AN464

Software Driver routines for the Motorola MC68HC05 CAN Module

Kenneth Terry,
Motorola Ltd.,
East Kilbride, Scotland

Introduction

The Controller Area Network (CAN) protocol was developed by Robert Bosch GmbH. It describes a serial communications protocol which has been designed to support interrupt-driven, real-time control applications, primarily in the automotive sector. There are currently four MC68HC05 MCUs which support the CAN protocol. These are the MC68HC05X4 and the MC68HC05X16 (and their EPROM derivatives), the MC68HC705X4 and the MC68HC705X16.

The software described in this application note comprises a number of driver routines which provide an interface between application software residing in the MCU ROM (or EPROM) and the CAN module. The routines allow for the initialisation of the CAN module, the transmission of messages, previously stored in RAM, and the automatic handling of received messages. If required, they allow for automatic response to requests for data from other CAN nodes without intervention from the application software. They also allow for optional vectoring into the application software when specific CAN interrupts occur. Messages to be transmitted can be queued and automatically transferred to the CAN module when its transmit buffers become available. Once a message is entered into the transmit queue, no further action is required from the application software. This allows the application software to cycle through a number of tasks at a fixed rate. If during those tasks it becomes necessary to transmit more than one message on the CAN bus, the driver routines allow this to be done without the application software being tied up waiting for the CAN transmit buffers to become available.

The CAN driver routines have been written to run on the MC68HC05X4 but can be easily adapted to run on any HC05 CAN MCU.

CAN Protocol Overview

The Motorola CAN module supports the CAN protocol defined in the Bosch CAN Specification Revision 1.2 (and Revision 2.0 Part A). The following is a brief overview of the major components of the protocol. Refer to the above documents for a full description of CAN.

The Bosch specification divides a CAN implementation into three layers. These are:

- Object Layer
- Transfer Layer
- Physical Layer

The Physical layer defines the actual media used for the transmission of the signals between CAN nodes, and the electrical specifications of the media. Most commonly the physical layer will consist of a two-wire differential bus, but this is not defined within the CAN specification.

The CAN protocol defines the main functions of the Transfer Layer. These consist primarily of framing control, message arbitration, error checking, error signalling and fault confinement. The bit timing is also defined within the transfer layer.

The Object Layer provides all required message filtering as well as status and message handling. The object layer will only pass on to the next layer, in this case the user's application software, messages which are relevant to the local CAN node.

Information is sent on the CAN bus in fixed format messages which are routed onto the bus by means of an identifier. The identifier does not indicate a destination for the data, but defines the data contained within a message such that it can be acted upon by a number of nodes within the network. The identifier is also used in the arbitration process, used to resolve bus access conflict when two or more nodes start to transmit a message at the same time. Data is sent on the bus in standard NRZ format. The bus has two allowable logical states; dominant and recessive. In the case where a dominant bit and a recessive bit are transmitted at the same time by two separate nodes, the dominant level will prevail. This is necessary to allow the arbitration process to work. A transmitting node monitors the bus when it is sending the message identifier. If it detects a dominant bit on the bus as it is transmitting a recessive bit, it assumes a higher priority identifier is being transmitted at the same time by another node, whereupon it stops transmitting and switches to receive mode.

The message transfer is controlled by the use of four different frame types. These are:

- Data Frame – carrying data from a transmitter to the receivers
- Remote Frame – transmitted as a request for data
- Error Frame – transmitted by any node (on detecting a bus error) to all other nodes on the bus
- Overload Frame – used by a node to request a delay between successive data or remote frames

The Motorola CAN module supports all the above frame types with the exception of the overload frame. It will recognise and respond to an overload frame, but will not generate one.

Figure 1 shows the format of the Data Frame.

The Data Frame is divided into a number of different fields.

The arbitration field is 12 bits long and contains the identifier for the frame. This is 11 bits long (for CAN Revision 1.2). The last bit of the arbitration field is the RTR bit and this is used to indicate whether a data frame or a remote frame is being transmitted. It is transmitted dominant to indicate a data frame, and recessive to indicate a remote frame. As the RTR bit is also used as part of the arbitration process, a data frame has higher priority in the case where a data and a remote frame, with the same identifier, are transmitted simultaneously.

The control field is 6 bits long. In CAN Rev. 1.2, the first two bits of the control field are transmitted recessive and the last four bits are used to provide the data length code. This determines the number of data bytes which will be transmitted in the Data Field and this can range from zero to a maximum of eight.

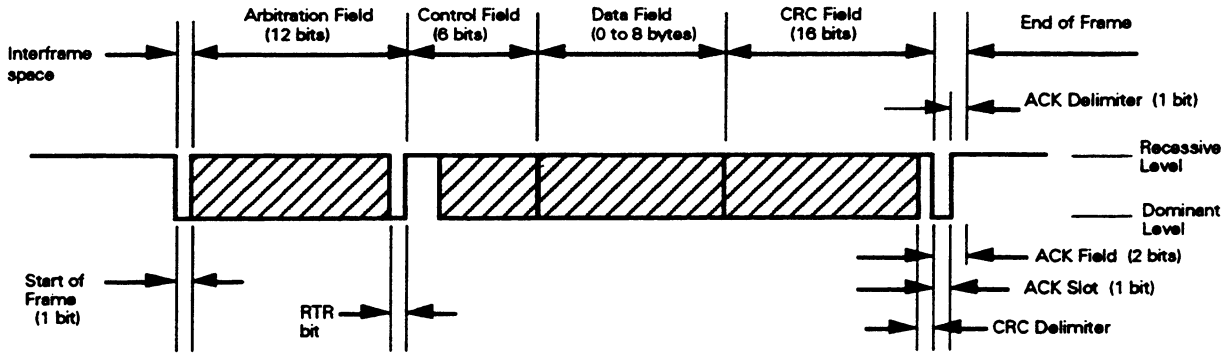


Figure 1 CAN Data Frame

The Data Field contains the data to be transmitted within the Data Frame. The bytes are transmitted MSB first.

The CRC Field contains a CRC sequence followed by a CRC field delimiter. The CRC sequence is derived from a cyclic redundancy code calculated from the bit stream making up the arbitration field, control field and data field. The CRC delimiter is a single recessive bit.

The Ack field is two bits long. The transmitting node will send two recessive bits during the Ack field. All nodes correctly receiving the message portion preceding the Ack field will indicate that they have done so by transmitting a dominant bit during the 1st bit time of the Ack field. This indicates to the transmitter that at least one node has correctly received the transmitted message. The second bit of the Ack field is the Ack delimiter, which is recessive.

Each data frame (and remote frame) is delimited by an end of frame sequence of 7 recessive bits.

A Remote Frame takes exactly the same form as a data frame with two exceptions:

- The RTR bit is sent recessive in a Remote Frame
- A Remote Frame has no data field

The Remote Frame does contain a control field but its value has no relevance due to the lack of a data field. The Remote Frame is used as a request for data. Figure 2 shows the format of the Remote Frame.

There are several errors that can be detected and signalled by the nodes on the CAN network. These include the following:

Bit Error – A transmitting unit monitors the bus as it is transmitting. If it detects a bit on the bus that does not correspond to the bit it is transmitting it perceives this as an error. The exceptions to this are during the transmission of the arbitration field and during the 1st bit time of the Ack field.

Stuff Error – A transmitting node automatically inserts a bit of opposite polarity into the transmitted bit stream after 5 consecutive identical bits have been sent. Any receiving node detecting 6 consecutive identical bits will perceive this as a stuff error.

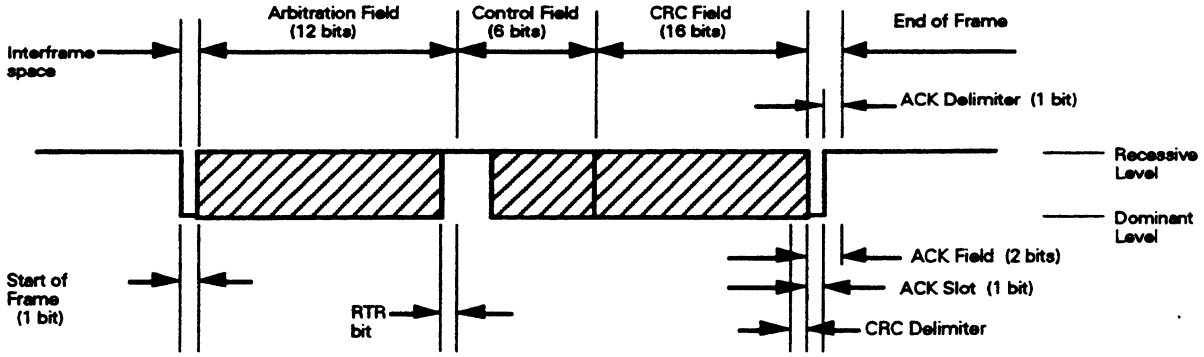


Figure 2 Remote Frame

CRC Error – Each receiving node recalculates the CRC for a received message and compares it with the received CRC. A CRC error occurs when the two values do not match.

Form Error – Any variation from the fixed frame format in a transmitted message will result in a form error.

Acknowledgement Error – This is detected by the transmitter if it does not see a dominant bit during the first bit time of the Ack field.

A node detecting an error can signal this by sending an error frame (See figure 3). The error frame consists of two different fields. The first field consists of the superposition of error flags sent by the various nodes on the network. The second field is the error delimiter. There are two types of error flags that can be sent; passive and active. The active error flag consists of 6 dominant bits. This flag will either violate the fixed form of the Ack field or end of frame, or will violate the bit stuffing rules applied to all fields before the CRC delimiter. This will indicate to the other nodes on the network that an error has occurred. The other nodes will in turn also transmit an error flag and the result will be the superposition of error flags on the bus. The length of this part of the frame can range from 6 to 12 bits. Each node will start sending recessive bits after it has completed transmitting its error flag and monitor the bus until it detects a recessive bit. It will then send a further 8 recessive bits to generate the error frame delimiter. A passive error flag consists of 6 recessive bits.

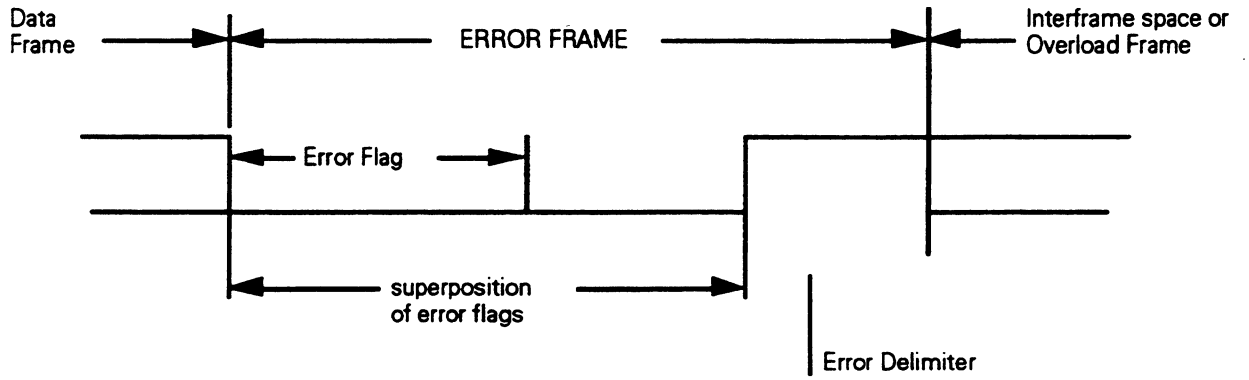


Figure 3 CAN Error Frame

The type of error flag sent by a node is determined by the state of the node. The node can be in one of three states. These are:

- Error Active

An error active node sends an active error flag in response to a detected error

- Error Passive

An error passive node sends a passive error flag in a response to a detected error.

- Bus Off

An node which is 'bus off' does not participate in bus activities

The state of each node is defined by a number of fault confinement rules which exist to allow temporary bus errors to be distinguished from permanent failures. A permanent failure is deemed to have occurred when an average of one in eight messages are corrupted. If a node continues to see a failure over a period of time then the node removes itself from the bus.

To allow permanent failures to be recognised, there are two error counts (transmit and receive) implemented in CAN. In general the error counts are incremented by a fixed amount when a transmitted or received message is corrupted, and decremented by a fixed amount if a message is transmitted or received without error. If both error counts are below 127 then the node is error active. When either of the error counts becomes greater than 127 the node becomes error passive. If either of the error counts becomes greater than 255 then the node becomes 'bus off' and it takes no further part in CAN bus activity until 128 occurrences of 11 consecutive bits have been monitored on the bus. Refer to the Bosch CAN specification for full details of the fault confinement rules.

Bit Timing

Figure 4 shows the bit time divided into a number of segments. These segments are derived from a number of fixed time units, time quanta, derived from the clock input to the CAN module. In the Motorola CAN module the time quanta length is derived from the i/p clock using a dedicated programmable prescaler.

