PM7344

S/UNI-MPH

S/UNI-MPH with QDSX Reference Design

Issue 2: June 1997

CONTENTS

LIST OF FIGURES

LIST OF TABLES

REFERENCES

- American National Standards for Telecommunications, ANSI T1.107 (1988), "Digital Hierarchy - Formats Specifications"
- American National Standards for Telecommunications, ANSI T1.107 (Draft 1995), "Digital Hierarchy - Formats Specifications"
- American National Standards for Telecommunications, ANSI T1.231 (1993), "Digital Hierarchy - Layer 1 In-Service Digital Transmission Performance Monitoring"
- American National Standards for Telecommunications, ANSI T1.403 (1994), "Network-to-Customer Installation — DS1 Metallic Interface"
- Integrated Device Technology, Data Book (1995), "Specialized Memories, FIFOs & Modules"
- Microchip Technology Inc., Data Book (1994)
- Microchip Technology Inc., DS00566A (1993), "Using the Port B Interrupt-on-Change as an External Interrupt"
- Microchip Technology Inc., DS300271 (1995), "MPSIM User's Guide"
- Microchip Technology Inc., DS33014D (1995), "MPASM User's Guide"
- PMC-Sierra, Data Book (Issue 6), "PM7344 S/UNI-MPH Saturn Quad T1/E1 Multi-PHY User Network Interface", PMC-940873
- PMC-Sierra, Data Book (Issue 2) "PM4314 QDSX Quad T1/E1 Line Interface Device", PMC-950739
- PMC-Sierra, Reference Design (October 1995), "D3MX Module of PM4944 M13 Reference Design", PMC-951046
- PMC-Sierra, Reference Design (January 1996), "RCMP Reference Design", PMC-960148
- PMC-Sierra, "Saturn compatible interface for ATM PHY layer and ATM layer devices, Level 2", PMC-940102
- ATM Forum, "User Network Interface specification", Ver 3.1

Changes to This Issue

- The schematic diagram in Appendix C have been changed to reflect an error in Issue 1 of this document. The Issue 1 schematics indicated to tie ALE of the PM-7344 S/UNI-MPH and the PM-4314 QDSX to ground. The correct operation is to tie ALE high to VDD_D.

- The register initialization in Table 14 sets up the S/UNI-MPH for single rail mode, as does the initialization routine in the firmware in Appendix D. The schematic diagram also indicates a design for single rail mode. However, the description of Table 14 indicated that the initialization was for dual rail mode. That description has been changed to read single rail.

- The register initialization in Table 14 and the initialization firmware in Appendix D indicate to set register 0x04 in the S/UNI-MPH to a value of 0x48. It should read 0x08. Table 14 and the initialization firmware in Appendix D have been changed to reflect this.

OVERVIEW

This document describes a possible implementation of the S/UNI-MPH with QDSX reference design.

The S/UNI-MPH with QDSX Reference Design embodies PMC-Sierra's guidelines and suggestions for designing an interface for DSX-1 signals to a SCI-PHY™ Multi-PHY ATM cell bus using PMC-Sierra products (such as the PM7344 S/UNI-MPH and the PM4314 QDSX).

In addition to the PM7344 S/UNI-MPH and the PM4314 QDSX devices, the MPH Module incorporates an on-board microcontroller (Microchip PIC16C74) for providing the local maintenance functions — including termination of the ESF datalink in the DS1 overhead.

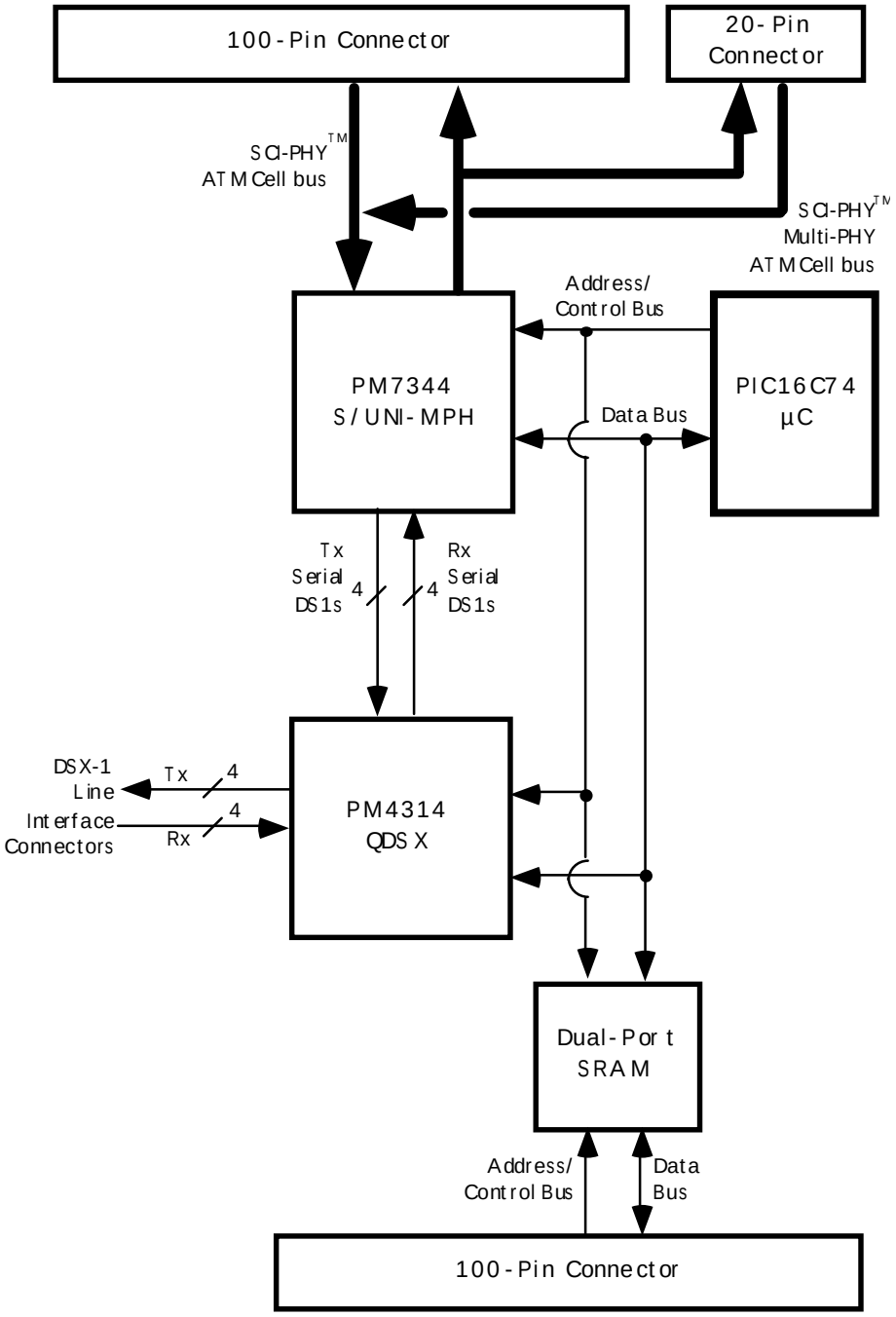
The DSX-1 line interface functions are provided by the QDSX and the DS-1 framing functions are provided by the S/UNI-MPH. The S/UNI-MPH also provides the ATM cell processing functions associated with the PHY layer, including the implementation of a SCI-PHY multi-PHY interface to the ATM layer device(s). The physical connection is provided through 100-pin and 20-pin connectors, pin-compatible with the RCMP reference design (PMC-960148).

The S/UNI-MPH Module communicates with a host system using a 100-pin connector. The pin-out of this connector is compatible for connection with the D3MX Module. The host does not have a direct connection to the microprocessor port of the S/UNI-MPH. Rather, a dual-port RAM, shared by the host and the local PIC16C74, is used to pass control and status information about the S/UNI-MPH to and from the host, where the actual register accesses to the S/UNI-MPH are performed by the PIC16C74.

The advantage to this architecture is that the host system is not burdened by the low-level monitoring and control functions.

FUNCTIONAL DESCRIPTION

Block Diagram



PM7344 S/UNI-MPH

The PM7344 SATURN Quad T1/E1 Multi-PHY User Network Interface (S/UNI-MPH) is a monolithic integrated circuit that implements the T1/E1 processing and ATM mapping functions for four 1.544 Mbit/s or 2.048 Mbit/s ATM User Network Interfaces. It can also be used in conjunction with external framing devices, to implement ATM user network interfaces for other bit rates. For example, a quad J2 (6.312 Mbit/s) interface can be realized with four external J2 framers and a single S/UNI-MPH. It is fully compliant with both ANSI and ITU requirements and ATM Forum UNI specifications. The S/UNI-MPH is software configurable, allowing feature selection without changes to external wiring.

On the receive side, when configured for T1 processing, the S/UNI-MPH recovers clock and data and can be configured to frame to either of the common DS-1 signal formats; SF or ESF. Clock recovery may also be bypassed. The S/UNI-MPH also supports detection of various alarm conditions such as loss of signal, pulse density violation, red alarm, yellow alarm, and AIS alarm. The S/UNI-MPH detects and indicates the presence of yellow and AIS patterns and also integrates yellow, red, and AIS alarms as per industry specifications.

Performance monitoring with accumulation of CRC-6 errors, framing bit errors, line code violations, and loss of frame events is provided. The S/UNI-MPH also detects the presence of in-band loopback codes, ESF bit oriented codes, and detects and terminates HDLC messages on the ESF data link.

On the receive side, when configured for E1 processing, the S/UNI-MPH recovers clock and data and can be configured to frame to a basic G.704 2048 kbit/s signal or also frame to the signaling multiframe alignment signal and the CRC multiframe alignment signal. Clock recovery may also be bypassed.

The S/UNI-MPH also supports detection of various alarm conditions such as loss of signal, loss of frame, loss of signaling multiframe, loss of CRC multiframe, and reception of remote alarm signal, remote multiframe alarm signal, alarm indication signal, and timeslot 16 alarm indication signal. The S/UNI-MPH detects and indicates the presence of remote alarm and AIS patterns and also integrates red and AIS alarms as per industry specifications.

Performance monitoring with accumulation of CRC-4 errors, far end block errors, framing bit errors, and line code violation is provided. The S/UNI-MPH also detects and terminates HDLC messages on a data link. The data link may be extracted from timeslot 16 or may be extracted from the national bits.

For both T1 and E1 configurations, the S/UNI-MPH interprets the received frame alignment and extracts the transmission format payload which carries the received ATM cell payload.

The S/UNI-MPH frames to the ATM payload using cell delineation. HCS error correction is optionally provided. Idle/unassigned cells may be dropped according to a programmable filter. Cells are also dropped upon detection of an uncorrectable header check sequence error. The ATM cell payloads are descrambled.

