

**MOTOROLA**  
**SEMICONDUCTOR**  
**TECHNICAL DATA**

**AN1224**

**Example Software Routines for the Message Data  
Link Controller Module on the MC68HC705V8**

By Chuck Powers  
Multiplex Applications

**INTRODUCTION**

The message data link controller (MDLC) module is a serial multiplex communication module designed to communicate on an automotive serial multiplex bus in a format compatible with the Society of Automotive Engineers (SAE) *Recommended Practice J1850-Class B Data Communication Network Interface*. The MDLC communicates using variable pulse width modulation (VPW) bit encoding at a transmission rate of 10.4 kilobits per second (kbps), which is also compatible with General Motors' Class 2 multiplex communication protocol. The MDLC handles all of the communication duties, including complete message buffering, bus access, arbitration and error detection. Interrupts of the CPU occur only when a complete message has been received error-free from the multiplex bus or following a successful transmission onto the multiplex bus.

This application note describes a basic set of MDLC driver routines for communicating on a Class B serial multiplex bus using the MC68HC705V8, a multi-purpose microcontroller unit (MCU) based upon Motorola's industry-standard MC68HC05 CPU. Methods will be outlined for initializing the MDLC for proper communication, and for transferring data in and out of the MDLC's transmit and receive buffers. Although these driver routines have been written for use with the MC68HC705V8, the methods described are readily applicable to any Motorola MC68HC05 or MC68HC08 microcontroller which contains the MDLC module.

**J1850 OVERVIEW**

The increase in the complexity and number of electronic components in automobiles has caused a massive increase in the wiring harness requirements for each vehicle. This has resulted in a demand for ways to reduce the amount of wiring required for communication between various components.

The SAE Recommended Practice J1850 was developed by the Society of Automotive Engineers as a recommended practice for medium speed (Class B) serial multiplex communication for use in the automotive environment. The use of serial multiplex communication has given automotive system designers one method of actually increasing the amount and type of data which can be shared between various components in the automobile while reducing the amount of wiring necessary to connect these components. This is done by connecting each component, or node, to a serial bus consisting of either a single wire or a twisted pair of wires. Each node collects whatever data is useful to itself or other nodes (wheel speed, engine RPM, oil pressure, etc.), and then transmits this data onto the multiplex bus, where any other node which needs this data can receive it. This results in a significant improvement in data sharing while at the same time eliminating the need for redundant sensing systems.

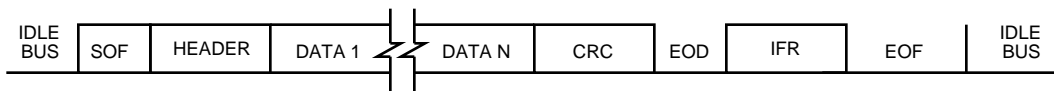
The J1850 protocol encompasses the lowest two layers of the International Standards Organization (ISO) open system interconnect (OSI) model, the data link layer and the physical layer. It is a multi-master system, utilizing the concept of carrier sense multiple access with collision resolution (CSMA/CR), whereby any node can transmit if it has determined the bus to be free. Non-destructive arbitration is



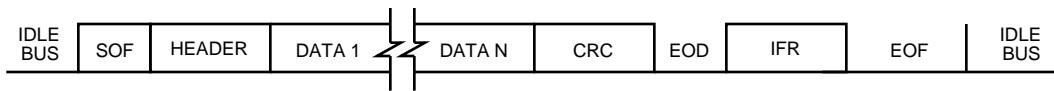
performed on a bit-by-bit basis whenever multiple nodes begin to transmit simultaneously. J1850 allows for the use of a single or dual wire bus, two data rates (10.4 kbps or 41.7 kbps), and two bit encoding techniques (pulse width modulation (PWM) or variable pulse width modulation (VPW)).

## Features

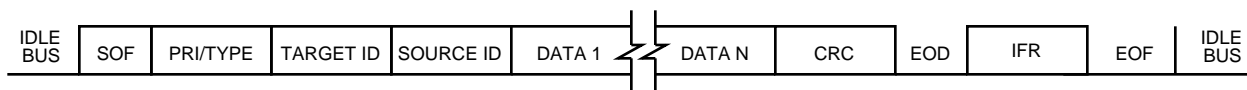
A J1850 message, or frame, consists of a start of frame (SOF) delimiter, a one- or three-byte header, zero to eight data bytes, a cyclical redundancy check (CRC) byte, an end of data (EOD) delimiter, and an optional in-frame response byte, followed by an end of frame (EOF) delimiter. Frames using a single-byte header are transmitted at 10.4 kbps, using VPW modulation, and contain a CRC byte for error detection (see Figure 1a). Frames using a one-byte consolidated header or a three-byte consolidated header can be transmitted at either 41.7 kbps or 10.4 kbps, using either PWM or VPW modulation techniques, and also contain a CRC byte for error detection (see Figures 1b and 1c).



**Figure 1a. Single-Byte Header Frame Format**



**Figure 1b. One-Byte Consolidated Header Frame Format**



**Figure 1c. Three-byte Consolidated Header Frame Format**

Each frame can contain up to 12 bytes (VPW) or 101 bit times (PWM), with each byte transmitted MSB first. The optional in-frame response can contain either a single byte or multiple bytes, with or without a CRC byte. Table 1 summarizes the allowable features of the J1850 protocol. The requirements of each individual network determine which features are used.

**Table 1. J1850 Protocol Options**

<b>Feature</b>	<b>1 &amp; 3-Byte Headers</b>	<b>1 &amp; 3-Byte Headers</b>	<b>Single-Byte Header</b>
Bit Encoding	PWM	VPW	VPW
Bus Medium	Dual Wire	Single Wire	Single Wire
Data Rate	41.7 kbps	10.4 kbps	10.4 kbps
Data Integrity	CRC	CRC	CRC

## **Frame Headers and Addressing**

As outlined above, a J1850 frame can contain one of three types of headers, depending upon a particular system's requirements. The single-byte header incorporates the frame priority/type and target address into a single byte. A one-byte consolidated header also consolidates the frame priority/type and target address into a single byte, with bit 4 = 1 to indicate that it is a one-byte consolidated header. The three-byte header places the frame priority/type into the first byte, the target address of the intended receiver(s) into the second byte, and the source address of the frame originator into the third byte. In the priority/type byte of the three-byte header, bit 4 = 0 to indicate it is a three-byte header.

Frames transmitted on a J1850 network can be either physically or functionally addressed. Since every node on a J1850 network must be assigned a unique physical address, a frame can be addressed directly to any particular node by making that node's physical address the target address of the frame. This is useful in applications such as diagnostic requests, where a specific node's identification may be important. Functional addressing is used when the data being transmitted can be identified by its particular function, rather than its intended receiver(s). With this form of addressing, a frame containing data is transmitted with the function of that data encoded in the header of the frame. All nodes which require the data of that function can then receive it at the same time. This is of particular importance to networks where the physical address of the intended receivers is not known, or could change, while their function remains the same. An example of data that would be functionally addressed is wheel speed, which could be of interest to multiple receivers, each with a different physical address. Functionally addressing the wheel speed data would allow it to be transmitted to all intended receivers in a single frame, instead of transmitting the data in a separate frame for each receiver.

## **Error Detection**

Every frame transmitted onto a J1850 network contains a single CRC byte for error detection. This byte is usually produced by shifting the header and data bytes through a preset series of feedback shift registers. The resulting byte is then inserted in the frame following the data bytes. Any node which receives the frame then shifts the header, data and CRC bytes through an identical series of feedback shift registers, with an error-free frame always producing the result \$C4. Although the CRC byte is normally generated with hardware, the CRC calculation can be done in software. Any frame in which the error detection byte does not produce the proper result is discarded by all receivers, and any in-frame response, if required, is not transmitted.

## **Arbitration**

Arbitration on the multiplex bus is accomplished in a non-destructive manner, allowing the frame with the highest priority to be transmitted, while any transmitters which lose arbitration simply stop transmitting and wait for an idle bus to begin transmitting again. If multiple nodes begin to transmit at the same time,

arbitration begins with the first bit following the SOF delimiter, and continues with each bit thereafter. Whenever a transmitting node detects a dominant bit while transmitting a recessive bit, it loses arbitration, and immediately stops transmitting. This is known as "bitwise" arbitration. Since an active bit dominates a passive bit (a "0" dominates a "1"), the frame with the lowest value will have the highest priority, and will always win arbitration, i.e., a frame with priority 000 will win arbitration over a frame with priority 001. This method of arbitration will work regardless of how many bits of priority encoding are contained in the frame. Frequently, messaging strategies are utilized which ensure that all arbitration is resolved by the end of the frame header.

## In-Frame Response

The optional in-frame response (IFR) portion of a frame follows the EOD delimiter, and contains one of three types of information. The first type of IFR contains a single I.D. byte from a single receiver, indicating that at least one node received the frame. The I.D. byte is usually the physical address of the responding node. The second type of IFR contains multiple I.D. bytes from multiple receivers, indicating which receivers actually received the frame. In this case, the number of response bytes is limited only by the overall J1850 frame length constraints. The third type of IFR contains data bytes, with or without a CRC byte, from a single receiver. This type of IFR usually occurs during the IFR portion of a frame in which that data is requested. The CRC byte, if included in the IFR, is calculated and decoded in an identical manner to the frame CRC, except the transmitter and receiver roles are reversed. In VPW modulation, the in-frame response byte is preceded by a normalization bit, which is required to return the bus to the dominant state prior to transmitting the first bit of the IFR.

## Modulation

As previously mentioned, J1850 frames can be transmitted using two different modulation techniques, pulse width modulation (PWM) or variable pulse width modulation (VPW). The modulation technique used is dependent upon the desired transmission bit rate and the physical makeup of the bus. The PWM technique is primarily used with a bit rate of 41.7 kbps, and a bus consisting of a differential twisted pair. VPW modulation is used with a bit rate of 10.4 kbps and a single wire bus.

For more detailed information on the features of J1850, refer to *SAE Recommended Practice J1850-Class B Data Communication Network Interface*. Because this document is still subject to modification, the user should ensure that the most recent version is referenced.

## MC68HC705V8 MICROCONTROLLER

The MC68HC705V8 MCU is a multipurpose HCMOS MCU based on the industry standard MC68HC05 CPU. The available user memory on the MC68HC705V8 includes 12K of EPROM, 512 bytes of RAM (including stack) and 128 bytes of EEPROM. In addition to the MDLC module, other features include an internal power supply, a serial peripheral interface port, a 6-bit pulse width modulation port, a 16-bit timer with one input capture and one output compare, a multi-function core timer, a 16-channel 8-bit analog to digital (A/D) converter and an onboard watchdog system (refer to **Figure 2 MC68HC705V8 Block Diagram**). A few of the major features of the MC68HC705V8 are outlined below. For a detailed description of the features and operation of the MC68HC705V8, refer to the *MC68HC705V8 Product Specification*.