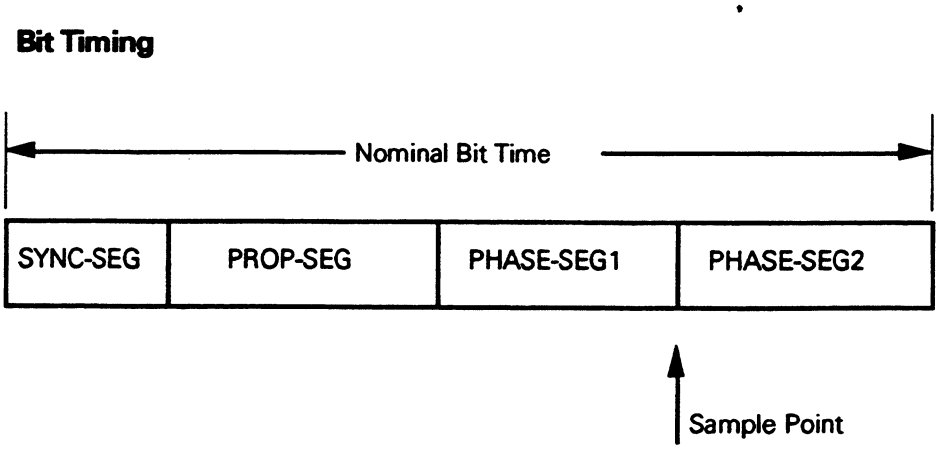


Figure 4 CAN Bit Timing

The SYNC_SEG is one time quantum long and is used to allow synchronisation between the various nodes on the bus.

The PROP_SEG, which can be from 1 to 8 time quanta in length, is used to compensate for physical delays on the CAN bus and within the output driver and receiver circuits of the nodes. The PROP_SEG must be at least twice the maximum signal propagation delay time between any two nodes on the network.

The PHASE_SEG1 and PHASE_SEG2, which can be from 1 to 8 time quanta long, can be respectively lengthened or shortened to allow synchronisation or resynchronisation with other nodes on the bus. The amount by which the bit length is altered is dependant on the edge positions of the incoming bits with respect to the SYNC_SEG position, as perceived by the host CAN node logic.

Synchronisation occurs whenever a recessive to dominant edge is detected on the bus during a bus idle period. Here the CAN module logic will set its bit timing so that the incoming edge lies within the SYNC_SEG portion of the bit time. This process is referred to as "Hard Synchronisation" in the Bosch CAN Specification.

Resynchronisation occurs on every recessive to dominant edge of an incoming message. There is also an option available for resynchronisation to occur on dominant to recessive edges. During resynchronisation, if the incoming edge lies outside the SYNC_SEG, then PHASE_SEG1 can be lengthened or PHASE_SEG2 can be shortened depending on whether the edge lies before the sample point or after the sample point. The maximum amount by which either PHASE_SEG1 or PHASE_SEG2 can be altered is known as the RESYNCHRONISATION JUMP WIDTH. This is programmable up to a maximum of 4 time quanta.

Motorola CAN Module

Refer to the MC68HC(7)05X4 Advance Information for full details of the Motorola CAN module.

The CAN module includes all the hardware necessary to implement the CAN Transfer layer and meets all the requirements of the Bosch specification. Figure 5 shows how the CAN module buffers are arranged in the MCU memory map (MC68HC05X4 and MC68HC05X16).

The module contains one full transmit buffer with registers to hold the identifier, the RTR bit, the data length code and up to eight data bytes. The eight most significant bits of the identifier are held in the Transmit Buffer Identifier (TBI) register located at $2A. The remaining bits of the identifier, the RTR bit and the data length code are located in the Remote Transmission Request and Data Length Code (RTRDL) register, located at $2B.

The Transmit Data Segment (TDS) registers are located from $2C to $33 and are used to hold the data bytes to be transmitted.

There are two full receive buffers. They are arranged in a double buffered configuration so that both buffers are accessed in the same area of the memory map. When the first buffer is filled it can be read by the CPU as a second incoming message is being transferred by the CAN receiver logic into the second receive buffer. Setting the Release Receive Buffer (RRB) bit in the CAN Command Register will cause the Receive Buffer currently being accessed by the CPU to be released and the second buffer to be moved into the memory map.

The buffering arrangement of the CAN module requires CPU intervention for each single message either received or transmitted.

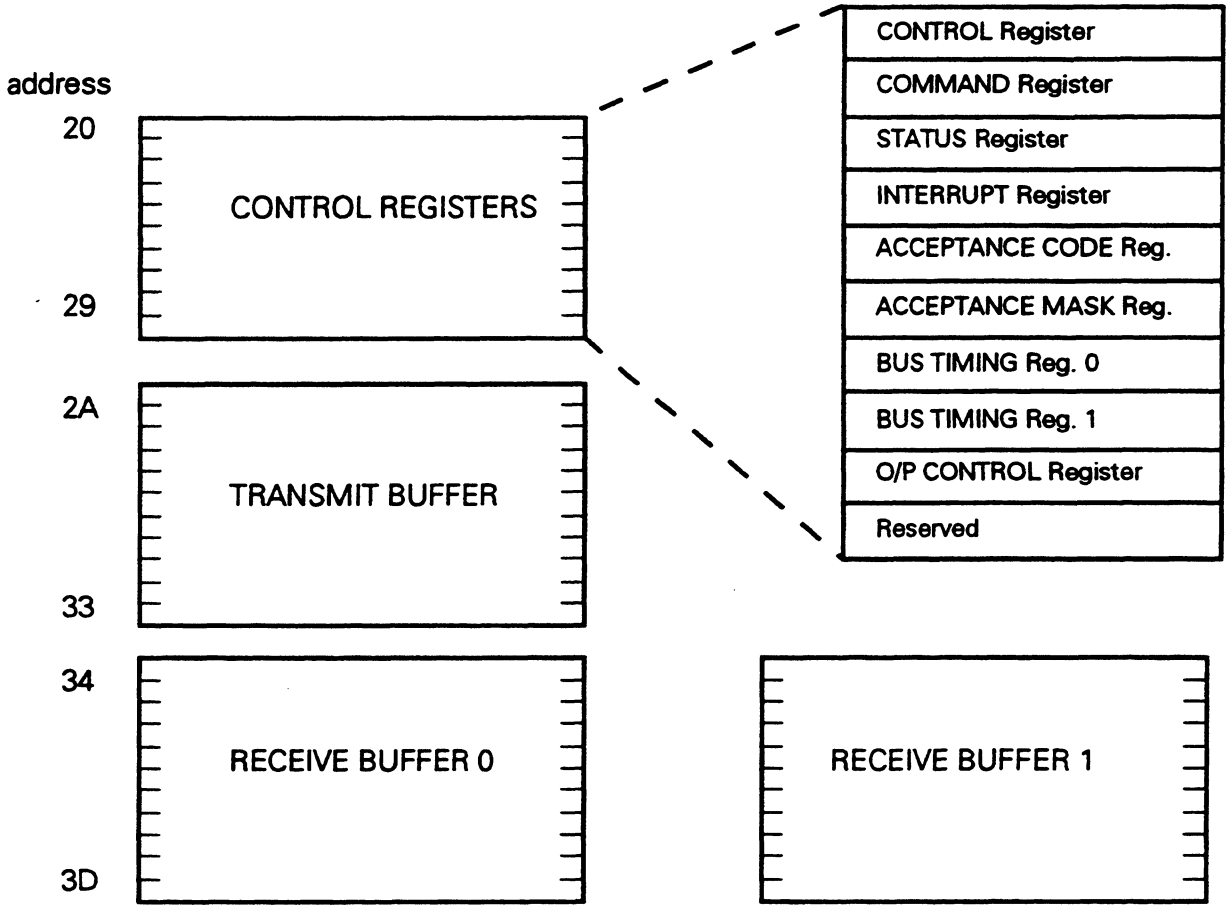


Figure 5 CAN Module Buffer and Register Map

The Control register provides local mask bits for the CAN module interrupts. In addition it contains the Reset Request bit which is set to disable the CAN module operation and allow access to the message filtering, bus timing and output control registers.

The Command register is a write only register which contains the Release Receive Buffer (RRB) and Transmit Request (TR) bits.

The Status register provides information on a number of conditions which can occur in the CAN module. These include information on whether or not the last requested message has completed transmission, new data has been received or the transmit buffer can be accessed to store new data. It also indicates if the CAN module has become 'off bus', as well as giving limited information on the state of the error counters.

The Interrupt register can be read to determine the source of a CAN interrupt. There are five CAN interrupts that can occur. These include: Wake Up; Data Overrun (a third message being received before either of the Receive Buffers have been released); Error; Transmit Complete and Receive.

The Acceptance Code and Acceptance Mask registers are used to provide limited message filtering on the eight most significant bits of the identifier. If a message is received with an identifier outside of the acceptance range of the CAN node then the node will respond by transmitting a dominant bit in the correct position in the Ack field, but it will not transfer the message to the receive buffers or indicate to the CPU that a new message has been received.

The Bus Timing registers are used to select a suitable baud rate prescaler value to provide an appropriate tSCL value which is then used to derive the bit time and position of the sample point within the bit. The Bus Timing registers allow two values, TSEG1 and TSEG2, to be defined. TSEG1 is the sum of PHASE_SEG1 and the PROP_SEG. TSEG2 is equal to PHASE_SEG2. The Bus Timing Registers also define the size of the RESYNCHRONISATION JUMP WIDTH.

The Output Control Registers are used to determine the configuration of the output drivers on the CAN transmit pins. The output drivers can be selected for pull-up, pull-down, or push-pull operation by selectively enabling or disabling the P type and N type transistors in the output driver circuits. They also provide a number of options for the way in which the data is transmitted. The most usual configuration is for complementary levels to be transmitted on the Tx0 and Tx1 pins (for two-wire differential operation).

CAN Driver Interface

The driver routines have, as far as possible, been optimised for maximum speed of operation. In some cases this has resulted in using more ROM than would otherwise be required. An attempt has been made to minimise the CPU overhead required to service the CAN module. The routines provide an interface between the user's application software and the CAN module hardware. The combination of the driver routines and the module hardware provide all functions associated with all layers up to and including the object layer and allow messages to be handled in a structured manner.

There are seven main routines in the driver software. The routines INIT and CANINIT must be called by the user application software to initialise the CAN module and a number of RAM registers used by the driver software. SENDDAT and SENDREM are called by the user's application software to initiate either a data frame or a remote frame transmission respectively. None of the other routines are called by the user's application software. CANIRQ is executed in response to a CAN module interrupt and will direct the program flow into the appropriate driver routine depending on the source of the interrupt. TRANSMIT is called by either SENDDAT, SENDREM or CANIRQ and, handles the transfer of message transmit data into the CAN module and all subsequent actions required for message transmission. RECEIVE is called by CANIRQ and is responsible for the filtering of incoming messages and the transfer of received message data from the CAN module to MCU memory. These routines are described in detail later.

There are essentially six main component memory areas used in the interface between the User Application software and the CAN Driver software. They exist in the general MCU memory, either RAM,ROM or EPROM. These component areas are:

1 – The Message Buffers (RAM)
2 – The Message Definition Table (RAM, ROM or E(E)PROM))
3 – The User Interrupt Jump Table (RAM, ROM or E(E)PROM))
4 – The User Interrupt Enable Register (RAM)
5 – The Control/Status Register Area (RAM)
6 – The Transmit Queue (RAM)

The Message Buffers

The Message Buffers hold data which has been received, or data which is to be transmitted, by the CAN module. Each Message Buffer has a specific identifier assigned to it. This is the identifier which is transmitted in the arbitration field of the CAN message frame. This will correspond to either a node, a group of nodes, or to a specific function which can be acted upon by the nodes within the CAN network. For the purposes of this application note the host node is defined as the MCU in which the CAN driver routines and the application software are resident. All Message Buffers must be located in Page 0 memory.

Only messages with an identifier, for which there is a corresponding Message Buffer assigned, can be transmitted by the host node. If the host node receives a message with an identifier for which there is no corresponding Message Buffer, then the driver software will not store the message contents. There is no limit, other than RAM size, to the number of Message Buffers that can be defined by the user.

Each Message Buffer is subdivided into a Message Transmit Buffer and a Message Receive Buffer. When the user application software is required to send a message with a particular identifier, it needs to place the data to be transmitted in the appropriate Message Transmit Buffer before calling the SENDDAT routine. All messages sent or received by the host node will have a specific identifier assigned to them. A message received by the CAN module will be checked by the driver software to see if its identifier matches any of the identifiers assigned to the Message Buffers. If a match is found then the data contents are transferred by the driver software to the appropriate Message Receive Buffer. If no match is found then the message is discarded.

Message Definition Table

The Message Definition Table must be generated by the user for the specific application. The table can reside in ROM, EPROM or RAM. In the case of the HC05X4, with its limited RAM, the Message Definition Table would most likely be located in ROM. The Message Definition Table assigns an identifier to each Message Buffer and allocates RAM for the Receive and Transmit buffers in each Message Buffer. The structure of the table is shown in Figure 6.