Valid, assigned cells are written to a four cell FIFO buffer. These cells are read from the FIFO using a synchronous 8 bit wide datapath interface with a cell-based handshake. Counts of received ATM cell headers that are errored and uncorrectable, those that are errored and correctable and all passed cells are accumulated independently for performance monitoring purposes. A multi-PHY interface allows the four receive FIFOs (one for each T1 or E1 port) to be serviced via a single 8 bit wide bus, address, and control lines.

On the transmit side, when configured for T1 processing, the S/UNI-MPH generates framing for SF and ESF DS1 formats. The S/UNI-MPH can also generate in-band loopback codes, ESF bit oriented codes, and transmit HDLC messages on the ESF data link.

On the transmit side, when configured for E1 processing, the S/UNI-MPH generates framing for a basic G.704 2048 kbit/s signal. The signaling multiframe alignment signal may be optionally inserted and the CRC multiframe structure may be optionally inserted. HDLC messages on a data link can be transmitted. The data link may be inserted into timeslot 16 or may be inserted into the national bits.

For both T1 and E1 configurations, the S/UNI-MPH generates the transmitted frame and inserts the transmit ATM cell payload into the transmission format payload appropriately.

ATM cells are written to an internal programmable-length 4-cell FIFO using a synchronous 8 bit wide datapath interface. Idle/unassigned cells are automatically inserted when the internal FIFO contains less than one cell. The S/UNI-MPH generates of the header check sequence and scrambles the payload of the ATM cells. Each of these transmit ATM cell processing functions can be enabled or bypassed. A multi-PHY interface allows the four transmit FIFOs (one for each T1 or E1 port) to be serviced via a single 8 bit wide bus, address, and control lines.

The S/UNI-MPH is configured, controlled and monitored via a generic 8-bit microprocessor bus interface. The S/UNI-MPH also provides a standard 5 signal P1149.1 JTAG test port for boundary scan board test purposes.

For a complete description of the S/UNI-MPH, please refer to PMC-Sierra's PM7344 databook (document number PMC-940873).

PM4314 QDSX

The PM4314 QDSX Quad T1/E1 Line Interface Device is a monolithic integrated circuit that supports DSX-1 and CEPT E1 compatible transmit and receive interfaces for four 1.544 Mbit/s or 2.048 Mbit/s data streams.

In the receive direction, clock and data are recovered from the received signal using a digital phase-locked loop that provides excellent high frequency jitter accommodation. The recovered data is decoded using B8ZS, HDB3, or AMI line code rules. Loss of signal and line code violations are detected. Line code violations are accumulated in internal counters.

In the transmit direction, each quadrant of the QDSX can transmit either a DS-1/E1 stream encoded using B8ZS, HDB3, or AMI line code rules. The digital data is converted to high drive, dual rail RZ pulses that drive the DSX-1/E1 interface through a coupling transformer. The shape of the pulses is programmable to ensure that the DSX-1/E1 pulse template is met after the signal is passed through different cable lengths or types. Driver performance monitoring is provided and can generate interrupts upon driver failure.

A jitter attenuation function comprised of a digital phase-locked loop and data FIFO is available for use in either the transmit or receive path of each channel.

Diagnostic loopback is provided and the loopback may be invoked past the analog transmit outputs using the driver performance monitors or invoked prior to the conversion to analog. Line loopback with jitter attenuation is provided and may be enabled for automatic operation based on detected inband loopback codes.

The QDSX detects framed or unframed inband loopback code sequences from the received input pulses. Any arbitrary code from three to eight bits in length can be declared to be the activate and deactivate codes by writing to configuration registers. The inband loopback code detector can optionally be moved to the transmit side where it detects inband loopback codes in the unipolar input transmit data stream. For framed inband loopback code sequences, it is expected that the framing bit overwrites the inband loopback code bit.

The QDSX may insert unframed inband loopback code sequences (programmable codes from three to eight bits in length) into the transmit or receive path of each channel. The QDSX detects framed or unframed inband loopback code sequences in either the transmit or receive path of each channel. Two arbitrary codes can be searched for simultaneously, each programmable between three to eight bits in length.

The QDSX may insert an unframed 2^{15}-1 O.151 compatible pseudo-random bit sequence into the transmit or receive path of each channel. The QDSX can detect an

unframed 2^{15}-1 O.151 compatible pseudo-random bit sequence in either the transmit or receive path of each channel.

The QDSX operates in conjunction with external line coupling transformers, resistors, and capacitors. An external crystal oscillator may be used for high speed timing generation. The QDSX is configured, controlled, and monitored using registers that are accessed via a generic microprocessor interface.

Internal high speed timing for all quadrants of the QDSX is provided by a common 37.056 MHz or 49.152 MHz master clock. This master clock rate is required for applications where QDSX provides jitter attenuation.

For a complete description of the QDSX, please refer to PMC-Sierra's QDSX Data Book (document number PMC-950857).

Dual-Port RAM

This reference design uses a high-speed 2K by 8-bit dual-port static RAM with internal interrupt logic (for inter-processor communications). Many manufacturers (such as Integrated Device Technology, and Cypress Semiconductors) produce pin-compatible versions of this device.

The dual-port RAM has two ports with separate control, address and data pins that permit independent, asynchronous access for both reads and writes to any location in memory.

Memory Map

Tables 1 to 4 define the memory map for the dual-port RAM. The memory map is organized to provide a generic flexible interface to the S/UNI-MPH physical layer control and status functionality.

Table 1. Dual-port RAM Memory Map

Ram Address (hex)	Function (length)
000	RFDL receive buffer and status, quadrant 1 (80h)
080	RFDL receive buffer and status, quadrant 2 (80h)
100	RFDL receive buffer and status, quadrant 3 (80h)
180	RFDL receive buffer and status, quadrant 4 (80h)
200	XFDL transmit buffer and status, quadrant 1 (80h)
280	XFDL transmit buffer and status, quadrant 2 (80h)
300	XFDL transmit buffer and status, quadrant 3 (80h)

Ram Address (hex)	Function (length)
380	XFDL transmit buffer and status, quadrant 4 (80h)
400	PMON shadow registers, quadrant 1 (13h)
413	PMON shadow registers, quadrant 2 (13h)
426	PMON shadow registers, quadrant 3 (13h)
439	PMON shadow registers, quadrant 4 (13h)
496	New signaling value (HOST->PIC)
497	Quadrant to affect with signaling command (HOST->PIC)
498	DS0 channel to affect with signaling command (HOST->PIC)
499	New idle code (HOST->PIC)
7DE	Contains firmware revision #
7DF	Contains firmware version #
7E0	Specifies BOC to transmit (HOST->PIC)
7F0	Interrupting quadrant (PIC->HOST)
7F1	Latest received BOC (PIC->HOST)
7F8	Current RFDL interrupting quadrant (PIC->HOST)
7F9	Current XFDL interrupting quadrant (PIC->HOST)
7FA	Quadrant on which to transmit BOC (HOST->PIC)
7FB	Register R/W data (HOST<->PIC)
7FC	Register R/W address LSB (HOST<->PIC)
7FD	Register R/W address MSB (HOST<->PIC)
7FE	Host Mailbox (PIC->HOST)
7FF	PIC16C74 Mailbox (HOST->PIC)

Table 2. RFDL Memory block of 80H bytes

Relative Address (hex)	Function
00 - 7D	Packet Data (maximum 126 bytes)
7E	Channel Status
7F	Packet Length

Table 3. XFDL Memory block of 80H bytes

Relative Address (hex)	Function
00 - 7E	Packet Data (maximum 127 bytes)
7F	Packet Length

Table 4. PMON Memory block of 13H bytes

Relative Address (hex)	Function
0	Frame error count
1	Far end block error count LSB
2	Far end block error count MSB
3	CRC count LSB
4	CRC count MSB
5	Line code violation count LSB
6	Line code violation count MSB
7	RXCP uncorrectable HCS error count LSB
8	RXCP uncorrectable HCS error count MSB
9	RXCP correctable HCS error count LSB
A	RXCP correctable HCS error count MSB
B	RXCP idle/unassigned cell count LSB
C	RXCP idle/unassigned cell count MSB
D	RXCP receive cell count LSB
E	RXCP receive cell count MSB
F	TXCP transmit cell count LSB
10	TXCP transmit cell count MSB
11	Reserved
12	Reserved

Mailbox Communication

Each port has one address location assigned as a "mailbox." When a port writes to its mailbox, the other port will be interrupted. This interrupt is cleared when the mailbox is read by the interrupted port. These mailboxes can be used as a control and status channel. By defining functions for certain values passed in the mailbox, each port can initiate actions of the other port. Additionally, a port can pass high-priority status information through the mailbox.

HOST-TO-PIC16C74 COMMUNICATION

The host sends control commands through the microcontroller's mailbox (location 7FF). Table 5 shows the mailbox codes for host-to-PIC16C74 messaging. Following the table is a description of each command and further processing (if necessary) required of the host.

Table 5. Host-to-PIC16C74 Mailbox Codes

Function	Code (hex)
Read S/UNI-MPH register	01
Write S/UNI-MPH register	02
Start FDL packet transmission	03
Start BOC transmission	04
Stop BOC transmission	05
Reserved	06
Reserved	07
Reserved	08
Reserved	09
Reserved	0A
Reserved	0B
Line Loopback Activate	0C
Line Loopback Deactivate	0D
Payload Loopback Activate	0E
Payload Loopback Deactivate	0F
Reserved	10
Reserved	11
Diagnostic Loopback Activate	12
Diagnostic Loopback Deactivate	13
Start AIS transmission	14
Stop AIS transmission	15
Read QDSX register	16
Write QDSX register	17

READ S/UNI-MPH REGISTER (01)

This function is useful for diagnostic purposes to read the value of a S/UNI-MPH register. To read a value from a specific S/UNI-MPH register perform the following steps:

1. Write the LSB of the address of the register to RAM location 7FC.
2. Write the MSB of the address of the register to RAM location 7FD.
3. Write the mailbox command 01 to the PIC16C74 Mailbox (location 7FF).
4. When the Requested S/UNI-MPH Register I/O Complete code (FF) is received in the Host Mailbox (7FE), the read register value will be available in RAM location 7FB.

WRITE S/UNI-MPH REGISTER (02)

This function is useful for diagnostic purposes to write a S/UNI-MPH register with a value. To write a value from a specific S/UNI-MPH register perform the following steps:

1. Write the LSB of the address of the register to RAM location 7FC.
2. Write the MSB of the address of the register to RAM location 7FD.
3. Write the data byte value to RAM location 7FB.
4. Write the mailbox command 02 to the PIC16C74 Mailbox (location 7FF).

