISTER 1

7	6	5	4	3	2	1	0
BASE ADDRESS + 1							
RmtPnd		TsRqst		MsgLst - CPUUpd		NewDat	

The four fields (flags), and their functions are:

- RmtPnd (Remote Frame Pending)
- TsRqst (Transmit Request)
- MsgLst - CPUUpd (Message Lost) ; For objects designated for receive.
  Or (CPU Updating) ; For Objects designated for transmit.
- NewDat (New Data)

ARBITRATION REGISTERS

(BASE ADDRESS +2 - BASE ADDRESS +5)

The arbitrations registers hold the identifier of the message. All the bits of these registers are used, with the exception of bits 0, bit 1 and bit 2 of Arbitration Register 3, which are reserved. The Arbitration process has been discussed in detail in Chapter 2 and an example of it's use can be seen in Chapters 13 and 14

MESSAGE CONFIGURATION REGISTER

(BASE ADDRESS + 6)

7	6	5	4	3	2	1	0
DLC				Dir	Xtd	Reserved	

These three fields (flags), and their functions are:

- DLC (Data Length Code)
- Dir (Direction)
- Xtd (Extended or Standard Identifer)

SERIAL RESET ADDRESS REGISTER FF

This register is only needed if the Uc is not capable of issuing a correct CS, and was not used in this project.

MEMORY MAP OF THE INTEL CAN CHIP

The location of all of the registers previously described can be seen in the summarised memory map of the Intel 82527. Each memory location on the 82527 is only 2 bytes in length, however the full address as used in this project is shown in the map. As mentioned in detail in the chapter dealing with the CS problem, the 82527 has been connected to the memory locations responding to RAM 6000h to 60FFh, on the EVB.

A Memory map representing the allocation of the 82527 registers within its RAM follows overpage.

MEMORY MAP OF THE INTEL 82527 AS USED ON THE HC11EVB

Memory Location	Register Function	Memory Location	Register Function
6000h	Control Register	606Fh	Reserved
6001h	Status Register	6070 - 607Eh	Message 7
6002h	CPU Interface Register	607Fh	Reserved
6003h	Reserved	6080 - 608Eh	Message 8
6004-6005h	High Speed Read	608Fh	Reserved
6006 - 6007h	Global Mask - Standard	6090 - 609Eh	Message 9
6008 - 600Bh	Global Mask - Extended	609Fh	P1CONF
600C - 600Fh	Message 15 Mask	60A0 - 60AEf	Message 10
6010 - 601Eh	Message 1	60AFh	P2CONFIG
601Fh	CLOCKOUT Register	60B0 - 60BEf	Message 11
6020 - 602Eh	Message 2	60BFh	P1IN
602Fh	Bus Config Register	60C0 - 60CEh	Message 12
6020 - 603Eh	Message 3	60CFh	P2IN
603Fh	Bit Timing Reg.0	60D0 - 60DEh	Message 13
6040 - 604Eh	Message 4	60DFh	P1OUT
604Fh	Bit Timing Reg.1	60E0 - 60EEh	Message 14
6050 - 605Eh	Message 5	60EFh	P2OUT
605Fh	Interrupt Register	60F0 - 60FEh	Message 15
6060 - 606Eh	Message 6	60FFh	Serial Reset Address

PROGRAM CONSIDERATIONS FOR REGISTERS

The purpose and characteristics of some of the registers must be considered for two reasons:

- Initialising the 82527. To establish correct communications between the 82527 to Host and the 82527 to the CAN bus
- Using the 82527 in a CAN bus for communication. Deciding upon the characteristics of the messages, and the handling of them.

Some consideration of the possible values that can be assigned to each register are listed:

6000 CONTROL REGISTER

The default value after a hardware reset is 01h. This register contains the Init bit. This is initially set to 1 on reset. This means that software initialisation is enabled. "This allows the user to configure the 82527 RAM without the chip participating in any CAN bus transmissions." [13]

- 41 = Write access to config registers; No interrupts; Soft initialisation enabled
- 01 = No write access, rest the same as above

6001 STATUS REGISTER

BOff Bus Off Status 1-busoff, 0- not busoff

Goes busoff when error counter reaches 256. "During busoff, no messages can be received or transmitted. The only way to exit this state is by resetting the Init bit in the Control register." [13]

6002 CPU INTERFACE REGISTER

"The default value of the CPU Interface Register in hardware reset is E1H." [13]
"After hardware reset is 62h."[13]

- 40 = SCLK = XTAL /2 ; Normal Operation, must be loaded after reset; MCLK = SCLK; No power down; No sleep; Normal operation, PIN24 = INT#

6016 MESSAGE CONFIG REGISTERS

- 8C = DLC = 8, DIR = 1 (Transmit, "when TXRqst is set, the MO will be transmitted") XTD = 1 " This message identifier will use an extended 29 bit message identifier."

6026 MESSAGE CONFIG REGISTERS

- 84 = DLC = 8, DIR = 0, "Receive. When TXRqst is set, a remote frame will be transmitted. When a message is received with a matching

identifier, the message will be stored in the message object." XTD = 1 " This message identifier will use an extended 29 bit message identifier."

602F BUS CONFIGURATION REGISTER

48 = " The Input comparator is bypassed and the RX0 input is regarded as the valid bus input, (DcR0 must be set to 0)." ; A logical 1 is interpreted as recessive at the RX0 input. DcT1 is set one hence "Disables the TX1 output driver. This mode is for use with a single wire bus line, or in the case of a differential bus when the two bus lines are shorted together." (Note: Later this setting became 42)

603F BIT TIMING REGISTER

41 = SJW = 1; BRP =1

604F BIT TIMING REGISTER

67 = One sample is used to determine the value of each bit; The sample point is taken after seven quantisation periods; the bit ends seven quantisation periods after the sample point.

ARBITRATION REGISTERS

The identifer bits are assigned to each message in these registers. The identifier starts (ID0) at bit3 of Arbitration Register 3 and ends (ID28) at bit 7 of Arbitration Register 0.

Now that the registers and the effects of assigning certain values to them have been investigated the next step is to attempt to connect the two CAN devices together

CLOCK AND FREQUENCY OF CAN BUS

There are different clocks to be considered for the entire system. There is the system clock which governs the frequency of the internal operations of the 82527, the bit timings and therefore the CAN bus. The memory clock governs the frequency of the external communications between the HC11 and the 82527, via

the parallel interface.

The frequency of the system clock is determined by the following formulas which obtain their values from the various bits in the CPU Interface Register (02h), previously described in this chapter.

F(sysCLK) = F(XTAL) / (1+ DSC bit)

As the XTAL is 16.0 Mhz and the DSC bit is set to one(sys(clk)=XTAL/2) then

F(sysCLK) = 16MHz / (1+1)

F(sysCLK) = 8 Mhz

The maximum frequency of the memory clock is determined from the following formulae:

F(memCLK) = F(SCLK) / (1 + DMC bit)

= F(XTAL) / [(1+ DSC bit) * (1 + DMC bit)]

The DMC bit is set to 0 so

F(memCLK) = 8 / (1+0)

= 8 Mhz

This is the maximum frequency of the memCLK, however the HC11 EVB has a crystal of 8 Mhz which gives the external parallel Address/Data bus a lower frequency of 2MHz.

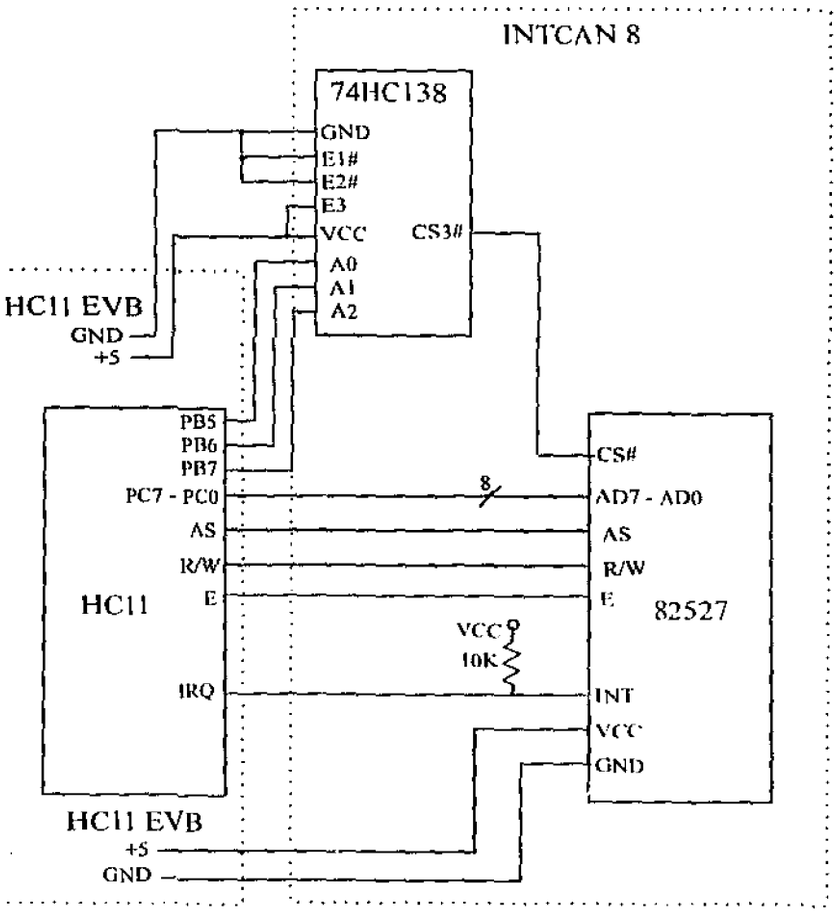
CHAPTER NINE

DESIGN OF SECOND CAN BOARD AND CONNECTION OF THE SYSTEM

ADDITION OF 74HC138 TO DESIGN

The second board was re-designed to deal with the addition of the 74HC138 to the circuit board. The diagram of the schematic circuit design which follows this text, illustrates the connection between the Intcan 8 board and the HC11 EVB. The CAN bus connections or the RS485 driver are not shown.

CONNECTION BETWEEN HC11 AND 82527



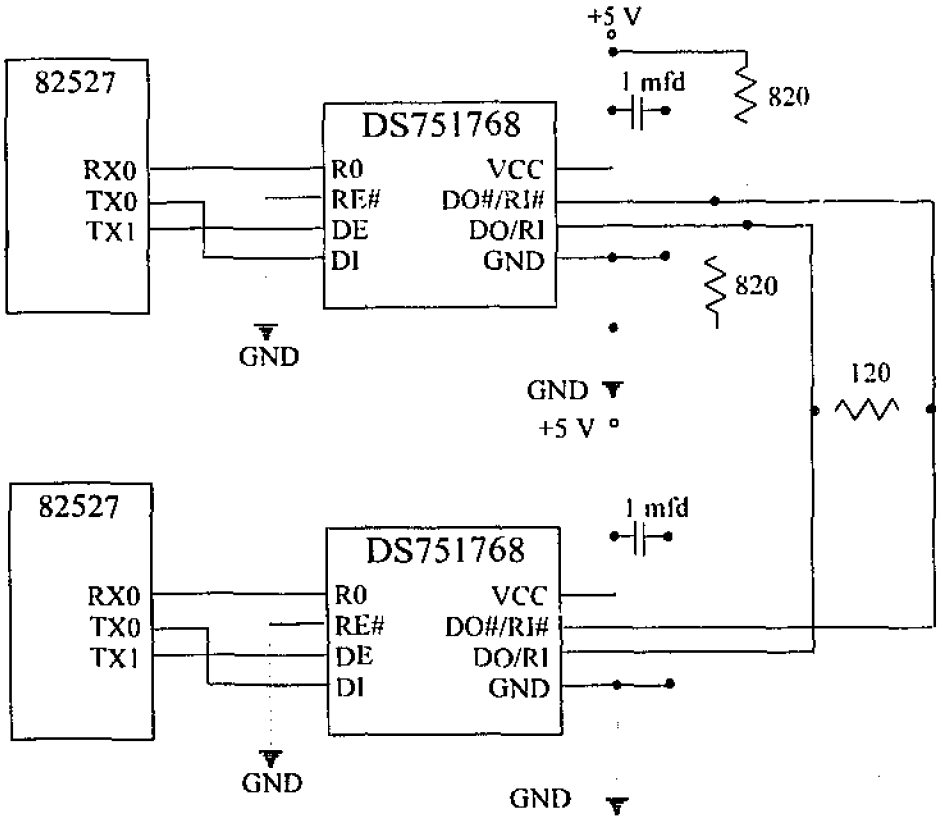
INITIAL CAN BUS DESIGN

The schematic circuit design of the connections between the 82527 and the CAN bus was one which was based on a design sourced on the Internet. The original design shown on the Internet can be found at:

http://www.teleport.com/~efa/cannet.htm.

This design used for this project is illustrated below:

CIRCUIT DIAGRAM - CAN BUS CONNECTION



These two schematic circuit diagrams were then combined to form the complete circuit which made up the original INTCAN 8 circuit board.

After this circuit was completed, and the board manufactured, it was tested for 82527 to Host communications. As these communications tested correctly, the next step was to attempt the Node to Node communications across the CAN bus.

(NOTE : After further investigation this cicuit had to be modified. This modification is dealt with in Chapter 12.

CHAPTER TEN

INITIAL DESIGN OF SOFTWARE TO ESTABLISH NETWORK (CAN to CAN Communications)

The CAN boards were connected together and an attempt was made to establish communication between them.

This chapter deals with the software developed to enable this to happen.

82527 INITIALISATION PROCEDURE

Before the INTEL could operate it needed to be configured. This required many factors to be determined, most importantly the following:

- The type of physical bus
- The timing relationships of the bits
- The frequency of the CAN bus
- The number of samples per bit

Obviously the majority of these are determined by the existing Serial bus, to which the CAN chip will become a new node.

In this case however there was no existing serial bus so many of these decisions had to be decided upon prior to any initialisation process being undertaken.

The relevant registers have been described in the previous chapter

82527 INITIALISATION PROGRAM

```
ldda    #$40
staa    $6002
ldaa    #$48
staa    $6000
ldaa    #$48
staa    $602fh
ldaa    #$41
staa    $603f
```

```
ldaa    #$67
staa    $604f
ldaa    #$01
staa    $6000
```

TRANSMITTING A MESSAGE

The TxRqst bit "is set by the CPU to indicate the MO data should be transmitted. Conditions required to transmit a data frame :

1) Init bit = 0
2) MsgVal bit = 1
3) direction = transmit
4) NewDat bit = 1
5) TsRqst = 1"

"To program a transfer request, the Control 1 register of the message object should have the TxRqst and NewDat bits set to "1". Therefore, this register may be written with the value 66h to initiate a transmission."

RECEIVING A MESSAGE

"Set the CPUUpd and RXIE bits in the message object control register to"1". Set the DIR bit in the message configuration register to "1". A remote frame will be received by this message object."

ACTIONS TAKEN TO RESOLVE PROBLEMS WITH CAN COMMUNICATIONS

Upon connection of the system it was found that the serial CAN Communications was not functioning correctly. The following points highlight the many different steps that were taken in an effort to establish a successful transmission and reception of a MO. Some of the following steps were more logical in their nature than others. This was because any hypothesis, regardless of how likely it was or not the cause of the problem, was tried.

- The termination resistor from the original module was removed. This does affect the impedance of the network. This did not resolve the problem.

- The Status register was examined. The transmission module (original) had an LEC error of 3. This is an acknowledgement error, indicating that the message transmitted by the device was not acknowledged by the other node. The receiver had no listed error.

- The global mask on the receive module was set to all 0's. This was also done on the transmitter, but this also failed to resolve the problem

- The value of the Bus configuration Register 2F, was changed from a value of 48h to 42h, to enable the TX1 and the RX1 to be enabled.

- The Bus configuration Register was changed from 2F to 2A, thus causing TX1 to be enabled and RX1 to be disabled.

- The Arbitration bits on the receive module were changed to 0's

- The Control 1 register for the transmit message was changed from 65 to 66, This was to tell the 82527 that the data for the MO is new, ie NewDat is set to 1- (10).