The table consists of a total number of i entries, the number i being defined by the user application. Each entry corresponds to a specific Message Buffer. All entries comprise 8 bytes and contain 5 separate components. These components are:

The Identifier (4 bytes) – This assigns an identifier value to the corresponding Message Buffer. Messages transmitted from, or received into, the Message Buffer will contain the identifier value assigned in the Message Definition Table. Four bytes of each entry have been assigned to the identifier. This exceeds the requirements of the identifier used in the current CAN Rev. 1.2 specification. However, the extra bytes will, if required, allow for easy modification of the driver routines to accommodate the Rev. 2.0 CAN protocol with its 29-bit identifier. They also simplify the software required to increment through the table when searching for an appropriate Message Buffer for a newly received message. The MSB of the 11-bit identifier (bit 10) is placed in bit 7 of byte 2 in the table entry (MBDF + 2 for Message Buffer 0) and the LSB of the identifier is placed in bit 5 of byte 3 (MBDF + 3 for Message Buffer 0). All other bits in the table identifier bytes are cleared to 0.

Receive Buffer Pointer (1 byte) – This byte points to the start of the Message Receive Buffer.

Transmit Buffer Pointer (1 byte) – This byte points to the start of the Message Transmit Buffer.

Message Buffer No.	Address					
0	MBDF	Identifier 0	Rx Buf.0 Ptr.	Tx Buf.0 Ptr.	Tx Buf 0 Siz	Rx Buf 0 Siz
1	MBDF+8	Identifier 1	Rx Buf.1 Ptr.	Tx Buf.1 Ptr.	Tx Buf 1 Siz	Rx Buf 1 Siz
2	MBDF+16					
i	MBDF+8i	Identifier i	Rx Buf.i Ptr.	Tx Buf.i Ptr.	Tx Buf i Siz	Rx Buf i Siz

Figure 6 Message Definition Table

Transmit Buffer size (1 byte) – The lower nybble of this byte is used to determine the number of bytes assigned to the Message Transmit Buffer. This can be from zero to eight. The upper nybble is not used.

Receive Buffer Size (1 byte) – The lower nybble of this byte is used to determine the number of bytes assigned to the Message Receive Buffer. This can be from zero to eight. The upper nybble is not used.

Each Message Buffer is assigned a number (n), which is dependant on its position within the Message Definition Table. The first entry is assigned the number 0 and the last entry is assigned the number (i-1).

One possible disadvantage of placing the Message Definition table in ROM is that the size of the transmitted messages, and possibly of the received messages, cannot be modified once the application software is operating. However, in most applications the message sizes would be predefined and fixed. Another disadvantage is the inability to add further identifiers. If this is a problem then the easiest solution is to place the table in RAM or, if using the HC05X16, in EEPROM.

The ability to place the Message Transmit and Receive Buffers anywhere in the first 256 bytes of the memory map means that they do not necessarily have to be placed in RAM. For example, on the HC05X16, the SCI control, status and data registers could be assigned as Receive and Transmit Buffers for a particular identifier, allowing the SCI to be controlled remotely by another node on the CAN bus.

The User Interrupt Jump Table (UIJT)

Figure 7 shows the structure of the User Interrupt Jump Table. The first entry is a single byte indicating the number of Message Buffers assigned in the Message Definition Table. This is followed by a series of extended jump instructions. The jump addresses are defined by the user and allow the driver software to vector into the application software when specific CAN interrupts occur. Routines pointed to by the jump addresses must be terminated with an RTS instruction to allow a proper return to the driver software. The facility to vector into the application software is selectively enabled for each CAN interrupt through the User Interrupt Enable register. The normal driver functions carried out in response to a particular CAN interrupt will be executed prior to the jump to the application software.

MCST	Number of Messager Buffers (i)
MCST+$01	$CC (Ext JMP inst) + 2 byte User Transmit Routine Address
MCST+$04	$CC + 2 byte User Receive Routine Address
MCST+$07	$CC + 2 byte User Error Routine Address
MCST+$0A	$CC + 2 byte User Overflow Routine Address
MCST+$0D	$CC + 2 byte User Wakeup Routine Address

Figure 7 User Interrupt Jump Table

The Message Definition Table and the User Interrupt Jump table must be generated by the user to suit their particular application. Additionally, the user must allocate space in suitable areas of memory for the Message Buffers, the Control/Status register area, the User Interrupt Enable register and the Transmit Queue.

User Interrupt Enable Register (USINTE)

This register contains the following:

Bit 0 – Receive Interrupt Enable (RIEN)
Bit 1 – Transmit Interrupt Enable (TIEN)
Bit 2 – Error Interrupt Enable (ERIEN)
Bit 3 – Overflow Interrupt Enable (OFIEN)
Bit 4 – Wake-Up Interrupt Enable (WUIEN)

Any of these bits will, if set, cause the driver software to vector to the appropriate JMP instruction in the User Interrupt Jump Table when an interrupt occurs. When the bits are cleared only the normal driver functions will be carried out in response to an interrupt.

The Control/Status Registers

Each Message Buffer has two control/status registers assigned to it.

These are M(n)CS1 and M(n)CS2 (where n is the Message Buffer number). The control/status registers are arranged sequentially in memory, starting from the user defined address MCSOFF (See Figure 8). Figure 9 shows contents of the control/status registers.

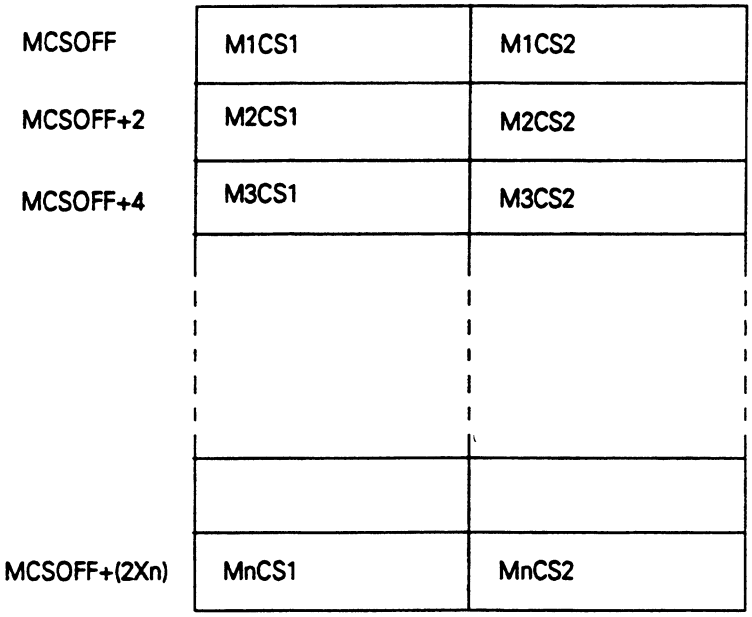


Figure 8 Message Control/Status Register Location

M(n)CS1

7	6	5	4	3	2	1	0
TPND	REMF	RBFU	RBOF	AUTO TX	RREC	ATPND	DERF

M(n)CS2

7	6	5	4	3	2	1	0
—	—	—	—	DSIZ3	DSIZ2	DSIZ1	DSIZ0

Figure 9 M(n)CS1/2 Registers

These registers are modified by both the application software and the driver routines. They can be read by the application software to provide Message Buffer status information. They also allow the application software to control the operation of the Message Buffers.

M(n)CS1

The bit functions of M(n)CS1 are as follows:

7 – Transmit Pending (TPND) – This indicates that a message is in the transmit queue awaiting transmission. The Message Transmit Buffer contents should not be altered or updated while this bit is set. This bit is cleared by the driver software after the Message Transmit Buffer (n) contents have been transferred to the CAN module transmit buffer. It should not be modified by the application software.

6 – Remote Frame Received (REMR) – This bit is set by the driver software to indicate that a remote frame, with an identifier corresponding to Message Buffer (n), has been received. It is cleared when the Transmit Buffer (n) contents are transferred to the CAN module transmit buffer to be transmitted as a data frame. It does not require to be modified by the application software.

5 – Receive Buffer Full (RBFU) – This bit is set to indicate that a new message has been received, with an identifier corresponding to Message Buffer (n), and stored in the Message Receive Buffer (n). When the application software has retrieved the received message data contents, it should clear this bit so that subsequent messages, with the same identifier, can be stored.

4 – Receive Buffer Overflow (RBOF) – This bit is set by the driver routines to indicate that a new message has been received but has not been stored in the Message Receive Buffer because RBFU is set.

3 – Auto Transmit (AUTOTX) – When this bit is set by the application software, the driver software will respond automatically to a received remote frame by entering the Message Buffer number into the transmit queue. This bit is not modified by the driver software.

2 – Remote Request (RREQ) – This bit is used by the driver software to cause a remote frame, with an identifier corresponding to Message Buffer (n) to be transmitted. It should not be modified by the application software.

1 – Automatic Response Pending (ATPND) – This bit is set by the driver software to indicate that a remote frame has been received and that a message has been automatically entered into the transmit queue in response. It is cleared when the Transmit Buffer (n) contents are transferred to the CAN module to be transmitted as a data frame.

M(n)CS2

Bits 0 to 3 of M(n)CS2 are used to indicate the number of data bytes that were contained in the last received message. If this number is greater than the size of the allocated Message Receive Buffer then the remaining portion of the received data will be dropped and only the bytes for which there is space will be stored.

The Transmit Queue

The transmit queue is a simple FIFO buffer located in RAM and used by the driver software to determine the next message to be transmitted. Each entry in the queue takes the form of a Message Buffer number which is loaded into the end of the queue whenever the SENDDAT, SENDREM and, under certain conditions, RECEIVE routines are executed. When the TRANSMIT routine is executed it unloads the first entry from the start of the queue. The location and size of the queue are determined by the byte equate

constants QSTA and QEND. QPT0 and QPT1 are RAM locations which point to respectively, the first entry into the queue and the next free location in the queue. As the pointers are incremented past the last location within the queue buffer, they are reset so that they wrap around to the start of the queue buffer.

Figure 10 illustrates how the Transmit Queue would operate. In this example up to 10 entries can be entered into the queue. There are 5 messages awaiting transmission. The RAM register QPT1 points to the next entry to be unloaded by the TRANSMIT routine. QPT2 points to the free location where the next message number should be loaded into the queue.

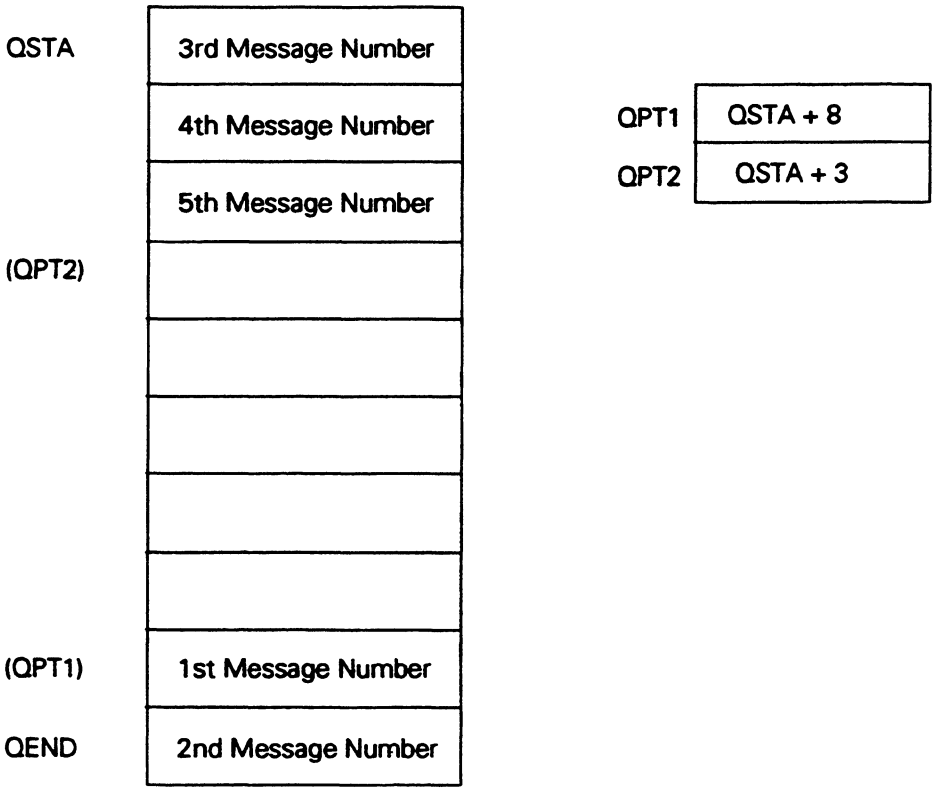


Figure 10 Transmit Queue Example

There is no requirement for the user application software to access the Transmit queue. All queue entries are loaded and unloaded by the driver routines.

CAN Driver Routines

Flowcharts for the main routines are included at the end of this application note.

The SENDDAT routine is called by the application software to request a data frame transmission. It should be entered with the required Message Buffer number n held in the accumulator. It loads the message number into the transmit queue and, if the CAN module transmitter is inactive, calls the TRANSMIT routine to initiate the transmission. If the CAN module is currently transmitting, the routine returns and allows the CANIRQ routine to initiate the transmission.