START FDL PACKET TRANSMISSION (03)

After FDL packet data (up to 127 bytes) is placed in a quadrant's XFDL transmit buffer (in dual-port RAM), this command can be used to initiate transmission. Once initiated the packet transmission will be handled by the PIC16C74. To transmit an FDL packet perform the following steps:

1. Write the packet data to the quadrant's XFDL transmit buffer.
2. Write the packet's length to the Packet Length byte of the quadrant's XFDL transmit buffer.
3. Write the mailbox command 03 to the PIC16C74 Mailbox (location 7FF).

START BIT-ORIENTED CODE TRANSMISSION (04)

To send a bit-oriented code on a quadrant's facility data link perform the following steps:

1. Write the 6-bit BOC (least significant bit transmitted first) to RAM location 7E0.
2. Write the quadrant number (0-3) to RAM location 7FA.
3. Write the mailbox command 04 to the PIC16C74 Mailbox (location 7FF).

The bit-oriented code will be transmitted until stopped using the Stop Bit-Oriented Code Transmission command.

STOP BIT-ORIENTED CODE TRANSMISSION (05)

To stop the transmission of a bit-oriented code on a quadrant's facility data link perform the following steps:

1. Write the quadrant number (0-3) to RAM location 7FA.
2. Write the mailbox command 05 to the PIC16C74 mailbox (location 7FF).

LINE LOOPBACK ACTIVATE (0C)

To activate line loopback perform the following steps:

1. Write the quadrant number (0-3) to RAM location 7FA.
2. Write the mailbox command 0C to the PIC16C74 mailbox (location 7FF).

LINE LOOPBACK DEACTIVATE (0D)

To deactivate line loopback perform the following steps:

1. Write the quadrant number (0-3) to RAM location 7FA.
2. Write the mailbox command 0D to the PIC16C74 mailbox (location 7FF).

PAYLOAD LOOPBACK ACTIVATE (0E)

To activate payload loopback perform the following steps:

1. Write the quadrant number (0-3) to RAM location 7FA.
2. Write the mailbox command 0E to the PIC16C74 mailbox (location 7FF).

PAYLOAD LOOPBACK DEACTIVATE (0F)

To deactivate payload loopback perform the following steps:

1. Write the quadrant number (0-3) to RAM location 7FA.
2. Write the mailbox command 0F to the PIC16C74 mailbox (location 7FF).

DIAGNOSTIC LOOPBACK ACTIVATE (12)

To activate diagnostic loopback perform the following steps:

1. Write the quadrant number (0-3) to RAM location 7FA.
2. Write the mailbox command 12 to the PIC16C74 mailbox (location 7FF).

DIAGNOSTIC LOOPBACK DEACTIVATE (13)

To deactivate diagnostic loopback perform the following steps:

1. Write the quadrant number (0-3) to RAM location 7FA.
2. Write the mailbox command 13 to the PIC16C74 mailbox (location 7FF).

START AIS TRANSMISSION (14)

To start AIS transmission perform the following steps:

1. Write the quadrant number (0-3) to RAM location 7FA.
2. Write the mailbox command 14 to the PIC16C74 mailbox (location 7FF).

STOP AIS TRANSMISSION (15)

To stop AIS transmission perform the following steps:

1. Write the quadrant number (0-3) to RAM location 7FA.

2. Write the mailbox command 15 to the PIC16C74 mailbox (location 7FF).

READ QDSX REGISTER (16)

This function is useful for diagnostic purposes to read the value of a QDSX register. To read a value from a specific QDSX register perform the following steps:

1. Write the LSB of the address of the register to RAM location 7FC.
2. Write the MSB of the address of the register to RAM location 7FD.
3. Write the mailbox command 16 to the PIC16C74 Mailbox (location 7FF).
4. When the Requested QDSX Register I/O Complete code (FF) is received in the Host Mailbox (7FE), the read register value will be available in RAM location 7FB.

WRITE QDSX REGISTER (17)

This function is useful for diagnostic purposes to write a QDSX register with a value. To write a value from a specific QDSX register perform the following steps:

1. Write the LSB of the address of the register to RAM location 7FC.
2. Write the MSB of the address of the register to RAM location 7FD.
3. Write the data byte value to RAM location 7FB.
4. Write the mailbox command 17 to the PIC16C74 Mailbox (location 7FF).

PIC16C74 TO HOST COMMUNICATION

The microcontroller sends alarm and status information to the host through the host's mailbox (location 7FE). Table 6 shows the mailbox codes for microcontroller-to-host messaging. Following the table is a description of each message and further processing (if necessary) required of the host.

Because the PIC16C74 takes a finite time to process information, it will only indicate one interrupt at a time. The host must process the mailbox before the next mailbox code is written by the PIC16C74. With the current code, the shortest delay between consecutive mailbox interrupts occurs when multiple channels indicate a simple alarm (e.g. RED). Therefore, the host must process all mailbox interrupts within approximately 50 µs, otherwise some information will be lost (overwritten).

Table 6. PIC16C74-to-Host Mailbox Codes

Code (hex)	Meaning
01	Reserved
02	Reserved
03	Reserved
04	Reserved
05	Reserved
06	AIS asserted
07	AIS cleared
08	YELLOW alarm asserted
09	YELLOW alarm cleared
0A	Reserved
0B	Reserved
0C	Reserved
0D	Reserved
0E	Valid BOC detected
0F	Return to idle BOC detected
10	New FDL packet received (Q1)
11	New FDL packet received (Q2)
12	New FDL packet received (Q3)
13	New FDL packet received (Q4)
14	In-Band Line-Loopback-Activate Detected
15	In-Band Line-Loopback-Deactivate Detected
20	FDL packet has been successfully transmitted (Q1)
21	FDL packet has been successfully transmitted (Q2)
22	FDL packet has been successfully transmitted (Q3)

Code (hex)	Meaning
23	FDL packet has been successfully transmitted (Q4)
25	Loss of clock activity asserted
26	Loss of clock activity cleared
27	Loss of cell delineation asserted
28	Loss of cell delineation cleared
29	Transmit FIFO overrun
2A	Receive FIFO overrun
2B	Change of cell alignment
FF	Requested S/UNI-MPH or QDSX register I/O completed
80	PMON statistics updated

AIS ASSERTED (06)

This message is sent when the S/UNI-MPH detects that an AIS is being received. The originating quadrant can be determined by reading RAM location 7F0. The AIS detection will take 1.5 seconds to integrate in the presence of a continuously received AIS pattern.

AIS CLEARED (07)

This message is sent when the S/UNI-MPH detects that AIS is no longer being received. The originating quadrant can be determined by reading RAM location 7F0. The AIS clear will take 16.8 seconds to integrate in the presence of a continuously received framed pattern.

YELLOW ALARM ASSERTED (08)

This message is sent when the S/UNI-MPH detects that an ESF YELLOW alarm is being received. The originating quadrant can be determined by reading RAM location 7F0. The YELLOW detection will take 425 ms to integrate in the presence of a continuously received YELLOW pattern.

YELLOW ALARM CLEARED (09)

This message is sent when the S/UNI-MPH detects that the YELLOW alarm is no longer being received. The originating quadrant can be determined by reading RAM location 7F0. The YELLOW clear will take 425 ms to integrate in the presence of continuously received frames not containing the YELLOW pattern.

VALID BOC DETECTED (0E)

This message is sent when the S/UNI-MPH detects a valid BOC on the received FDL. The originating quadrant can be determined by reading RAM location 7F0. The received BOC can be read from RAM location 7F1. A BOC is considered valid if it is repeated at least 8 times in a window of 10 consecutively received BOCs.

If the BOC matches a standardized loopback activate or deactivate command, the PIC16C74 will automatically respond by putting the quadrant into the appropriate mode. Note that as specified in ANSI T1.403, a quadrant will not be put into the Line Loopback mode until the Line-Loopback-Activate BOC has been removed (to ensure that the loopback command is not looped back itself).

For a list of the standardized loopback commands, see the Extended Superframe Loopbacks subsection of the functional description for the firmware.

IDLE BOC DETECTED (0F)

This message is sent when the S/UNI-MPH decides that no valid BOC is being received on the FDL. The originating quadrant can be determined by reading RAM location 7F0.

NEW HDLC PACKET RECEIVED (10, 11, 12, OR 13)

This message is sent when the S/UNI-MPH is finished receiving a new HDLC packet. A separate code is assigned to each originating quadrant. The packet length, channel status and packet data can be read from the quadrant's RFDL receive buffer.

FINISHED TRANSMITTING HDLC PACKET (20, 21, 22, OR 23)

This message is sent when the S/UNI-MPH has finished transmitting an HDLC packet. This allows the host to know when it may construct another packet for transmission. A separate code is assigned to each originating quadrant.

IN-BAND LINE-LOOPBACK-ACTIVATE DETECTED (14)

This message is sent when the S/UNI-MPH detects a standardized in-band Line-Loopback-Activate request. The originating quadrant can be determined by reading RAM location 7F0. The PIC16C74 will automatically respond to the request, putting the quadrant into a Line Loopback mode.

IN-BAND LINE-LOOPBACK-DEACTIVATE DETECTED (15)

This message is sent when the S/UNI-MPH detects a standardized in-band Line-Loopback-Deactivate request. The originating quadrant can be determined by reading RAM location 7F0. The PIC16C74 will automatically respond to the request, taking the quadrant out of the Line Loopback mode.

LOSS OF CLOCK ACTIVITY ASSERTED (25)

This message is sent when the S/UNI-MPH indicates a loss of clock activity on any of the XCLKA, RCLKIxA, TCLKIA, TFCLKA, RFCLKA inputs. The PIC microcontroller polls the clock activity register in the S/UNI-MPH (register 0EH) to determine clock status. The message is only sent once.

LOSS OF CLOCK ACTIVITY CLEARED (26)

This message is sent when the S/UNI-MPH indicates clock activity has been restored to all of the XCLKA, RCLKIxA, TCLKIA, TFCLKA, RFCLKA inputs. The PIC microcontroller polls the clock activity register in the S/UNI-MPH (register 0EH) to determine clock status.

LOSS OF CELL DELINEATION ASSERTED (27)

This message is sent when the S/UNI-MPH detects a loss of cell delineation. (The receive cell processor of the S/UNI-MPH has been out of cell delineation for the number of cell periods determined by register 84H of the S/UNI-MPH). The originating quadrant can be determined by reading RAM location 7F0.

LOSS OF CELL DELINEATION CLEARED (28)

This message is sent when the S/UNI-MPH detects that cell delineation has been regained. (The receive cell processor of the S/UNI-MPH has been in cell delineation for the number of cell periods determined by register 84H of the S/UNI-MPH). The originating quadrant can be determined by reading RAM location 7F0.