- It became evident that the previous step had already been carried out and that the error may be solely with the receiver module. The 82527 must be transmitting the message correctly and then setting NewDat to '0'

- Hardware check. Both output pins of the 485 chip on the original board were pulsing. However only one of the accompanying pins on the new module was pulsing. This indicated the likelihood of a possible error with the CAN bus line.

- The transmitter module was reset and as a result the 485 outputs stopped pulsing. The transmitter software was then loaded on the new module and the module was then disconnected from the bus. The new software was then executed and both of the 485 pins started pulsing as they did with the

original module. This indicated that the previous step did not reveal any error.

- The next step was to Re-evaluated the whole board as it was considered that the fault was with the hardware. The circuit was further examined and this revealed that the Board was biasing the low line high and biasing the high line low. In order to make the circuit board match the design as specified by the (netsite), the components were relocated.

- The board still did not work. An IC socket was then used. The RS485 chip was removed from the board and the spare socket was inserted into this place. The two output pins however were bent out. This gave access to these two lines and open-circuited them to the rest of the board circuitry. The same was done to the other board. Next the two outputs were directly connected together.

After the Last Step the CAN Boards Began Communicating.

Although the communications between one CAN node and the other became functional, the connection of one RS485 driver chip to the other, as described above, was only temporarily used until this part of the circuit was redesigned, as can be seen in the section called "CORRECTION OF THE CAN BUS CONNECTIONS" in Chapter 12

MEMORY MAP FOR COMMUNICATIONS

The following Memory Map was used to indicate the values stored at each memory location to allow for an interaction between two different CAN nodes to take place. This map is for the (Remote or Transmitter or Transceiver) :

MEMORY MAP OF THE 82527 - CONFIGURED FOR COMMUNICATION

	00	01	02	03	04	05	06	07	08	09	A	B	C	D	E	F
6000	02		40	X			00	00	00	00	00	78				
6001	55	55	00	00	00	08	8C									
6002	95	56	00	00	00	10	84									42
6003	55	55														41
6004	55	55														67
6005	55	55														
6006	55	55														
6007	55	55														
6008	55	55														
6009	55	55														
600A	55	55														
600B	55	55														
600C	55	55														
600D	55	55														
600E	55	55														
600F	55	55														

NOTE: The X indicates that the contents of this register are read only.
In this table Message Object One (MO1) is configured for the transmit of a remote frame with an identifier of value 1, while MO2 is configured for the reception of a message with an identifier of value 2.

Conclusion

The communications from one CAN node to the other was now able to be carried out. Further software investigation into the methods of message transmission was the next step in the development of the network.

Clearly the fault was with the biasing and impedance of the CAN bus lines. Therefore further investigation of the connections was necessary. While this temporary hardware configuration was used, the 82527 to CAN bus side of the circuitry clearly needed to be corrected properly, the resulting circuit correction can be seen in chapter 12

CHAPTER ELEVEN

USING CAN IN A REAL TIME PROCESS

INTRODUCTION

The CAN network is able to be used in a number of ways. A process may continue normal operation flow and the program may simply look occasionally at one of the message objects in the CAN'S CAM. In this way if a message needs to be dealt with, the program will be able to deal with the message when it is ready. This is fine for many applications which are not time critical. Other messages need to be dealt with as soon as they arrive.

In a real time system, an important message object can be set to trigger the Interrupt line of the microcontroller. The interrupt can be dealt with as soon as the currently executing instruction has finished. The interrupt service routine can then retrieve the new message from the CAN'S CAM and the message can be dealt with. The original program can then resume from the next instruction prior to the interrupt occurring.

ISSUES TO CONSIDER WHEN USING INTERRUPTS

For any interrupts to be generated at all, the 'global interrupts enable bit'(bit IE), in the Control Register (00h) had to be set. However the EIE and the SIE (Status Change Interrupt Enable) bits have not been set as the program is only designed to deal with the generation of interrupts due to the reception of new messages, not CAN bus errors.

The message object that is to generate an interrupt must have the RXIE bit set in it's Control 0 Register.

The 82527 stores in the Interrupt Register (5Fh) a value which is related to the number of the message object which generated it. This message object can be determined by retrieving the contents of the Interrupt Register and subtracting 2 from it. The IntPnd bit of the Control Register 0 for this message object will also be set by the 82527.

HC11 SOLAR TRACKER - CAN CONTROLLED

To demonstrate and successfully test the CAN network it needed to be used in a real situation. The best process that could be chosen was one which would use all the various different data frames and one which used both message objects which produced interrupts and those which did not. The process was to demonstrate that the data field contained within a specific MO would be used to give Uc on the network information of the state of other microcontrollers on the network, and to issue various other Uc's with different commands from the same MO. To illustrate it was decided to link the CAN up to a **SOLAR TRACKER** which was controlled via a HC11 microcontroller.

The design of the solar tracker was the result of the author's previous project design for Microcomputer Systems.
Its accompanying microcontroller, which controlled the tracker, is herein after referred to as the PRIMARY CONTROLLER.

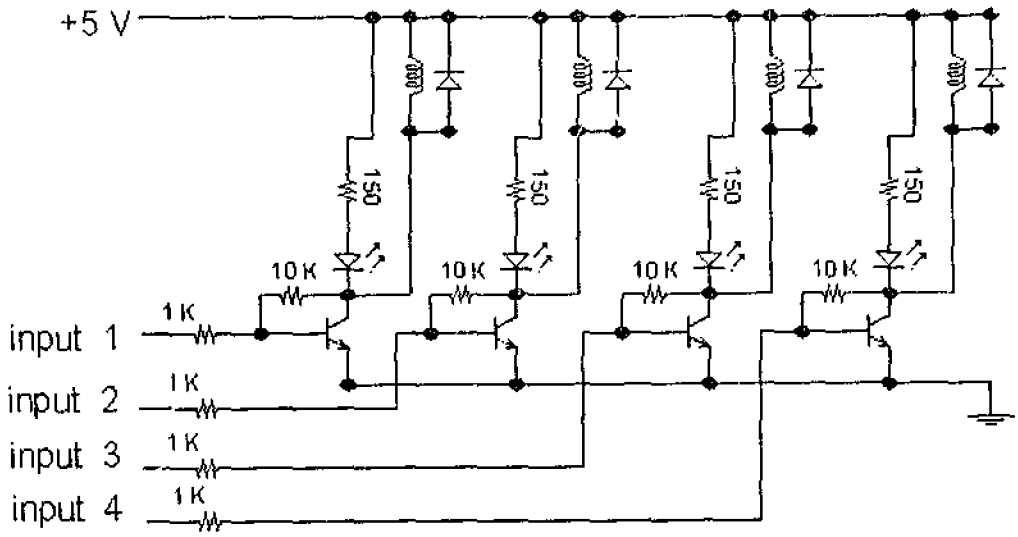
BRIEF DESCRIPTION OF SOLAR TRACKER

The solar tracker has been designed to move a panel with its surface plane normal to the Sun's rays as the Sun moves from the East to the West. The tracker has several features built into it. These features include:

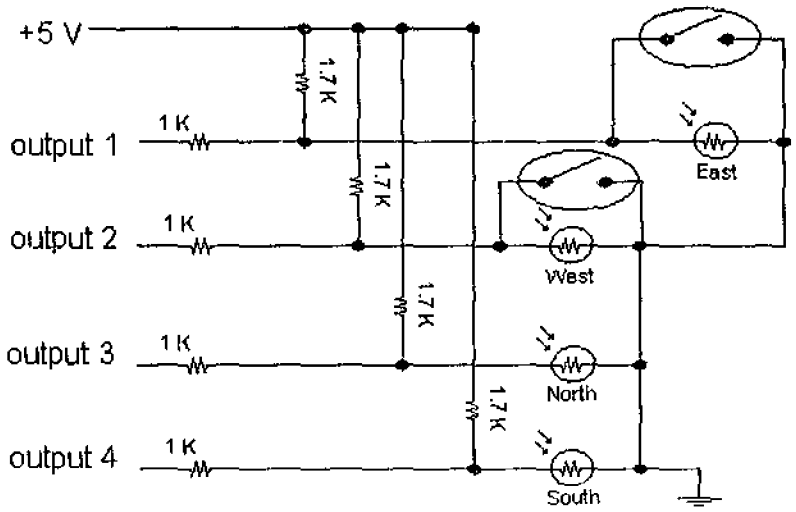
- The ability to align the panel to the East in readiness for sunrise at the close of the day.
- The ability to halt operation of the panel movement due to the pressing of a halt button.
- The ability to detect the maximum angle of operation both East and West.
- The ability to track the Sun through the entire estimate from rental solstice to summer solstice.
- The ability to operate correctly in various sun conditions from bright day to cloudy day.

HARDWARE DESCRIPTION OF SOLAR TRACKER

The circuit diagram below represents the stepper motor controlling circuitry:



The circuit diagram below represents the sunlight sensing circuit, complete with the addition of the maximum tilt angle mercury switches.



SOFTWARE DESCRIPTION OF SOLAR TRACKER

The original software designed for the solar tracker was extensively modified and extended to allow for a networking capability between the Primary and Remote Controllers. This software is dealt with in detail in chapter 15 under the heading of 'Software Design of Primary Controller'

TASK OBJECTIVE

To remotely control the solar tracker via the use of the new CAN network. This is to be implemented by the Remote system being capable of issuing a set of commands overriding those of the Primary Controller and also being able to monitor the status of the solar tracker.

The Primary Controller is the one which is hardwired up to the Solar Tracker module, while the Remote Controller is the one which is located some distance away and whose function is to issue various commands to, and receive data from the Primary Controller.

In order to use the various functions of the CAN network (as previously discussed) the various tasks which the Remote Controller should be able to issue the Primary Controller were :

- **HALT OPERATION**
- **RESUME OPERATION**
- **STOW THE TRACKER FOR MORNING OPERATION**
- **OBTAIN OPERATION DATA OF SOLAR TRACKER**

The HALT function is to demonstrate the use of MO's which produce interrupts upon reception. The RESUME and STOW functions are to demonstrate that one MO can be used to issue different tasks to the Uc's on the network, and that although these commands can be responded to when the Uc is ready, they do not interrupt the Uc. The fourth operation POSDATA, is to demonstrate how a Remote Data Frame can be used to request data from other Uc's on the Network.

All of these tasks are to be issued via the use of momentary contact buttons on the Remote Controller. The operation which is to receive data back from the solar tracker Primary Controller is the fourth selection. This data is then to be displayed on the LCD display on the remote HC11.

A more detailed description of these tasks follows:

- **HALT OPERATION**

When this command is issued to the Primary Controller, the tracker is to cease operation immediately. The Primary Controller will then await further instructions from the Remote Controller. For the Primary Controller to respond immediately to this request indicates that interrupts must be used. This means the message object which is chosen for this purpose is to have the interrupt generation upon receipt of MO flag set (RXIE). The Primary Controller could then analyse the data received by this MO and carry out the issued instructions. For this project however it will not use the data in this MO, it will simply go into an algorithm waiting for further instructions.

- **RESUME AND STOW OPERATIONS**

The Resume and Stow commands will only take effect when the Primary Controller has previously been issued with a HALT command. These instructions will be issued to the Uc using the same MO's. This means that once the MO has been received the data bytes of the message must be analysed. The Primary Controller is to respond appropriately to the command. As the reception of this MO is not to generate an interrupt the MO must frequently be looked at from within the controllers algorithm. Compared to the Interrupt generating MO's, this is an intensive use of the Uc. In this example however this is not a problem as the Uc is not attending to any other operation at this stage. A 'passive' reception of the MO like this for the initial HALT operation would not be acceptable for this reason as well as the fact the HALT command is to be responded to immediately upon it's reception.

- **POSITIONAL DATA OPERATION**

This operation is to demonstrate and test the use of remote data frames. This command is to be issued be the remote controller at any time, not only when the tracker has been halted. The command is not intended to affect the operation of the Primary Controller in any way. The remote frame whose reception is set so as to not generate an interrupt, is an ideal MO for this situation. A set of nine different data bytes contained within the one Message Object are to be received from the controller Uc. These different Data bytes are to indicate the stage of

operation of the controller.

This is achieved by the insertion in the Solar Tracker algorithm of code segments whose purpose is to update the data within this MO. This is done at set times within the algorithm and is not influenced by the remote Uc in any way. This ensures that there is no interference in the operation of the tracker by the reception of this remote data request MO. It should be noted that Remote Request MO's do not need to work in this way, they can in fact, generate an interrupt upon reception or transmission.

IMPLEMENTATION CONSIDERATIONS

Note: The various messages used in this project are in actuality distinguished by their individual identity or arbitration bits. These messages can be located in any one of the Content Accessible Memory locations. However to simplify the descriptions of the various messages used in this project, it was decided to leave each one of the various messages in its own message object location. Thus when a message object number is referred to, it should be regarded as a reference to a particular message.

Firstly the message objects need to be determined. Keeping in mind that the lower the number of the MO, the higher it's transmission priority over the network. The following table lists the Message Objects and their characteristics, in order of priority.

MESSAGE OBJECT No	FUNCTION	GENERATE INTERRUPTS
1	For obtaining the positional data - Solar Tracker	No
2	Send the halt command to Solar Tracker	Yes
3	Indicated whether to Resume or Stow	No

The decision whether to use all eight data bytes or simply to use one has to be made. There are only a small list of commands and for this reason only one data byte needs to be used. However to test the functionality of the CAN network all eight bytes will be transmitted, with only data byte 7 being used.

The speed of the network has been chosen to be 125Kbps. This is substantially slower than the maximum of 1Mbps which it is capable due to the small length of the serial bus wire. It was chosen however because in a 'real' situation the length of this cable would be significantly greater (Thereby reducing the maximum data rates).

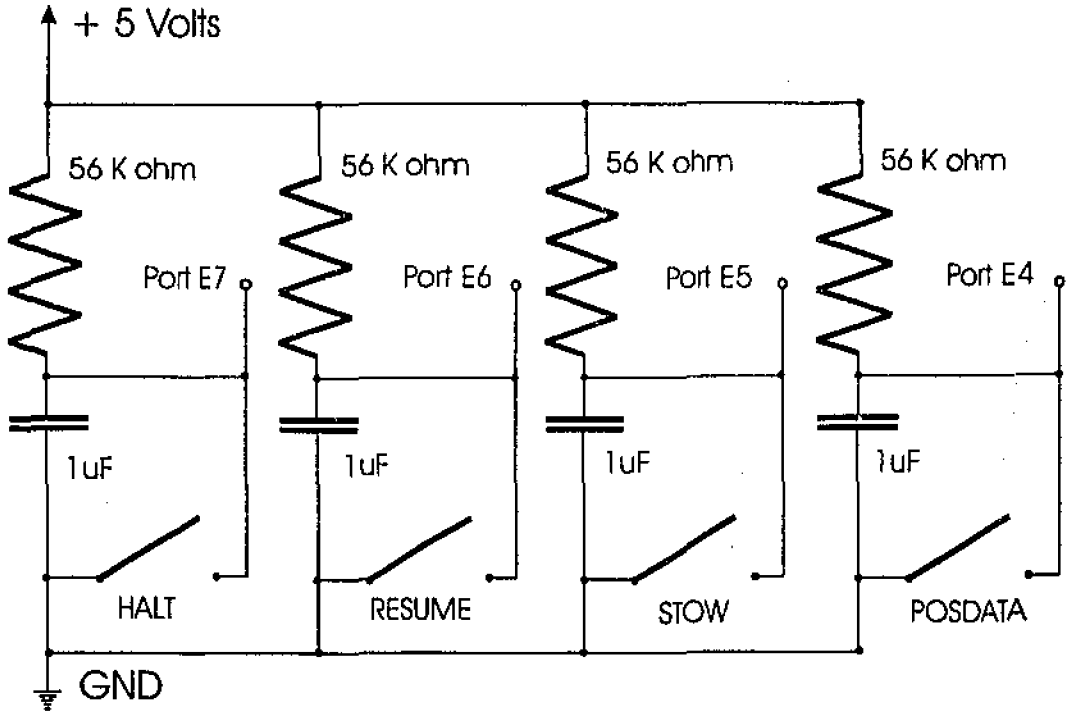
COMMAND ISSUING BUTTONS (HARDWARE & SOFTWARE VIEW)

As there are to be 4 major different functions which the network will use, for simplicity there will be four buttons. These buttons are to be connected to the remote HC11. The obvious choice of button mechanism is a momentary contact, rather than a push on, push off type. These buttons will be connected directly to the PORT E lines as indicated in the table below:

BUTTON	PORT E LINE	BINARY VALUE	HEX VALUE
HALT	7	10000000	80
RESUME	6	01000000	40
STOW	5	00100000	20
POSDATA	4	00010000	10

The algorithm for checking the state of these buttons will need to monitor Port E. Depending on the resultant value seen at Port E, the exact button pressed will be known.