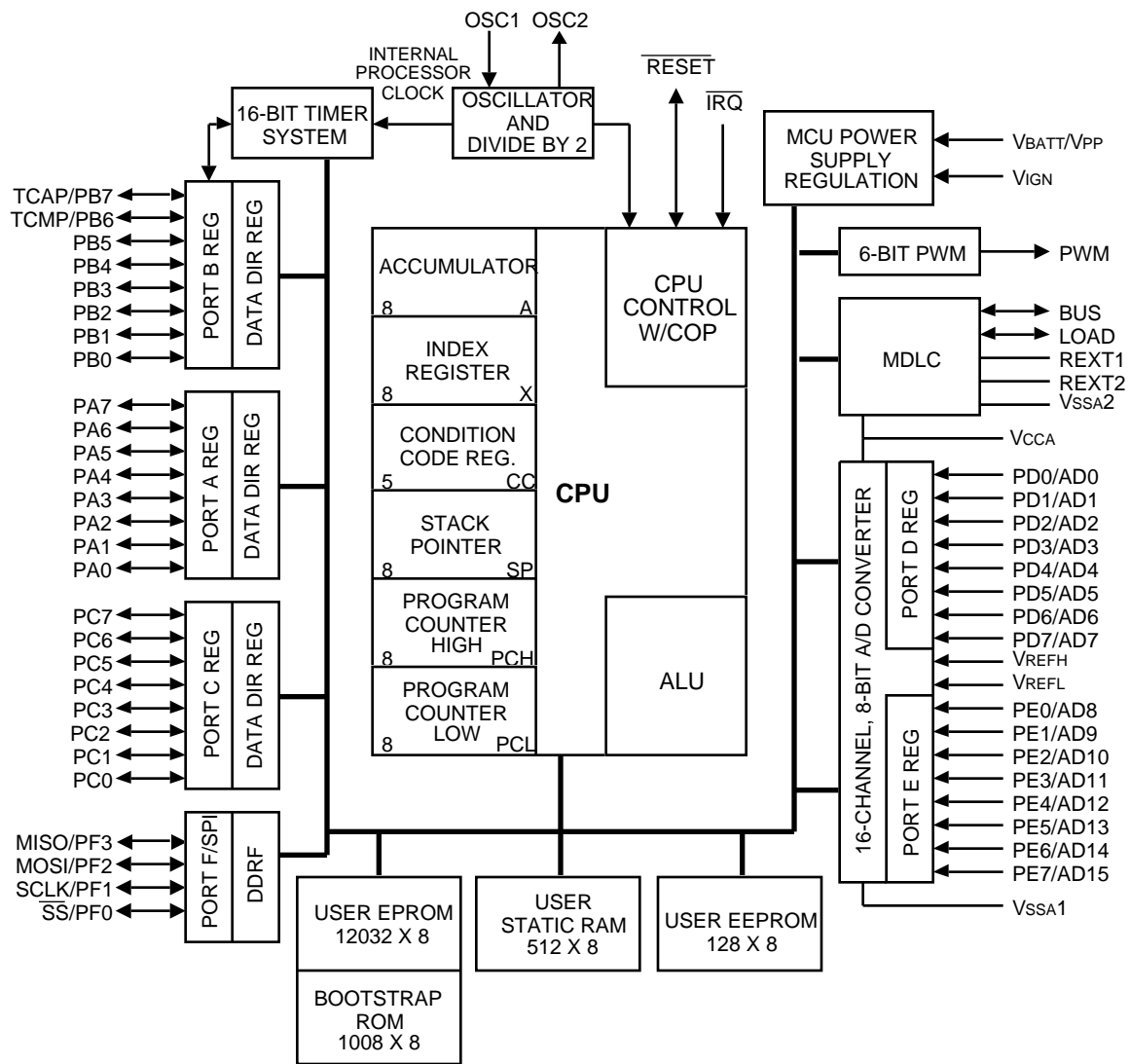


Figure 2. MC68HC705V8 Block Diagram

## Input/Output

The MC68HC705V8 contains 28 bi-directional I/O lines, divided into three 8-bit I/O ports, designated ports A, B and C, and one 4-bit I/O port, designated port F. Port A inputs and port C inputs can also be used as IRQ interrupt sources, while port B shares functions with the 16-bit timer and port F shares functions with the serial peripheral interface (SPI) port. Two 8-bit input-only ports are also available, designated as ports D and E. These two ports share functions with the 16-channel A/D converter. The direction of ports A, B, C, and F are controlled by four data direction registers, one for each I/O port. This allows each I/O line to be individually configured by the user as either an input or output. The ports and data direction registers are contained in the first page of the MCU memory map, and can be read or written to directly by the user.

## Internal Power Supply

The MC68HC705V8 contains a fully integrated power supply which can be used to produce the regulated +5-V supply for the device. This power supply includes a primary and secondary voltage regulator. The primary voltage regulator uses a regulated DC supply input (up to 16 V) to produce a regulated +5-V supply for all internal digital circuitry, except for the  $V_{IGN}$  logic and power supply control logic. If the REGDIS bit in the mask option register (MOR) is set, enabling the on-chip voltage regulation system, the primary regulator will be enabled once a rising edge is detected on the  $V_{IGN}$  pin, or on the BUS pin of the MDLC module if the MDLCPU bit in the MOR is set.

**Following a reset, if the internal power supply is enabled, the user should set bit 0 of the miscellaneous (MISC) register. This will ensure that the primary regulator is not inadvertently disabled by a subsequent read-modify-write command to the MISC register.** If at any time the user does wish to disable the primary regulator, this can be done by clearing bit 0 of the MISC register. The primary regulator will then remain disabled until a rising edge is detected on the  $V_{IGN}$  pin. Once enabled, in addition to supplying  $V_{DD}$  to the internal circuitry, the primary regulator is also connected to the external  $V_{DD}$  pins, to allow external decoupling and to supply +5 V to the MDLC and A/D converter analog circuitry, via  $V_{CCA}$ , as well as providing a limited supply for external components. The secondary voltage regulator, which cannot be disabled through software, is used by the MC68HC705V8 to supply a regulated +5-V supply to the  $V_{IGN}$  and power supply control logic.

### NOTE

Even if the internal power supply is disabled,  $V_{BATT}$  must still be supplied to the MC68HC705V8 if the user wishes to use the MDLC module.

When laying out the power supply circuitry, the user should always follow good layout techniques to minimize EMI, and to ensure a stable power supply to the MCU and other components. These techniques include adequate decoupling of power supply pins, the use of a single point ground, and ground and power planes where possible. Refer to **Figure 3 MC68HC705V8 Power Supply and Multiplex Bus Interface Circuit** for an example MC68HC705V8 power supply connection circuit.

## Pulse Width Modulation

The MC68HC705V8 contains a 6-bit pulse width modulation (PWM) system with two user programmable prescalers. The PWM system can generate outputs ranging from 0% to 100% duty cycle. With the programmable prescalers and 6-bit counter, the user can attain a wide range of PWM output frequencies ranging from  $f_{osc}/64$  to  $f_{osc}/8192$ .

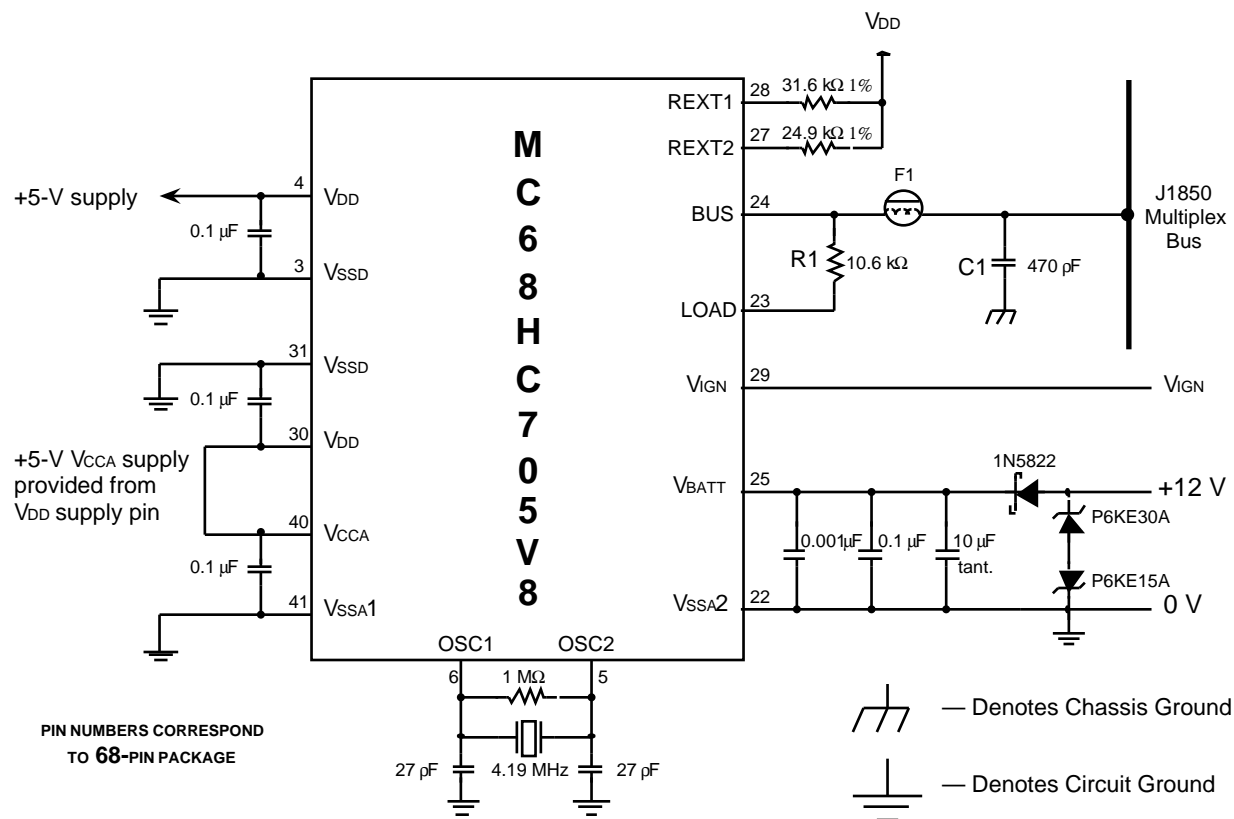
## Serial Peripheral Interface

The MC68HC705V8 contains a full-function serial peripheral interface (SPI) port, which can be used for high speed serial communication with other peripherals or MCUs. The SPI is a full-duplex, three-wire synchronous serial interface, with programmable clock phase and polarity, which can transmit and receive data at up to 1.05 MHz in Master Mode, or at up to 2.1 MHz in Slave Mode, depending upon the oscillator frequency.