TRANSMIT FIFO OVERRUN (29)

This message is sent when the S/UNI-MPH detects that the transmit cell FIFO has overrun. The originating quadrant can be determined by reading RAM location 7F0.

RECEIVE FIFO OVERRUN (2A)

This message is sent when the S/UNI-MPH detects that the receive cell FIFO has overrun. The originating quadrant can be determined by reading RAM location 7F0.

CHANGE OF CELL ALIGNMENT (2B)

This message is sent when the S/UNI-MPH detects a change of cell alignment. The originating quadrant can be determined by reading RAM location 7F0.

PMON STATISTICS UPDATED (80)

This message is sent following the update of PMON statistics in shadow RAM.

REG TRANSFER COMPLETE (FF)

This message is sent to indicate that the requested register read of the S/UNI-MPH or QDSX has been completed.

PIC16C74 Microcontroller

The Microchip PIC16C74 is a low-cost, high-performance, CMOS, fully-static EPROM-based 8-bit microcontroller. It employs a Harvard RISC-like architecture with 14-bit instruction words. Its two stage instruction pipeline allows most instructions to be executed in a single cycle. The PIC16C74 has enhanced core features, an eight-level deep stack, and multiple internal and external interrupt sources.

The PIC16C74 has 192 bytes of on-chip user RAM and a 4k by 14-bit EPROM for program memory.

It has 33 I/O pins, each of which can source/sink 25mA.

The PIC16C74 has multiple timers which can be configured to generate internal interrupts at variable intervals.

The PIC16C74 also has a number of peripheral features (A/D converters, capture/compare/PWM modules, and two serial ports) which are not used in this reference design.

In the S/UNI-MPH Module, the PIC16C74 is devoted to the local maintenance, performance monitoring, failure monitoring and diagnostic functions related to the four DS1 physical layers terminated by the PM7344 S/UNI-MPH and PM4314 QDSX.

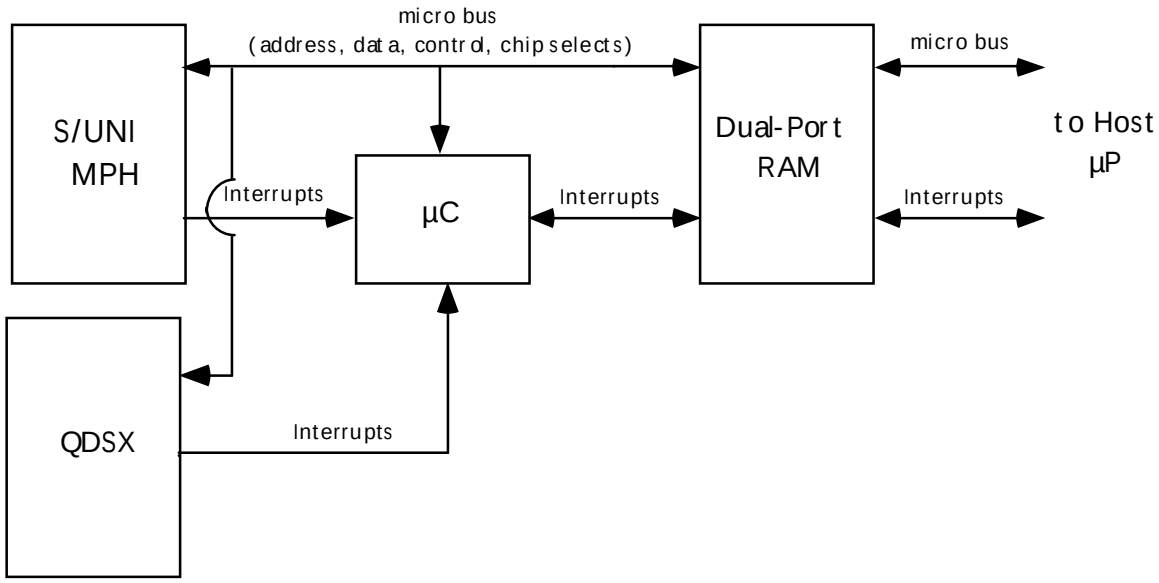
Firmware

Example firmware for the PIC16C74 is included in Appendix D. The firmware was developed in Assembly language using Microchip's PC-hosted symbolic assembler, MPASM (v1.30.01), available free from Microchip's bulletin board (MCHIPBBS on Compuserve) or Web site (http://www.microchip.com). The firmware was simulated using MPSIM (v5.20), also available free from Microchip. The PIC16C74 can be programmed on universal programmers from Sprint and Data I/O. Additionally, Microchip sells low cost programmers for this device.

External Memory Accesses

The PIC16C74 does not have a microprocessor bus suitable to access registers and memory within the S/UNI-MPH and dual-port RAM. Therefore I/O pins on the PIC16C74 are defined to act as the address bus, data bus, chip selects, RDB, WRB and other signals necessary. Since the PIC16C74 is only accessing three external devices, generates the chip selects instead of relying on external decode logic. Figure 1 shows a block diagram of the connections.

Figure 1. Block Diagram of PIC16C74 Connections



The firmware in Appendix D includes subroutines that operate dedicated I/O pins as the micro bus during external accesses. Since the interrupt service routines will often alternate between accessing the S/UNI-MPH, QDSX, and the dual-port RAM, it saves instruction cycles to have separate subroutines dedicated to each, where the address would be taken from different pre-defined registers. There are six general access routines: RD_MPH, RD_RAM, RD_QDSX, WR_MPH, WR_RAM, and WR_QDSX. All four routines share a common Data register so that data transfers could be optimized.

Initialization

The firmware initializes the S/UNI-MPH to enable the appropriate interrupts, set the framing format, configure the interfaces, set the timing options, and initialize the HDLC serial controllers and the signaling blocks. The S/UNI-MPH is initialized for Extended Superframe Format (ESF) in all quadrants.

The QDSX is initialized in much the same way as the S/UNI-MPH. Many of the features of the QDSX are not enabled, however, as they are already available in the S/UNI-MPH.

The dual-port RAM is initialized with the version and revision number of the firmware. The microcontroller-side mailbox is read to clear any spurious interrupt.

Finally, the initialization routine enables the interrupts within the PIC16C74, and falls into a main loop.

Dual-Port Mailboxes

The dual-port RAM contains two mailbox registers, one in each direction, for passing data between the two ports. When the micro bus on one port writes a data byte to its mailbox address, the other port is interrupted. Using these mailboxes, a proprietary messaging scheme can be arranged. These interrupts from the dual-port RAM are processed by the PIC16C74.

Datalink Service Routines (RFDL/XFDL)

The ESF facility datalink is a 4 kbit/s channel provided for three functions: a remote alarm indication (YELLOW Alarm), a bit-oriented code FEAC channel, and an HDLC datalink. The YELLOW Alarm and FEAC processing are explained in the Maintenance Functions subsection.

The PM7344 S/UNI-MPH provides detection and generation of the HDLC packet overhead, by using its internal RFDL and XFDL functional blocks. The RFDL and XFDL functional blocks generate interrupt indications on the common INTB output.

The procedures for handling the interrupts from the RFDL and XFDL blocks is explained in the Operations section of the S/UNI-MPH data book. The firmware implements these procedures providing routines for initiating the transmission of packets constructed by the host (using the Start FDL Transmission mailbox command) as well as interrupting the host when a complete packet is received (indicated with the New FDL Packet Received mailbox messages).

Additionally, the firmware automatically constructs and transmits ANSI T1.403 one second performance report packets based on performance monitoring data it has collected. If the firmware is already transmitting a packet, the performance report will be transmitted after the current packet is finished. The performance report functionality is further explained in the Maintenance Functions subsection.

Maintenance Functions

There are a number of maintenance functions specified in ANSI T1.107, T1.231, and T1.403. These include YELLOW Alarm, Alarm Indication Signal (AIS), RED Alarm, Loopbacks, and Performance Monitoring (including Performance Reports).

YELLOW ALARM

A DS1 YELLOW Alarm must be sent to the remote DS1 equipment in response to a RED Alarm state. This is performed automatically by the S/UNI-MPH.

The firmware automatically interrupts the host when a YELLOW Alarm is detected in a receive channel.

ALARM INDICATION SIGNAL (AIS)

The AIS must be sent upon loss of originating signal, or when any action is taken that would cause service disruption (such as loopbacks).

The firmware will not automatically transmit AIS in response to any condition. However, it will interrupt the host when an AIS is detected in a receive channel.

LOSS OF CELL DELINEATION

The PIC16C74 firmware alerts the host whenever a loss of cell delineation is indicated by the S/UNI-MPH. Loss of cell delineation is determined by counting the number of cell periods for which the receive cell processor is out of delineation, and comparing to the RXCP LCD Count Threshold, register 84H.

LOOPBACKS

Loopbacks are used by carriers and customers as a maintenance tool to aid problem resolution. The carrier uses loopbacks for trouble isolation, and the customer uses them for CI-to-CI testing.

There are two loopbacks defined by ANSI T1.403: Line and Payload. In SF, only Line Loopback is applicable, but in ESF, both loopbacks apply.

Both Line and Payload loopbacks are supported in the S/UNI-MPH.

Superframe (SF) Loopbacks

In SF, loopbacks are controlled by in-band signaling. There are two in-band codes defined: Loopback Activate and Loopback Deactivate. The Loopback Activate code is a framed "00001..." repeating pattern. The Loopback Deactivate code is a framed "001..." repeating pattern. The IBCD functional block in the S/UNI-MPH can be configured (during initialization) to look for these patterns and interrupt when one is detected. In response to these interrupts, the firmware enables or disables Line Loopback.

Extended Superframe (ESF) Loopbacks

In ESF, loopbacks are controlled by bit-oriented codes sent over the facility datalink. (Note some provisions are made for framed or unframed in-band codes to control ESF Line Loopbacks.) There are six bit-oriented codes defined relating to loopbacks:

- Line-Loopback-Activate: 00001110 11111111
- Line-Loopback-Deactivate: 00111000 11111111
- Payload-Loopback-Activate: 00010100 11111111
- Payload-Loopback-Deactivate: 00110010 11111111
- Universal-Loopback-Deactivate: 00100100 11111111
- Loopback-Retention: 00101010 11111111

The RBOC TSB in the S/UNI-MPH can detect any valid bit-oriented code, generating an interrupt when one is detected or removed.

The Line Loopback should not be activated until the Line-Loopback-Activate code is removed, so that the activate code is not looped back to the network interface (NI). Payload Loopback can be activated immediately on detection of the Payload-Loopback-Activate code since the datalink is regenerated — hence the bit-oriented codes will not be looped back.