When the button is depressed the button will temporarily connect the port line to ground. However as the button monitoring algorithm is running in a loop, it is theoretically possible to press and release one of the buttons so quickly that the algorithm may not be taking a sample of the port while this occurs. This means that the activation of the button may not be noticed. To avoid this situation and to help eliminate any bounce problems the following circuit was implemented.



Notice that the activation of one of the buttons actually sends the input line to ground or 0. When the button is the normally open position, the input line has a high input due to the pull-up resistor. The capacitor in the circuit is to help keep the input level in a low level for a short period of time once the button has been depressed. This is according to the T=CR, this gives a time constant of around 0.05s or 50 ms which is ample time for the button depress to be recognised. The full diagram of the button circuitry is shown above

OBTAINING POSITIONAL DATA FROM TRACKER

By looking at the last data byte in message object 1, the operation of the tracker can be determined. This is illustrated in the following table:

DATA VALUE	MEANING	DATA VALUE	MEANING
20	Full East by remote	70	Halt - remote
30	stall operation - controller	80	Moving East - controller
40	normal tracking - controller	90	East position - controller
50	East position by remote	A0	Full East - controller
60	Moving East - remote		

Note : The East position is the position the tracker is in when the tracker has gone full East and then backed off until the East tilt sensor has been deactivated. The Full East position is when the tracker has gone fully East and has not yet backed off until the East tilt sensor has been deactivated.

When the POSDATA command is issued by the remote user, the remote algorithm will tell the remote CAN chip to issue a remote data frame for MO1. The remote algorithm will then look at the data stored in the last byte of the data field of MO 1 in the controllers CAM. The data in this byte will have one of 9 values, the algorithm will then display the operation of the tracker by matching this value to an operation, in accordance with the previous table.

LCD DISPLAY OF MESSAGE

The LCD display chosen for the purpose of displaying the tracking operation of the Primary Controller is the Powertip . This device is a two line display which is serially controlled by a 74HC251 chip, incorporated in the HC11 EVB. The memory address for this device is located at 8000. There is no CS line running to the device from the 74HC138. Instead address lines A15, A13, and A14 are connected directly to inputs S0, S1 and S2 respectively on the 74HC251.
The 74HC251 is an 8 input multiplexor, however all the digital inputs with the exception of input I1 are connected to ground. The only non zero multiplexor input is I1 and this is selected when S0 is 1, S1 and S2 are zero. Due to this configuration only address values in the range of 8000 to 9FFF may produce a useful (non-zero) R/W input to the LCD driver.
When a valid address is on the bus, the E clock of the HC11 (I1 input) will be selected and driven through the Z output to the R/W of the LCD driver.
The LCD driver functions in the following manner;
8000 is used as both the locations for the Instruction / Control register and the Address register / Counter busy flag. This location can be used for two purposes because one function the Address Counter/Busy flag is a read only register while the Instruction/Control register is a write only register. Memory location 8001 is used for the Data register and it is a read / write register. The Address counter

uses bits 0-6 to display the address of the internal cursor, while bit 7 is the busy flag, which when high indicates that the LCD driver is busy.
The driver accepts data in an American Standard Code for Information Interchange (ASCII) format. The control characters used in the program for the Remote Uc are as follows:

$80 : Returns the internal cursor to the top line home position
$C0 : Returns the internal cursor to the home position of the bottom line
$38 : Initialises the LCD driver to two line operation mode
$0C : Resets the LCD driver

More programming details of the LCD driver are contained in the notes pertaining to the remote Uc program.

CHAPTER TWELVE

COMPLETE ELECTRONIC DESIGN OF THE TRACKER / CAN NETWORK

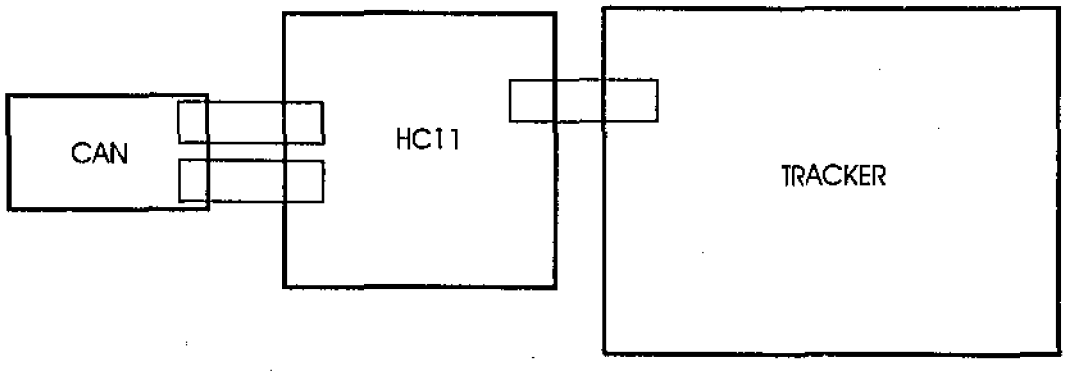
The design of the whole electronics of the project can naturally be broken up into two main sections, the Primary Uc and the Remote Uc. The only connection between these two sections is via the two line serial bus.

HARDWARE DESIGN OF THE PRIMARY UNIT

The complete configuration of the remote system consists of:

- The CAN board
- The HC11 EVB
- The Chip Select Adaptor board
- The Solar Tracker

The layout of this system can be seen in the following diagram.



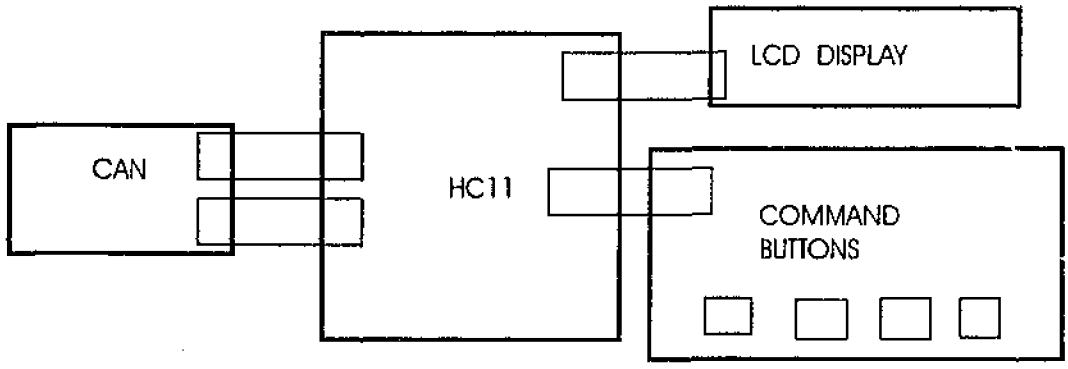
LAYOUT OF PRIMARY SYSTEM

HARDWARE DESIGN OF THE REMOTE UNIT

The complete configuration of the remote system consists of:

- The CAN board
- The HC11 EVB
- The LCD display module
- The 4 command issuing buttons.

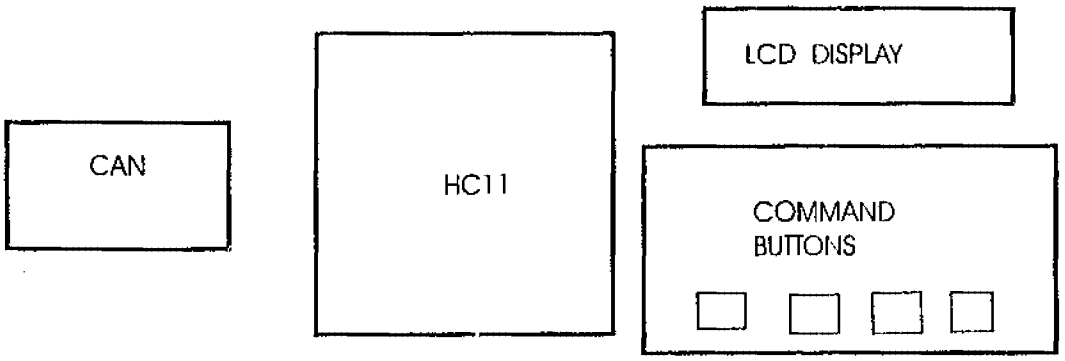
Because this system works as a complete unit, it will enable the user to command the Primary Controller to either accept various commands or to give the positional Data via the CAN bus. The layout of this system is shown in the following diagram;



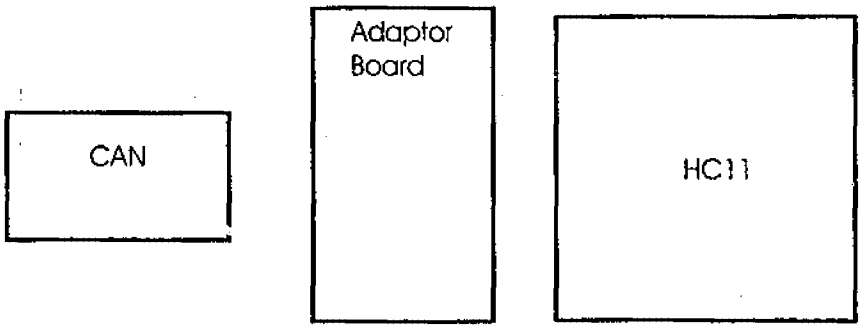
LAYOUT OF REMOTE SYSTEM

These two distinct functioning units were then connected together to form the complete system which is represented by a block diagram over page.

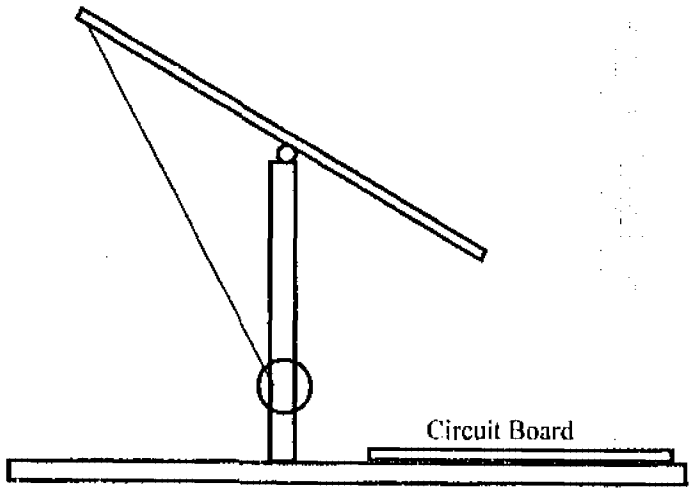
DIAGRAMMATIC REPRESENTATION OF THE MAJOR COMPONENTS OF THE COMPLETE SYSTEM



REMOTE CONTROLLER



PRIMARY CONTROLLER

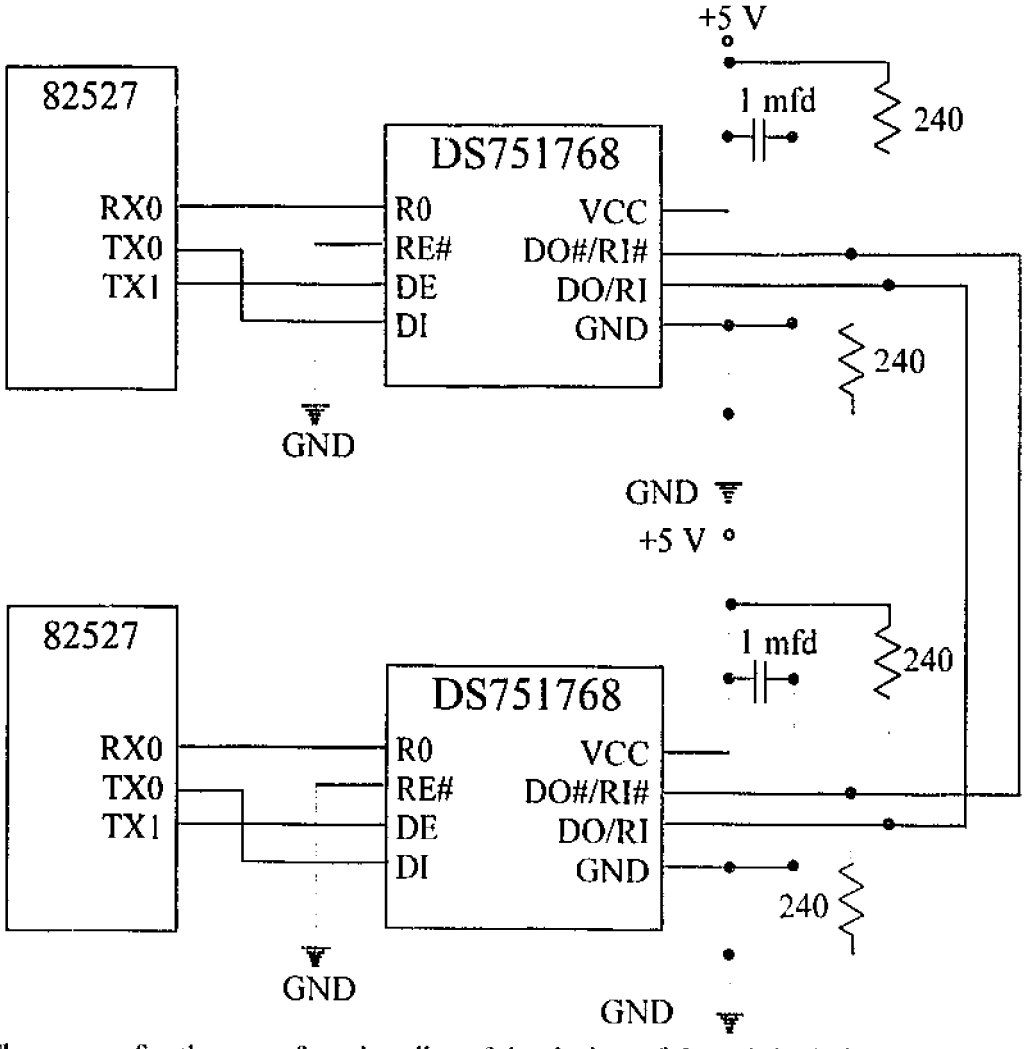


SOLAR TRACKER

CORRECTION OF THE CAN BUS CONNECTIONS

As indicated in chapter 10, the circuit design of the CAN bus connections needed to be reconsidered. The CAN to CAN communications up to this point had been working as a straight through connection from one RS485 driver to the other. This did not allow for a correct functioning of the system as no thought had been given to the correct biasing of the lines. The new circuit was designed acknowledging that the two differential bus lines needed to be correctly biased and have the correct impedance. This revised circuit showing the connections from the 82527 to the RS485 driver chip, down to the CAN bus are shown below:

CIRCUIT CONNECTION BETWEEN 82527, RS485 DRIVER AND CAN BUS



The reason for the non- functionality of the design of the original circuit is because of the incorrect values used for the biasing resistors, and the exceptionally high value of the resistor used for the impedance matching of the serial bus line.

CHAPTER THIRTEEN

SOFTWARE DESIGN OF REMOTE CONTROLLER

INTRODUCTION

This chapter deals with the program which controls the REMOTE unit. The purpose of the remote Uc is to receive positional data from the tacker and to issue commands to the Primary Controller.

The program consists of the mainline algorithm and a procedure which deals with the displaying of positional data on the LCD unit.

Both of these two main sections of code are described here and flowcharts are included in the description for conceptual ease.

The complete Assembly language listing of the 'Remote Controller' is included at the back of the chapter and detailed comments are attached to the code for further clarification of the operation.