## MDLC OVERVIEW

The message data link controller (MDLC) module is a communications module which has been designed to handle all of the necessary communication functions associated with transmitting and receiving frames

on a J1850-compatible serial multiplex bus. When a message is ready for transmission onto the multiplex bus, the CPU simply stores the entire message into the MDLC transmit buffer, loads the number of bytes to be transmitted into the MDLC transmit control register (MTCR), and then returns to normal application processing. The MDLC performs all of the necessary bus acquisition, frame transmission, arbitration and error detection to ensure that only complete, error-free frames are transmitted. When frames are received from the multiplex bus, the MDLC module performs the necessary bit and symbol decoding and frame verification, interrupting the CPU only when complete messages have been received from the multiplex bus error-free. Also, because all of the major components necessary for multiplex bus communication are integrated onto the MDLC module, including the analog transceiver, only a few inexpensive external discrete components are required to interface the MDLC module to the multiplex bus physical layer.



NOTE: Ferrite Bead F1 used for suppression of EMI above 5 MHz on the bus line; characteristics of ferrite bead should be similar to TDK# HF70ACB453215.

Figure 3. MC68HC705V8 Power Supply and Multiplex Bus Interface Circuit

### Control and Status Registers

The MDLC contains two registers which allow the user to configure the module and control message transmission, and two registers for relaying status information about the module and about received messages to the CPU. The MDLC Control Register (MCR) contains control bits which are used to set the receive mode of the MDLC module, to select the rate of transmission onto the multiplex bus, to enable MDLC

interrupts of the CPU, and to control how the MDLC module will operate when the WAIT instruction has been executed by the CPU. The MCR also contains the transmit abort (TXAB) control bit, which enables the user to abort a transmission which is in progress, allowing a different message, possibly one of higher priority, to then be transmitted. The MDLC Transmit Control Register (MTCR) is used to indicate to the MDLC module that a new message is in the transmit buffer awaiting transmission, and how many bytes are to be transmitted.

The MDLC Status Register (MSR) contains two status bits, one which indicates when a message has been transmitted successfully onto the multiplex bus, and another to indicate that a new message has been received from the multiplex bus. If MDLC interrupts of the CPU have been enabled, whenever either of these two bits is set an interrupt of the CPU will occur. When a received message is made available to the CPU, the MDLC Receive Status Register (MRSR) contains the number of bytes in the received message.

## Transmit Message Buffer Operation

The MDLC contains a single buffer for storing messages for transmission onto the multiplex bus, located in the MC68HC705V8 I/O registers memory map at locations \$20-\$2A. This transmit (Tx) buffer is an 11-byte buffer into which the CPU loads all necessary header and data bytes to be transmitted onto the multiplex bus. This allows the MDLC to transmit the maximum frame length allowed by J1850 (11 bytes + CRC byte). The CRC byte is automatically calculated and appended onto the frame by the MDLC following transmission of the last data byte.

The Tx buffer can only hold one complete message at a time. Once a complete message has been loaded into the Tx buffer, the CPU then writes the number of bytes to be transmitted into the MTCR. Once this is done, the user should never attempt to write further data to the Tx buffer until the message has been successfully transmitted, or discarded. If the transmission attempt has been successful, the TXMS bit in the MSR will be set, and an interrupt of the CPU will occur, if interrupts are enabled. If the CPU wishes to transmit a new message before the current one has been transmitted, it can empty the Tx buffer by setting the transmit abort (TXAB) bit in the MCR. This will clear the Tx buffer, aborting any transmission in progress. Once the TXAB bit has been reset, indicating that the transmit abort process is complete, the CPU can then begin loading new message bytes into the Tx buffer.

If an error is detected during a transmission onto the multiplex bus, the MDLC will immediately halt transmission. No attempt will be made to retransmit the message, and the Tx buffer will be cleared and again made available to the CPU. No indication will be given to the CPU that the transmission was unsuccessful. Any time a loss of arbitration is detected during a transmission, the MDLC will also immediately halt transmission. However, once an idle bus condition is detected, the MDLC will attempt to retransmit the message onto the multiplex bus.

## Receive Message Buffers Operation

The MDLC contains two buffers for storing messages received from the multiplex bus. Each buffer can hold up to 11 bytes, allowing the MDLC to also receive the maximum frame length allowed by J1850 (11 bytes + CRC byte). The receive buffers are both located at memory locations \$34-\$3E. These receive (Rx) buffers can each store a complete, maximum-length J1850 message (without the CRC). The MDLC will only place a message which has been received from the multiplex bus error-free into an Rx buffer. Any message in which an error is detected will be discarded, and the CPU will not be notified. Once the MDLC has placed a complete message in an Rx buffer, it makes this Rx buffer available to the CPU while denying the CPU access to the other Rx buffer until the next message has been received. Since only one of these Rx buffers can be accessed by the CPU at a time, to the user there appears to be only a single Rx buffer. This "ping-pong" action allows the MDLC to store a message being received from the multiplex bus in one Rx buffer while the user is retrieving a previously received message from the other Rx buffer.

When a message is received by the MDLC and placed into an Rx buffer, the user is notified via the RXMS bit in the MSR register, and an interrupt of the CPU is generated, if MDLC interrupts are enabled. Any



access of the MRSR will clear the interrupt and reset the RXMS bit in the MSR. Once the user has retrieved any data bytes of interest from a received message, a write to the MRSR will release the Rx buffer back to the MDLC module. If another message has already been placed in the other Rx buffer, this buffer will then be made available to the CPU, the RXMS bit will again be set, and another interrupt of the CPU will be generated. Once the MDLC has stored a received message in each Rx buffer, it will ignore any further frames being transmitted onto the multiplex bus until the user releases one of the Rx buffers.

The MDLC module can also be used to receive messages containing large blocks of data. This type of message violates the J1850 message frame length constraints, and will normally be used only in a manufacturing or diagnostic environment. To utilize this feature, the user sets the receive block mode (RXBM) bit in the MCR. When this bit is set, once a valid SOF delimiter is detected by the MDLC, it will begin loading data bytes into an Rx buffer. As soon as that Rx buffer is filled, the MDLC will make that buffer available to the CPU and begin filling the other buffer. This will continue until the MDLC detects a valid EOD symbol. Each time an Rx buffer is filled, the number of bytes contained in the MRSR will only reflect the number of bytes in that Rx buffer, not a cumulative total. The MDLC will also calculate a cumulative CRC byte. Once the EOD symbol is received, the MDLC will verify that the cumulative CRC is equal to  $\$C4$ , just as with a normal message reception. Once the complete message has been received error-free, the MDLC will clear the RXBM bit in the MCR and return to the normal reception mode. If the MDLC detects an error during a block mode reception, the RXBM bit will also be cleared, and the MDLC will again return to the normal mode of operation. The MDLC cannot be used to transmit block mode messages.

Due to the nature of the J1850 bus, each node must receive every frame it transmits to ensure proper arbitration and error-detection. Therefore, the MDLC module will receive, and pass along to the user, every message that it successfully transmits on to the multiplex bus. This feature can allow a node's software to handle a received message that it has transmitted in the same way it would handle a message received from another node in the system. This feature can also be used to determine whether or not a loss of arbitration or a transmission error has occurred during a transmission attempt.

## Error Detection

The MDLC uses a variety of methods to ensure the data transmitted onto or received from the multiplex bus is error-free. These include a digital input filter, CRC generation and checking, and a constant monitoring of bit and symbol timing, as well as message framing.

All data received from the multiplex bus passes through a digital filter. This filter removes short noise pulses from the input signal, which could otherwise corrupt the data being received. The "cleaned up" signal is then passed to the symbol decoder, which decodes the data stream, determining what each bit or symbol is, whether it is of the proper length, and that the message is framed properly.

The CRC byte is calculated by the MDLC as it transmits a frame onto the multiplex bus and is then appended to the message following the data portion of the frame. The CRC of any message the MDLC receives, including messages it has transmitted, is checked, and if it is not correct, the frame is discarded.

For more information on the different methods of error detection and notification used by the MDLC module, refer to the *MC68HC705V8 Product Specification*.

## Physical Layer Transceiver

One unique feature of the MC68HC705V8 is the analog transceiver necessary for interfacing the MDLC module to the multiplex bus physical layer, which is integrated directly onto the MCU. Because of this, only a few discrete components are necessary to complete the connection to the physical layer, and to ensure protection from the extreme transients common in the automotive environment. The MDLC transceiver is designed to transmit and receive messages with a dominant voltage level of 7 V (nominal), as specified in the J1850 document. Two external resistors are used to "tune" the rise and fall times of the waveform, and

a resistor and capacitor are required to ensure the appropriate bus loading for the node. The transceiver design allows communication to be maintained with up to a 2-V ground offset between nodes, and also allows the MDLC to survive a variety of fault conditions, including a short between the multiplex bus line and  $V_{BATT}$ . Refer to **Figure 3 MC68HC705V8 Power Supply and Multiplex Bus Interface Circuit** for an example circuit for connecting the MC68HC705V8 to the multiplex bus.

#### NOTE

The load capacitor (C1) in this circuit should be connected between the module multiplex bus pin and chassis ground of the module, and should be mounted as close to the module bus pin as possible. Also, the load resistor (R1) and load capacitor should be adjusted to provide the appropriate multiplex bus loading, depending upon the total number of nodes connected to the multiplex bus.

## MC68HC705V8 SOFTWARE ROUTINES

The example software routines in Appendix A perform three functions: initialize the MC68HC705V8 and MDLC module for J1850 communication, transmit messages stored in MCU RAM onto the multiplex bus, and receive, filter and store messages received from the multiplex bus into MCU RAM. The software listing in Appendix A contains a simple program which demonstrates how each of these functions may be performed. When a reset occurs, the MCU and MDLC will be initialized. Following this, a message will be transmitted onto the multiplex bus which contains a three-byte header and a single data byte. Once the message transmission is initiated, the software will enter a delay loop while the transmission takes place, and then increment the data byte and loop back to transmit the message again. When messages are received from the multiplex bus, the MDLC interrupt service routine will filter the messages based on the type of message and the target address (2nd byte) of the message. Several example target I.D.s have been selected, although the user can change, add or remove I.D.s from this list as desired. Messages passing the filtering criteria will be stored in receive buffers in MCU RAM. Below is a brief description of each segment of the example software and how it works. This routine uses less than 250 bytes of programmable memory, plus 62 bytes of user RAM, although the amount of RAM required depends upon the number of data bytes contained in each received message.