The deactivation for both loopbacks can be accomplished in several ways:

- upon detection of the Universal-Loopback-Deactivate code
- upon detection of AIS
- upon the receipt of two occurrences of the one-second performance report messages separated by uninterrupted IDLE code (this is not implemented in firmware).

Additionally, the Loopback-Retention code is optionally sent in framed test signals during loopbacks as a positive confirmation of the presence of a Line Loopback.

The firmware responds to the loopback codes as required. To transmit loopback codes the host should use the Start BOC Transmission mailbox command.

Performance Monitoring

All the DS1 performance monitoring parameters required by ANSI T1.231 are available within registers of the S/UNI-MPH.

Therefore, with a timer interrupt setting the accumulation interval, the PIC16C74 can transfer all the performance monitoring indicators and parameters to known memory locations in the dual-port RAM. The host system may then process these parameters. In addition to the performance report, the firmware collects cell performance counters from the S/UNI-MPH and copies them to dual port RAM for use by the host.

Additionally, ANSI T1.403 specifies that a performance report should be sent once each second using a bit-assigned structure within a HDLC packet transmitted over the facility datalink.

Therefore, a timer interrupt is enabled within the microcontroller to interrupt once each second. The service routine for the timer interrupt collects the required performance parameters and constructs performance report packets which are then automatically transmitted.

The PIC16C74 indicates when it is finished its one second functions by sending the PMON Statistics Updated mailbox message.

The performance report packet is 15 octets long, including the opening and closing HDLC Flags. The format is shown in Table 7.

Table 7. Performance Report Packet Format

Octet No.	Octet Contents	Interpretation
1	01111110	Opening HDLC Flag
2	00111000	From CI: SAPI=14, C/R=0, EA=0
	00111010	From NI: SAPI=14, C/R=1, EA=0
3	00000001	TEI=0, EA=1
4	00000011	Unacknowledged Frame
5, 6	Variable	Data for latest second (T')
7, 8	Variable	Data for previous second (T'-1)
9, 10	Variable	Data for earlier second (T'-2)
11, 12	Variable	Data for earlier second (T'-3)
13, 14	Variable	CRC-16 Frame Check Sequence (FCS)
15	01111110	Closing HDLC Flag

The format for each second's two octets of data (transmitted right to left) is shown in Table 8.

Table 8. Performance Report Data Byte Structure

G3	LV	G4	U1	U2	G5	SL	G6
FE	SE	LB	G1	R	G2	Nm	NI

Where G1=1 indicates that exactly one CRC Error Event occurred during the interval,
G2=1 indicates that between two and five (inclusive) CRC Error Events occurred during the interval,
G3=1 indicates that between six and ten (inclusive) CRC Error Events occurred during the interval,
G4=1 indicates that between 11 and 100 (inclusive) CRC Error Events occurred during the interval,
G5=1 indicates that between 101 and 319 (inclusive) CRC Error Events occurred during the interval,
G6=1 indicates that more than 319 CRC Error Events occurred during the interval,
SE=1 indicates that one or more Severely Errored Framing Events occurred during the interval,
FE=1 indicates that one or more Frame Sync Bit Error Events occurred during the interval,
LV=1 indicates that one or more Line Code Violation Events occurred during the interval,
SL=1 indicates that one or more Slip Events occurred during the interval. This bit is always 0 for the S/UNI-MPH.
LB=1 indicates that Payload Loopback is active,
U1, U2=0 under study for synchronization,
R=0 Reserved
NmNI One-second report modulo-4 counter

IMPLEMENTATION DESCRIPTION

The schematic diagram of the S/UNI-MPH Module is contained in Appendix C. This section explains that schematic diagram.

Hardware

The S/UNI-MPH Module schematic shows four main functional blocks: a PIC16C74 microcontroller, dual-port RAM, a PM4314 QDSX, and a PM7344 S/UNI-MPH. Additionally, the schematic contains connectors, timing sources and miscellaneous "glue" circuitry.

PIC16C74 Microcontroller

Figure 2 shows how the I/O pins of the PIC16C74 have been defined for this reference design.

Figure 2. PIC16C74 Port Map

PORTA	unused	unused	QDSX CE	RAM CE	MPH CE	A10	A9	A8
bit	7	6	5	4	3	2	1	0
PORTB	LED 4	LED 3	LED 2	RAM INT	LED 1	unused	BUSY	EXT. INT
bit	7	6	5	4	3	2	1	0
PORTC	A7	A6	A5	A4	A3	A2	A1	A0
bit	7	6	5	4	3	2	1	0
PORTD	D7	D6	D5	D4	D3	D2	D1	D0
bit	7	6	5	4	3	2	1	0
PORTE	unused	unused	unused	unused	unused	unused	WRB	RDB
bit	7	6	5	4	3	2	1	0

Although the PIC16C74 is a simple microcontroller to program, there are some issues which can cause difficulties for the first-time programmer. Here is a list to be aware of:

- The internal register space (data memory) is partitioned into 2 banks. Accessing any particular register requires that the bank bit (bit 5 of the STATUS register) be set accordingly prior to the access. Care must be taken to select the register in the right bank, otherwise bugs which are hard to isolate will occur. Care must also be taken to preserve the state of the bank bit during interrupts.

- The internal program memory is partitioned into 2 pages. The page that is currently being executed is specified by the page bit (bit 3 of the PCLATH register). When a branch or call is made, the page bit is copied into the program counter. When making cross-page subroutine calls, the page bit must be set according to the location of the subroutine.

  The entire program counter is pushed onto the stack when a call is made, so when returning there is no need to program the page bit. This means that if a subroutine which manipulates the page bit is called, the state of the page bit may not be known upon return from that subroutine. Therefore, it is good programming practice to explicitly restore the page bit upon returning from a cross-page subroutine call to reflect the page from which the call was made.

- The PIC16C74 has four I/O pins (bits 4-7 of PORTB) which have an interrupt-on-change feature. They can be used as edge-triggered external interrupts. Each time the port is read (with any read or read-modify-write instruction) the state of these pins is latched into comparison logic. If at any time there is a pin transition such that a mismatch between the previous latched value and the current value occurs, an interrupt is asserted.

  The proper procedure for clearing the mismatch condition is explained in a Microchip application note (document number DS00566A) entitled "Using the Port B Interrupt on Change as an External Interrupt."

  There is a shortcoming in the interrupt logic of the PIC16C74 which makes it possible that transitions on the pins are occasionally missed. To work around this issue the interrupt-on-change pins should be occasionally polled for changes.

- Care must be taken when globally disabling interrupts because there exists the possibility that an interrupt will occur while the global interrupt enable bit is being cleared. To ensure that interrupts have been disabled, the following code fragment is recommended:

```
DISABLEINTS
        BCF             INTCON, GIE
        BTFSC    INTCON, GIE
        GOTO     DISABLEINTS
```

```
        . . .
```

- Because of complications involving register file bank swapping and preservation of the zero (Z) bit in the STATUS register, the following code fragment is the recommended procedure for state preservation in an interrupt service routine:

```
MOVWF    TEMP_W
SWAPF    STATUS, W
BCF              STATUS, RP0
MOVWF    TEMP_STATUS
MOVF             PCLATH, W
MOVWF    TEMP_PCLATH
```

And the following code is recommended to restore the saved registers before returning from an interrupt service routine:

```
MOVF             TEMP_PCLATH, W
MOVWF    PCLATH
SWAPF    TEMP_STATUS, W
MOVWF    STATUS
SWAPF    TEMP_W, F
SWAPF    TEMP_W, W
```

Dual-Port RAM

The dual-port RAM is shared by the host system and the local PIC16C74 microcontroller. The PIC16C74 performs all the local maintenance functions (including termination of the facility datalink), while the dual-port RAM is used to pass information to and from the host system. This arrangement greatly reduces the real-time burden on the host system.

The dual-port RAM holds, as a minimum, the following information:

- packets received by the S/UNI-MPH over the ESF facility datalink
- packets to be transmitted by the S/UNI-MPH over the ESF facility datalink, including T1.403 performance reports constructed by the PIC16C74.
- status of the S/UNI-MPH. This includes performance monitoring indications and parameters as specified in ANSI T1.231.
- channel-associated signaling for each of the received DS0 channels.

Each S/UNI-MPH Module has been given 2 kB of space because they must accumulate information on *four* S/UNI-MPH channels. For example, permanent allocation of 128 bytes per facility datalink (for each of transmit and receive) uses up 1 kB alone. The remaining 1 kB of memory should be sufficient for the status and performance monitoring information.

The two ports on the dual-port RAM are denoted the "Left" and "Right" ports. In this design, the Left Port is connected to the 100-pin connector which carries the host microprocessor bus (which is decoded, de-multiplexed and buffered on the D3MX Module). The Right Port is connected to the PIC16C74's microprocessor bus (shared with the microprocessor port of the S/UNI-MPH).

The two ports are entirely independent from each other and allow asynchronous Read and Write accesses from either port.

Each port also has an interrupt indication line used for processing a "mailbox" communications channel within the dual-port RAM. Each port has a memory location, which, when it is written to, alerts the other port via the interrupt indication signal. Therefore, a protocol can be used to pass information through these mailboxes. Using the mailboxes allows:

- the host to "peek" and "poke" registers in the S/UNI-MPH and QDSX even though it is not directly connected to their microprocessor ports,
- the host to initiate different configuration routines,
- the host to initiate different diagnostic routines, and
- the PIC16C74 to interrupt the host during alarm conditions and when HDLC packets or BOCs are received on the ESF datalink.

S/UNI-MPH Functional Block

The S/UNI-MPH is a full-featured quadruple ATM User Network Interface which provides most standard DS1 framing functions and performance monitoring.

The full register set of the S/UNI-MPH, including Test Mode registers are accessible to the PIC16C74 block. For a full description of these registers, refer to the S/UNI-MPH Data Book (PMC-940873).

The backplane signals of the S/UNI-MPH are connected to SCI-PHY multi-PHY ATM Cell bus, where the transport stream can be further processed.

QDSX Functional Block

The QDSX is a full-featured quadruple T1/E1 line interface unit which provides a G.703-compliant interface for 1544 kbit/s and 2048 kbit/s rates.

The full register set of the QDSX, including Test Mode registers are accessible to the PIC16C74 block. For a full description of these registers, refer to the QDSX Data Book (PMC-950739).

In this reference design, the QDSX is being used solely to provide a DSX-1 interface for the four duplex DS-1 serial data streams.

Line Interface Circuitry

The line interface circuitry consists of the transformers, connectors and passive networks necessary to interface the QDSX device to cables carrying G.703-compliant 1544 kbit/s signals. This circuitry reflects recommendations in the QDSX data book.