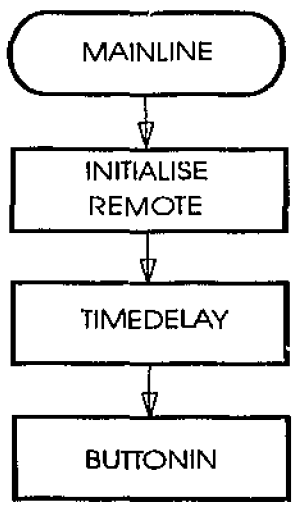
MAIN ALGORITHM

This procedure begins operation by initialising the port, to which the buttons are connected. Then the data from this port is retrieved. The algorithm can then, depending upon the value retrieved and the button pressed operate in one of five different ways:

- Transmitting halt message.
- Transmit resume message.
- Transmit stall message.
- Send remote frame MO1.
- Wait for another button to be pressed.

The above operations have previously been discussed in detail and will be further analysed in the following flow charts related to each of these operations.

The flow chart for the MAINLINE of the Remote operation is shown below:



INITIALISE REMOTE

The 82527 is configured to set up correct communications between the 82527 and the HC11, and the Serial bus communications between the CAN to CAN communications. This is carried out by the following sequence of steps:

- Set CPU interface SCLK = XTAL /2 and MCLK = SCLK
- Enable write access to configuration registers to allow the following to be implemented:
  - Enable TX1 Transmitter
  - Bypass the input comparator
  - Set the SJW and Baud Rate Prescalar to 1
  - Set to 1 sample period per bit and make TSEG1 = 7 and TSEG2 =6
  - Disable any further access to this register
- Set all MO's Config 0 and Config 1 registers to #$55 - Reset
- Set the Global Mask to 'Don't Care'.(accept any identifer), except the last four bits of every message identifer
- Set for extended message objects
- Set Message Object 1 to Receive; Message Object 2 to Transmit; and Message Object 3 to Transmit
- Set the identifier bits of each Message Object to 1 for MO1, 2 for MO3 and 3 for MO3
- End the initialisation procedure of the 82527

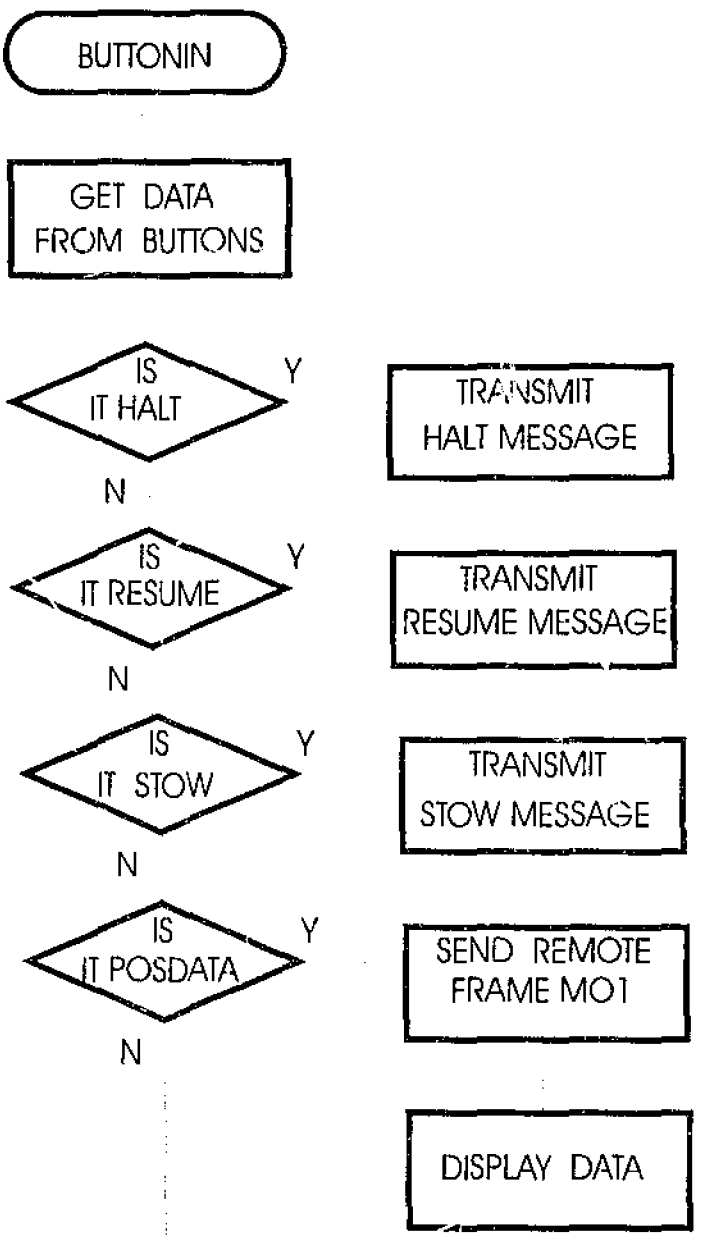
TIMEDELAY

The purpose of the time delay is to cause a brief delay after the initialisation of the Host to 82527 and the CAN to CAN communications. This delay ensures sufficient time for the correct function of the system.

This procedure loads the accumulator with #$FFFFh (65535 decimal). As each delay loop takes `2.5uS to complete, it corresponds to a delay of 2.5uS X 65535 = 163mS delay every time this section of code is executed.

BUTTONIN

The following flow chart illustrates the operation of the Buttonin Algorithm.



GET DATA FROM BUTTONS

The hardware is designed such that when a button is pressed the input to Port E will go low '0'. The data is retrieved from Port E. To convert this data to a form which is conceptually easy to understand, and to make the data easy for use with the compare instructions which follow this routine, the data was manipulated as follows. An 'Exclusive Or' with FFH operation is performed on the data. Then an 'And' operation with F0h is performed to change the last four bits of the data to 0. The button which was pressed is now represented by a '1' in one of the bit locations 4,5,6, or 7. For an example, if button 2 was pressed, data would be seen at Port E as 11010000 and this data would be manipulated as follows:

□	Original data	11010000
□	Exclusive Or	11111111
□	Gives	00101111
□	And with	11110000
□	Gives a result of	00100000

IS IT HALT

The Manipulated Port E data is compared to 80h. If it is then the procedure 'Transmit Halt Message' is executed, or else the flow of execution continues to procedure 'Is It Resume'.

IS IT RESUME

The Manipulated Port E data is compared to 40h. If it is then the procedure 'Transmit Resume Message' is executed, or else the flow of execution continues to procedure 'Is It Stow'.

IS IT STOW

The Manipulated Port E data is compared to 20h. If it is then the procedure 'Transmit Stow Message' is executed, or else the flow of execution continues to procedure 'Is It POSDATA'.

IS IT POSDATA

The Manipulated Port E data is compared to 10h. If it is then the code segment 'Send Remote Frame MO1' is executed, or else the flow jumps back to 'Get Data From Buttons'.

TRANSMIT HALT MESSAGE

Unlike the previous routines, no code needs to be sent with this MO as the function of the Message is self evident, due to the interrupt it generates when it is received. The steps of this code are listed below:

- MO 2 is made invalid so that it can be updated
- The TxRqst, CPUUpd and the NewDat bits are set.
- The message is set to valid again so that it is ready for transmission

TRANSMIT RESUME MESSAGE

The following steps are performed by this code segment:

- MO3 is made invalid so that it can be updated
- Data Byte 8 of MO3 is altered to 55h
- The MO is made valid and therefore is ready for transmission

TRANSMIT STOW MESSAGE

The following steps are performed by this code segment:

- MO3 is made invalid so that it can be updated
- Data Byte 8 of MO3 is altered to FFH
- The MO is made valid and therefore is ready for transmission

SEND REMOTE FRAME MO1

Unlike the previous code segments, no data is stored in the MO1 by this procedure before being transmitted, as it is a remote Frame, which is requesting data. The following steps are performed by this code segment:

- MO1 is made invalid so that it can be updated
- The MO is made valid and therefore is ready for transmission

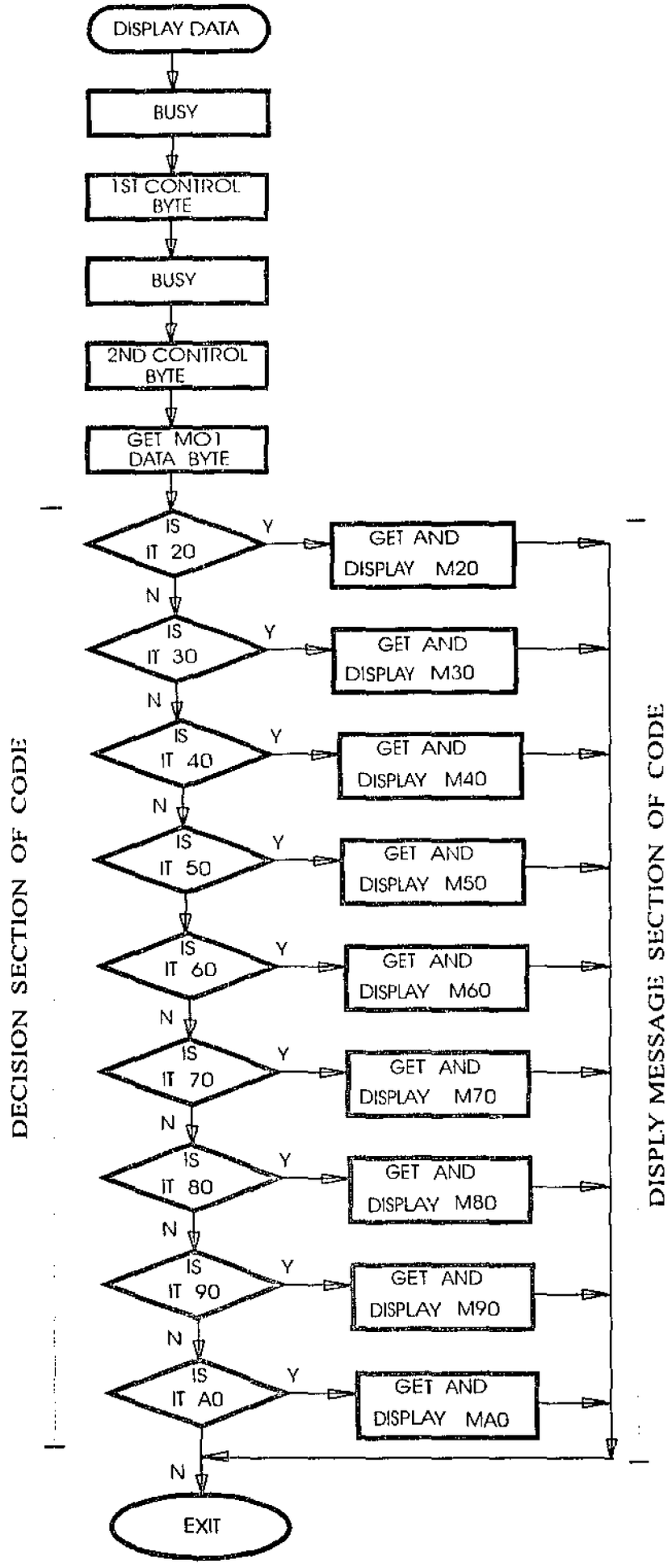
Following the execution of this procedure the 'Display Data' code segment is executed.

DISPLAY DATA ALGORITHM

This procedure begins operation by sending two different control bytes into the LCD display unit. Each of these are :

- The first command initialises the LCD driver to two line operation
- The second resets the LCD driver.
- The algorithm then retrieves the data from the MO1 and depending upon it's value it decides which message to display. This is accomplished by a series of Compare (cmpa) instructions

This Algorithm is represented by the Flow Chart over the page.


DISPLAY DATA
BUSY
1ST CONTROL BYTE
BUSY
2ND CONTROL BYTE
GET MO1 DATA BYTE
IS IT 20
Y
GET AND DISPLAY M20
N
IS IT 30
Y
GET AND DISPLAY M30
N
IS IT 40
Y
GET AND DISPLAY M40
N
IS IT 50
Y
GET AND DISPLAY M50
IS IT 60
Y
GET AND DISPLAY M60
N
IS IT 70
Y
GET AND DISPLAY M70
N
IS IT 80
Y
GET AND DISPLAY M80
N
IS IT 90
Y
GET AND DISPLAY M90
N
IS IT A0
Y
GET AND DISPLAY MA0
N
EXIT
DECISION SECTION OF CODE
DISPLY MESSAGE SECTION OF CODE

BUSY

Waits until the LCD driver is ready before proceeding to the rest of the program.

1ST CONTROL BYTE

The first control byte is loaded into the LCD driver to tell it to operate in tow line operation mode.

2ND CONTROL BYTE

The second control byte resets the LCD driver

GET MO1 DATA BYTE

The message byte is retrieved from MO1 (601Eh).

DECISION SECTION OF CODE

These 9 choices decide, depending on the value of the message which message to display. The data retrieved from MO1 is compared to a series of values until a match is found.

DISPLAY MESSAGE SECTION OF CODE

These 9 functions all display the message relating to the value retrieved from DF00h.

The program listing of the REMOTE MICROCONTROLLER follows overpage:

PROGRAM FOR THE REMOTE MICROCONTROLLER

```
**************************************************************
*This is the Transmit program                                *
**************************************************************
        org    $c000              ;Starting  Address of Program
begin:    jsr    Initialise      ;Initialise 82527
        jsr    timedelay          ;Timedelay
        jsr    timedelay          ;Timedelay
        jsr    buttonin           ;Deal with Message Objects
        jmp    begin              ;End of mainloop
        end
**************************************************************
*          Initialise the 82527 registers                    *
**************************************************************
Initialise:   ldaa   #$40         ;Set CPU interface SCLK = XTAL/2
        staa   $6002              ;and MCLK = SCLK
        ldaa   #$41               ;Set the CCE = 1 so write access
        staa   $6000              ;to config registers
        ldaa   #$42               ;Bypasses input comparator,one is
        staa   $602f              ;recessive. TX1 is enabled
        ldaa   #$41               ;Gives SJW =1
        staa   $603f              ;and BRP = 1
        ldaa   #$67               ;Spl=0:one sample period, and
        staa   $604f              ;TSEG2=6. and TSEG1=7
        ldaa   #$01               ;CCE=0:CPU has no write access to the
        staa   $6000              ;configuration registers
        ldaa   #$55               ;Reset control reg 0 and 1
        staa   $6010              ;in all the messages.
        staa   $6020              ;All the messages are
        staa   $6030              ;set to invalid.
        staa   $6040        ;
        staa   $6050        ;
        staa   $6060        ;
        staa   $6070        ;
        staa   $6080        ;
        staa   $6090        ;
        staa   $60A0        ;
        staa   $60B0        ;
        staa   $60C0        ;
        staa   $60D0        ;
        staa   $60E0        ;
        staa   $60D0        ;
        staa   $60E0        ;
        staa   $60F0        ;
        staa   $6011        ;
        staa   $6021        ;
```

```
staa  $6031      ;
staa  $6041      ;
staa  $6051      ;
staa  $6061      ;
staa  $6071      ;
staa  $6081      ;
staa  $6091      ;
staa  $60A1      ;
staa  $60B1      ;
staa  $60C1      ;
staa  $60D1      ;
staa  $60E1      ;
staa  $60D1      ;
staa  $60E1      ;
staa  $60F1      ;
ldaa  #$00             ;Set global Mask Standard and
staa  $6006            ;extended to 00 which is
staa  $6007            ;'don't care'
staa  $6008            ;
staa  $6009            ; -Using extended mask registers-
staa  $600a            ;
ldaa  #%01111000       ;Set the last four bits of mask to
staa  $600b            ;'care'
ldaa  #$84             ;Transmit and receive using extended
staa  $6016            ;Message 1 is receive &
ldaa  #$8c             ;Message 2 is transmit &
staa  $6026            ;
staa  $6036            ;Message 3 is transmit
ldaa  #$00             ;Load the arbitration registers for
staa  $6012            ;Messages 1 - make '1'
staa  $6013            ;
staa  $6014            ;
ldaa  #%00001000       ;
staa  $6015            ;
ldaa  #$00             ;load the arbitration registers for
staa  $6022            ;message 2 - make '2'
staa  $6023            ;
staa  $6024            ;
ldaa  #%00010000       ;
staa  $6025            ;
ldaa  #$00             ;load the arbitration registers for
staa  $6032            ;message 3 - make '3'
staa  $6033            ;
staa  $6034            ;
ldaa  #%00011000       ;
staa  $6035            ;
ldaa  #$00             ;Set Init bit to 0
staa  $6000            ;otherwise no coms
rts                    ;End of initialisation
```

```
*****************************************************************
*        Retrieval and sending of Message Objects               *
*****************************************************************
buttonin:    ldaa   $100a          ;Port E data
             eora   #$ff           ;Not the accumulator
             anda   #$f0           ;And with 11110000
             cmpa   #$80h          ;Is it halt
             beq    halt           ;Branch if equal
             cmpa   #$40           ;Is it Resume
             beq    resume         ;Branch if equal
             cmpa   #$20           ;Is it Stow
             beq    stow           ;Branch if equal
             cmpa   #$10           ;Is is Posdata
             beq    posdata        ;Branch if equal
             jmp    buttonin       ;Loop for valid input
halt:        ldaa   #$55           ;Transmit MO2
             staa   $6020          ;Make MO invalid for update
             ldaa   #$ff           ;this data not needed
             staa   $602e          ;
             ldaa   #$66           ;Set TxRqst = 1 & CPUUpd = 1 &
             staa   $6021          ;NewDat = 1 & RmtPnd = 0
             ldaa   #$95           ;Config0 > MSGVAL=1 or no
             staa   $6020          ;coms as message not valid BW !!!!!
             jmp    buttonin       ;
resume:      ldaa   #$55           ;Make Mo invalid for update
             staa   $6030          ;
             ldaa   #$55           ;Set data byte 8 of MO3
             staa   $603e          ;to 55h
             ldaa   #$66           ;Set TxRqst = 1 & CPUUpd = 1 &
             staa   $6031          ;NewDat = 1 & RmtPnd = 0
             ldaa   #$95           ;Config0 > MSGVAL=1 or no
             staa   $6030          ;coms as message not valid BW !!!!!
             jmp    buttonin       ;
stow:        ldaa   #$55           ;Make Mo invailid for update
             staa   $6030          ;
             ldaa   #$ff           ;Set data byte 8 of MO3
             staa   $603e          ;to FFh
             ldaa   #$66           ;Set TxRqst = 1 & CPUUpd = 1 &
             staa   $6031          ;NewDat = 1 & RmtPnd = 0
             ldaa   #$95           ;Config0 > MSGVAL=1 or no
             staa   $6030          ;coms as message not valid BW !!!!!
             jmp    buttonin       ;
posdata:     ldaa   #$55           ;Make MO invalid so it can be updated
             staa   $6010          ;
             ldaa   #$66           ;Set TxRqst = 1 &
             staa   $6011          ;Send remote frame
             ldaa   #$95           ;
             staa   $6010          ;MsgVal again
             nop                   ;LCD program to go here
```

```
*****************************************************************
*          Lcd program                                          *
*****************************************************************
lcd:      jsr   busy
          ldaa  #$38              ;First Control Byte
          staa  $8000             ;
          jsr   busy              ;Is the display busy
          ldaa  #$0c              ;Second control byte
          staa  $8000             ;
*****************************************************************
*          Which Message to Display                             *
*****************************************************************
which:     ldaa  $601e
twenty:    cmpa  #$20
          bne   thirty
          ldx   #m120             ;First ascii character was d000
          jsr   top line
          ldx   #m220
          jmp   nextlin
thirty:    cmpa  #$30
          bne   forty
          ldx   #m130             ;First ascii character was d000
          jsr   top line
          ldx   #m230
          jmp   nextlin
forty:     cmpa  #$40
          bne   fifty
          ldx   #m140             ;First ascii character was d000
          jsr   top line
          ldx   #m240
          jmp   nextlin
fifty:     cmpa  #$50
          bne   sixty
          ldx   #m150             ;First ascii character was d000
          jsr   top line
          ldx   #m250
          jmp   nextlin
sixty:     cmpa  #$60
          bne   seventy
          ldx   #m160             ;First ascii character was d000
          jsr   top line
          ldx   #m260
          jmp   nextlin
seventy:   cmpa  #$70
          bne   eighty
          ldx   #m170             ;First ascii character was d000
          jsr   top line
          ldx   #m270
          jmp   nextlin
```

```
eighty      cmpa  #$80
            bne   ninety
            ldx   #m180          ;First ascii character was d000
            jsr   top line
            ldx   #m280
            jmp   nextlin
ninety:     cmpa  #$90
            bne   a0
            ldx   #m190          ;First ascii character was d000
            jsr   top line
            ldx   #m290
            jmp   nextlin
a0:         cmpa  #$a0
            bne   a1
            ldx   #m1a0          ;First ascii character was d000
            jsr   top line
            ldx   #m2a0
            jmp   nextlin
a1:         jmp   buttonin
*****************************************************************
top line:   jsr   busy           ;Is the display busy
            ldaa  #$80           ;Top line of display
            staa  $8000          ;
            jsr   busy
output:     ldaa  $0,x           ;get first character
            staa  $8001          ;Display it
            cmpa  #$31           ;Check for end of line
            beq   finish         ;Finish line one
            jsr   busy
            inx
            jmp   output
*****************************************************************
nextlin:    jsr   busy
            ldaa  #$c0           ;next line control character
            staa  $8000
            jsr   busy
out2:       ldaa  $0,x           ;get first character
            staa  $8001          ;display it
            cmpa  #$31           ;Check for end of line 2
            beq   finish2        ;Finish line two
            jsr   busy
            inx
            jmp   out2
*****************************************************************
finish:     rts
*****************************************************************
finish2:    jmp   buttonin
*****************************************************************
delaytim:   ldy   #$2000         ;time delay routine
```

```
here:     dey                    ;
          bne   here             ;loop for delay
          rts                    ;end delay procedure
*****************************************************************
busy:     ldaa  $8000            ;If busy keep looping
          anda  #$80             ;
          cmpa  #$80             ;
          beq   busy             ;
          jsr   delaytim         ;
          rts                    ;end busy wait
*****************************************************************
*         Timedelay                        *
*****************************************************************
timedelay:  ldx   #$ffff
loopy:    dex
          bne   loopy
          rts
*****************************************************************
m120      fcc   " beg of 20   1"
m220      fcc   " end of 20   1"
m130      fcc   " beg of 30   1"
m230      fcc   " end of 30   1"
m140      fcc   " beg of 40   1"
m240      fcc   " end of 40   1"
m150      fcc   " beg of 50   1"
m250      fcc   " end of 50   1"
m160      fcc   " beg of 60   1"
m260      fcc   " end of 60   1"
m170      fcc   " beg of 70   1"
m270      fcc   " end of 70   1"
m180      fcc   " beg of 80   1"
m280      fcc   " end of 80   1"
m190      fcc   " beg of 90   1"
m290      fcc   " end of 90   1"
m1a0      fcc   " beg of a0   1"
m2a0      fcc   " end of a0   1"
          end
```

CHAPTER FOURTEEN

SOFTWARE DESIGN OF PRIMARY CONTROLLER

CONTROLLER ALGORITHM

The purpose of the Primary Controller algorithm is two-fold :

1) To control the tracker in an autonomous mode
2) To accept commands from the network and to give details of the operational status of the tracker to the network.

This has been achieved by the adaption of a program which was written for the HC11 to control the tracker in an autonomous way. This program required extensive modification, however virtually any program could be modified in a similar way. The new procedures are described in this section and a flow chart indicating the execution flow for each procedure is included. and in addition, individual sections of the code are described with respect to the function of each.

The control algorithm has been designed to execute and run the Tracker in an autonomous mode, but on reception of an interrupt from the 82527 (which indicates that a new MO2 message has arrived), the Interrupt service routine will execute to deal with the command message. Once this has been dealt with, the execution of the original Mainline program will continue from where it was, before the reception of the interrupt.

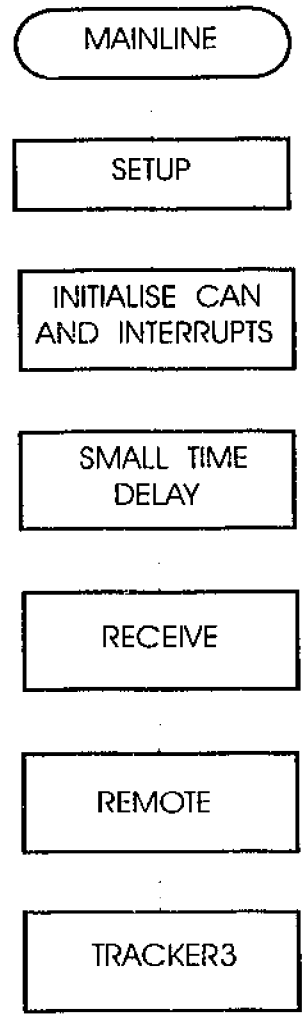
Note for a detailed description of the program, see the detailed comments attached to the program listings.

MAINLINE

The original solar tracking program has been altered to :

- Open communications between the Uc and the 82527
- Initialise the 82527 chip for serial bus communications
- Update the MO's in the CAN'sCAM to indicate the current operation being undertaken by the Uc
- Respond to New Commands from the Remote Controller

The execution of the MAINLINE program is split up into sections of code who's functions and organisation are illustrated in the following diagram.



The functions of the various sections of code in the previous diagram are described here under.

SETUP

This section of code organises the HC11 to deal with the reception of interrupts which it receives from the 82527 whenever an MO2 message is received. This is achieved by the following steps:

- Alter the INT Vector Address to 'flashagain'
- Set the HC11 to enable the reception of interrupts

INITIALISE

This section of code deals with the initialising of the 82527 to deal with the Host to Interface communications and the network CAN to CAN communications. The functions carried out in this section of code are listed below:

- Set CPU interface SCLK = XTAL /2 and MCLK = SCLK
- Enable write access to configuration registers to allow the following to be implemented:
  - Enable TX1 Transmitter
  - Bypass the input comparator
  - Set the SJW and Baud Rate Prescalar to 1
  - Set to 1 sample period per bit and make TSEG1 = 7 and TSEG2 =6
  - Disable any further access to this register
- Set all MO's Config 0 and Config 1 registers to #$55 - Reset
- Set the Global Mask to 'Don't Care',(accept any identifer), except the last four bits of every message identifer
- Set for extended message objects
- Set Message Object 1 to transmit; Message Object 2 to receive; and Message Object 3 to Receive
- Set the identifier bits of each Message Object to 1 for MO1, 2 for MO3 and 3 for MO3
- End the initialisation procedure of the 82527

SMALL TIME DELAY

The purpose of the time delay is to cause a brief delay after the initialisation of the Host to 82527 and the CAN to CAN communications. This delay ensures sufficient time for the correct function of the system.

This procedure loads the accumulator with #$FFFFh (65535 decimal). As each delay loop takes `2.5uS to complete, it corresponds to a delay of 2.5uS X 65535 = 163mS delay every time this section of code is executed.

RECEIVE

This section sets up the 82527 for the reception of MO2, and MO3

- Set MO2 and MO3 to valid and indicates it is new data
- Set the MO2 & MO3 configuration registers

REMOTE

This section organises the 82527 for the transmission of MO1

- Set MO1 to invalid so it can be reconfigured, without interrupting the Serial Bus, and restricts the transmission of the object which may contain invalid data, due to the fact that it is being updated.
- Gets the Current tracker status data from DF00h and places it in the lowest byte location of MO1
- Sets MO1 to indicate it has new data
- Sets MO1 to a valid message again, pending to be sent over the network

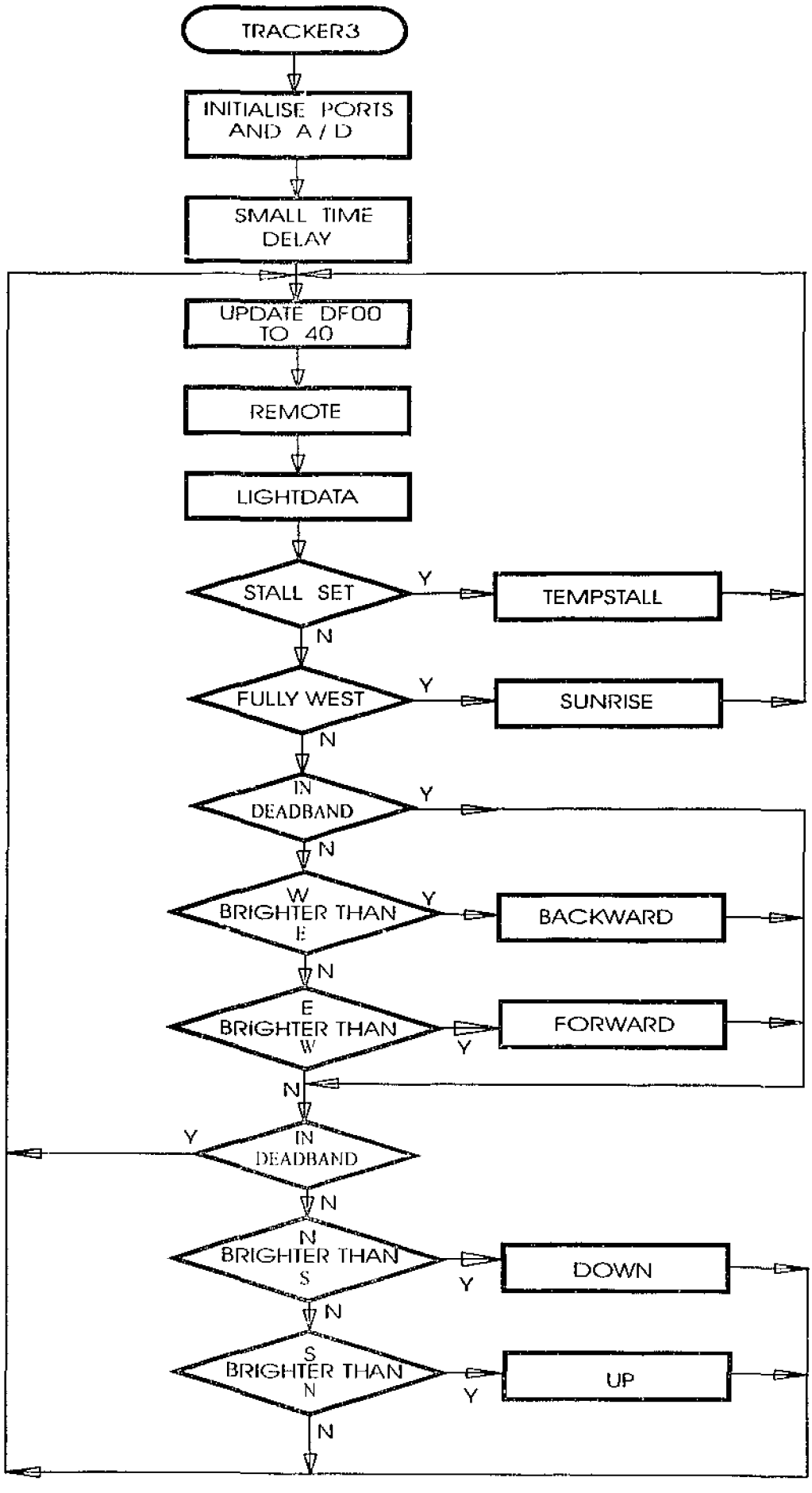
TRACKER 3 ALGORITHM

The Tracker 3 algorithm is the section of code which was the original solar tracking program. The main modification to this code was to install a set of code instructions at various major procedures and streams of program flow.
The primary purpose of these code segments was to enable a data storage area on the user RAM (DF00) to be continually updated. The data stored at this address represents the current function of the solar tracker.
This would causes very little interruption to the execution of the original program as each of these inserted code segments only consisted of a load instruction.
After every instance of the data at DF00 being updated, a procedure called REMOTE is called. The purpose of this procedure is to transfer the data stored at DF00 into the 82527's CAM in Message Object 1.

The main sections of code are described in detail. However to aid in the understanding of these descriptions the following points should be noted:

- The deadband is used to stop unnecessary continued minute adjustments of the solar tracker panel
- The CdS cells are interfaced into the tracker sensing circuit in such a way that the ratio of incident light on each cell is inversely related to the voltage level seen at the A/D converter input. Hence the statement E > W (E voltage value greater than West voltage value) actually means that the sensor has more incident light on it than the East sensor.