The SENDREM routine is almost identical to the SENDDAT routine except that it sets the RREQ bit in the appropriate M(n)CS1 register. This indicates to the TRANSMIT routine that a remote frame should be transmitted.

The TRANSMIT routine is called by SENDREM, SENDDAT or CANIRQ. This routine, after checking that the CAN module transmit buffer is free, takes the next pending Message Buffer number from the transmit queue and fetches the correct identifier from the Message Definition Table. This is then loaded into the CAN module transmit buffer. The state of the RREQ bit in M(n)CS1 is checked and the RTR bit in the CAN module transmit buffer is set accordingly, to select either a remote frame or data frame transmission. The TPND, REMR and ATPEND bits in M(n)CS1 are cleared. If a data frame is to be sent, the Message Transmit Buffer contents are transferred to the CAN transmit buffer. The TR bit in the CAN Command register is then set to initiate the message frame transmission.

The RECEIVE routine is called by CANIRQ when a CAN Module Receive interrupt occurs. The routine will search through the Message Definition Table to find an identifier which matches the identifier of the incoming message. If the end of the table is reached and no match is found the message is dropped and the CAN module receive buffer is released. No indication is made to the application software that the message was received.

If the incoming message is a remote frame, and AUTOTX in M(n)CS1 is set, then Message Buffer (n) is loaded into the transmit queue and ATPND, REMR, and TPND are set. If AUTOTX is clear then only REMR is set. The CAN receive buffer is then released and the routine returns.

If the incoming message is a data frame then RBFU is checked to determine if the Message Receive Buffer is available. If RBFU is clear then the routine will store the incoming message size in the appropriate M(n)CS2 register. The message data contents are then transferred to the Message Receive Buffer and the RBFU bit in M(n)CS1 is set. The CAN module receive buffer is then released and the routine returns.

If RBFU is set then the routine sets RBOF and continues to search through the Message Definition Table for another Message Buffer entry with the correct identifier. It is possible for more than one Message Buffer to be assigned to a particular identifier. This might be done if it is likely that the application software will not be able to process the data from a received message in sufficient time to be able to release the Message Receive Buffer before a second message arrives with the same identifier

It should be noted that a large part of the execution time of this routine is spent searching the Message Definition Table for an entry with the correct identifier for the received message. In order to minimise the CPU resource used by the routine, the most commonly received identifiers should have entries at the start of the Message Definition Table.

The CANIRQ routine is executed in response to a CAN module interrupt. The source of the interrupt is determined from the CAN Interrupt register. If a receive or transmit interrupt occurs then the appropriate routine is called. The CANIRQ routine can also vector into the application software, depending on the state of the User Interrupt Register. No routines are provided for handling overrun or error interrupts as the way these are dealt with will be very much dependant on the application.

The INIT routine is used to initialise the M(n)CS1/2 and the USINTE registers.

The CANINIT routine is used to set up the CAN module registers. This routine is provided as an example only and is designed to operate with the interface circuit shown in Figure 11. The CAN register set up requirements will be dependant on the application.

The Acceptance Code and Acceptance Mask registers are configured to accept identifiers in the range $700 to $77F. The identifiers used in the sample Message Definition Table, in the software listing, all lie within this range.

The bit time set up in this routine is 10 µS (4 MHz clock i/p to the CAN module). The t_{SCL} period is defined by the Baud Rate Prescaler bits BRP0 to BRP4 in Bus Timing Register 0. The CANINIT routine sets this register to $01 giving a T_{SCL} value of 1 µS for a clock input frequency of 4 MHz. The TSEG1 and TSEG2 segments of the bit time are defined by Bus Timing Register 1. The CANINIT routine sets this register to $34. TSEG1 = 5 x t_{SCL}, giving a PROP_SEG value of 1 t_{SCL} and a PHASE_SEG 1 value of 4 x t_{SCL}. The PROP_SEG value is not explicitly defined but must be accounted for when defining the length of TSEG1. TSEG2 (which defines PHASE_SEG2) = 4 x t_{SCL}. The SAMP bit in BusTiming Reg. 1 is 0, selecting the single sample per bit option. The selected resynchronisation jump width is defined by SJW1 and SJW0 in Bus Timing Register 0 and is 4 x t_{SCL}. This bit timing meets the requirements for the maximum oscillator tolerance, which is achieved when the length of the phase buffer segments is the same as the resynchronisation jump width, and the propagation delay time is equal to 1 x t_{SCL} (i.e., the maximum possible proportion of the bit time is allocated for resynchronisation). The trade off in this case is the reduced time that can be allowed for propagation delays. This conforms to the bit timing, recommended in the CAN specification, for slave CAN devices without programmable bit timing.

The output drivers are configured from the Output Control Register. The CANINIT routine initialises this register to $AA. The o/p driver for Tx0 is configured for pulldown operation by clearing OCTP0 and setting OCTN1. This allows Tx0 to drive the CANH bus line low (0.3 V approx.) for a dominant bit. For a recessive bit, the bus line is pulled high (3.25 V) by the termination network. The o/p driver for Tx1 is configured for pullup operation by setting OCTP1 and clearing OCTN1. This allows the CANL bus line to be driven high (4.7 V approx.) for a dominant bit. For a recessive bit the CANL line is held low (1.25 V) by the termination network. The OCMODE1 bit is set and OCMODE0 bit cleared to select 'Normal mode 1' transmission, where the bit stream is output on both Tx0 and Tx1. The OCPOL1 bit is set and the OCPOL0 bit cleared. This causes the bit stream output on Tx0 to be transmitted normally and the bit stream on Tx1 to be transmitted inverted, providing the required differential signal.

The receiver network shown in Figure 11 uses a resistor voltage divider network, referenced to VDD/2, to increase the common mode range specified for the comparator input pins to be increased by a factor of 6 on the actual bus.

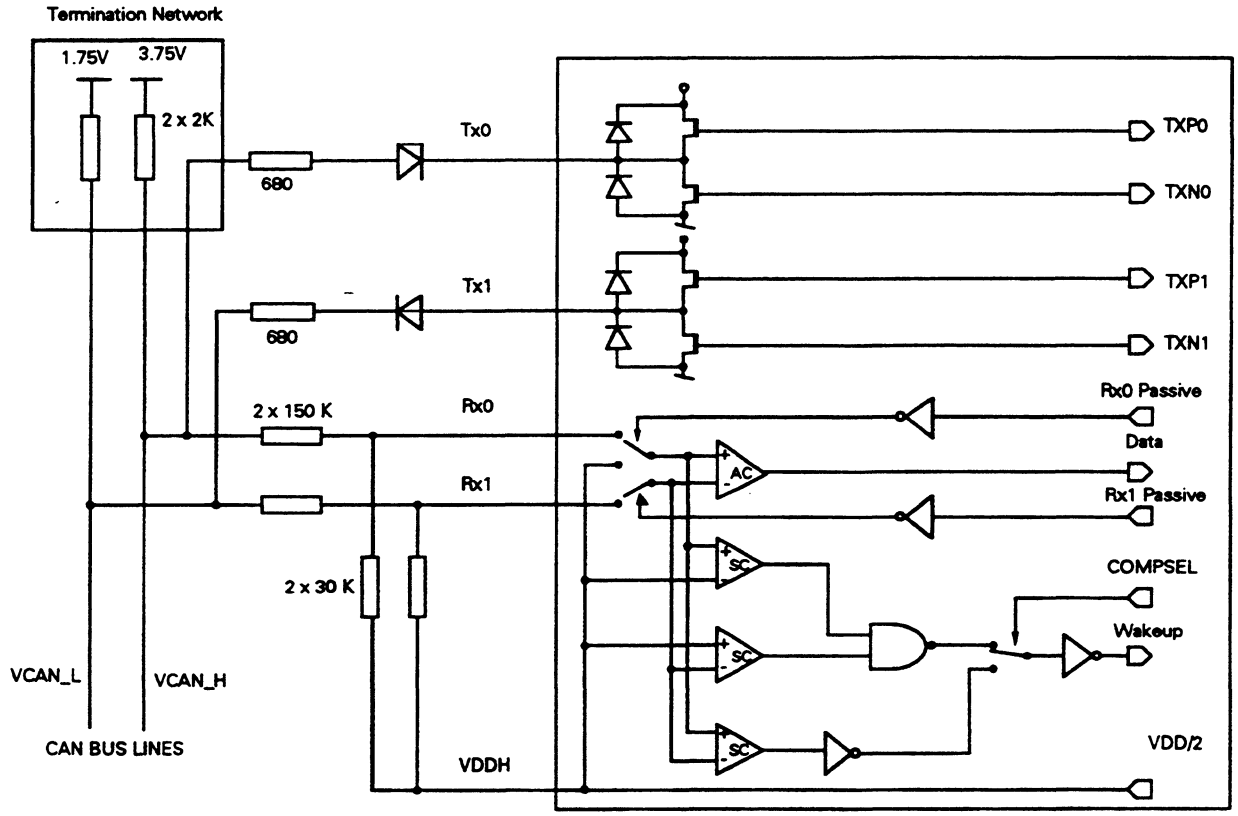


Figure 11 Physical Interface Example

Processor Overhead

The driver routines were written, as far as possible, to minimise the processing time required by the CPU. The original aim in developing this software was to have routines which would allow the CAN module to operate on a 100% loaded bus (i.e.. receiving and storing messages transmitted in a constant stream with the minimum inter frame space) operating at a minimum rate of 100 Kbits per second. In practice it is unlikely that any bus would be anywhere near 100% loaded. However, this is a worst case situation which must be allowed for.

A remote frame is 44 bits in length. To this can be added the inter frame space of 3 bits. Therefore, using a 10 µS bit time, the shortest possible time for a remote frame transmission is 470 µS. This is assuming that the frame requires no stuff bits to be inserted. Each data frame transmitted takes a minimum time of 470 µS plus an additional 80 µS for each data byte transmitted. Therefore an 8 byte data frame takes 1.11 mS to be transmitted.

The TRANSMIT routine execution time was both calculated and measured at 118 µS with a CPU internal operating frequency of 2 MHz. This does not change for varying numbers of data bytes, except in the case where a data frame is sent with 0 bytes, where the routine takes 84 µS. Because of the single transmit buffer the host node is unable to send a continuous stream of frames. There is a 118 µS delay between

each frame. However, when the host node is transmitting single byte data frames at its maximum rate, approximately 17.6% of the CPU's time is spent servicing the TRANSMIT driver routine. For 8 byte data frames the amount of CPU time required for the TRANSMIT driver routine is approximately 9.6%.

There are two factors which affect the execution time of the RECEIVE routine; (1) the size of the Message Receive Buffers and (2) the position of the Message Buffer, required for the incoming message identifier, within the Message Definition Table. The RECEIVE routine execution time was calculated and measured for a variety of Message Buffer positions (1 to 5) and a range of sizes (0 to 8 bytes) for the Message Receive Buffer. Figure 12 summarises these results.

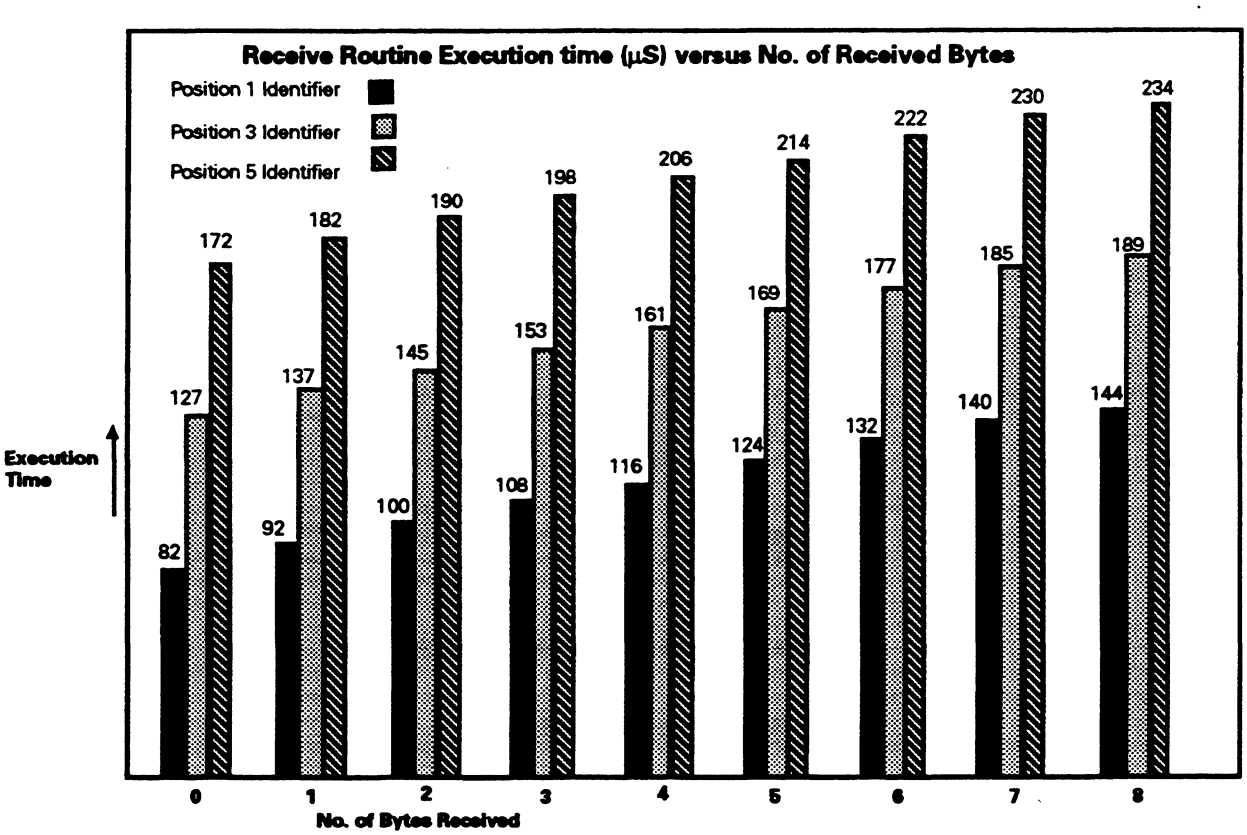


Figure 12 CPU Overhead for RECEIVE Routine

It can be seen from Figure 12 that, for a continuous stream of single byte data frames being transmitted on the bus, the CPU time required to service the RECEIVE routine is approximately 16.7%, for frames corresponding to a position 1 Message Buffer, and 33.1% for frames corresponding to a position 5 Message Buffer. For a stream of 8 byte frames, the CPU time required would be 12.8% , for position 1 Message Buffer frames, and 21.1 % for position 5 Message Buffer frames. These figures are somewhat artificial as it is unlikely that the situations described would ever occur, but they are intended only to give an indication of the required CPU overhead. In a real application it is unlikely that the bus would be loaded by more than 30 % to 40 %.