### Mask Option Register

When the device is programmed, the user must also program the mask option register (MOR). This register allows the user to enable or disable certain device features which would normally be enabled or disabled with mask options on a ROM device.

For these software routines, the MOR byte is programmed to enable the internal voltage regulator, to clamp the  $V_{DD}$  pins to  $V_{SS}$  internally when the primary voltage regulator is disabled by software, to allow the primary voltage regulator to power up when a rising edge is detected on the BUS pin of the MDLC module, to enable the low voltage reset (LVR) option, to disable entry into the STOP mode of operation, to set the external interrupt request sensitivity to negative-edge only, and to enable the COP watchdog timer.

### MCU Initialization

The MCU initialization routine initializes the necessary MCU registers and modules, and also initializes the MCU RAM used by these communication routines. It should be necessary to run this routine only following a reset of the MCU. All of the setup procedures described below can be performed by calling the subroutine V8\_RST.

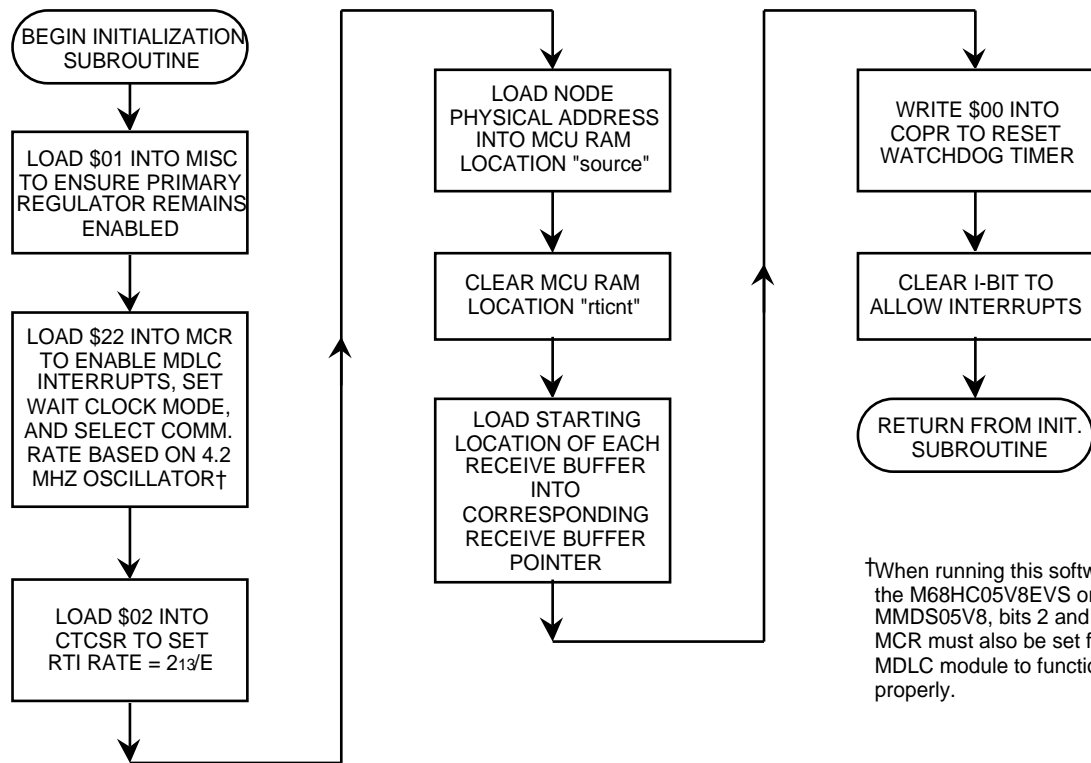


Figure 4. MC68HC705V8 Reset Subroutine

Following a reset of the MC68HC705V8, three MCU registers are initialized for use by these driver routines. The miscellaneous (MISC) register is initialized with output compare disabled and power down control enabled, enabling the primary voltage regulator. The MDLC control register (MCR) is initialized with receive block mode disabled, R1:0 adjusted for 4.19 MHz oscillator, interrupts enabled, and wait clock mode disabled. **In this example bits 3 and 2 of the MCR are also set. This is necessary for this software to be run on the M68HC05V8EVS.** These bits will be ignored by the MC68HC705V8 running in Single-Chip Mode. This routine also initializes the core timer control and status register (CTCSR) to set the RTI interrupt rate to once every  $2^{13}/E$ , or every 3.90 ms. At this time, RTI interrupts are not enabled.

Next, the physical address of the node is stored in MCU RAM location "source". This RAM location corresponds to the third byte in the RAM transmit buffer. This ensures that the physical address of the node will be transmitted as the third byte of each message transmitted by the MDLC, as required by the three-byte message header format outlined in SAE J1850. The MCU RAM location "rticnt" is then cleared, preparing it for use with the MDLC message transmit subroutine.

The only other MCU RAM initialization required is the initialization of the received message buffer pointers (RMBP) pointing to the functional message receive buffers. Each RMBP is loaded with the starting address of a corresponding functional message receive buffer. In this example, there are four functional message receive buffers, one for each functional message I.D. However, the number of functional I.D.s, and buffers, can be increased by the user, with the only limit being the amount of RAM available and the amount of time the user can spend sorting received messages.

Once this is complete and the I-bit is cleared, enabling interrupts, the MC68HC705V8 is loaded and ready for multiplex communication. Refer to **Figure 4 MC68HC705V8 Reset Subroutine** for a graphical representation of the reset sequence.

## Transmitting

To transmit a message onto the multiplex bus using the MDLC, the CPU simply stores the message bytes and message byte count into the correct MCU RAM locations and then calls the TRANSMIT subroutine. The software handles moving the data from MCU RAM storage into the MDLC Tx buffer, initiating the message transmission, and determining when the message has been transmitted successfully.

When the application has data to be transmitted onto the multiplex bus, the user stores the message bytes, including the header bytes, into MCU RAM, beginning at location "txbuf". The total number of bytes in the message to be transmitted is then loaded into the RAM location "txcount". The user then calls the subroutine TRANSMIT. This subroutine will transfer the new message bytes to the MDLC module for transmission onto the multiplex bus, and then load the number of bytes to be transmitted into the MTCR, initiating the message transmission. **Figure 5 Example Transmit Sequence** illustrates the sequence used by this example software to transmit a message onto the multiplex bus.

Once the message transmission has been initiated, the TRANSMIT subroutine will then enable the real time interrupt (RTI) function of the Core Timer. The RTI interrupt function is used to monitor the amount of elapsed time since the initiation of the message transmission. Following this, each time an RTI interrupt occurs, the RTI interrupt service routine will increment a counter stored in MCU RAM location "rticnt".

When the message is successfully transmitted, a transmission successful (TXMS) interrupt of the CPU will be generated. When this occurs, the software will then disable the RTI interrupt, clear the RTI counter in MCU RAM location "rticnt", and write \$00 to the MTCR, clearing the interrupt. Following this, the MDLC module will be ready for another message transmission. For an outline of the TXMS interrupt service routine, refer to the transmit interrupt portion of **Figure 7b MDLC Interrupt Service Routine (concluded)**.

However, if the transmission successful (TXMS) interrupt does not occur before 8 RTI interrupts have been counted, the transmission will be aborted. This gives the MDLC module approximately 30 ms to successfully complete the transmission of a message. Due to the uncertainty of when the RTI interrupt is enabled, the time allowed for the transmission to complete will vary between 7 and 8 RTI interrupts, or between approximately 27 ms and 31 ms. This time period can easily be adjusted by the user by increasing or decreasing the maximum RTI count in the RTI interrupt service routine. Also, if so desired, the user can easily initiate another transmission attempt when the RTI counter reaches 8, rather than simply aborting the transmission. Refer to **Figure 6 Real Time Interrupt Service Routine** for an outline of the RTI interrupt handling sequence.