The following flow chart is a description of the Tracker 3 algorithm, with the additional changes to deal with CAN communications.


TRACKER3
INITIALISE PORTS AND A / D
SMALL TIME DELAY
UPDATE DF00 TO 40
REMOTE
LIGHTDATA
STALL SET
Y
TEMPSTALL
N
FULLY WEST
Y
SUNRISE
N
IN DEADBAND
Y
N
W BRIGHTER THAN E
Y
BACKWARD
N
E BRIGHTER THAN W
Y
FORWARD
N
Y
IN DEADBAND
N
N BRIGHTER THAN S
Y
DOWN
N
S BRIGHTER THAN N
Y
UP
N

The functions of the various sections of code in the previous diagram are described hereunder.

INITIALISE PORTS AND A/D CONVERTER

- Set Port A to include bit 7 as an output line.
- Include all of Port D lines for output
- Scan group one of the analogue to digital converters

SMALL TIME DELAY

See the description of the 'SMALL TIME DELAY' procedure under the previous 'MAINLINE ALGORITHM' section. For this particular procedure however the time delay value has been altered to a delay of 1.3mS. This is achieved by Decrementing 34h loops = 2.5uS X 52 = 1.3 mS of time delay.

UPDATE DF00 TO 40

Update DF00 to 40 which is used to indicate that the tracker is continuing normal operation.

REMOTE

The functions of this code have already been explained under the description of the 'MAINLINE' Code.

LIGHTDATA

This procedure has been previously discussed

STALL SET

Has the stall switch on the tracker been set ?

- Is N < 10

FULLY WEST

Has the tracker gone to it's full west position. If so then begin tracking east and store for sunrise.

- Is W<10

IN DEADBAND

Was the difference between the East and West sensors great enough to warrant moving the panel in a East/West motion? Or should the next N/S comparisons been gone to ?

- Is E-W < 5

W BRIGHTER THAN E

Determines whether the West sensor has more incident light on it than the East sensor. if so, then track West (Backwards). The code looks as if it says the opposite but the greater the light on the CdS cell the lower the A/D value, due to the circuit design of the tracker sensor board.

- IS E > W

E BRIGHTER THAN W

Determines whether the East sensor more incident light on it than the West, if so, then track East (Forward).

- IS E < W

IN DEADBAND

Determines whether the difference between the North and South sensors is so small as to not warrant a North/South movement?

- Is S - N < 5

NORTH BRIGHTER THAN SOUTH

Determines whether the North sensor has more incident light on it than the South sensor; if so, then track North (Down).

- IS S > N

S BRIGHTER THAN N

Determines whether the South sensor has more incident light on it then the North sensor? ; If so, then track South (Up).

- Is S < N

SUNRISE ALGORITHM

The following is a description of the SUNRISE algorithm. The main modification to the original procedure was to include sections of code where the CAM of the 82527 was updated to indicate to the network the current functions being carried out by the Controller.

A General Description of the SUNRISE function is as follows:

When the solar panel has reached the full West position, which would be set to correspond to the angle of the sun as it sets below the horizon, the realigning of the panel towards the East to meet the sunrise the following morning, could begin. (It was considered that the insignificant amount of sunlight received by the panel, between it's full West position, and it's position when sunset was complete, would generate negligible power.)

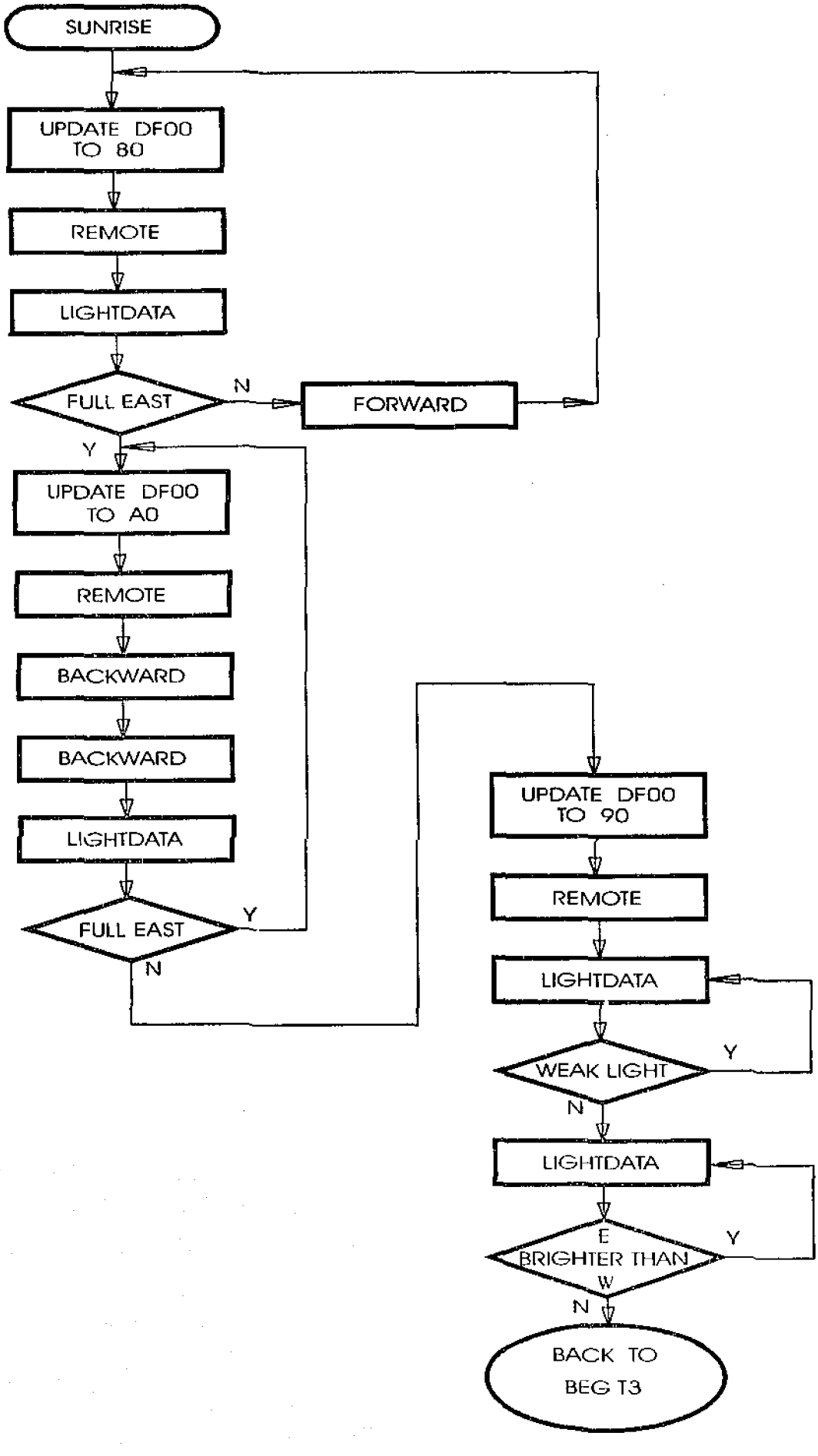
This realigning was achieved by programming the following sequence into the microcontroller.

The East/West motor would revolve the panel back to the East as soon as the West mercury switch indicated full tilt. This rotation would not alter the North / South elevation and would continue regardless of what information the light sensors were detecting. Once the East mercury switch indicted that the panel had reached the full East tilt position, the East rotation would stop. For correct operation to be able to continue in the morning the East mercury switch would then have to be deactivated otherwise the East light sensor would be unable to give correct information to the microcontroller. For this reason it was necessary for the microcontroller to rotate the East/West stepper motor rotate back towards the west until the mercury rolled back off the contacts.

The panel would now be ready for the sunrise to occur. At this stage all the LDR sensors became operational, providing that the no rotation mode has been selected by the user. The solar panel however does not begin the normal tracking operation immediately. If this occurred, as soon as the sun appeared, the tracker may try to continue to track further East past the design limit, and the mercury switch would then be reactivated. This program would not be expecting this and as the input to the microcontroller would be ~0 volts (indicating more than maximum possible sunlight) it would continue to drive itself to destruction.

For this reason the tracker is programmed to wait until the sun has risen sufficiently high enough in the horizon until the West sensor detects more sunlight than the East sensor and then it continues the normal tracking operation.

The flowchart of this algorithm is shown overpage:


SUNRISE
UPDATE DF00 TO 80
REMOTE
LIGHTDATA
FULL EAST
N
FORWARD
Y
UPDATE DF00 TO A0
REMOTE
BACKWARD
BACKWARD
LIGHTDATA
FULL EAST
Y
N
UPDATE DF00 TO 90
REMOTE
LIGHTDATA
WEAK LIGHT
Y
N
LIGHTDATA
E
BRIGHTER THAN
W
Y
N
BACK TO BEG T3

The functions of the various sections of code in the previous diagram are described here under.

UPDATE DF00 TO 80

As the name of this section of code implies, 80 is stored at DF00, waiting to be transmitted to the network. This indicates to the network that the tracker has begun to track East for sunrise.

REMOTE

Previously Described

LIGHTDATA

Previously Described

FULLEAST

Determines whether the tracker has moved to the full East position. If it hasn't then keep going east or else proceed to the rest of the program

UPDATE DF00 TO A0

Tells the network that the network is in the Full East position

- Store A0 at DF00

BACKWARD

Previously Described

LIGHTDATA

Previously Described

UPDATE DF00 TO 90

Tells the network that the tracker is in East Position mode of operation.

FULLEAST

- IS E < 10

WEAKLIGHT

[] W > F0

E BRIGHTER THAN W

Remember that sensor data is inversely related to light intensity

INTERRUPT ALGORITHM

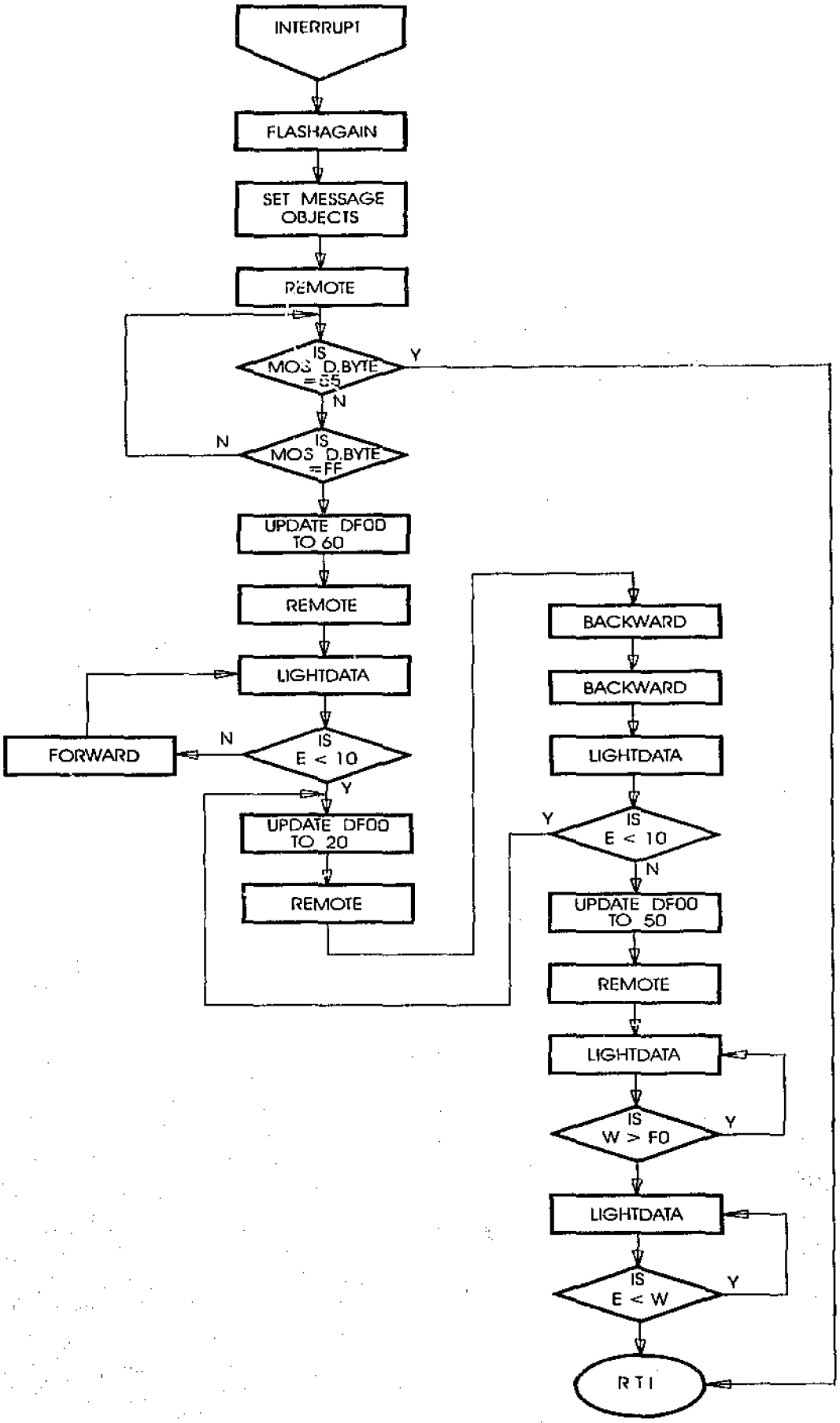
Whenever an interrupt is received by the HC11, the main primary controller algorithm pauses execution and the Interrupt algorithm takes precedence. At this stage the microcontroller will wait for a new command message to be received. The microcontroller will then interrogate the new message and depending upon it's value (listed below) will do one of the following:

- 55h. When this value is received the algorithm will end the execution of the interrupt routine and will end with the RTI instruction.

- FFH. The message object one will be updated to store 60h. This is the command to tell the tracker to stow the solar panel. The panel will track East until the East sensor receives an input value less than 10. At this stage the panel has reached the maximum position in which it can move East. Message object one is then updated to 20 H. The panel then moves two full cycles West and will continue to do this until the East sensor is greater than ten. Once this occurs the message object one is updated to 50 h. at this stage the tracker will remain steady until the West sensor reaches a value less than F0h. then the algorithm will halt the operation of the panel until the East Sensor contains a value less than the West Sensor, when this occurs the procedure will complete operation with the RTI instruction.

The purpose of this procedure, as the name implies, is to service the occurrence of an interrupt. The interrupt is generated at the HC11 whenever a message is received and stored in the CAM of the 82527 at the MO2 location.
This routine is the first one to be executed upon the occurrence of an interrupt.

The flow chart is over page


INTERRUPT
FLASHAGAIN
SET MESSAGE OBJECTS
REMOTE
IS MOS D.BYTE =55
Y
N
IS MOS D.BYTE =FF
N
UPDATE DF00 TO 60
REMOTE
LIGHTDATA
IS E < 10
N
FORWARD
Y
UPDATE DF00 TO 20
REMOTE
BACKWARD
BACKWARD
LIGHTDATA
IS E < 10
Y
N
UPDATE DF00 TO 50
REMOTE
LIGHTDATA
IS W > F0
Y
LIGHTDATA
IS E < W
Y
RTI

The functions of the various sections of code in the previous diagram are described here under.

Flashagain

- Resets the interrupt line
- Clears the data byte of MO3
- Puts normal tracking info into location DF00
- Runs the remote procedure

Remote

- Update the MO1 - tracking data CAN CAM: puts the data stored at DF00 into MO1, ar.. ets it to active.
- Runs the Action procedure

Action

- Waits for a valid data byte to arrive at MO3 in the CAN CAM, while this is happening the tracker has halted operation.
- The valid data will be either Resume operation ($55) or do a remote stow ($FF).

Gostow

- Update DF00 to moving east($60) and call remote, then proceeds to Sune.

Sune

- Get the lightdata from the tracker sensors and uses the following code in Rdymrn to set in sunrise pos.

Rdymrn

- Once in the backed off East position(sunrise) sets DF00h to East position - by remote($50) and calls remote.