The transformer shown is a new product, the T1008 from Pulse Engineering. Other manufacturers which produce suitable transformers are BH Electronics, Filtran, Midcom, and Schott. The characteristics of the transformer should match those recommended in the QDSX data book.

Timing Distribution

The high-speed timing to the S/UNI-MPH Module is sourced from an on board oscillator, or optionally from the host 100-pin connector.

The Utopia interface timing is sourced from the 100-pin connector P2.

The S/UNI-MPH and QDSX require a 37.056MHz high-speed clock. The PIC16C74 requires a high-speed clock, with 20MHz being the maximum frequency. It is recommended that all clocks be sourced from a host circuit, since that provides the lowest cost solution.

100-Pin Connector P1: Host Interface

This 100-pin connector is used to connect the S/UNI-MPH Module to a Host device. This connector carries a microprocessor control, address, and data bus, as well as power to the S/UNI-MPH Module. The signals are defined in the following table.

In Table 9, the signal type is one of: I (input), O (output), I/O (bi-directional), N/C (no-connect), PWR (power), or GND (ground). The direction of the signal is with reference to this design.

Table 9. 100-Pin P1Connector: Host Interface Pin Definitions

Pin Name	Type	Pin	Function
GND	GND	A1, A2	Ground
N/C	N/C	A3	No connection
INTB	O	A4	Interrupt indication (active low).
TFPI	I	A5	Transmit frame position. This is an 8kHz signal used to synchronize the frame alignment of the framer backplanes for synchronous switching applications.
TCLKI	I	A6	Transmit clock input. This is a clock (either 1.544MHz or 2.048MHz) used to synchronize the timing of the framer backplanes for synchronous switching applications.
GND	GND	A7, A8	Ground

Pin Name	Type	Pin	Function
N/C	N/C	A9-A14	No connection
VCC	PWR	A15, A16	+5V
N/C	N/C	A17, A18	No connection
GND	GND	A19, A20	Ground
N/C	I	A21	No connection
N/C	I	A22	No connection
GND	GND	A23, A24	Ground
N/C	I	A25	No connection
N/C	I	A26	No connection
GND	GND	A27, A28	Ground
N/C	I	A29	No connection
N/C	I	A30	No connection
GND	GND	A31, A32	Ground
N/C	I	A33	No connection
N/C	I	A34	No connection
GND	GND	A35, A36	Ground
N/C	O	A37	No connection
N/C	O	A38	No connection
GND	GND	A39, A40	Ground
N/C	O	A41	No connection
N/C	O	A42	No connection
GND	GND	A43, A44	Ground
N/C	O	A45	No connection
N/C	O	A46	No connection
VDD	PWR	A47, A48	+5V
N/C	N/C	A49	No connection
RSTB	I	A50	Hardware Reset (active low)
N/C	O	A51	No connection
N/C	O	A52	No connection
GND	GND	A53, A54	Ground
A[3:0]	I	A55-A58	Address Bus [3:0]
GND	GND	A59, A60	Ground
A[5, 4]	I	A61, A62	Address Bus [5, 4]
VDD	PWR	A63, A64	+5V
A[7, 6]	I	A65, A66	Address Bus [7, 6]
GND	GND	A67, A68	Ground
A[10:8]	I	A69-A71	Address Bus [10:8]
N/C	N/C	A72	No connection
GND	GND	A73, A74	Ground
N/C	N/C	A75-A78	No connection
GND	GND	A79, A80	Ground
RWB	I	A81	Read/Write Bar signal of the microprocessor control bus.
N/C	N/C	A82	No connection
VDD	PWR	A83, A84	+5V
CSB	I	A85	Chip Select (active low). Selects the dual-port RAM's Left Port on the S/UNI-MPH Module for microprocessor access.
N/C	N/C	A86	No connection
GND	GND	A87, A88	Ground

Pin Name	Type	Pin	Function
D[3:0]	I/O	A89-A92	Data Bus [3:0]
GND	GND	A93, A94	Ground
D[7:4]	I/O	A95-A98	Data Bus [7:4]
XCLK EXT	I	A99	High-Speed Clock (37.056MHz) for S/UNI-MPH
MCLK	I	A100	High-Speed Clock (20.000MHz) for PIC16C74

100-Pin Connector P21: SCI-PHY Interface

This 100-pin connector is used to connect the S/UNI-MPH Module to a SCI-PHY MultiPHY device such as the RCMP reference design. This connector carries an 8-bit bidirectional PHY bus interface and power for optional use by the devices on the S/UNI-MPH Module.

In Table 10, the signal type is one of: I (input), O (output), I/O (bi-directional), N/C (no-connect), PWR (power), or GND (ground). The direction of the signal is with reference to this design.

Table 10. 100-Pin Connector P21: SCI-PHY Interface Pin Definitions

Pin Name	Type	Pin	Function
GND	GND	A1, A2	Ground
TDAT[0]	I	A3	Transmit Cell Data
TDAT[4]	I	A4	Transmit Cell Data
TDAT[1]	I	A5	Transmit Cell Data
TDAT[5]	I	A6	Transmit Cell Data
VDD_D	PWR	A7, A8	Decoupled VCC
TDAT[2]	I	A9	Transmit Cell Data
TDAT[6]	I	A10	Transmit Cell Data
TDAT[3]	I	A11	Transmit Cell Data
TDAT[7]	I	A12	Transmit Cell Data
GND	GND	A13	Ground
TXPRTY	I	A14	Transmit bus parity
TSOC	I	A15	Transmit start of cell
GND	GND	A16	Ground
GND	GND	A17	Ground
N/C	N/C	A18	No Connection
N/C	N/C	A19	No Connection
GND	GND	A20,A21	Ground
TFCLK	I	A22	Transmit FIFO write clock
N/C	N/C	A23	No Connection
GND	GND	A24, A25	Ground
GND	GND	A26	Ground
N/C	N/C	A27	No Connection
GND	GND	A28	Ground
GND	GND	A29	Ground
RDAT[4]	O	A30	Receive Cell data
RDAT[0]	O	A31	Receive Cell data

Pin Name	Type	Pin	Function
RDAT[5]	O	A32	Receive Cell data
RDAT[1]	O	A33	Receive Cell data
RXPRTY	O	A34	Receive bus parity
VDD_D	PWR	A35, A36	Decoupled VCC
RDAT[2]	O	A37	Receive Cell data
RDAT[6]	O	A38	Receive Cell data
RDAT[3]	O	A39	Receive Cell data
RDAT[7]	O	A40	Receive Cell data
GND	GND	A41	Ground
N/C	N/C	A42	No Connection
RSOC	O	A43	Receive start of cell
GND	GND	A44,A45	Ground
N/C	N/C	A46,A47	No Connection
GND	GND	A58,A49	Ground
RFCLK	O	A50	Receive FIFO read clock
N/C	N/C	A51	No Connection
GND	GND	A52,A53	Ground
N/C	N/C	A54,A55	No Connection
GND	GND	A56,A57	Ground
N/C	N/C	A58-A62	No Connection
VDD_D	PWR	A63, A64	Decoupled VCC
N/C	N/C	A65-A68	No Connection
GND	GND	A69, A70	Ground
N/C	N/C	A71-A74	No Connection
GND	GND	A75, A76	Ground
N/C	N/C	A77, A80	No Connection
GND	GND	A81,A82	Ground
N/C	N/C	A83, A84	No Connection
VDD_D	PWR	A85, A86	Decoupled VCC
N/C	N/C	A87-A89	No Connection
GND	GND	A90, A91	Ground
N/C	N/C	A92, A93	No Connection
GND	GND	A94, A95	Ground
N/C	N/C	A96-A98	No Connection
GND	GND	A99	Ground
GND	GND	A100	Ground

20-Pin Connector P22: SCI-PHY MultiPHY Interface

This 20-pin connector is used to address multiple PHYs on the S/UNI-MPH Module This connector carries addressing and protocol information.

In Table 11, the signal type is one of: I (input), O (output), I/O (bi-directional), N/C (no-connect), PWR (power), or GND (ground). The direction of the signal is with reference to this design.

Table 11. 20-Pin Connector P22: Multi-PHY Interface Pin Definitions

Pin Name	Type	Pin	Function
N/C	N/C	A1, A2	Not Connected
RCAMPH	I/O	A3	Receive multiPHY cell available
RRDMPH	I	A4	Active low receive multiPHY read enable
N/C	N/C	A5-A7	Not Connected
RRA1	I	A8	Receive read address MSB
N/C	N/C	A9	Not Connected
RRA0	I	A10	Receive read address LSB
TWA0	I	A11	Transmit write address LSB
N/C	N/C	A12	Not Connected
TWA1	I	A13	Transmit write address MSB
N/C	N/C	A14-16	Not Connected
TWRMPHB	I	A17	active low Transmit MultiPHY write enable
TCAMPH	I	A18	Transmit multiPHY cell available
N/C	N/C	A19,20	Not Connected

Firmware

Appendix D contains the source code listing of the assembly language firmware used to program the PIC16C74. This section gives a brief overview of the code in Appendix D, describing the functional routines and associated implementation issues.

Note: The source code in Appendix D is meant as a proof-of-concept that an inexpensive, simple microcontroller is capable of handling all the physical layer functions associated with the four DS1 channels of a PM7344 S/UNI-MPH device. *It is the responsibility of the person(s) using or adapting the example code to ensure that the resulting system operation complies with both standardized and proprietary requirements.*

The manufacturer of the PIC16C74, Microchip, provides free firmware development software for the assembler (MPASM) and simulator (MPSIM). The firmware for this reference design was developed and tested using that free software.

An efficient way of processing events that may have a low frequency of occurrence is to use interrupt-driven routines (instead of polling). The events (alarm conditions, timers, datalink servicing, and dual-port mailbox events) on the S/UNI-MPH and the dual-port RAM will, on average, be low frequency. Therefore, the PIC16C74 firmware developed for this reference design is based on an interrupt-driven scheme.

When the PIC16C74 detects an interrupt indication on its external interrupt input pins, it determines the source of the interrupt by polling its internal interrupt source register to determine if it was the S/UNI-MPH, QDSX or the dual-port RAM that interrupted. If it was the S/UNI-MPH or QDSX, then the registers of both devices need to be further polled to determine what event caused the interrupt. Once the PIC16C74 has determined the interrupt source, it can jump to a routine specific to that interrupt.

The *P16C74.INC* File

The *p16C74.inc* file is the standard include file for the PIC16C74 provided by Microchip which contains labels for all the special registers and bits of the microcontroller.

The *MACROS.INC* File

The *macros.inc* file defines some macros which are generally useful when using the PIC16C74.

MACROS

Table 12 describes the macros defined in the *macros.inc* file.