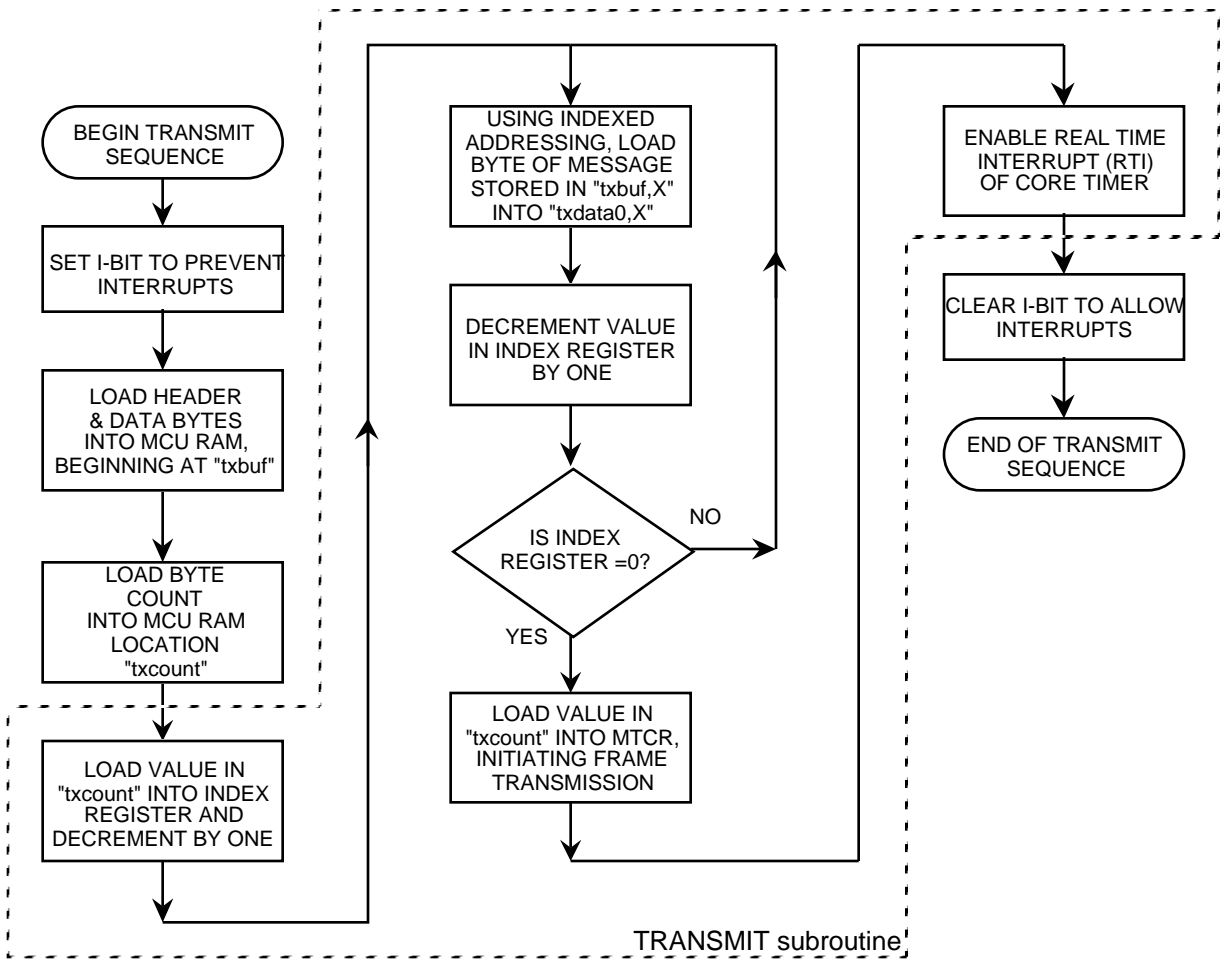
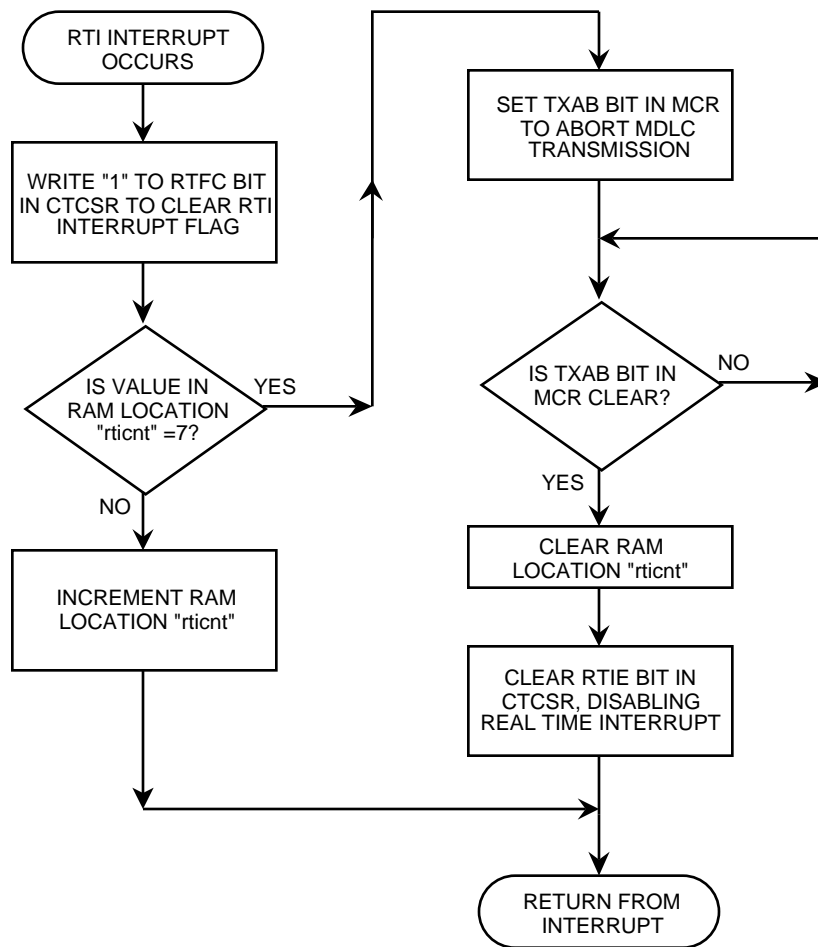


Figure 5. Example Transmit Sequence



**Figure 6. Real Time Interrupt Service Routine**

## Receiving

When the MDLC has received an error-free message from the multiplex bus, the MDLC interrupt service routine performs the necessary message filtering and data retrieval. If the received message passes the message filtering criteria, the message data bytes are stored in MCR RAM. If the received message does not meet the message filtering criteria, the message is discarded.

As soon as a received message is stored in one of the MDLC's two Rx buffers, a received message (RXMS) interrupt of the CPU is generated. The interrupt service routine first tests bit 2 of the first byte of the message. The first byte of the message is the Priority/Type byte in the 3-byte consolidated header format defined in SAE J1850. Bit 2 is the Type bit, which is used to indicate whether the message is functionally addressed or physically addressed. If this bit is set, indicating that the message is a physically addressed message, the second byte of the message, the target address byte, is compared to the node address. If the target address of the message matches the physical address of the node, the message data bytes are retrieved and stored in MCU RAM beginning at location "nbuff", and the third byte of the message, the physical address of the message originator, is stored in MCU RAM location "nsource". If the target address of the received message does not match the physical address of the node, a write to the MRSR is performed, clearing the RXMS interrupt and releasing the Rx buffer back to the MDLC, effectively discarding the message.

If the Type bit of the first byte is clear, indicating that the message is a functionally addressed message, the second byte of the message, the target address byte, is then retrieved. This byte is compared to each functional I.D. for which a Received Message Buffer has been reserved. If a match is found, the Received Message Buffer Pointer corresponding to that functional I.D. is loaded into MCU RAM location "buff\_ndx". The data bytes are then retrieved and stored in the MCU RAM message buffer which corresponds to that functional I.D. The target and source address bytes of the message are not retained. It is not necessary to retain these bytes of the message, since a logical assumption is that the functional I.D. must be known to the receiver already, and the source address is of no use since the function, and not the source, of the message data is what is important.

When a message of interest to the user is received, the RXMS interrupt will be cleared when the number of received message bytes in the MRSR is read prior to retrieval of the message data bytes. Once the data bytes of interest are retrieved, then the MRSR is written to, releasing the Rx buffer back to the MDLC. If a message which does not pass the message filtering criteria is received, the write to the MRSR to release the Rx buffer and discard the message also serves to clear the RXMS interrupt generated by that message.

This procedure results in each MCU RAM receive buffer containing the latest data received for each specific functional I.D., and for the most recently received node-to-node message. The CPU can access this data whenever it needs updated information. Whenever this stored data is accessed, however, the user should first set the I-bit to inhibit interrupts. If a receive interrupt is serviced while the CPU is accessing this stored data, it is possible that the CPU could end up reading partial data from two different received messages.

Refer to **Figure 7a MDLC Interrupt Service Routine (cont.'d)** and **Figure 7b MDLC Interrupt Service Routine (concluded)** for a flow diagram of the sequence followed when an MDLC interrupt of the CPU occurs.

## Error Handling

The basic driver routines in Appendix A do not contain extensive error handling procedures. Since any message in which an error is detected is discarded by the receiver, the user has no need (or way) to monitor receive errors as they occur. The user can monitor, to a limited extent, the status of the multiplex bus through the use of a test message, or by monitoring time between received messages.

When transmitting a message, the user can determine whether a loss of arbitration or a transmission error has occurred by checking messages received after a transmission has been originated. If messages of higher priority are continually received, the user may wish to abort the transmission and retry using a higher message priority. If more than one message of lower priority is detected following the initiation of a message transmission, the user may assume a transmit error occurred, and initiate another attempt to transmit the message, if so desired.

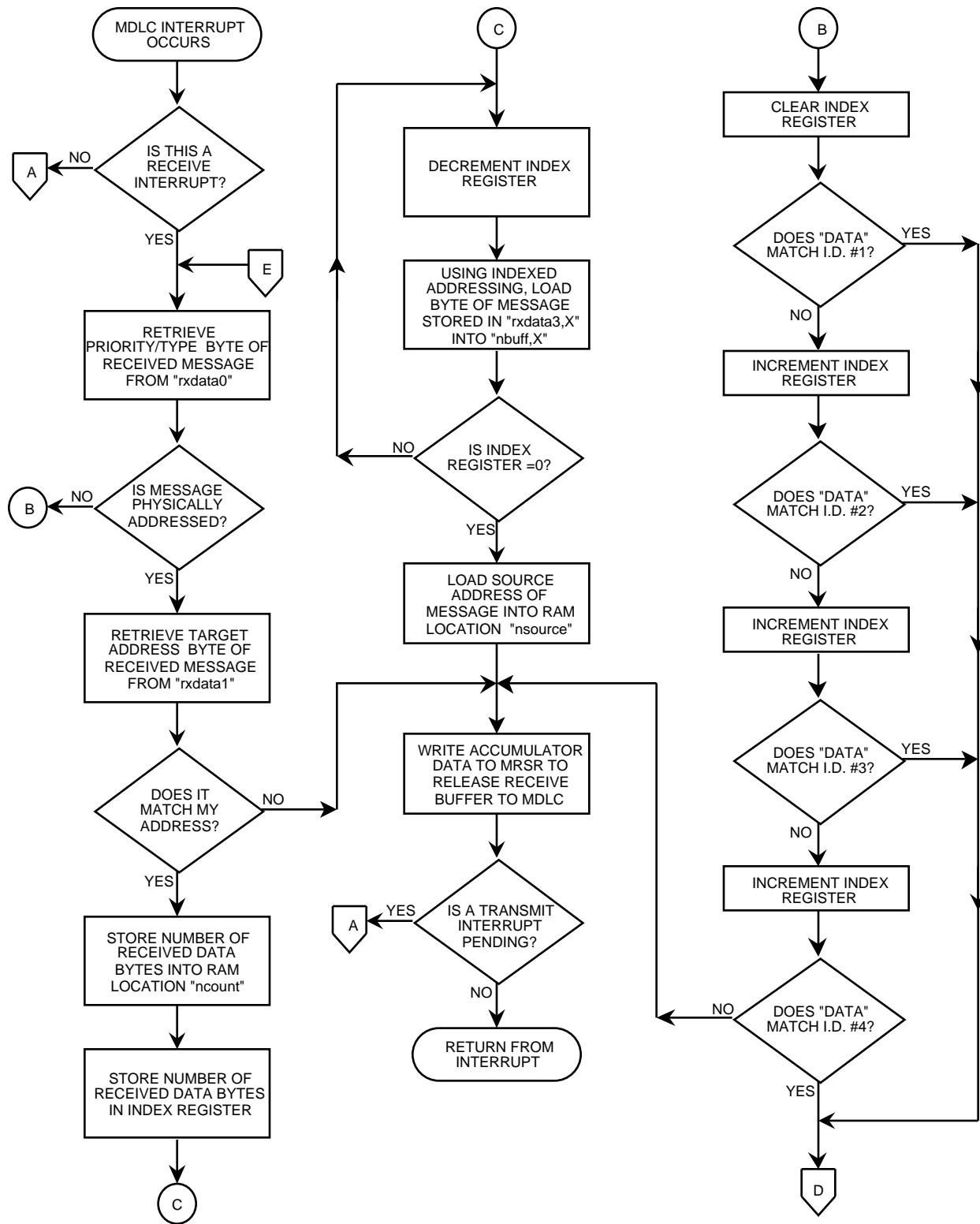


Figure 7a. MDLC Interrupt Service Routine (cont.'d)