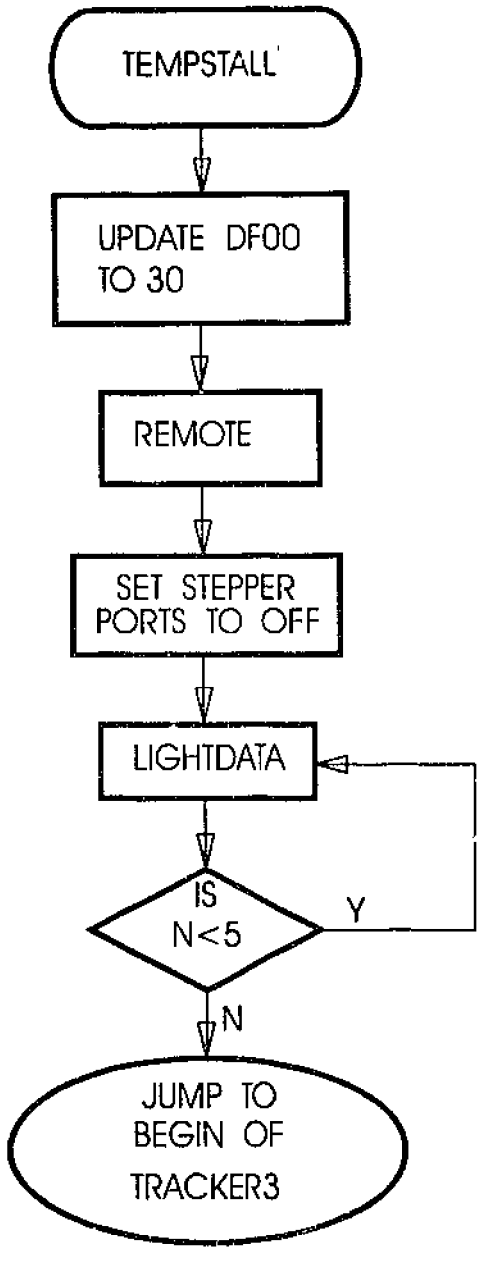
Sunp&Moren

Will stay in this procedure, hence halting the operation of the tracker until Moren sufficient light is on the West sensor to begin tracking normal tracking operation again.

TEMPSTALL ALGORITHM

The Tempstall or 'No Rotation' function of the controller is initiated at the controller site. This mode is set by flicking a switch which in effect shorts the terminals of the North LDR. The microcontroller then detects that the sensor has gone to an unexpected 'forbidden' voltage level. It responds to the situation by going back to the Hold procedure. The hold procedure simply waits until the sensor has gone back to the normal operating voltage levels.

The flowchart can be seen overpage


TEMPSTALL
UPDATE DF00 TO 30
REMOTE
SET STEPPER PORTS TO OFF
LIGHTDATA
IS N<5
Y
N
JUMP TO BEGIN OF TRACKER3

The functions of the various sections of code in the previous diagram are described here under.

UPDATE DF00 TO 30

Updates DF00h with 30h, indicating that the temporary stall routine is beginning operation

REMOTE

The remote routine issues the message to the network that the controller is in temporary stall mode

SET STEPPER PORTS TO OFF

All the output lines to the stepper motors output, are turned off ~ 0 volts.

LIGHTDATA

Obtains new data from the CdS cells on the panel mounting plane and stores this data as

D000- EAST Sensor

D001- WEST Sensor

D002- SOUTH Sensor

D003- NORTH Sensor

This is accomplished by setting up index Accumulator X as a source address and index Accumulator Y as a destination address. Data is retrieved from X, (which points to one group of the A/D converters which are part of Port E), and then stores the data at the destination pointed to by Y. This process occurs one location at a time. After each read and write cycle the source and destination registers are incremented by 1. This loop occurs 4 times so all of the light data is transferred one byte at a time from the A/D converter to the memory locations as indicated above.

IS N<5

The data for the North CdS cell is compared to the number 05h. The program will continue to retrieve execute the lightdata procedure and compare to 05h until

the data is greater, at which time the execution of this loop is terminated.

JUMP TO BEGIN OF TRACKER3

As the name implies the program execution jumps to BegT3. Which is the top of the Tracker 3 program.

PROGRAM FOR THE PRIMARY MICROCONTROLLER

```
*****************************************************************
*       This is the program to be added to the solar tracker    *
*****************************************************************
        org   $c000             ;Starting  Address of Program
        ldaa  #$7e              ;alter the IRQ interrupt vector
        staa  $00ee
        ldd   #flashagain       ;where to jump on interrupt
        std   $00ef             ;
        ldaa  #$00              ;alter CCR to enable interrupts
        tap                     ;puts acca into ccr
begin:    jsr   Initialise      ;Initialise 82527
        jsr   timedelay
        jsr   receive           ;Set for MO2 & MO3
        jsr   remote            ;Set up for MO1
        nop                     ;was jsr    flashled4
        nop                     ;was jsr    flashled4
wait:     nop                   ;was jsr    flashled5
        nop                     ;this is where the tracker
        nop                     ;program will go
        jmp   tracker3          ;
        jmp   wait              ;waiting for interrupt
        nop                     ;main prgram loop area
        end
*****************************************************************
*         Initialise the 82527 registers                        *
*****************************************************************
Initialise:  ldaa  #$40         ;Set CPU interface SCLK = XTAL/2
        staa  $6002             ;and MCLK = SCLK
        ldaa  #$41              ;Set the CCE = 1 so write access
        staa  $6000             ;to config registers
        ldaa  #$42              ;Bypasses input comparator,one is
        staa  $602f             ;recessive, TX1 enabled
        ldaa  #$41              ;Gives SJW =1
        staa  $603f             ;and BRP = 1
        ldaa  #$67              ;Spl=0:one sample period, and
        staa  $604f             ;TSEG2=6, and TSEG1=7
        ldaa  #$01              ;CCE=0:CPU has no write access to the
        staa  $6000             ;configuration registers
        ldaa  #$55              ;Reset control reg 0 and 1
        staa  $6010             ;in all the messages.
        staa  $6020             ;All the messages are
        staa  $6030             ;set to invalid.
        staa  $6040             ;
        staa  $6050             ;
```

```
staa  $6060      ;
staa  $6070      ;
staa  $6080      ;
staa  $6090      ;
staa  $60A0      ;
staa  $60B0      ;
staa  $60C0      ;
staa  $60D0      ;
staa  $60E0      ;
staa  $60D0      ;
staa  $60E0      ;
staa  $60F0      ;
staa  $6011      ;
staa  $6021      ;
staa  $6031      ;
staa  $6041      ;
staa  $6051      ;
staa  $6061      ;
staa  $6071      ;
staa  $6081      ;
staa  $6091      ;
staa  $60A1      ;
staa  $60B1      ;
staa  $60C1      ;
staa  $60D1      ;
staa  $60E1      ;
staa  $60D1      ;
staa  $60E1      ;
staa  $60F1      ;
ldaa  #$00           ;Set global Mask Standard and
staa  $6006          ;extended to 00 which is
staa  $6007          ;'don't care'
staa  $6008          ;
staa  $6009          ; -Using extended mask registers-
staa  $600a          ;
ldaa  #%01111000     ;Set the last four bits of mask to
staa  $600b          ;'care'
ldaa  #$8c           ;Transmit and receive using extended
staa  $6016          ;Message 1 is transmit &
ldaa  #$84           ;Message 2 & Message 3 is receive
staa  $6026          ;
staa  $6036          ;
ldaa  #$00           ;Load the arbitration registers for
staa  $6012          ;Message 1 - make '1'
staa  $6013          ;
staa  $6014          ;
ldaa  #%00001000     ;
staa  $6015          ;
ldaa  #$00           ;load the arbitration registers for
```

```
        staa   $6022            ;Message 2 - make '2'
        staa   $6023            ;
        staa   $6024            ;
        ldaa   #%00010000       ;
        staa   $6025            ;
        ldaa   #$00             ;load the arbitration registers for
        staa   $6032            ;Message 3 - make '3'
        staa   $6033            ;
        staa   $6034            ;
        ldaa   #%00011000       ;
        staa   $6035            ;
        ldaa   #$02             ;Reset Init bit & Set IE bit
        staa   $6000            ;otherwise no coms or interrupts
        rts                     ;End of initialisation
*****************************************************************
*       Receive procedures                                      *
*****************************************************************
receive:    ldaa   #$56         ;Set message valid;New data
        staa   $6021            ;for message two and
        staa   $6031            ;   message three
        ldaa   #$99             ;Config0 > msgval=1,TXIE=0,
        staa   $6020            ;RXIE=1, IntPnd=0 MO2
        ldaa   #$95             ;MO3
        staa   $6030
        rts
*****************************************************************
*       MO1 - Remote frame                                      *
*****************************************************************
remote:     ldaa   #$55         ;Message not valid -for update
        staa   $6010            ;
        ldaa   $df00            ;Put Tracker data here
        staa   $601e            ;
        ldaa   #$56             ;NewDat but not transmit
        staa   $6011            ;
        ldaa   #$95             ;Message Valid again
        staa   $6010            ;
        rts
*****************************************************************
*       Timedelay                                               *
*****************************************************************
timedelay:ldx   #$ffff
loopy:    dex
        bne    loopy
        rts
*****************************************************************
*  This is where the Stow or Resume procedures would be located   *
*  This routine is loaded whenever MO2 is received                *
*****************************************************************
        nop
```

```
            nop
flashagain:  nop                ; was ldaa  $605f
            nop                 ; was staa  $c710
            ldaa  #$99          ;Reset int IntPnd bit in the
            staa  $6020         ;C0 reg of MO2 & 5Fh data erased
            ldaa  #$55          ;MO3 no longer active so can
            staa  $6030         ;be updated
            ldaa  #$00          ;Clear data byte of MO3
            staa  $603e         ;
            ldaa  #$95          ;Make MO3 active again
            staa  $6030         ;
            ldaa  #$70          ;remote frame info
            staa  $df00         ;remote frame info
            jsr   remote        ;update remote MO
action:      ldaa  $603e        ;Get data from MO3 and do action
            cmpa  #$55          ;If it is Resume then leave
            beq   go            ;interrupt routine
            cmpa  #$ff          ;If it is Stow then stow and leave
            beq   gostow
            jmp   action        ;Loop until choice is made
gostow:      ldaa  #$60         ;remote frame info
            staa  $df00         ;remote frame info
            jsr   remote        ;update remote MO
            nop                 ;The tracker procedure follows
sune:        jsr   lightdata
            ldaa  $d000
            cmpa  #$10          ;See if fully East yet
            blo   rdymn         ;If yes then rdymorn
            jsr   forward       ;If no keep going
            jmp   sune
rdymn:       ldaa  #$20         ;remote frame info
            staa  $df00         ;remote frame info
            jsr   remote        ;update remote MO
            jsr   backward      ;back off extreme a bit
            jsr   backward
            jsr   lightdata     ;Go west until tilt sensor
            ldaa  $d000         ;Is deactivated
            cmpa  #$10          ;see above remote MO
            blo   rdymn         ;data 20 sent in loop
            ldaa  #$50          ;Remote frame info
            staa  $df00         ;Remote frmame info
            jsr   remote        ;update remote MO
sunp:        jsr   lightdata    ;Has West got morning light
            lda   $d001         ;If no than stay
            cmpa  #$f0
            bhi   sunp
moren:       jsr   lightdata    ;If yes than compare to east
            ldaa  $d000
            cmpa  $d001         ;If East still brighter than
```

```
        blo    moren            ;West than stay else begin
        nop                     ; was jmp begin
go:       rti                   ; end of canon program
*****************************************************************
*    The following is the modified tracker3.asm program        *
*    org mofified, c500 data modifed to d000  etc              *
*****************************************************************
tracker3:    nop                ;the modified tracker3
        nop                     ;program follows
        ldaa   #%10000000       ;Include PA7 as output line
        staa   $1026            ;
        ldaa   #$f8             ;Value to set the mask
        staa   $100c            ;output value to OC1M mask enable
        ldaa   #$ff             ;set all the bits for port d
        staa   $1009            ;as output
        ldab   #26              ;26 loops of the time loop
        ldaa   #$34             ;scan group one AtoDs
        staa   $1030
loop:     decb
        bne    loop
begt3:     ldaa   #$40          ;Remote frame info
        staa   $df00            ;Remote frame info
        jsr    remote           ;update remote frame info
        jsr    lightdata        ;obtain data for light levels
        ldaa   $d003            ;test for stall
        cmpa   #$10
        blo    stall
        ldaa   $d001
        cmpa   #$10
        blo    fullwest         ;has tracker gone fully West
        ldaa   $d000            ;begin deadband East/West
        suba   $d001
        cmpa   #$5
        blo    updown
        ldaa   $d000            ;begin East/West compare
        cmpa   $d001
        bhi    bward            ;these are the
        blo    fward            ;east west movements
fward      jsr    forward       ;Go East
        jmp    updown
bward      jsr    backward      ;Go West
updown:      ldaa   $d002       ;begin deadband North/South
        suba   $d003
        cmpa   #$5
        blo    begt3
        ldaa   $d002            ;begin North/South compare
        cmpa   $d003
        bhi    jumpdown         ;these are the
        blo    jumpup           ;North / South movements
```

```
            jmp   begt3
jumpup:     jsr   up
            jmp   begt3
jumpdown:   jsr   down
            jmp   begt3
fullwest:   jmp   sunrise
stall:      jmp   tempstall
*****************************************************************
*     Surise Routine to align for sunrise after sunset          *
*****************************************************************
sunrise:    ldaa  #$80          ;remote frame info
            staa  $df00         ;remote frame info
            jsr   remote        ;update remote MO
            jsr   lightdata
            ldaa  $d000
            cmpa  #$10          ;See if fully East yet
            blo   rdymorn       ;If yes then rdymorn
            jsr   forward       ;If no keep going
            jmp   sunrise
rdymorn:    ldaa  #$a0          ;remote frame info
            staa  $df00         ;remote frame info
            jsr   remote        ;update remote MO
            jsr   backward      ;back off exreme a bit
            jsr   backward
            jsr   lightdata     ;Go west until tilt sensor
            ldaa  $d000         ;Is deactivated
            cmpa  #$10
            blo   rdymorn
            ldaa  #$90          ;remote frame info
            staa  $df00         ;remote frame info
            jsr   remote        ;update remote MO
sunup:      jsr   lightdata     ;Has West got morning light
            lda   $d001         ;If no than stay
            cmpa  #$f0
            bhi   sunup
moresun:    jsr   lightdata     ;If yes than compare to east
            ldaa  $d000
            cmpa  $d001         ;If East still brighter than
            blo   moresun       ;West than stay else begin
            jmp   begt3
*****************************************************************
*     Routine to obtain light level data                        *
*****************************************************************
lightdata:  ldaa  #04
            staa  $dc00
            ldx   #$1031
            ldy   #$d000
more:       ldaa  0,x
            staa  0,y
```

```
            inx
            iny
            dec   $dc00
            bne   more
            rts
*****************************************************************
*     The following temporarily stalls the tracker - idle state  *
*****************************************************************
tempstall:    ldaa  #$30          ;Remote frame info
            staa  $df00           ;Remote frame info
            jsr   remote          ;update remote frame info
            ldaa  #%00000000      ;no output to stepper motors
            staa  $100d
            staa  $1008
goyet:        jsr   lightdata
            ldaa  $d003
            cmpa  #$5
            blo   goyet
            jmp   begt3
*****************************************************************
*     The following controls the way the motor does a forward movement  *
*****************************************************************
*           This makes the tracker go east
forward:      ldaa  #%01000000    ;First step of motor
            staa  $100d           ;output data to OC1D and port
            jsr   slower
            ldaa  #%01100000      ;Second step of motor
            staa  $100d           ;Output data to OC1D and port
            jsr   slower
            ldaa  #%00100000      ;Third step of motor
            staa  $100d           ;Output data to OC1D and port
            jsr   slower
            ldaa  #%00110000      ;Fourth step of motor
            staa  $100d           ;Output data to OC1D and port
            jsr   slower
            ldaa  #%00010000      ;Fivth step of motor
            staa  $100d           ;Output data to OC1D and port
            jsr   slower
            ldaa  #%00011000      ;Sixth step of motor
            staa  $100d           ;Output data to OC1D and port
            jsr   slower
            ldaa  #%00001000      ;Seventh step of motor
            staa  $100d           ;Output data to OC1D and port
            jsr   slower
            ldaa  #%01001000      ;Eight step of motor
            staa  $100d
            jsr   slower
            rts
*****************************************************************
```

```
*     The following proceedure is to control the motors reverse          *
*     revolution                                                         *
*********************************************************************
*     This makes the tracker go west
backward:    ldaa  #%01000000    ;First step of motor
             staa  $100d         ;output data to OC1D and port
             jsr   slower
             ldaa  #%01001000    ;Second step of motor
             staa  $100d         ;Output data to OC1D and port
             jsr   slower
             ldaa  #%00001000    ;Third step of motor
             staa  $100d         ;Output data to OC1D and port
             jsr   slower
             ldaa  #%00011000    ;Fourth step of motor
             staa  $100d         ;Output data to OC1D and port
             jsr   slower
             ldaa  #%00010000    ;Fivth step of motor
             staa  $100d         ;Output data to OC1D and port
             jsr   slower
             ldaa  #%00110000    ;Sixth step of motor
             staa  $100d         ;Output data to OC1D and port
             jsr   slower
             ldaa  #%00100000    ;Seventh step of motor
             staa  $100d         ;Output data to OC1D and port
             jsr   slower
             ldaa  #%01100000    ;Eighth step of motor
             staa  $100d
             jsr   slower
             rts
*********************************************************************
*     The following controls the way the motor does a up movement    *
*********************************************************************
up:          ldaa  #%00000100    ;First step of motor
             staa  $1008         ;output data to OC1D and port
             jsr   slower
             ldaa  #%00001100    ;Second step of motor
             staa  $1008         ;Output data to OC1D and port
             jsr   slower
             ldaa  #%00001000    ;Third step of motor
             staa  $1008         ;Output data to OC1D and port
             jsr   slower
             ldaa  #%00011000    ;Fourth step of motor
             staa  $1008         ;Output data to OC1D and port
             jsr   slower
             ldaa  #%00010000    ;Fivth step of motor
             staa  $1008         ;Output data to OC1D and port
             jsr   slower
             ldaa  #%00110000    ;Sixth step of motor
             staa  $1008         ;Output data to OC1D and port
```

```
        jsr   slower
        ldaa  #%00100000      ;Seventh step of motor
        staa  $1008           ;Output data to OC1D and port
        jsr   slower
        ldaa  #%00100100      ;Eight step of motor
        staa  $1008
        jsr   slower
        rts
*****************************************************************
*     The following controls the way the motor does a down movement     *
*****************************************************************
down:   ldaa  #%00000100      ;First step of motor
        staa  $1008           ;output data to OC1D and port
        jsr   slower
        ldaa  #%00100100      ;Second step of motor
        staa  $1008           ;Output data to OC1D and port
        jsr   slower
        ldaa  #%00100000      ;Third step of motor
        staa  $1008           ;Output data to OC1D and port
        jsr   slower
        ldaa  #%00110000      ;Fourth step of motor
        staa  $1008           ;Output data to OC1D and port
        jsr   slower
        ldaa  #%00010000      ;Fivth step of motor
        staa  $1008           ;Output data to OC1D and port
        jsr   slower
        ldaa  #%00011000      ;Sixth step of motor
        staa  $1008           ;Output data to OC1D and port
        jsr   slower
        ldaa  #%00001000      ;Seventh step of motor
        staa  $1008           ;Output data to OC1D and port
        jsr   slower
        ldaa  #%00001100      ;Eight step of motor
        staa  $1008
        jsr   slower
        rts
*****************************************************************
*     The following procedure is to control the time interval between     *
*     succesive steps of the stepper motor                                *
*****************************************************************
slower: ldx   #$ffff
loopy:  dex
        bne   loopy
        rts
*****************************************************************
*     End of Operations                                                   *
*****************************************************************

finish:  end
```