Figure 13 illustrates a possible worst case situation for required CPU overhead. The timings shown are for a transmission rate of 100 Kbits/s.

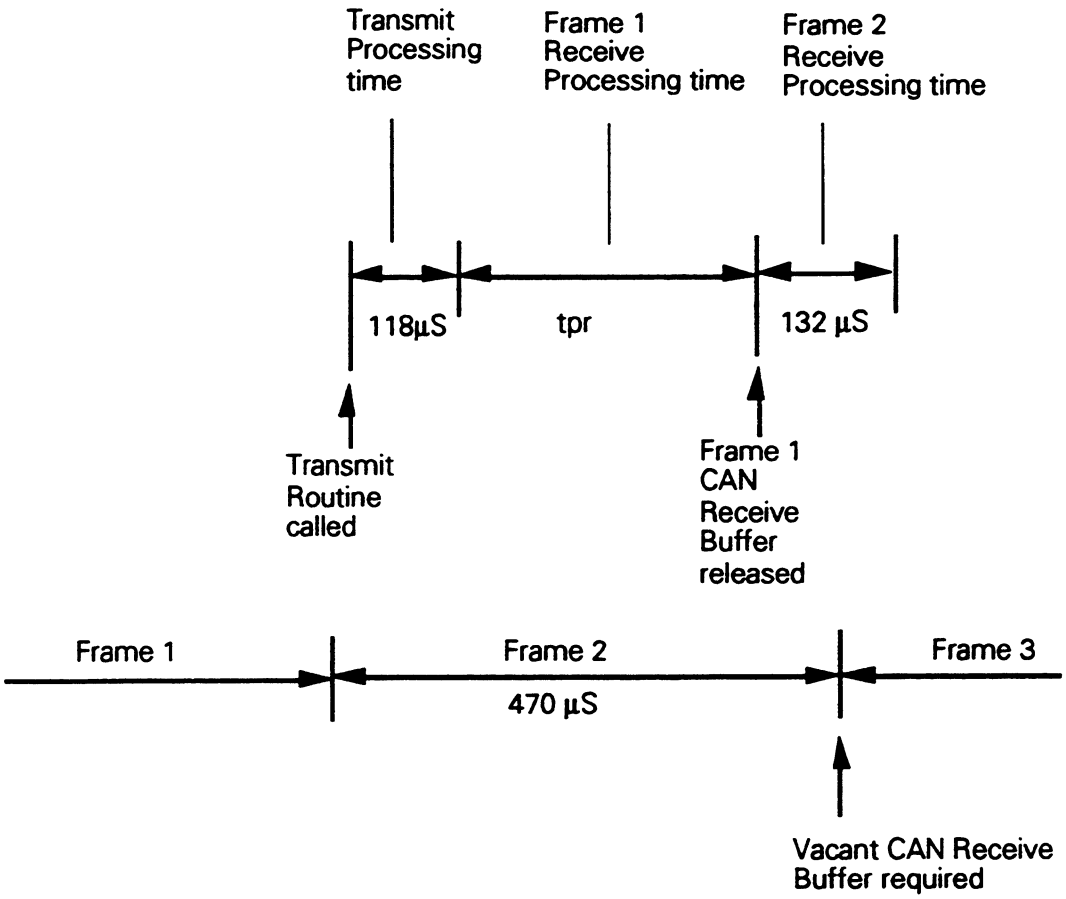
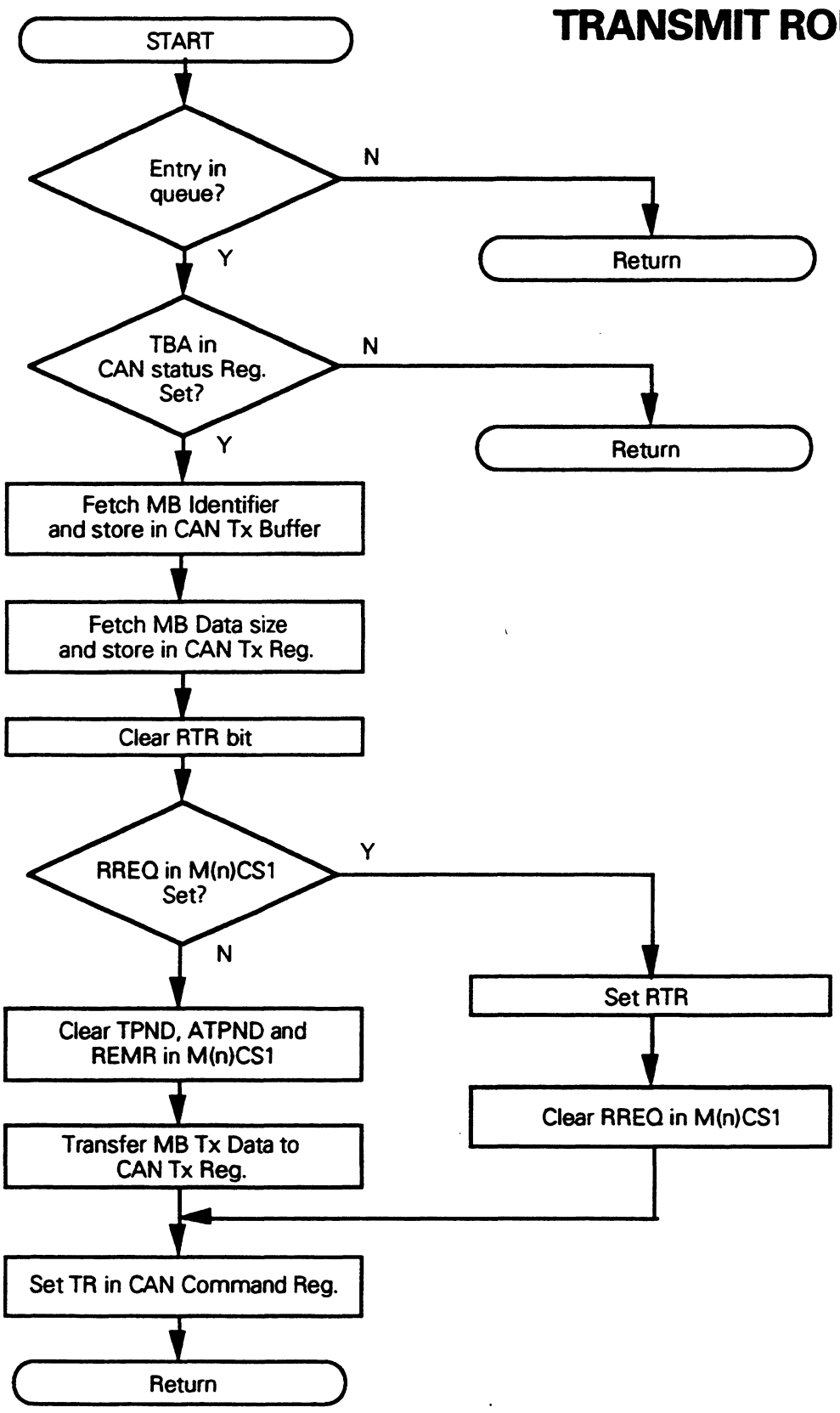
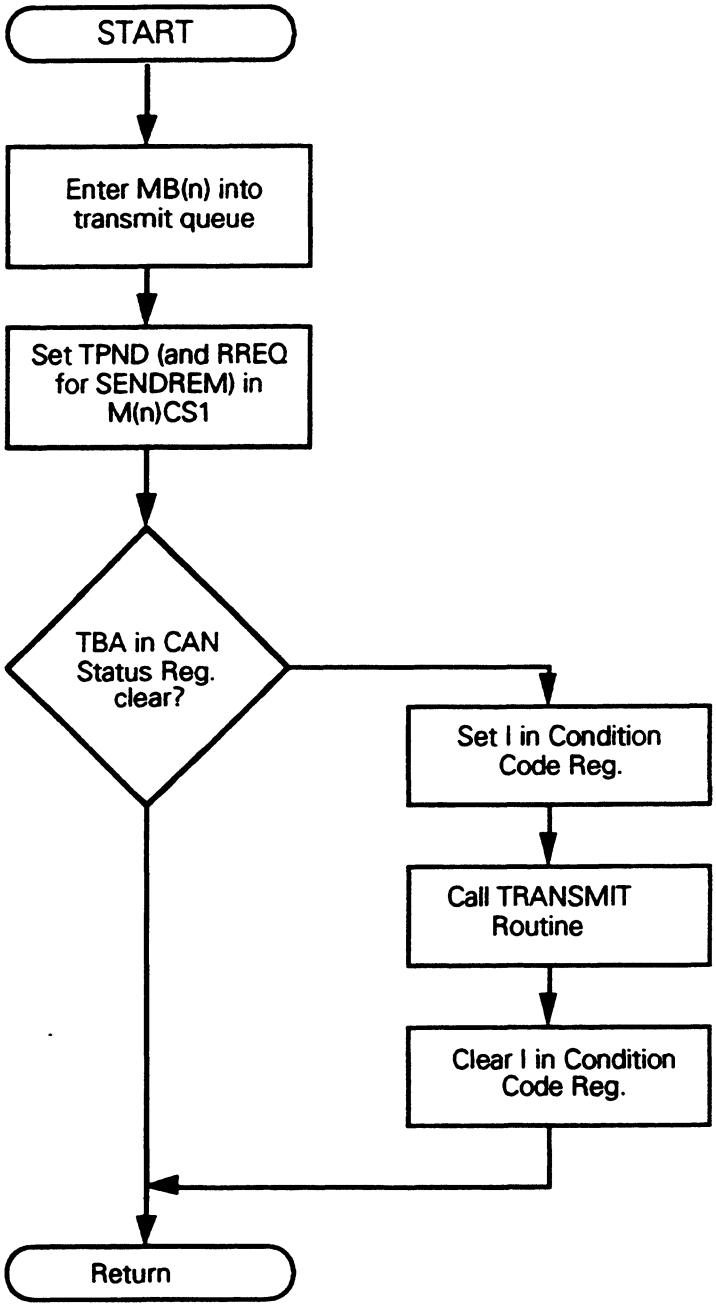


Figure 13 Worst Case Timing Requirements for Received Frames at 100 Kbits/s Transmission Rate