Table 12. Macros in *MACROS.INC*

Macro	Arguments	Description
BANK0	none	Selects the Bank 0 register space.
BANK1	none	Selects the Bank 1 register space
PAGE0	none	Selects Page 0 of program memory
PAGE1	none	Selects Page 1 of program memory
INTSOFF	none	Disables all interrupts by clearing the global interrupt enable, and testing to ensure that it is cleared.
INTSON	none	Enables all interrupts by setting the global interrupt enable.

The *MPH.ASM* File

The *mph.asm* file is the main source file containing the macros, constants and routines specific to this reference design.

CONSTANTS

The first portion of the *mph.asm* code contains a number of CBLOCK and CONSTANT statements which assign labels to the following:

- user registers within the Bank 0 register space.
- external LEDs driven by bits in the Port B register.
- S/UNI-MPH direct access registers.
- QDSX direct access registers.
- S/UNI-MPH and QDSX indirect access registers.
- dual-port RAM memory locations as per memory map

Following this are CONSTANT statements defining dual-port RAM mailbox codes for host to microcontroller and microcontroller to host communication and FEAC bit-oriented codes. Also included here are a number of miscellaneous constant definitions whose purpose is explained in the code.

MACROS

Table 13 describes the macros used to perform I/O operations with the S/UNI-MPH, QDSX, and the dual-port RAM.

Table 13. Description of Macros in *MPH.ASM*

Macro	Arguments	Description
WR_RAM	*address*, *value*	Writes *value* (byte) to the *address* in the dual-port RAM. Only the lower 11 bits of *address* are used.
WR_RAM_L	*register*, *value*	Writes *value* (byte) to quadrant specific RAM. Only the lower 7 bits are used, the upper 4 bits in the RAM address register (bit 7 of ADDR_RAM_LO and bits 0-3 of ADDR_RAM_HI) must be setup prior to invoking this macro.
WR_RAM_D	*address*	Writes the value in the data register (DATA_REG) to the *address* in the dual-port RAM. Only the lower 11 bits of the *address* argument are used.
WR_RAM_DL	*register*	Writes the value in the data register to quadrant specific RAM. Only the lower 7 bits are used, the upper 4 bits in the RAM address register must be setup prior to invoking this macro.
WR_MPH	*register*, *value*	Writes *value* (byte) to the *register* in the MPH. Only the lower 9 bits of the *register* argument are used.
WR_MPH_L	*register*, *value*	Writes *value* (byte) to a MPH *register* in a particular quadrant. Only the lower 7 bits are used, the upper 2 bits in the MPH address register (bit 7 of ADDR_MPH_LO and bit 0 of ADDR_MPH_HI) must be setup prior to invoking this macro.

Macro	Arguments	Description
WR_MPH_DL	*register*	Writes the value in the data register to a MPH *register* in a particular quadrant. Only the lower 7 bits are used, the upper 2 bits in the MPH address register must be setup prior to invoking this macro.
WR_QDSX	*register*, *value*	Writes *value* (byte) to the *register* in the QDSX. Only the lower 9 bits of the *register* argument are used.
WR_QDSX_L	*register*, *value*	Writes *value* (byte) to a QDSX *register* in a particular quadrant. Only the lower 6 bits are used, the upper 3 bits in the QDSX address register (bits 6&7 of ADDR_QDSX_LO and bit 0 of ADDR_QDSX_HI) must be setup prior to invoking this macro.
WR_QDSX_DL	*register*	Writes the value in the data register to a QDSX *register* in a particular quadrant. Only the lower 6 bits are used, the upper 3 bits in the QDSX address register (bits 6&7 of ADDR_QDSX_LO and bit 0 of ADDR_QDSX_HI) must be setup prior to invoking this macro.
RD_RAM	*address*	Reads from the *address* in the dual-port RAM. Only the lower 11 bits of the *address* argument are used. The value is returned in the data register and the W register.
RD_RAM_L	*register*	Reads from quadrant specific RAM. Only the lower 7 bits are used, the upper 4 bits in the RAM address register (bit 7 of ADDR_RAM_LO and bits 0-3 of ADDR_RAM_HI) must be setup prior to invoking this macro. The value is returned in the data register and the W register.
RD_MPH	*register*	Reads from the *register* in the MPH. Only the lower 9 bits of the *register* argument are used. The value is returned in the data register and the W register.
RD_MPH_L	*register*	Reads from a MPH *register* in a particular quadrant. Only the lower 7 bits are used, the upper 2 bits in the MPH address register must be setup prior to invoking this macro. The value is returned in the data register and the W register.
RD_QDSX	*register*	Reads from the *register* in the QDSX. Only the lower 9 bits of the *register* argument are used. The value is returned in the data register and the W register.
RD_QDSX_L	*register*	Reads from a QDSX *register* in a particular quadrant. Only the lower 6 bits are used, the upper 3 bits in the QDSX address register must be setup prior to invoking this macro. The value is returned in the data register and the W register.

ROUTINES

The following subsections describe each of the routines in the *mph.asm* file.

Main Loop

The processor loops infinitely waiting for the 1s time flag to be set. The 1s timer flag is set by a routine which is invoked using the internal timer interrupt. During each loop, it also checks the clock activity monitor for loss of clock activity. It must poll this activity because the clock activity monitor has no interrupt. When the 1s flag is set the processor executes the PMON subroutine, updating performance monitoring statistic and cell performance statistic shadow registers in RAM, and returns to the main loop. Here also the watchdog timer is cleared.

Performance Monitoring

Shadowing of performance monitoring statistics in the dual-port RAM and construction of one second performance reports (for transmission on the facility datalink) is done by the PMON subroutine.

The PMON subroutine writes to the Global PMON Update register of the MPH to trigger the update of performance statistics. The routine then polls the S/UNI-MPH waiting for the PMON block of the S/UNI-MPH to indicate that it has updated the performance statistics. When the S/UNI-MPH is ready, the routine reads the data from the S/UNI-MPH registers to local PIC registers.

The GENERATE_PRM subroutine is called to construct the performance report for each quadrant. These performance reports are formatted as specified in the ANSI T1.403 standard. The GENERATE_PRM subroutine reads the performance monitoring statistics from the S/UNI-MPH PMON registers and constructs the octet pair for the last second accordingly. Besides the S/UNI-MPH PMON data, the GENERATE_PRM subroutine also reads the Diagnostics register (to see if the quadrant is currently in a Payload Loopback mode). An eight-byte circular buffer is maintained for each quadrant such that the octets for the previous three seconds are available (in addition to the current second's octets), kept in the proper order. The GENERATE_PRM subroutine also copies the performance monitoring and cell performance statistics to RAM for use by the host system.

A modulo-4 counter is copied into the Nl and Nm bits in the performance report. This helps the far end equipment handle packet corruption (the performance reports use an unacknowledged packet protocol).

Once constructed, the performance reports will be transmitted by the XFDL interrupt service routine. Because of possible contention between one second performance reports and other host-initiated HDLC packets, if the datalink is busy (i.e. already

transmitting a packet) then the performance report is flagged pending and will be automatically transmitted when the datalink becomes available. The same protocol is also used when the host initiates packet transmission while a one second performance report is currently in transmission (i.e. the host-initiated packet will pend until the one second performance report is finished transmission).

Read Access to S/UNI-MPH

The READ_MPH subroutine is used by the RD_MPH and RD_MPH_L macros. The address for the MPH register is copied to the address latch (PORT C and bits 0-2 of PORT A) and the control lines WRB and CSB2 are toggled to perform a read cycle. The data bus (PORT D) is latched into the data register (DATA_REG).

Read Access to QDSX

The READ_QDSX subroutine is used by the RD_QDSX and RD_QDSX_L macros. The address for the QDSX register is copied to the address latch (PORT C and bits 0-2 of PORT A) and the control lines WRB and CSB3 are toggled to perform a read cycle. The data bus (PORT D) is latched into the data register (DATA_REG).

Read Access to Dual-Port RAM

The READ_RAM subroutine is used by the RD_RAM and RD_RAM_L macros. The RAM address is copied to the address latch (PORT C and bits 0-2 of PORT A) and the control lines WRB and CSB1 are toggled to perform a read cycle. The data bus (PORT D) is latched into the data register (DATA_REG).

Write Access to S/UNI-MPH

The WRITE_MPH subroutine is used by the WR_MPH, WR_MPHL and WR_MPH_DL macros. The address for the S/UNI-MPH register is copied to the address latch (PORT C and bits 0-2 of PORT A) and the data register (DATA_REG) is copied to the data bus (PORTD). The control lines WRB and CSB2 are toggled to perform a write cycle.

Write Access to QDSX

The WRITE_QDSX subroutine is used by the WR_QDSX, WR_QDSXL and WR_QDSX_DL macros. The address for the QDSX register is copied to the address latch (PORT C and bits 0-2 of PORT A) and the data register (DATA_REG) is copied to the data bus (PORTD). The control lines WRB and CSB3 are toggled to perform a write cycle.

Write Access to Dual-Port RAM

The WRITE_RAM subroutine is used by the WR_RAM, WR_RAM_D, WR_RAM_DL, and WR_RAM_L macros. The RAM address is copied to the address latch (PORT C and bits 0-2 of PORT A) and the data register (DATA_REG) is copied to the data bus (PORTD). The control lines WRB and CSB1 are toggled to perform a write cycle.

Initialize Microcontroller

The INIT_MICRO routine makes all control pins inactive, turns off the LEDs and clears the user registers. Timer 0 is configured and the 1 second timer counter is initialized The input/output state of each pin is set. PORT A is configured as a digital input (can be a A/D port). PORT B is read to initialize the interrupt on change logic.

Initialize Dual-Port RAM

The INIT_RAM routine reads the micro's dual-port RAM mailbox to clear any spurious interrupt. The version and revision numbers are copied to their respective locations in RAM.

Initialize S/UNI-MPH

The S/UNI-MPH is first initialized by writing to bit 7 of register 0x0C. Note that the bit must be explicitly set and then cleared to complete the reset operation. The INIT_MPH routine is called four times on start-up, once per quadrant. Table 14 shows the S/UNI-MPH registers initialized and the value to which they are set. The S/UNI-MPH is initialized for the Extended Superframe Format (ESF) for all channels.