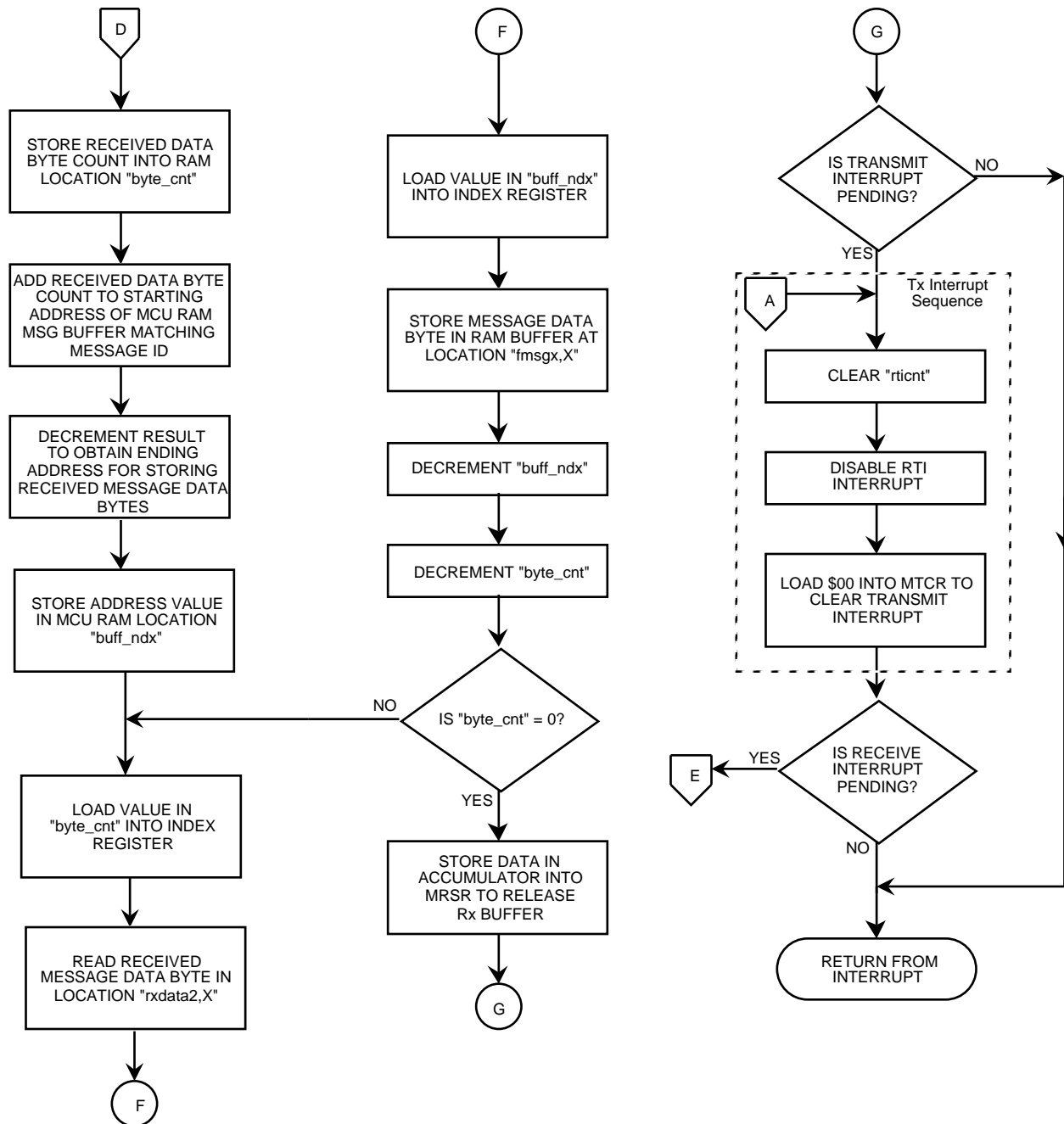


Figure 7b. MDLC Interrupt Service Routine (concluded)

## SUMMARY

The software driver routines in Appendix A are intended as examples which can be used as a starting point for the development of application software which includes the use of the MDLC module. They should allow the user to quickly construct a basic application using the MC68HC705V8 for communication onto a J1850 multiplex bus, but do not provide a full range of error detection procedures, or otherwise utilize all of the features the MDLC module contains for transmitting and receiving message frames on a J1850-compatible multiplex bus. For a detailed description of the functions of the MDLC module, refer to the *MC68HC705V8 Product Specification (General Release)*.

## REFERENCES

*MC68HC705V8 Product Specification*, Motorola, 1993

*MC68HC05 Applications Guide*, M68HC05AG/AD, Motorola, 1989

*Society of Automotive Engineers Recommended Practice J1850-Class B Data Communication Network Interface*, J1850, SAE, 1993

*Society of Automotive Engineers Recommended Practice J2178 Class B Communication Network Messages*, J2178, SAE, 1992

# Appendix A

MDLCCODE.ASM                    Assembled with IASM    08/05/1993   04:57   PAGE 1

```
1 *****
2 *
3 *                    MC68HC705V8 MDLC Module                    *
4 *                    Example Driver Software                    *
5 *
6 *        This software is an example of a basic software routine which        *
7 * can be used with the Message Data Link Controller (MDLC) module on the        *
8 * Motorola MC68HC705V8 microcontroller. This simple example program loads        *
9 * a message into MCU RAM and then calls the transmit subroutine TRANSMIT        *
10 * which loads the message into the MDLC module transmit buffer and then        *
11 * initiates the message transmission. Once the message transmission is        *
12 * initiated, the Real Time Interrupt (RTI) function on the MCU's Core        *
13 * Timer is activated. If the transmission successful interrupt does not        *
14 * occur before 8 RTI interrupts have occurred (about 30ms), the trans-        *
15 * mission is aborted. When the transmission successful interrupt does        *
16 * occur, the RTI interrupt is disabled. While waiting for the message to        *
17 * complete successfully or to be aborted, the program simply waits in a        *
18 * time delay loop.
19 *        This basic routine interprets received messages as either physically*
20 * addressed or functionally addressed. In J1850, a functionally addressed *
21 * message is indicated if Bit 2 of the Pri/type byte (1st byte) of the *
22 * message is clear, while for physically addressed messages, Bit 2 will *
23 * be set. No other message type determination is made, and no in-frame *
24 * response byte is (or can be) returned.
25 *        When a message is received, the MDLC interrupt service routine will *
26 * compare the target address of the received message with 4 pre-defined *
27 * message identifiers. If the target address of the message matches one *
28 * of the four, the message is retrieved and saved in a buffer reserved for*
29 * that message ID. If the target address of the received message does not *
30 * match, the message is discarded. Also, if the received message is a *
31 * physically addressed message, the target address of the received *
32 * message is compared with the assigned node address, and if it matches *
33 * the message is then retrieved and stored in a RAM buffer reserved for *
34 * physically addressed messages, along with the source address of the *
35 * message originator.
36 *        Though simple, this software demonstrates all of the basic features *
37 * of the MDLC module, and how they can be used for performing communi- *
38 * cation across a J1850-compatible multiplex bus.
39 *
40 *                    Revision History
41 *
42 *        Rev 0.1 (initial release)            Chuck Powers            5/03/93        *
43 *        Rev 0.2 Updated transfer routines                                            *
44 *                    between MCU RAM and the                                            *
45 *                    MDLC transmit and receive                                            *
46 *                    buffers.                    Chuck Powers            7/08/93        *
47 *        Rev 0.3 Added initialization of                                            *
48 *                    MISC register                    Chuck Powers            8/05/93        *
49 *
50 *****
51
```

```

52 *****
53 ***** Equates *****
54 *****
55
56 *** MC68HC705V8 Register Equates ***
57
0000 58 porta equ $00 ;Port A Data Register
0000 59 portb equ $01 ;Port B Data Register
0000 60 portc equ $02 ;Port C Data Register
0000 61 portd equ $03 ;Port D Data Register
62
0000 63 ddra equ $04 ;Port A Data Direction Register
0000 64 ddrb equ $05 ;Port B Data Direction Register
0000 65 ddrc equ $06 ;Port C Data Direction Register
66
0000 67 ctcsr equ $08 ;Core Timer Control/Status Register
0000 68 ctcount equ $09 ;Core Timer Count Register
69
0000 70 mcr equ $0e ;MDLC Control Register
0000 71 msr equ $0f ;MDLC Status Register
0000 72 mtcrcr equ $10 ;MDLC Transmit Control Register
0000 73 mrcsr equ $11 ;MDLC Receive Status Register
0000 74 txdata0 equ $20 ;Beginning of MDLC Tx Buffer
0000 75 rxdata0 equ $34 ;Beginning of MDLC Rx Buffer(s)
0000 76 rxdata1 equ $35 ;Target Address storage in MDLC Rx Buffer(s)
0000 77 rxdata2 equ $36 ;Source Address storage in MDLC Rx Buffer(s)
0000 78 rxdata3 equ $37 ;Beginning of MDLC Rx Data Bytes
79
0000 80 misc equ $2f ;Miscellaneous Register
81
82 *** MCR Bit Assignments ***
83
0000 84 rxbm equ 7 ;Receive Block Mode Enable
0000 85 txab equ 6 ;Transmit Abort
0000 86 rl equ 5 ;\ J1850 Bus Communication
0000 87 r0 equ 4 ;/ Rate Select Bits
0000 88 ie equ 1 ;MDLC Interrupt Enable
0000 89 wcm equ 0 ;MDLC Wait Clock Mode Select
90
91 *** MSR Bit Assignments ***
92
0000 93 txms equ 3 ;Transmitter Successful Indicator
0000 94 rxms equ 2 ;Receiver Successful Indicator
95
96 *** MISC Bit Assignments ***
97
0000 98 igns equ 7 ;Ignition Status
0000 99 oce equ 6 ;Output Compare Enable
0000 100 pdc equ 0 ;Power Down Control Indicator
101
102 *** CTCSR Bit Assignments ***
103
0000 104 ctovf equ 7 ;Core Timer Overflow Flag
0000 105 rtif equ 6 ;Real Time Interrupt Flag
0000 106 tofe equ 5 ;Timer Overflow Interrupt Enable
0000 107 rtie equ 4 ;Real Time Interrupt Enable
0000 108 tofc equ 3 ;Timer Overflow Flag Clear Bit
0000 109 rtfc equ 2 ;RTI Interrupt Flag Clear Bit
0000 110 rtl equ 1 ;\ Real Time Interrupt
0000 111 rt0 equ 0 ;/ Rate Select Bits
112
113 *** Receive Control Bit Assignments
114
0000 115 type equ 2 ;Rxdata0, Bit 2 (Received Msg Type Indicator)
116