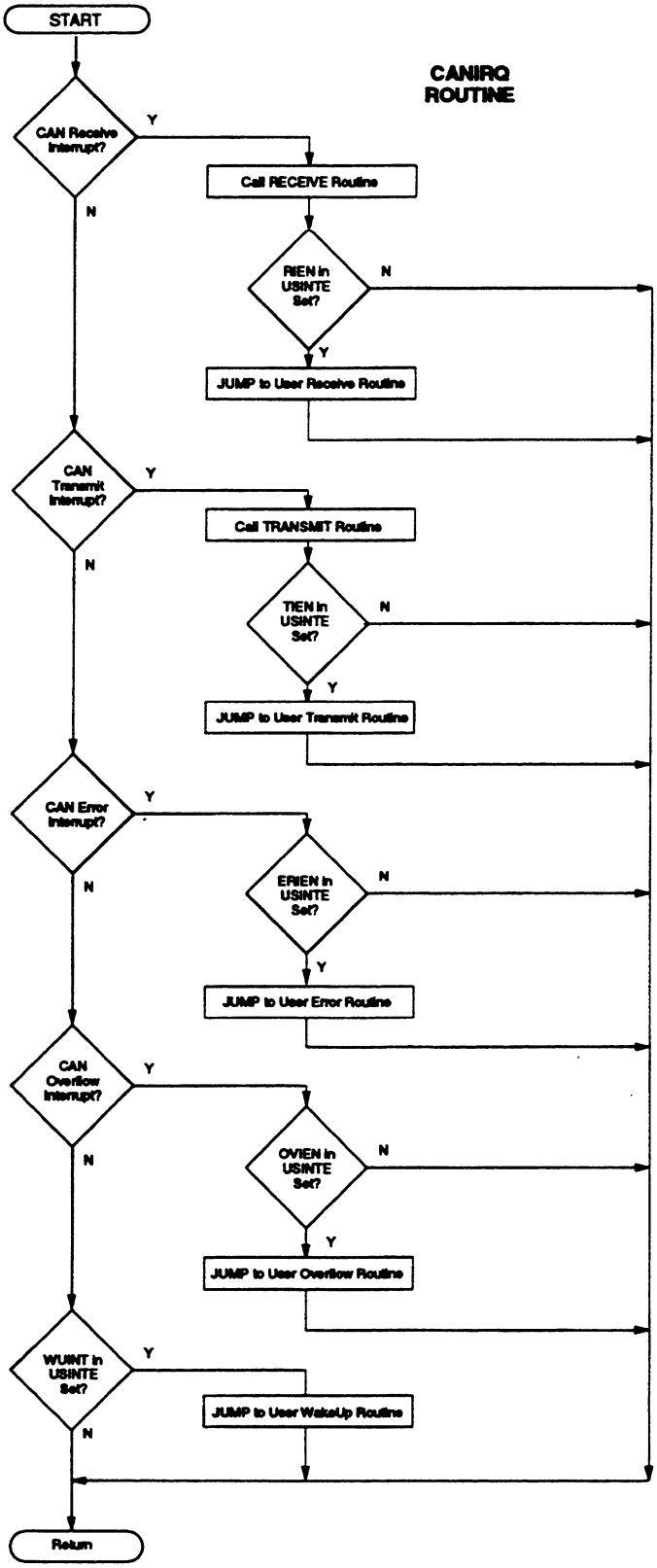
The host node starts to process a data frame transmission just prior to the CAN module completing reception of an 8 byte, position 5 Message Buffer data frame (Frame 1). The received data frame is immediately followed by a remote frame (Frame 2), which is in turn immediately followed by a further frame (Frame 3). In order for Frame 3 to be correctly received, the CPU must release the CAN receive buffer occupied by Frame 1 before Frame 3 transmission starts. In order for this to happen, the TRANSMIT routine execution time and the RECEIVE Routine execution time, for handling Frame 1, must be less than the transmit time for Frame 2 which is 470 µS. This means that the receive routine processing time t_{pr} must be less than 352 µS. The receive routine execution time when receiving an 8 byte data frame with a position 1 identifier is 144 µS. This execution time increases by 22.5 µS for each increment in the table position. This means that, under worst case conditions, the driver software can support up to 10 table entries and bus transmission rate of 100 Kbits/s without losing any incoming messages.