CONCLUSION

The initial proposal to use the National Semiconductor DP8390 Local Area Network Interface Controller and interface it with a Motorola MC68HC11 was abandoned because after investigation it was realised that to fully utilise the capabilities of the Ethernet (IEEE 802.3) high speed communications system, whose attributes are in the area of large data transfer, it should be interfaced with at least a 16-bit microcontroller. As the aim of this project was to form a communications system using a small 8-bit microcontroller it was considered unsuitable for utilisation in this project.

The consequent research and investigation into the selection of the most suitable serial interface controller and connection method resulted in the following progressive conclusions:

- The CAN system was chosen as the most suitable to be interfaced with an 8-bit microcontroller for the purposes of this project.
- The Intel 82527 CAN controller was selected.
- The 74HC138 was effectively utilised to ensure correct interfacing between the HC11 EVB and the 82527.
- A CAN Network was successfully established and tested in a real-time environment in a dual monitoring and control role.

A solar tracker was used for the purposes of testing the system in a real-time environment. The successful demonstration of the ability to monitor and possess an overriding control capability over a device already having its own PRIMARY control system, by an independent REMOTE control system, indicated that the system was successful.

THE FINAL AIM OF THIS PROJECT WAS ACHIEVED.

The potential of such a system is unlimited.

For example:

In large building complexes, where each unit or cell has its own airconditiong, mechanical services installation, and control systems, the ability to monitor and have an overriding control over such systems from a central control and security position, without expensive dedicated wiring looms for each circumstance, would be invaluable.

This would also apply to large hospital installations where comprehensive monitoring and control of major equipment and critical services is not only required at the specific locations but also desireable at one central control point.

Large industrial complexes with plant and equipment spread over the site could have comprehensive monitoring and management of their installation from one central location at minimum cost.

APPENDIX A

PCB CONSTRUCTION

PHOTORESIST PROCEDURE

1 Coat the board evenly with photoresist.

2 Allow to dry for ~ 15 minutes.

3 Dry further in oven for ~ 20 minutes at 60 - 80 degrees Celsius.

4 For double sided boards repeat steps 1 - 4 for other side.

5 Attach artwork to both sides of board.

6 Expose each side to UV source of light.

7 Develop the board in Caustic Soda or Developer solution.

8 Rinse in water to stop developing process.

9 Check board for defects, correct these and then etch board in acid solution.

PROBLEMS ENCOUNTERED WHEN DEVELOPING

The following problems were encountered during the production of the Printed Circuit Board (PCB).

- Image adhered to the clear plastic artwork sheet.
  Diagnosis - The photoresist on the board was not dry enough.
- Image failed to develop.
  Diagnosis - As the board needs to absorb moisture while setting it may have been affected by one of the following
  * The board was placed into the oven after 5 minutes drying time. This may have been insufficient time.
  * The temperature of the oven may have been too high
  * The humidity of the room may have been too low due to the fact that the room was heated

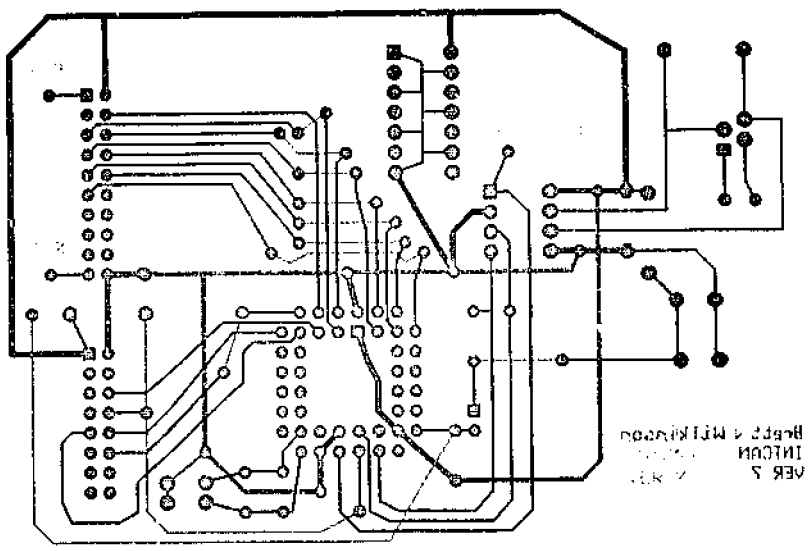
This could have been due to the fact the board was only allow to dry for 5 minutes before being put into the oven for drying or because the new oven used was too hot. Either of these may have reduced the water absorption which the board must do while setting. Another possibility is the fact that the room was too dry due to heating that was running both nights.

APPENDIX B

CIRCUIT BOARD DESIGNS

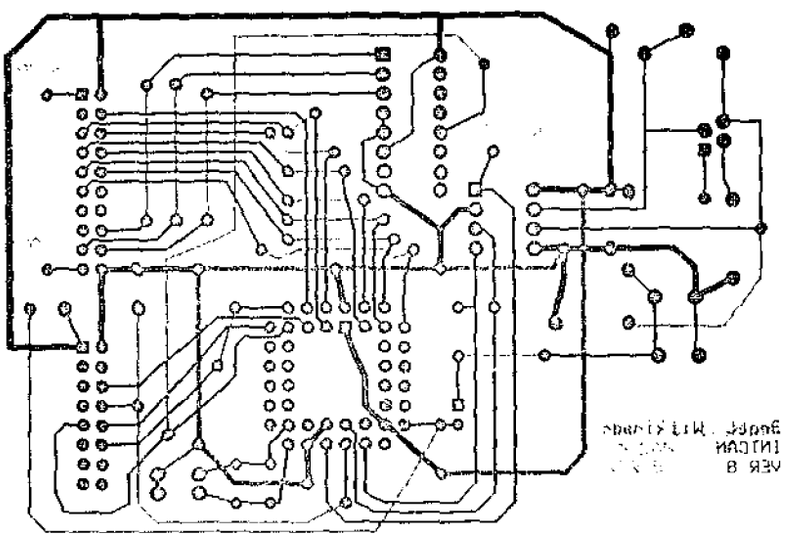
INTCAN 7 BOARD

The artwork below is of the INTCAN7 board, which was produced using the EDWIN Software Package



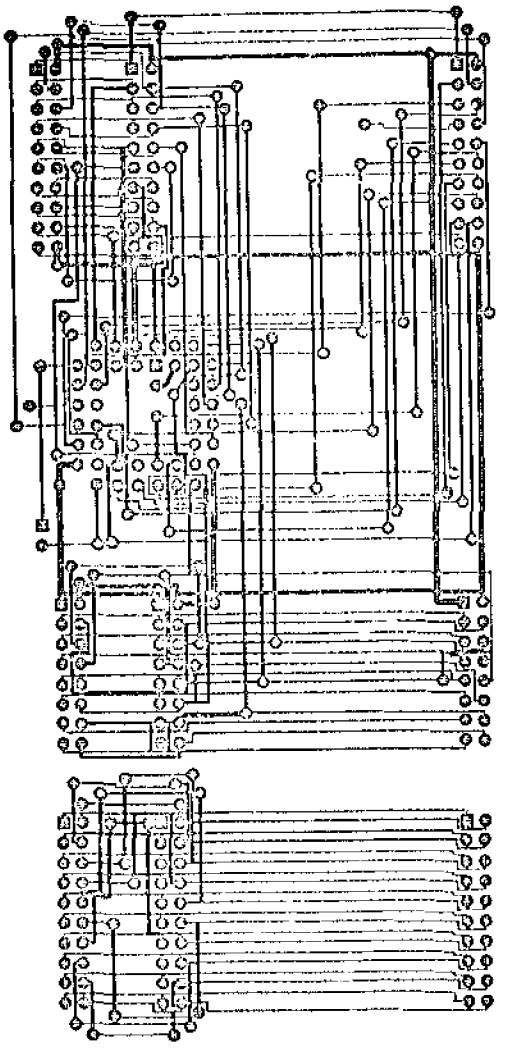
INTCAN 8 BOARD

The artwork below is of the INTCAN8 board, which was produced using the EDWIN Software Package



PORT REPLACEMENT BOARD UNIT

The artwork below is of the Port Replacement Unit Board, which was produced using the EDWIN Software Package



REFERENCES

1) Madron, T.W. (1994). Local Area Networks : New Technologies, Emerging Standards. 3rd Edition. New York: Wiley.

2) Motorola. (1996). Motorola Master Selection Guide 2.6-2
Available WWW:http://www.mot-sps.com/sps/general//chips.html

3) National Semiconductor (1993). DP8390 Network Iterface Controller: an Introductory Guide. Application Note 475.
Available WWW:http://www.nsc.com

4) Vincent Himpe (1997). I2C Faq.
Available WWW:http//www.ping.be/~ping0751/12cfaq.htm

5) SICAN Braunschweig (1997). I2C Bus Design Objects.
Available WWW:http://www.sican-bs.de/Pulblic/ChipTechnik/
DesignObjects/12cbusDo

6) Vincent Himpe (1997). I2C Faq.
Available WWW:http//www.ping.be/~ping0751/12carbit.htm

7) Intel Corporation (1997). Introduction to In-vehicle Networking [on-line].
Available WWW:
http://developer.intel.com/design/news/autob:bk.htm#A2

8) Kvaser, A.B. (1997). The CAN protocol [on-line].
Available WWW: http://www.kvaser.se/can/general.htm

9) Schofield, M.J. (1997). Controller Area Network - Implementations [on-line]. Available WWW:
http://www.omegas.co.uk/can/implement.htm

10) Himpe, V. (1997). I2C Faq.
Available WWW:http//www.ping.be/~ping0751/12cproto.htm

11) Tindell, K. (1997). CAN: Controller Area Network
Available WWW:www.nrtt.demon.co.uk/cantech.htm

12) Motorola Inc. (1986). Fast and LS TTL DATA.

13) Intel Corporation. (1995). 82527 Serial Communications Controller Architectural Overview. Order Number: 272410-003
Available WWW:http://www.developer.intel.com/design/

14) Bosch, R.(1991). CAN Specification Version 2.0.
PDF file Available WWW:http//www.bosch.com

15) Driscoll, F., Coughlin, R., & Villanucci, R. (1994). Data Acquisition and Process Control with the MC68HC11 Microcontroller. New York: Macmillan Publishing Company.

16) Schofield, M.J. (1997). Controller Area Network - Available Devices [on-line]. Available WWW: http://www.omegas.co.uk/can/devices.htm.

17) Scott, G. (1995). Interfacing an Intel 82527 Serial Communications Controller to a 68HC11. Application Note AP-722.
Available WWW:http://www.http://developer.intel.com/design/

18) Microcontroller Solutions (1996). HC11 Development Board Manual.

BIBLIOGRAPHY

Hallsall, F. (1995) Data Communications, Computer Networks and Open Systems. United States of America: Addison - Wesley.

Couch II, L. (1993) Digital and Analog Communication Systems. New York: Macmillan Publishing Company.

ACRONYMS AND ABRIDGMENTS

AS	Address Strobe
ASCII	American Standard Code for Information Interchange
BRP	Baud Rate Prescalar
CAL	CAN Application Layer
CAM	Content Addressable Memory
CAN	Controller Area Network
CPU	Central Processing Unit
CS	Chip Select
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DLC	Data Length Code
EEPROM	Electrically Erasable Programmable Read Only Memory
EIE	Error Interrupts Enable
EVB	Evaluation Board
IC	Integrated Circuit
ID	Unique Identifer
I/O	Input / Output
ISO	International Standards Organisation
LAN	Local Area Network
LCD	Liquid Crystal Display
LLC	Logical Link Control
MAC	Media Access Control
MC	Microcontroller
MISO	Master Input Slave Output
MO	Message Object
MOSI	Master Output Slave Input
NRZ	Non Return to Zero
OSI	Open Systems Interconnection
PLC	Programmable Logic Controller
PRU	Port Replacement Unit

RAM	Random Access Memory
ROM	Read Only Memory
R/W#	Read / Write line (Write active low)
SCI	Serial Communications Interface
SCK	Synchronous Serial Clock
SDS	Smart Distribution System
SIE	Status change Interrupt Enable
SJW	Synchronisation Jump Width
SPI	Serial Peripheral Interface
SS	Slave Select
STRA	Strobe A
STRB	Strobe B
UART	Universal Asynchronous Receiver Transmitter
Uc	Microcontroller
UV	Ultra Violet