```

```

117 *** General Equates ***
118
0000 119 ram      equ    $0040 ;Beginning of user RAM
0000 120 rom      equ    $0d00 ;Beginning of user ROM
0000 121 mdlc     equ    $1300 ;Beginning of MDLC Interrupt Service Routine
0000 122 ctimer   equ    $1000 ;Beginning of Core Timer Int. Service Routine
0000 123 sub      equ    $2000 ;Beginning of MDLC Subroutines
0000 124 vectors  equ    $3ff2 ;Beginning of user vectors
0000 125 none     equ    $0000 ;Bogus Location
0000 126 mor      equ    $3c00 ;Mask Option Register Location
0000 127 copr     equ    $3ff0 ;COP Watchdog Reset Location
128
129 *** Node Address ***
130
0000 131 node     equ    $77      ;MC68HC705V8 Node physical address
132
133 *** Functional Message I.D.'s ***
134
0000 135 id1      equ    $1a
0000 136 id2      equ    $20
0000 137 id3      equ    $5e
0000 138 id4      equ    $d3
139
140 *****
141 *****          HC05 RAM Storage Assignments          *****
142 *****
143
0040 144          org    ram
145
146 *** Variable Data Byte Storage ***
147
0040 148 data_byt rmb    $1      ;Tx Message Variable Data Byte Storage
149
150 *** Core Timer Registers ***
151
0041 152 rticnt   rmb    $1      ;RTI Interrupt Count Register
153
154 *** Transmit Message Buffer ***
155
0042 156 txcount  rmb    $1      ;Host Transmit Message Byte Count
0043 157 txbuf    rmb    $1      ;Start of Host Transmit Message Buffer
0044 158 target   rmb    $1      ;Tx Message Target Address
0045 159 source   rmb    $1      ;Tx Message Source Address
0046 160 data1    rmb    $1      ;Tx Message 1st Data Byte
0047 161 data2    rmb    $1      ;Tx Message 2nd Data Byte
0048 162 data3    rmb    $1      ;Tx Message 3rd Data Byte
0049 163 data4    rmb    $1      ;Tx Message 4th Data Byte
004A 164 data5    rmb    $1      ;Tx Message 5th Data Byte
004B 165 data6    rmb    $1      ;Tx Message 6th Data Byte
004C 166 data7    rmb    $1      ;Tx Message 7th Data Byte
004D 167 data8    rmb    $1      ;Tx Message 8th Data Byte
168
169 *** Received Message Storage ***
170
004E 171 buff_ndx rmb    $1      ;Temporary storage for receive message index
004F 172 byte_cnt rmb    $1      ;Temporary storage for received message count
173
0050 174 ncount   rmb    $1      ;RAM holding # of Data Bytes in last N-N msg
0051 175 nsource  rmb    $1      ;RAM holding source address of last N-N msg
0052 176 nbuff    rmb    $8      ;RAM holding last received N-N message
177
005A 178 fmsg1    rmb    $1      ;Pointer to RAM holding funct. message w/id1
005B 179 fmsg2    rmb    $1      ;Pointer to RAM holding funct. message w/id2
005C 180 fmsg3    rmb    $1      ;Pointer to RAM holding funct. message w/id3
005D 181 fmsg4    rmb    $1      ;Pointer to RAM holding funct. message w/id4
182
005E 183 buff1    rmb    $8      ;RAM holding last received message w/id1
0066 184 buff2    rmb    $8      ;Ram holding last received message w/id2
006E 185 buff3    rmb    $8      ;RAM holding last received message w/id3
0076 186 buff4    rmb    $8      ;RAM holding last received message w/id4
187

```

```

188 *****
189 *                               Main Program                               *
190 *****
191
192 *****
193 * This is the main line program for this example. Following a reset,      *
194 * this routine will reset the MC68HC705V8, preparing it for J1850        *
195 * communication, and then transmit a message made up of: Pri/type = $31 *
196 * Target Address = $A1, Source Address = $77, and a variable data byte. *
197 * Following the initiation of transmission, the variable data byte is    *
198 * incremented, and then a delay loop is entered which allows the trans- *
199 * mission to take place. Periodically during the delay, the COP timer is *
200 * cleared. Once the delay loop is complete, the Tx routine begins again.*
201 *****
0D00          202          org          rom
                203
0D00 [06] CD2000 204          jsr          v8_rst          ;Call subroutine to reset the MCU
                205
0D03 [05] 3F40   206          clr          data_byt
                207
0D05 [02] 9B     208 doover:   sei              ;Set I-bit to prevent unwanted interrupts
                209
0D06 [02] A631   210          lda          #$31          ;Load Pri/Type byte ($31) into
0D08 [04] B743   211          sta          txbuf          ;1st byte of RAM Tx buffer
                212
0D0A [02] A6A1   213          lda          #$a1          ;Load target address byte into
0D0C [04] B744   214          sta          target          ;second byte of RAM Tx buffer
                215
0D0E [03] B640   216          lda          data_byt      ;Load variable data byte into
0D10 [04] B746   217          sta          data1          ;first data byte location of RAM Tx buffer
                218
0D12 [02] A604   219          lda          #$04          ;Load number of bytes to be transmitted
0D14 [04] B742   220          sta          txcount        ;into RAM location TXCOUNT
                221
0D16 [06] CD202A 222          jsr          transmit        ;Call subroutine to transmit message
                223          ;onto multiplex bus
                224
0D19 [02] 9A     225          cli              ;Clear I-bit to allow interrupts
                226
0D1A [05] 3C40   227          inc          data_byt      ;Increment transmitted variable data byte
                228

0D1C [02] AE3F   229          ldx          #$3f          ;Begin delay loop...
                230
0D1E [02] A6FF   231 lp1:     lda          #$ff
0D20 [03] 4A     232 lp0:     decb
0D21 [03] 26FD   233          bne          lp0
                234
                235          *** Kick Watchdog ***
                236
0D23 [02] A600   237          lda          #$00          ;Load $00 into COPR to kick
0D25 [05] C73FF0 238          sta          copr          ;the COP watchdog
                239
0D28 [03] 5A     240          decx
0D29 [03] 26F3   241          bne          lp1          ;end delay loop
                242
0D2B [03] CC0D05 243 skip:   jmp          doover
                244

```

```

245 *****
246 *****          MCU Initialization Routine          *****
247 *****
248
249 *****
250 * This subroutine initializes all of the MCU modules and RAM locations *
251 * used in this example routine. This includes the MISC register, the *
252 * MDLC module, the Core Timer, the node address, and the RTI counter. *
253 * The COP Watchdog is then kicked for good luck! *
254 *****
255
2000      256          org      sub
257
2000 [02] 9B      258      v8_rst:  sei          ;To reset the MCU, JSR to here
259
260      *** Initialize Miscellaneous for Voltage Regulator Operation ***
261
2001 [02] A601    262          lda      #00000001  ;Output Compare disabled,
2003 [04] B72F    263          sta      misc      ;Primary voltage regulator enabled
264
265      *** Initialize MDLC for J1850 Comm. ***
266
2005 [02] A62E    267          lda      #00101110  ;B7-rxbr, B6-txab, B5:4-Rate Select
2007 [04] B70E    268          sta      mcr        ;B1-Interrupt Enable, B0-Wait Clock Mode
269          ;Also, MTST1:0=1 for EVS
270
271      *** Initialize Core Timer ***
272
2009 [02] A602    273          lda      #00000010  ;Initialize the Core Timer for using
200B [04] B708    274          sta      ctcsr     ;the RTI for timing transmissions
275
276      *** Initialize 05V8 Node Address ***
277
200D [02] A677    278          lda      #node      ;Store the node address into
200F [04] B745    279          sta      source    ;the correct location in the RAM
280          ;Transmit buffer
281      *** Clear RTI Counter ***
282
2011 [05] 3F41    283          clr      rticnt     ;Clear RTI counter, just to make sure
284          ;it begins life=0
285
286      *** Initialization of Receive Message Buffer Pointers ***
287
2013 [02] A65E    288          lda      #buff1     ;Load location of message buffer w/id1
2015 [04] B75A    289          sta      fmsg1     ;in message buffer pointer fmsg1
290
2017 [02] A666    291          lda      #buff2     ;Load location of message buffer w/id2
2019 [04] B75B    292          sta      fmsg2     ;in message buffer pointer fmsg2
293
201B [02] A66E    294          lda      #buff3     ;Load location of message buffer w/id3
201D [04] B75C    295          sta      fmsg3     ;in message buffer pointer fmsg3
296
201F [02] A676    297          lda      #buff4     ;Load location of message buffer w/id4
2021 [04] B75D    298          sta      fmsg4     ;in message buffer pointer fmsg4
299
300      *** Kick Watchdog ***
301
2023 [02] A600    302          lda      #$00       ;Load $00 into COPR to kick
2025 [05] C73FF0  303          sta      copr      ;the COP watchdog
304
2028 [02] 9A      305          cli          ;Clear I-bit, enabling interrupts
306
2029 [06] 81      307          rts          ;Return from Reset subroutine
308