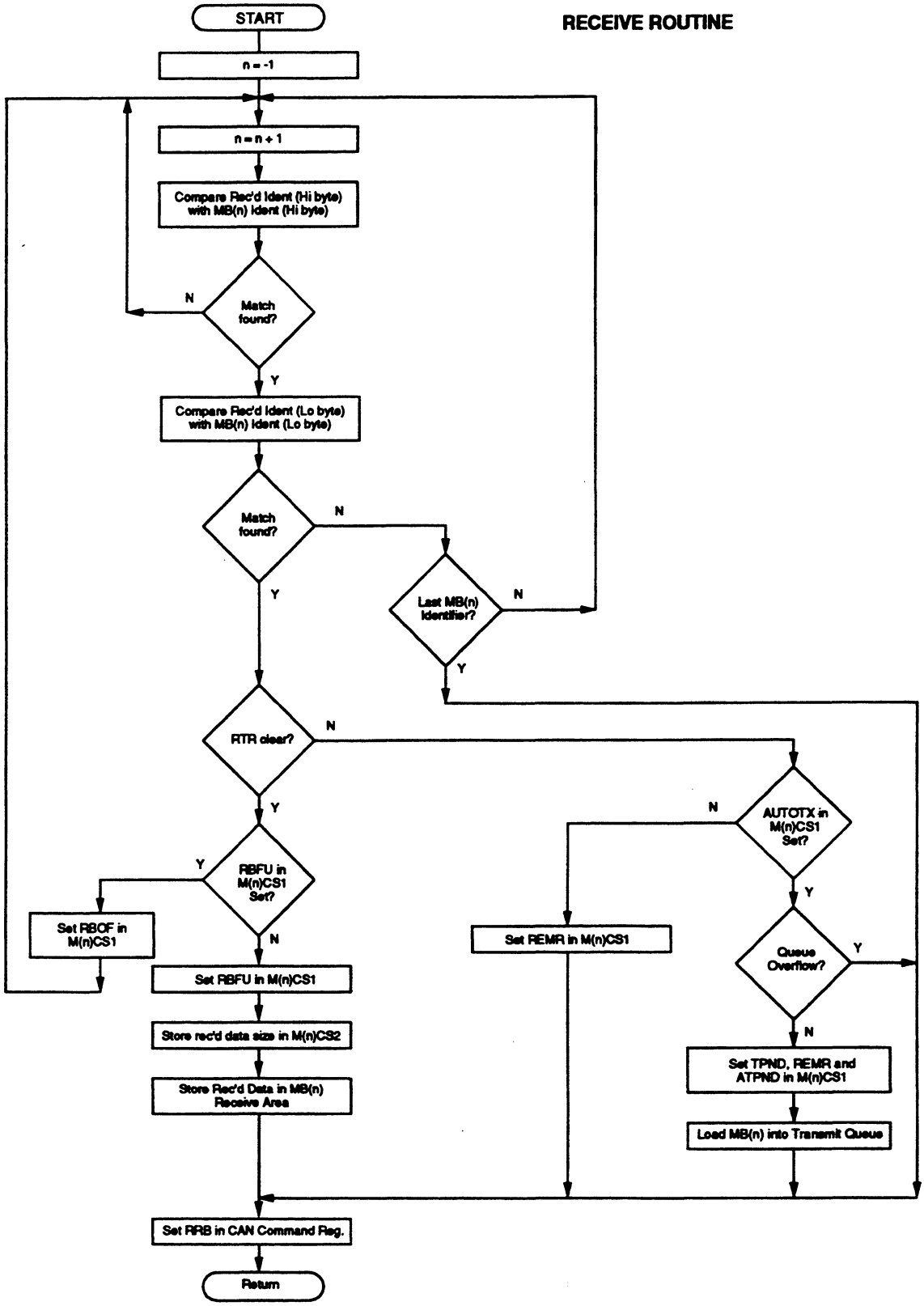
Transmit Routine



SENDDAT (SENDREM) Routine



CANIRQ Routine



Receive Routine

Driver Routines

```
0001                    ********************************************************************
0002                    *       CAN Module Driver Routines for the MC68HC05X4
0003                    *       Revision 1.0 - 16/3/92
0004                    *
0005                    *       This program contains CAN module driver routines which
0006                    *       provide an interface between application s/w running on
0007                    *       the HC05X4 and the on board CAN module. They allow messages
0008                    *       to be queued for transmission and automatically handle all
0009                    *       received messages. If required, the routines will provide
0010                    *       a response to remote requests from other CAN nodes, without
0011                    *       intervention from the application s/w.
0012                    *
0013                    ********************************************************************
0014                    *       Register Equates
0015 0000               PORTA   EQU   $00
0016 0001               PORTB   EQU   $01
0017 0004               DDRA    EQU   $04
0018 0005               DDRB    EQU   $05
0019 0003               PCR     EQU   $03       Port Configuration Register
0020 0012               TCR     EQU   $12       Timer Control Register
0021 0013               TSR     EQU   $13
0022 0019               TIMLO   EQU   $19
0023                    *
0024                    *       Bit Equates
0025                    *
0026 0005               TOF     EQU   5         Timer Overflow Flag
0027 0005               TOIE    EQU   5         Timer Overflow Interrupt Enable
0028 0004               TIMEN   EQU   4         Timer Enable
0029                    *
0030                    *       CAN Registers
0031                    *
0032 0020               CANCTRL EQU   $20       CAN Control Register
0033 0021               CANCOM  EQU   $21       CAN Command Register
0034 0022               CANSTAT EQU   $22       CAN Status Register
0035 0023               CANINT  EQU   $23       CAN Interrupt Register
0036 0024               CANACC  EQU   $24       CAN Acceptance Code Register
0037 0025               CANACM  EQU   $25       CAN Acceptance Mask Register
0038 0026               CANBT0  EQU   $26       CAN Bus Timing Register 1
0039 0027               CANBT1  EQU   $27       CAN Bus Timing Register 2
0040 0028               CANOPC  EQU   $28       CAN O/P Control Register
0041 002c               CANTX   EQU   $2C       Start of CAN Tx Buffer
0042 0036               CANRX   EQU   $36       Start of CAN Rx Buffer
0043 002a               IDENT   EQU   $2A       CAN Transmit Buffer Identifier Reg.
0044 0034               IDENTR  EQU   $34       CAN Receive Buffer Identifier Reg.
0045                    *
0046                    *       CAN DRIVER S/W REGISTERS and EQUATES
0047                    *
0048 0050                   ORG   $50
0049 0050               BUFNO1  RMB   1         General Purpose RAM Reg. used by Receive Routine
0050 0051               BUFSIZ  RMB   1         General Purpose RAM Reg. Used by Transmit Routine
0051 0052               QCOUNT  RMB   1         Queue Buffer Counter
0052 0053               QPT1    RMB   1         First Item Into Queue Pointer
0053 0054               QPT0    RMB   1         Next Item into Queue Pointer
0054 0055               QSTA    EQU   *         Start of Transmit Queue area
0055 0055               QBUF    RMB   5         Reserved area for Transmit Queue Buffer
0056 005a               QEND    EQU   *         End of Transmit Queue Area
0057 005a               CURTX   RMB   1         Temporary storage aea for TRANSMIT routine
0058 005b               IDOFF   EQU   *         Storage byte for rec'd identifier position
0059 005b               TXPNT   RMB   1
0060 005c               USINTE  RMB   1         User Interrupt Enable Register
0061 005d               GENSTAT RMB   1         MUX Communications General Status Register
0062                    *
0063 005e               MCSOFF  RMB   10        Message Control/Status Registers
0064                    *
0065                    *             M(n)CS1 and M(n)CS2 (n = 1,5)
0066                    *
0067 0068               MB      RMB   40         Reserved Area for Message Buffers
0068                    *
0069                    *
0070                    *       Bit Equates
0071                    *
0072                    *M(n)CS1
0073 0007               TPND    EQU   7           Set to indicate Tx Buffer locked
0074 0006               REMR    EQU   6           Set to indicate Remote frame rec'd
0075 0005               RBFU    EQU   5           Set to indicate Rx Buffer contains new data
0076 0004               RBOF    EQU   4           Set to indicate Rx Buffer overflow
0077 0003               AUTOTX  EQU   3           Set to indicate Auto Tx facility enabled
0078 0002               RREQ    EQU   2           Set to indicate Remote Request in Queue
```

```
0079 0001                      ATPND     EQU    1            Set to indicate auto resp. to remote frame
0080                           *                             is pending
0081                           *
0082                           *M(n)CS2
0083                           *
0084 0f00                           ORG    $0F00
0085                           ********************************************************
0086                           ********************************************************
0087                           *          Message Definition Table (Example)
0088                           ********************************************************
0089                           *          Message #0 - Identifier = $70F
0090                           *          Receive Area size = Transmit Area Size = 1 byte
0091 0f00 00 00 e1 e0 68 69    MBDF       FCB    $00,$00,$E1,$E0,MB,MB+1,$01,$01
     01 01
0092                           *
0093                           *          Message #1 - Identifier = $71F
0094                           *          Receive Area size = Transmit Area Size = 1 byte
0095 0f08 00 00 e3 e0 6a 6b         FCB    $00,$00,$E3,$E0,MB+2,MB+3,$01,$01
     01 01
0096                           *
0097                           *          Message #2 - Identifier = $72F
0098                           *          Receive Area size = Transmit Area Size = 1 byte
0099 0f10 00 00 e5 e0 6c 6d         FCB    $00,$00,$E5,$E0,MB+4,MB+5,$01,$01
     01 01
0100                           *
0101                           *          Message #3 - Identifier = $73F
0102                           *          Receive Area size = Transmit Area Size = 1 byte
0103 0f18 00 00 e7 e0 6e 6f         FCB    $00,$00,$E7,$E0,MB+6,MB+7,$01,$01
     01 01
0104                           *
0105                           *          Message #4(A) - Identifier = $74F
0106                           *          Receive Area size = Transmit Area Size = 8 bytes
0107 0f20 00 00 e9 e0 70 78         FCB    $00,$00,$E9,$E0,MB+8,MB+$10,$08,$08
     08 08
0108                           *
0109                           *          Message #4(B) - Identifier = $74F
0110                           *          Receive Area size = Transmit Area Size = 8 bytes
0111 0f28 00 00 e9 e0 80 88         FCB    $00,$00,$E9,$E0,MB+$18,MB+$20,$08,$08
     08 08
0112                           *
0113                           ********************************************************
0114                           *          MUX Communications Set Up Table
0115                           ********************************************************
0116
0117                           *
0118 0f30 05                   MCST       FCB    $05          Number of Messages
0119 0f31 cc 15 00                  FCB    $CC,$15,$00   Jump to User Receive routine at $1500
0120 0f34 cc 15 10                  FCB    $CC,$15,$10   Jump to User Transmit Routine at $1510
0121 0f37 cc 15 20                  FCB    $CC,$15,$20   Jump to User Error Routine at $1520
0122 0f3a cc 15 30                  FCB    $CC,$15,$30   Jump to User Overflow Routine at $1530
0123 0f3d cc 15 40                  FCB    $CC,$15,$40   Jump to User Wake Up Routine at $1540
0124                           *
0125                           *
0126                           ********************************************************
0127                           *          Subroutine TRANSMIT
0128                           *
0129                           *          This routine takes the next Message Buffer number in the
0130                           *          transmit queue and, if the CAN Tx buffer is free, loads
0131                           *          Message Buffer Transmit Area contents into the CAN Tx Buffer
0132                           *          along with the corresponding identifier. The TR bit in the
0133                           *          in the CAN Command reg.is set to start frame transmission.
0134                           *
0135                           ********************************************************
0136                           *
0137 0f40                      TRANSMIT   EQU    *
0138                           *
0139 0f40 b6 52                     LDA    QCOUNT
0140 0f42 27 70                     BEQ    TRANS5        Return if Queue empty
0141 0f44 be 53                     LDX    QPT1          Get offset for next Message in Q
0142 0f46 f6                        LDA    ,X            Load next pending Message Number
0143 0f47 5c                        INCX
0144 0f48 a3 5a                     CPX    #QEND         Check for QPT1 at end of Q Buffer
0145 0f4a 26 02                     BNE    TRANS2
0146 0f4c ae 55                     LDX    #QSTA         Set QPT1 to start of Q Buffer
0147 0f4e bf 53             TRANS2  STX    QPT1          Update QPT1 to next pending Message Number
0148 0f50 3a 52                     DEC    QCOUNT        Decrement Q Counter Reg.
0149                           *
0150 0f52 05 22 5f          TRANS1  BRCLR  2,CANSTAT,TRANS5 Check that TBA bit is set
0151                           *
0152 0f55 b7 5a                     STA    CURTX         Store Message Number
0153 0f57 97                        TAX                  Transfer Message Number to X reg.
```

```
0154 0f58 58                    LSLX                Multiply by 8 to get offset for MB Definition
0155 0f59 58                    LSLX                Table entry
0156 0f5a 58                    LSLX
0157                        *
0158 0f5b d6 0f 02              LDA     MBDF+2,X    Get Hi order bits of identifier
0159 0f5e b7 2a                 STA     IDENT
0160 0f60 d6 0f 06              LDA     MBDF+6,X    Get Tx Buffer size
0161 0f63 b7 51                 STA     BUFSIZ
0162 0f65 d6 0f 03              LDA     MBDF+3,X    Get low order bits of identifier
0163 0f68 a4 e0                 AND     #$E0        Clear all other bits including RTR bit
0164 0f6a bb 51                 ADD     BUFSIZ
0165 0f6c b7 2b                 STA     IDENT+1     Store Low Ident, RTR and Data size
0166 0f6e d6 0f 05              LDA     MBDF+5,X    Get Message (n) Transmit Buffer Pointer
0167 0f71 b7 5b                 STA     TXPNT
0168                        *
0169 0f73 54                    LSRX                Divide X reg. by 4 to get offset for M(n)CSx
0170 0f74 54                    LSRX
0171 0f75 e6 5e                 LDA     MCSOFF,X    Get M(n)CS1
0172 0f77 a4 04                 AND     #$04        Check for RREQ Set
0173 0f79 27 0a                 BEQ     TRANS3      Load data into CAN Tx Regs. if REMR
0174 0f7b 18 2b                 BSET    4,IDENT+1   Set RTR bit
0175 0f7d e6 5e                 LDA     MCSOFF,X
0176 0f7f a4 fb                 AND     #$FB        Clear RREQ in M(n)CS1
0177 0f81 e7 5e                 STA     MCSOFF,X
0178 0f83 20 2b                 BRA     TRANS4
0179                        *
0180
0181 0f85 e6 5e             TRANS3  LDA     MCSOFF,X    Get M(n)CS1
0182 0f87 a4 3d                 AND     #$3D        Clear TPND, REMR and ATPEND in M(n)CS1
0183 0f89 e7 5e                 STA     MCSOFF,X
0184                        *
0185 0f8b b6 51                 LDA     BUFSIZ      Get Tx Buffer size
0186 0f8d 27 21                 BEQ     TRANS4      Check if Buf. size is zero
0187 0f8f be 5b                 LDX     TXPNT       Load Tx Buffer pointer into X reg.
0188 0f91 f6                    LDA     ,X          Get 1st byte of Tx Buffer
0189 0f92 b7 2c                 STA     CANTX
0190 0f94 e6 01                 LDA     1,X
0191 0f96 b7 2d                 STA     CANTX+1
0192 0f98 e6 02                 LDA     2,X
0193 0f9a b7 2e                 STA     CANTX+2
0194 0f9c e6 03                 LDA     3,X
0195 0f9e b7 2f                 STA     CANTX+3
0196 0fa0 e6 04                 LDA     4,X
0197 0fa2 b7 30                 STA     CANTX+4
0198 0fa4 e6 05                 LDA     5,X
0199 0fa6 b7 31                 STA     CANTX+5
0200 0fa8 e6 06                 LDA     6,X
0201 0faa b7 32                 STA     CANTX+6
0202 0fac e6 07                 LDA     7,X
0203 0fae b7 33                 STA     CANTX+7
0204                        *
0205 0fb0 a6 01             TRANS4  LDA     #$01
0206 0fb2 b7 21                 STA     CANCOM      Set Transmission Request
0207 0fb4 81                TRANS5  RTS
0208                        *
0209                        *
0210                        ************************************************************
0211                        *           Subroutine RECEIVE
0212                        *
0213                        *           This routine is called when a CAN Receive Interrupt occurs.
0214                        *           It compares the identifier of the received message with the
0215                        *           the identifiers in the Message Definition Table and, if a
0216                        *           match is found, stores the received message in the appropriate
0217                        *           Message Buffer Receive Area. If a remote frame is received and
0218                        *           the AUTOTX bit in M(n)CS1 is set then the appropriate Message
0219                        *           Buffer number is entered into the Transmit queue to so that
0220                        *           a data frame is sent in response
0221                        *
0222                        ************************************************************
0223                        *
0224 0fb5 cc 10 86          RECV7   JMP     RECV2
0225 0fb8                   RECEIVE EQU     *
0226                        *
0227 0fb8 5f                    CLRX
0228 0fb9 bf 50                 STX     BUFNO1      Initialise Rx BUFNO Reg.
0229                        *
0230 0fbb b6 50             RECV1   LDA     BUFNO1
0231 0fbd c1 0f 30              CMP     MCST        Check for last Identifier comparison
0232 0fc0 24 f3                 BHS     RECV7
0233 0fc2 3c 50             RECV8   INC     BUFNO1
0234                        *
```

```
0235 0fc4 9f                TXA                 Increment X reg. by 8
0236 0fc5 ab 08             ADD    #$08
0237 0fc7 97                TAX
0238                    *
0239 0fc8 d6 0e fa          LDA    MBDF-6,X     Get High order byte of 1st Mes. Buf. Ident.
0240 0fcb b1 34             CMP    IDENTR       Compare with High order byte of Rx'ed Ident.
0241 0fcd 26 ec             BNE    RECV1
0242 0fcf b6 35             LDA    IDENTR+1     Get Low order byte of Rx'ed Ident.
0243 0fd1 a4 e0             AND    #$E0         Remove RTR and data size code bits
0244 0fd3 d1 0e fb          CMP    MBDF-5,X
0245 0fd6 26 e3             BNE    RECV1
0246 0fd8 b6 35             LDA    IDENTR+1
0247 0fda a4 10             AND    #$10         Check RTR bit
0248 0fdc 27 37             BEQ    RECV5        If RTR clear then transfer CAN Rx Buffer
0249                    *                              contents to Message (n) Receive Buffer.
0250                    *
0251 0fde be 50             LDX    BUFNO1       Get Message number n
0252 0fe0 5a                DECX
0253 0fe1 59                ROLX                Multiply by 2 for M(n)CS offset
0254 0fe2 e6 5e             LDA    MCSOFF,X     Get M(n)CS1
0255 0fe4 a4 08             AND    #$08         Check for AUTOTX bit set
0256 0fe6 27 25             BEQ    RECV6
0257 0fe8 b6 52             LDA    QCOUNT       Check for Q overflow
0258 0fea a1 05             CMP    #QEND-#QSTA
0259 0fec 24 c7             BHS    RECV7        No action if Queue is full
0260 0fee e6 5e             LDA    MCSOFF,X     Get M(n)CS1
0261 0ff0 aa c8             ORA    #$C8         Set TPND, REMR and ATPND in M(n)CS1
0262 0ff2 e7 5e             STA    MCSOFF,X
0263                    *
0264 0ff4 b6 50             LDA    BUFNO1
0265 0ff6 4a                DECA                Generate Message number
0266                    *
0267                    *          Enter message number into Transmit Queue
0268                    *
0269 0ff7 be 54             LDX    QPT0         Get position of next queue location
0270 0ff9 f7                STA    ,X           Load message number into queue
0271 0ffa 5c                INCX
0272 0ffb a3 5a             CPX    #QEND        Check for QPT1 at end of Q Buffer
0273 0ffd 26 02             BNE    RECV9
0274 0fff ae 55             LDX    #QSTA        Set QPT0 to start of Q Buffer
0275 1001 bf 54         RECV9   STX    QPT0          Update QPT0 to next free Q location
0276 1003 3c 52             INC    QCOUNT
0277                    *
0278                    *     TAX
0279                    *     LSLX                 Generate offset for M(n)CS1
0280                    *     LDA     MCSOFF,X     Get M(n)CS1
0281                    *     ORA     #$80         Set TPND
0282                    *     STA     MCSOFF,X
0283                    *
0284 1005 05 22 7e          BRCLR  2,CANSTAT,RECV2   Check for TBA bit clear
0285                    *                                TBA clear indicates tx in process
0286                    *                                therefore TRANSMIT will be called by
0287                    *                                CAN Interrupt service routine.
0288 1008 cd 0f 40          JSR    TRANSMIT     Initiate CAN Transmit Process
0289                    *
0290 100b 20 79             BRA    RECV2
0291                    *
0292 100d e6 5e         RECV6   LDA    MCSOFF,X      Get M(n)CS1
0293 100f aa 40             ORA    #$40         Set REMR bit
0294 1011 e7 5e             STA    MCSOFF,X
0295 1013 20 71             BRA    RECV2        Set RRB and return
0296                    *
0297 1015 bf 5b         RECV5   STX    IDOFF
0298                    *
0299 1017 be 50             LDX    BUFNO1       Get Message number n
0300 1019 5a                DECX
0301 101a 59                ROLX                Multiply by 2 for M(n)CS offset
0302 101b e6 5e             LDA    MCSOFF,X     Get M(n)CS1
0303 101d a4 20             AND    #$20         Check for RBFU set
0304 101f 27 0a             BEQ    RECV4
0305 1021 e6 5e             LDA    MCSOFF,X
0306 1023 aa 10             ORA    #$10         Set RBOF to indicate overflow
0307 1025 e7 5e             STA    MCSOFF,X
0308 1027 be 5b             LDX    IDOFF
0309 1029 20 97             BRA    RECV8        Continue to search for vacant Receive Buffer
0310
0311 102b e6 5e         RECV4   LDA    MCSOFF,X      Get M(n)CS1
0312 102d aa 20             ORA    #$20         Set RBFU in M(n)CS1
0313 102f e7 5e             STA    MCSOFF,X
0314 1031 e6 5f             LDA    MCSOFF+1,X   Get M(n)CS2
0315 1033 a4 f0             AND    #$F0         Set prev. data size to zero
```

```
0316 1035 b7 51                 STA     BUFSIZ          Temporary store
0317 1037 b6 35                 LDA     IDENTR+1        Get no. of rec'd data bytes
0318 1039 a4 0f                 AND     #$0F            Mask out Identifier and RTR
0319 103b bb 51                 ADD     BUFSIZ
0320 103d e7 5f                 STA     MCSOFF+1,X
0321 103f be 5b                 LDX     IDOFF
0322 1041 d6 0e ff              LDA     MBDF-1,X        Get Message Receive Buffer size
0323 1044 27 40                 BEQ     RECV2           If Buf. size 0 then return
0324 1046 b7 51                 STA     BUFSIZ
0325 1048 de 0e fc              LDX     MBDF-4,X        Get Message Receive Buffer Pointer
0326 104b b6 36                 LDA     CANRX           Get 1st data byte
0327 104d f7                    STA     ,X
0328 104e 3a 51                 DEC     BUFSIZ          Check for end of Message Receive Buffer
0329 1050 27 34                 BEQ     RECV2
0330 1052 b6 37                 LDA     CANRX+1         Get 2nd byte
0331 1054 e7 01                 STA     1,X
0332 1056 3a 51                 DEC     BUFSIZ          Check for end of Message Receive Buffer
0333 1058 27 2c                 BEQ     RECV2
0334 105a b6 38                 LDA     CANRX+2         Get 3rd byte
0335 105c e7 02                 STA     2,X
0336 105e 3a 51                 DEC     BUFSIZ          Check for end of Message Receive Buffer
0337 1060 27 24                 BEQ     RECV2
0338 1062 b6 39                 LDA     CANRX+3         Get 4th byte
0339 1064 e7 03                 STA     3,X
0340 1066 3a 51                 DEC     BUFSIZ          Check for end of Message Receive Buffer
0341 1068 27 1c                 BEQ     RECV2
0342 106a b6 3a                 LDA     CANRX+4         Get 5th byte
0343 106c e7 04                 STA     4,X
0344 106e 3a 51                 DEC     BUFSIZ          Check for end of Message Receive Buffer
0345 1070 27 14                 BEQ     RECV2
0346 1072 b6 3b                 LDA     CANRX+5         Get 6th byte
0347 1074 e7 05                 STA     5,X
0348 1076 3a 51                 DEC     BUFSIZ          Check for end of Message Receive Buffer
0349 1078 27 0c                 BEQ     RECV2
0350 107a b6 3c                 LDA     CANRX+6         Get 7th byte
0351 107c e7 06                 STA     6,X
0352 107e 3a 51                 DEC     BUFSIZ          Check for end of Message Receive Buffer
0353 1080 27 04                 BEQ     RECV2
0354 1082 b6 3d                 LDA     CANRX+7         Get 8th byte
0355 1084 e7 07                 STA     7,X
0356                    *
0357 1086 a6 04         RECV2   LDA     #$04
0358 1088 b7 21                 STA     CANCOM          Set RRB (Release Receive Buffer) in CANCOM
0359 108a 81                    RTS
0360                    *
0361                    ********************************************************
0362                    *       Subroutine SENDDAT
0363                    *       This routine is called by the user to initiate a data frame
0364                    *       transmission on the CAN bus. It should be called with the
0365                    *       required Message Buffer number in the accumulator. It enters
0366                    *       the number into the Transmit Queue. The TS bit in the
0367                    *       CAN status register is checked and if set the routine returns.
0368                    *       If TS is clear then TRANSMIT is called to initiate the CAN
0369                    *       transmit process.
0370                    ********************************************************
0371                    *
0372 108b 9b            SENDDAT SEI                     Stop interrupts
0373 108c be 54                 LDX     QPT0            Get position of next queue location
0374 108e f7                    STA     ,X              Load message number into queue
0375 108f 5c                    INCX
0376 1090 a3 5a                 CPX     #QEND           Check for QPT1 at end of Q Buffer
0377 1092 26 02                 BNE     SDAT1
0378 1094 ae 55                 LDX     #QSTA           Set QPT0 to start of Q Buffer
0379 1096 bf 54         SDAT1   STX     QPT0            Update QPT0 to next free Q location
0380 1098 3c 52                 INC     QCOUNT
0381                    *
0382 109a 97                    TAX
0383 109b 58                    LSLX                    Generate offset for M(n)CS1
0384 109c e6 5e                 LDA     MCSOFF,X        Get M(n)CS1
0385 109e aa 80                 ORA     #$80            Set TPND
0386 10a0 e7 5e                 STA     MCSOFF,X
0387                    *
0388 10a2 05 22 03              BRCLR   2,CANSTAT,SDAT2 Check for TBA bit clear
0389                    *                               TBA clear indicates tx in process
0390                    *                               therefore TRANSMIT will be called by
0391                    *                               CAN Interrupt service routine.
0392 10a5 cd 0f 40              JSR     TRANSMIT        Initiate CAN Transmit Process
0393 10a8 9a            SDAT2   CLI
0394 10a9 81                    RTS
0395                    *
0396                    *
```

```
0397                        ********************************************************
0398                        *           Subroutine SENDREM
0399                        *           This routine is called by the user to initiate a remote frame
0400                        *           transmission on the CAN bus. It should be called with the
0401                        *           required Message Buffer number in the accumulator. It enters
0402                        *           the number into the Transmit Queue. The TS bit in the
0403                        *           CAN status register is checked and if set the routine returns.
0404                        *           If TS is clear then TRANSMIT is called to initiate the CAN
0405                        *           transmit process.
0406                        ********************************************************
0407                        ********************************************************
0408                        *
0409 10aa 9b                SENDREM     SEI                 Stop interrupts
0410 10ab be 54                 LDX     QPT0            Get position of next queue location
0411 10ad f7                    STA     ,X              Load message number into queue
0412 10ae 5c                    INCX
0413 10af a3 5a                 CPX     #QEND           Check for QPT1 at end of Q Buffer
0414 10b1 26 02                 BNE     SREM1
0415 10b3 ae 55                 LDX     #QSTA           Set QPT0 to start of Q Buffer
0416 10b5 bf 54             SREM1       STX     QPT0        Update QPT0 to next free Q location
0417 10b7 3c 52                 INC     QCOUNT
0418                        *
0419 10b9 97                    TAX
0420 10ba 58                    LSLX                    Generate offset for M(n)CS1
0421 10bb e6 5e                 LDA     MCSOFF,X        Get M(n)CS1
0422 10bd aa 04                 ORA     #$04            Set RREQ
0423 10bf e7 5e                 STA     MCSOFF,X
0424                        *
0425 10c1 05 22 03              BRCLR   2,CANSTAT,SREM2     Check for TBA bit clear
0426                        *                                       TBA clear indicates tx in process
0427                        *                                       therefore TRANSMIT will be called by
0428                        *                                       CAN Interrupt service routine.
0429 10c4 cd 0f 40              JSR     TRANSMIT        Initiate CAN Transmit Process
0430 10c7 9a                SREM2       CLI
0431 10c8 81                    RTS
0432                        *
0433                        ********************************************************
0434                        *           Subroutine INIT - Initialises the M(n)CS1/2 and USINTE
0435                        *           Registers.
0436                        ********************************************************
0437                        *
0438 10c9 c6 0f 30          INIT        LDA     MCST            Fetch no. of Message Buffers
0439 10cc 49                    ROLA                    Multiply by 2
0440 10cd b7 50                 STA     BUFNO1
0441 10cf 5f                    CLRX                    Clear index for first M(n)CS Reg.
0442 10d0 4f                    CLRA
0443 10d1 e7 5e             INIT1       STA     MCSOFF,X        Clear M(n)CS(1/2)
0444 10d3 5c                    INCX
0445 10d4 b3 50         R^¿         CPX     BUFNO1          Check for last M(n)CS Reg.
0446 10d6 25 f9                 BLO     INIT1
0447 10d8 b7 5c                 STA     USINTE          Clear USINTE to disable user interrupts
0448 10da b7 5d                 STA     GENSTAT         Indicates normal 2 wire differential mode
0449 10dc a6 55                 LDA     #QSTA
0450 10de b7 54                 STA     QPT0            Initialise Transmit Queue
0451 10e0 b7 53                 STA     QPT1
0452 10e2 3f 52                 CLR     QCOUNT
0453 10e4 81                    RTS                                                 .
0454                        *
0455                        ********************************************************
0456                        *           Subroutine CANINIT - Initialises the CAN Module
0457                        *           Registers as follows:
0458                        *
0459                        *           tSCL = 1 uS @ 2 MHz internal operating frequency
0460                        *           tSJW = 4 x tSCL
0461                        *           tSEG1 = 5 x tSCL, tSEG2 = 4 x tSCL, tBIT = 10 x t SCL = 10 uS
0462                        *
0463                        *           o/p drivers set up for push/pull configuration.
0464                        *
0465                        *           Data o/p - opposite polarity on Tx0 and Tx1 for diffential
0466                        *           bus operation
0467                        *
0468                        *           Message Filtering set to accept identifier range $700 to $77F
0469                        *
0470                        *           Overrun, Error, Receive and Transmit Interrupts enabled
0471                        *
0472                        ********************************************************
0473                        *
0474 10e5 a6 61             CANINIT     LDA     #$61            Set Reset Request, disable interrupts
0475 10e7 b7 20                 STA     CANCTRL         select slow mode
0476 10e9 a6 e0                 LDA     #$E0            Identifier ID10 to ID3 values
0477 10eb b7 24                 STA     CANACC          Store in Acceptance Code Reg.
```

```
0478 10ed a6 0f                 LDA    #$0F
0479 10ef b7 25                 STA    CANACM        Store in CAN Acceptance Mask Reg.
0480                    *                                  Allows identifier range
0481 10f1 a6 01                 LDA    #$01
0482 10f3 b7 26                 STA    CANBT0        Bus Timing Reg. 0, tSCL = 1 uS @ 4 MHz
0483                    *                                  tSJW = 1 x tSCL
0484 10f5 a6 12                 LDA    #$12          Bus Timing Reg. 1
0485 10f7 b7 27                 STA    CANBT1        tSEG1 = 3tSCL, tSEG2 = 2tSCL, tBIT = 6 tSCL
0486                    *
0487 10f9 a6 fa                 LDA    #$FA          Set o/p control reg.
0488 10fb b7 28                 STA    CANOPC        Normal mode 1
0489                    *
0490 10fd a6 7e                 LDA    #$7E          Overrun, Error, Transmit and Receive
0491 10ff b7 20                 STA    CANCTRL       Interrupts enabled, Reset Req. cleared
0492                    *
0493 1101 9a                    CLI
0494 1102 81                    RTS
0495                    *
0496                    *
0497                    ********************************************************
0498                    *          CAN Module Interrupt Routine
0499                    *
0500                    *          - Checks all CAN Interrupt flags to determine source
0501                    *          of interrupt and calls appropriate service routine. RAM
0502                    *          reg. USINTE is checked and if the corresponding user interrupt
0503                    *          enable is set, the program executes a JSR instruction to the
0504                    *          appropriate instruction in the user JUMP table.
0505                    *
0506                    ********************************************************
0507                    *
0508 1103 b6 23         CANIRQ     LDA    CANINT        Fetch CAN Interrupt register
0509 1105 46                    RORA                 Check for Receive Interrupt flag set
0510 1106 25 0c                 BCS    RXINT
0511 1108 46                    RORA                 Check for Transmit Interrupt flag set
0512 1109 25 16                 BCS    TXINT
0513 110b 46                    RORA                 Check for Error Interrupt flag set
0514 110c 25 22                 BCS    ERRINT
0515 110e 46                    RORA                 Check for Overrun Interrupt flag set
0516 110f 25 29                 BCS    OVINT
0517 1111 20 31                 BRA    WUINT         Wake Up Interrupt assumed
0518 1113 80            INTEND     RTI
0519                    *
0520 1114 cd 0f b8      RXINT      JSR    RECEIVE
0521 1117 b6 5c                 LDA    USINTE
0522 1119 a4 01                 AND    #$01          Check for User Receive Interrupt Enable set
0523 111b 27 f6                 BEQ    INTEND
0524 111d cd 0f 31              JSR    MCST+1
0525 1120 80                    RTI
0526                    *
0527 1121 cd 0f 40      TXINT      JSR    TRANSMIT
0528 1124 b6 5c                 LDA    USINTE
0529 1126 a4 02                 AND    #$02          Check for User Transmit Interrupt Enable set
0530 1128 27 e9                 BEQ    INTEND
0531 112a cd 0f 34              JSR    MCST+4
0532 112d 11 00                 BCLR   0,PORTA       CPU Overhead analysis
0533 112f 80                    RTI
0534                    *
0535 1130 b6 5c         ERRINT     LDA    USINTE
0536 1132 a4 04                 AND    #$04          Check for User Error Interrupt Enable set
0537 1134 27 dd                 BEQ    INTEND
0538 1136 cd 0f 37              JSR    MCST+7
0539 1139 80                    RTI
0540                    *
0541 113a b6 5c         OVINT      LDA    USINTE
0542 113c a4 08                 AND    #$08          Check for User Overrun Interrupt Enable set
0543 113e 27 d3                 BEQ    INTEND
0544 1140 cd 0f 3a              JSR    MCST+$0A
0545 1143 80                    RTI
0546                    *
0547 1144 b6 5c         WUINT      LDA    USINTE
0548 1146 a4 10                 AND    #$10          Check for Wake Up Interrupt Enable set
0549 1148 27 c9                 BEQ    INTEND
0550 114a cd 0f 3d              JSR    MCST+$0D
0551 114d 80                    RTI
0552                    *
0553 1ffa                       ORG    $1FFA
0554 1ffa 11 03                 FDB    CANIRQ
0555                    *
0556                            END
0557                    *  END
```

This page intentionally left blank.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com