```

```

309 *****
310 ***** Subroutines *****
311 *****
312
313 *** MDLC Transmit Subroutine ***
314
315 *****
316 * This subroutine transfers the data in the RAM transmit buffer into *
317 * the MDLC Tx buffer, loads the number of bytes to be transmitted *
318 * into the MTCR, initiating the message transmission, and then *
319 * enables the RTI interrupt to count RTI's until either the message *
320 * is transmitted successfully, or 30ms has passed. If 30ms passes *
321 * without the message transmitting successfully, the message will *
322 * be aborted automatically by the software. *
323 *****
324
202A [03] BE42 325 transmit: ldx txcount ;Load the number of bytes to be transmitted
326 ;into the X-register
327
202C [03] 5A 328 nexttx: decx ;Now decrement X to get it to the correct
329 ;value to begin an indexed transfer of the
330 ;data from RAM into the MDLC Tx buffer
331
202D [04] E643 332 lda txbuf,x ;Load message data byte (last byte first)
202F [05] E720 333 sta txdata0,x ;into the MDLC Tx register
334
2031 [03] 5D 335 tstx ;Test the X-Register to see if it is 0
336
2032 [03] 26F8 337 bne nexttx ;If X<>0, indicating the last byte has not
338 ;been transferred, then go get the next byte
339
2034 [03] B642 340 lda txcount ;Otherwise, load Tx byte count into Acc.
341
2036 [04] B710 342 sta mtcrcr ;Then store in MTCR, initiating transmission
343
2038 [05] 1408 344 bset rtfc,ctcsr ;Clear RTI interrupt flag, in case it is set
345
203A [05] 1808 346 bset rtie,ctcsr ;Enable RTI interrupt to track how
347 ;long the transmission is pending
348
203C [06] 81 349 rts ;Then return from subroutine
350
351 *****
352 ***** MDLC Interrupt Routine Service *****
353 *****
354
355 *****
356 * The MDLC interrupt service routine handles both the transmit and *
357 * receive interrupts of the CPU generated by the MDLC module. When a *
358 * receive interrupt is serviced, the software automatically checks *
359 * the message type of the received message. If it is a physically *
360 * addressed message, the target address of the message is compared *
361 * to the node address, and if it matches, the message data bytes and *
362 * source address are retrieved and stored in RAM. If it is a funct- *
363 * ionally address message, the target address is compared to 4 pre- *
364 * viously defined target addresses and if there is a match, the data *
365 * bytes of the message are stored in that message ID's corresponding *
366 * RAM buffer. The source or target address are not retained, since *
367 * it is assumed that the receiver must already know what the message *
368 * pertains to. *
369 * When a transmit interrupt occurs, the MDLC module will *
370 * acknowledge the successful transmission by clearing the transmit *
371 * interrupt, and then disabling the RTI interrupt and clearing the *
372 * RTI counter register. The application is then ready for another *
373 * transmission. *
374 *****
375
1300 376 org mdlc
377
1300 [05] 040F03 378 brset rxms,msr,rxint ;If an receive interrupt, go to
379 ;receive interrupt service routine

```



```

380
1303 [03] CC1358 381          jmp     txint      ;Otherwise, go to transmit interrupt
382                ;service routine
383
384 *** Receive Message Successful Interrupt Handling Routine ***
385
1306 [05] 05341C 386 rxint:   brclr   type,rxdata0,funct ;Is Bit 2 of the Pri/Type byte
387                ;of the message set, indicating
388                ;a physically addressed message?
389
1309 [03] B635   390         lda     rxdata1   ;If physically addressed,
391                ;load Target Address into Acc.
392
130B [02] A177   393         cmp     #node      ;Does it match my Node Address?
394
130D [03] 2647   395         bne     rxdone   ;If not, ignore message and release Rx buffer
396
130F [03] B611   397         lda     mrsr     ;Otherwise, load received message
398                ;byte count into Acc.
399
1311 [02] A003   400         sub     #$03       ;Subtract $03 for correct # of data bytes
401
1313 [04] B750   402         sta     ncount    ;Then store in N-N message byte count storage
403
1315 [02] 97     404         tax           ;Also store # of data byte in X-register
405
1316 [03] 5A     406 nextn:   decx          ;Then decrement X, for correct offset
407                ;for indexed transfer of data from the
408                ;MDLC Rx buffer to the Node-Node RAM buffer
409
1317 [04] E637   410         lda     rxdata3,x  ;Move next data byte (last byte first) from
1319 [05] E752   411         sta     nbuff,x   ;MDLC Rx buffer into Node-Node RAM buffer
412

131B [03] 5D     413         tstx          ;Test the X-Register to see if it is 0
414
131C [03] 26F8   415         bne     nextn     ;If X<>0, indicating the last byte has not
416                ;been transferred, then go get the next byte
417
131E [03] B636   418         lda     rxdata2   ;Load Source Address of receive message
1320 [04] B751   419         sta     nsource   ;into RAM location NSOURCE
420
1322 [03] CC1356 421         jmp     rxdone   ;Then go release the Rx buffer and finish up
422
1325 [03] 5F     423 funct:   clrx          ;Clear Index Register
424
1326 [03] B635   425         lda     rxdata1   ;Compare target address of received
1328 [02] A11A   426         cmp     #id1       ;message with ID#1
427
132A [03] 2712   428         beq     get_msg   ;If it matches, go get message
429
132C [03] 5C     430         incx          ;Otherwise increment X-register, and then...
431
132D [02] A120   432         cmp     #id2       ;Compare target address with ID#2
433
132F [03] 270D   434         beq     get_msg   ;If it matches, go get message
435
1331 [03] 5C     436         incx          ;Otherwise increment X-register, and then...
437
1332 [02] A15E   438         cmp     #id3       ;Compare target address with ID#3
439
1334 [03] 2708   440         beq     get_msg   ;If it matches, go get message
441
1336 [03] 5C     442         incx          ;Otherwise increment X-register, and then...
443
1337 [02] A1D3   444         cmp     #id4       ;Compare target address with ID#4
445
1339 [03] 2703   446         beq     get_msg   ;If it matches, go get message
447
133B [03] CC1356 448         jmp     rxdone   ;If not, go release Rx buffer,
449                ;discarding the received message
450

```

```

133E [03] B611 451 get_msg: lda mrsr ;Read received message byte count
452
1340 [02] A003 453 sub #03 ;Subtract $03 to account for message header
454
1342 [04] B74F 455 sta byte_cnt ;And store in temporary storage
456
1344 [04] EB5A 457 add fmsg1,x ;Add starting address of appropriate
458 ;RAM message buffer to get ending address
459
1346 [03] 4A 460 deca ;Decrement to get correct address index
461
1347 [04] B74E 462 sta buff_ndx ;and store in temporary storage
463
1349 [03] BE4F 464 next_byt: ldx byte_cnt ;Then load message byte count into X-reg.
465
134B [04] E636 466 lda rxdata2,x ;Read message data byte from MDLC Rx buffer
467
134D [03] BE4E 468 ldx buff_ndx ;Now load message buffer index into X-reg.
469
134F [04] F7 470 sta ,x ;and store data byte into RAM message buffer
471
1350 [05] 3A4E 472 dec buff_ndx ;Decrement message buffer index
473
1352 [05] 3A4F 474 dec byte_cnt ;Decrement received byte count
475

1354 [03] 26F3 476 bne next_byt ;If not zero, go get next byte
477
1356 [04] B711 478 rxdone: sta mrsr ;Write data in Acc. into MRSR to clear
479 ;RXMS interrupt
480
481 *** Tx Message Successful Interrupt Handling Routine ***
482
1358 [05] 070F0E 483 txint: brclr txms,mrsr,mdlcclr ;Check to see if a Tx interrupt
484 ;is pending - if not, go return
485 ;from the interrupt
486
135B [05] 3F41 487 clr rticnt ;Clear RTI Counter register
488
135D [05] 1908 489 bclr rtie,ctcsr ;Then disable the RTI interrupt
490
135F [02] A600 491 lda #00 ;Write $00 into the MTCR
1361 [04] B710 492 sta mtcrr ;to clear TXMS interrupt
493
1363 [05] 050F03 494 brclr rxms,mrsr,mdlcclr ;If Rx interrupt is not pending,
495 ;then finish up interrupt routine
496
1366 [03] CC1306 497 jmp rxint ;Otherwise, go check out the
498 ;pending receive interrupt
499
1369 [09] 80 500 mdlcclr: rti ;Return from interrupt
501
502 *****
503 * Core Timer Interrupt Service Routine *
504 *****
505
506 *****
507 * This routine handles the Real Time Interrupt (RTI) interrupts *
508 * enabled by the routine which initiates MDLC transmissions. The *
509 * RTI interrupts occur every 3.9ms. Following the initiation of *
510 * a transmission by the MDLC module, if the TXMS bit is not set *
511 * before 9 RTI interrupts are recorded (approximately 30ms), the *
512 * transmission will be aborted. This routine records and counts *
513 * the number of RTI interrupts, and when the count reaches 9, *
514 * the message transmission is aborted automatically by software. *
515 *****
516
1000 517 org ctimer
518
519 *** RTI Interrupt Service Routine ***
520
1000 [05] 1408 521 bset rtfc,ctcsr ;Clear RTI interrupt flag


```

```

522
1002 [03] B641 523      lda      rticnt      ;Read current RTI count
524
1004 [02] A108 525      cmp      #$08       ;Have 8 RTI's already occurred?
526
1006 [03] 2403 527      bhs      dumptx     ;If so, go abort the transmission
528
1008 [05] 3C41 529      inc      rticnt     ;If not, increment RTICNT
530
100A [09] 80   531      rti              ;Then return from interrupt
532
100B [05] 1C0E 533 dumptx: bset      txab,mcr ;Set the Tx Abort bit in the MCR
534
100D [05] 0C0EFD 535 doabort: brset    txab,mcr,doabort ;Wait until Tx Abort bit clears,
536 ;indicating that the transmission
537 ;has successfully been halted
538

1010 [05] 3F41 539      clr      rticnt     ;Then clear the RTICNT register
540
1012 [05] 1908 541      bclr     rtie,ctcsr ;Also disable RTI interrupt
542
1014 [09] 80   543      rti              ;Then return from interrupt
544
545 *****
546 *****          MC68HC705V8 Mask Option Register          *****
547 *****
548
3C00      549      org      mor
550
3C00      79   551      fcb      %01111001 ;Regulator enabled, set Vdd clamp,
552 ;MDLC power up enabled, LVR enabled,
553 ;STOP mode disabled, IRQ edge sensitive,
554 ;COP enabled
555
556 *****
557 *****          MC68HC705V8 Reset Vectors          *****
558 *****
559
3FF2      560      org      vectors
561
3FF2      1000 562      fdb      ctimer     ;Core Timer interrupt vector
3FF4      0000 563      fdb      none       ;SPI interrupt vector
3FF6      1300 564      fdb      mdlc       ;MDLC interrupt vector
3FF8      0000 565      fdb      none       ;16-Bit Timer interrupt vector
3FFA      0000 566      fdb      none       ;IRQ interrupt vector
3FFC      0000 567      fdb      none       ;SWI interrupt vector
3FFE      0D00 568      fdb      rom        ;Reset vector
569
570 *****
571 *****          End of the World          *****
572 *****

```

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

**How to reach us:**

**USA/EUROPE:** Motorola Literature Distribution;

P.O. Box 20912; Phoenix, Arizona 85036. 1-800-441-2247

**JAPAN:** Nippon Motorola Ltd.; Tatsumi-SPD-JLDC, Tosikatsu Otsuki,

6F Seibu-Butsuryu-Center, 3-14-2 Tatsuni Koto-Ku, Tokyo 135, Japan. 03-3521-8315

**HONG KONG:** Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park,

51 Ting Tok Road, Tai Po, N.T., Hong Kong. 852-26629298

**MFAX:** RMFAX0@email.sps.mot.com -TOUCHTONE (602) 244-6609

**INTERNET:** <http://Design-NET.com>



**MOTOROLA**



AN1224/D