Table 14. S/UNI-MPH Initialization Register Values

Register	Location	Value
Receive configuration	00	00*
Transmit configuration	01	10
Receive interface config	03	04
Transmit interface config	04	08
Transmit timing options	07	00*
Source selection	0D	00*
CDRC configuration	10	00*
CDRC interrupt enable	11	00
ALMI configuration	14	10
ALMI interrupt status	15	07
T1 FRMR configuration	1C	10
RBOC Enable	30	05
IBCD interrupt enable	3D	30
IBCD activate code	3E	08
IBCD decativate code	3F	24
T1 TRAN config	40	30

Register	Location	Value
RXCP control	70	2C
RXCP framing control	71	40
RXCP interrupt enable	72	20
RXCP Idle/unassigned cell pat	73	00*
RXCP Idle/unassigned cell pat	74	00*
RXCP Idle/unassigned cell pat	75	00*
RXCP Idle/unassigned cell pat	76	00*
RXCP Idle/unassigned mask	77	FF
RXCP Idle/unassigned mask	78	FF
RXCP Idle/unassigned mask	79	FF
RXCP Idle/unassigned mask	7A	FF
RXCP user cell pattern	7B	00*
RXCP user cell pattern	7C	00*
RXCP user cell pattern	7D	00*
RXCP user cell pattern	7E	00*
RXCP user mask	7F	FF
RXCP user mask	80	FF
RXCP user mask	81	FF
RXCP user mask	82	FF
TXCP control	88	A4
TXCP interrupt enable	89	40
TXCP idle cell pattern	8A	00*
TXCP idle cell pattern	8B	00*
TXCP idle cell pattern	8C	00*
TXCP idle cell pattern	8D	00*
TXCP idle cell pattern	8E	55[1]
TXCP idle cell payload	8F	6A

* This value is already defaulted to upon reset.

[1] This is the correct HCS for an unassigned cell header of 00(H1) 00(H2) 00(H3) 00 (H4).
The correct HCS for an idle cell header of 00(H1) 00(H2) 00(H3) 01(H4) is 52H.

To summarize, the above register settings configure the S/UNI-MPH for T1 single rail, B8ZS, ESF, NRZ, and an idle/unassigned ATM header pattern of 00H, 00H, 00H, 00H, 55H.

Initialize QDSX

The QDSX is first initialized by writing to bit 7 of register 0x07. Note that the bit must be explicitly set and then reset to complete the reset operation. The INIT_QDSX routine is called four times on start-up, once per quadrant. Table 15 shows the QDSX registers initialized and the value to which they are set. The QDSX is initialized for the Extended Superframe Format (ESF) for all channels.

Table 15. QDSX Initialization Register Values

Register	Location	Value
CDRC Config	10	64
IBCD Config	20	04
XPLS Config	2C	40
IBCD activate code	22	08
IBCD decativate code	23	24

Interrupt Service

The INTHDLR routine deals with interrupts asserted by the S/UNI-MPH, QDSX, dual-port RAM and internally by timer 0. Each source's interrupt flag is polled in turn, and if asserted the corresponding interrupt service routine is called. Those registers saved on entry into the service routine are restored on exit.

The external (S/UNI-MPH or QDSX) interrupt is serviced as long as it is asserted to minimize latency when there are multiple pending S/UNI-MPH or QDSX interrupts. This is important because the interrupts are edge-triggered.

The RAM interrupt pin is also polled as a work-around to solve the problem that the interrupt-on-change logic of the PIC16C74 can occasionally miss edges.

S/UNI-MPH Interrupt Service

The MPH_INT routine determines the source of an S/UNI-MPH interrupt, both quadrant and block, and then branches to the appropriate service routine.

ALMI Interrupt Service

The ALMI interrupt status register is examined to determine if an AIS, YELLOW alarm, or RED alarm has been asserted or cleared. A message is written to the host's mailbox accordingly. In the case that an AIS has been asserted, all loopbacks are cleared.

CDRC Interrupt Service

The CDRC interrupt status register is examined to determine if an LOS has been asserted or cleared, or if a pulse density violation has been detected. Response to these interrupts has been commented out of the code to minimize the amount of messages sent to the host. However, it should be noted that the LOS defect will still be integrated to a RED alarm by other firmware.

FRMR Interrupt Service

The FRMR interrupt status register is examined to determine if the quadrant has gone in or out of frame, a framing error or severe framing error has occurred. Response to these interrupts has been commented out of the code to minimize the amount of messages sent to the host. As with the CDRC interrupt service, the OOF defect will still be integrated to a RED alarm by other firmware. Framing errors are counted as performance statistics.

RBOC Interrupt Service

The RBOC interrupt status register is examined to determine if a valid BOC has been detected or if the channel has gone idle. A message is written to the host's mailbox accordingly.

In the case that a new valid BOC has been detected, the BOC is copied to RAM and is tested to see if it is a loopback related request. Line loopback deactivate, payload loopback activate and deactivate and universal loopback deactivate are dealt with immediately. If a line loopback activate request is received, a bit flag is set using the routine QUAD_BIT_SET.

In the case the link has gone from transmitting a valid BOC to transmitting flags (idle state) the host is notified via its mailbox. If the line loopback bit flag is set then a line loopback is initiated.

IBCD Interrupt Service

The IBCD interrupt status register is examined to determine if an in-band line loopback activate or deactivate request has been received. The line loopback is initiated or cleared accordingly and a message is written to the host's mailbox.

RFDL Interrupt Service

The RFDL Receive Data register is read and stored locally. The RFDL Status register is read to determine if an overrun or abort has occurred. In the case of an overrun the overrun bit in the RAM copy of the RFDL status register is set. If an abort has occurred, the local link status is made inactive and the status and packet length count are reset. If the link has gone from inactive to active then the link status is set to active and the packet length counter is reset in preparation for a new packet. If this is the next byte of a packet currently being received, then the byte is copied into the RAM receive buffer, packet length counter incremented and the end of message bit in the RFDL status register is tested. If this byte is the end of the current message then the CRC error bit and NVB bits of the RAM RFDL status register are updated, local link status made inactive and packet length copied to RAM. If the packet length is greater than 3 (which

all valid packets with CRC enabled will be) then the host is notified of the new packet through its mailbox.

XFDL Interrupt Service

An XFDL interrupt will occur when either the XFDL needs another data byte for transmission, or if an underrun has occurred in the XFDL FIFO.

When an XFDL interrupt occurs, a flag (in the Q#_FLAGS register) is tested to see whether a performance report or a host-initiated packet is currently being transmitted so that the firmware knows from where to source the data.

- If a host-initiated HDLC packet is being transmitted, the XFDL Interrupt Status register is examined to determine if an underrun has occurred.

  > If an underrun has occurred the XFDL UDR bit is cleared and the packet is restarted (unless retransmission has already been attempted 10 times in which case the firmware will give up attempting to retransmit the packet). Note that the XFDL will automatically transmit an HDLC ABORT sequence during an underrun condition.

  > If an underrun has not occurred, the data pointer is compared with the packet length to determine if the end of message has been reached.

    - If the end of message has been reached, the EOM bit is set and a flag is tested to see if a performance report is pending.

      If a report is pending, a flag is programmed to indicate that a performance report is being transmitted.

      If a report is not pending, the XFDL interrupts are disabled and the FDL busy bit in the flags register is cleared. The host is notified through its mailbox that the host-initiated packet has finished transmitting.

    - If the end of message has not been reached, the next data byte is copied from the dual-port RAM to the XFDL Transmit Data register, and the data pointer is advanced.

- If a performance report is being transmitted, the XFDL interrupt status register is examined to determine if an underrun has occurred.

  > If an underrun occurred, the XFDL UDR bit is cleared and the transmission is aborted. The firmware will not attempt to retransmit the packet because each performance report contains three seconds worth of history (i.e. up to three

consecutive performance reports can be corrupted without losing any information).

> If an underrun has not occurred, a counter variable is examined to determine which octet of the performance report to transmit next. The first three octets (containing the LAPD overhead fields) are constants. The fourth through eleventh octets are taken from the circular buffer. If the eleventh octet has been transmitted then the EOM bit is set and the flags register is tested to see if a host-initiated HDLC packet is pending.

- If a host-initiated packet is pending, a flag is programmed to indicate a host-initiated packet is being transmitted.

- If a packet is not pending, the XFDL interrupts are disabled and the FDL busy bit in the flags register is cleared.

RXCP Interrupt Service

The RXCP interrupt status register is examined to determine if a receive FIFO overrun or loss of cell delineation has occurred, and the appropriate mailbox code is sent to the host.

TXCP Interrupt Service

The TXCP interrupt status register is examined to determine if a transmit FIFO overrun or change of cell alignment has occurred, and the appropriate mailbox code is sent to the host.

QDSX Interrupt Service

The QDSX_INT routine determines the source of a QDSX interrupt, both quadrant and block, and then branches to the appropriate service routine.

DJAT Interrupt Service

This routine currently takes no action.

CDRC Interrupt Service

This routine currently takes no action.

LCV Interrupt Service

This routine currently takes no action.

PRSM Interrupt Service

This routine currently takes no action.

IBCD Interrupt Service

This routine currently takes no action.

XPLS Interrupt Service

This routine currently takes no action.

RLSC Interrupt Service

This routine currently takes no action.

Dual-Port Mailbox Interrupt Service

The micro's mailbox is read to determine the request issued by the host. If a valid code is read the corresponding routine is executed.

Timer Interrupt Service

The 3ms flag is set and 1s timer counter decremented. If the 1s counter has reached zero then the 1s flag is also set and the timer LED (LED 4) is toggled.

APPENDIX A: DESIGN CONSIDERATIONS

Power Supply Voltage Transients

High currents drawn during IC switching causes power supply voltage transients due to the inductance of the power lines. The magnitude of the noise voltage can be reduced by minimizing the inductance of the power lines and by decreasing the magnitude of the transient currents. The power line inductance can be minimized by using a power plane. The transient currents on the power rails can be minimized by supplying the power from a local source such as a de-coupling capacitor near the circuit drawing the current.

The de-coupling capacitance and the inductance of the connection between the capacitor and the power pin determine the noise voltage at the power pin. The effectiveness of the de-coupling capacitor depends on the frequencies of the transients. Large "bulk" de-coupling capacitors are used to supply the low-frequency current variations and the small "noise-bypassing" capacitors are used to supply the high-frequency transient current that is required when the circuit is switching.

Ground Noise

Return currents and power supply transients during high current consumption produce most of the ground noise. Since ground noise cannot be controlled by de-coupling capacitors, the only way to minimizing the effect of ground noise is to minimize ground impedance. The best way to minimize ground impedance is to use a ground plane. It is not advisable to use ferrite beads in the ground path as this will inhibit the return currents from leaving and raise the ground noise level.

Noise-Bypassing at Power Pins

The S/UNI-MPH can generate a lot of simultaneous switching noise, especially if the line rate clocks are synchronous. It is important to provide a noise-bypassing capacitor at every power pin so that the switching currents can be supplied locally, thereby reducing the noise introduced into the power plane.

Values of Noise-Bypassing Capacitors

A rule of thumb is that the bulk noise-bypassing capacitor (placed where the power enters the circuit board) should have 10 times the value of all the noise-bypassing capacitors combined. Capacitors with low internal inductance should be used such as a tantalum electrolytic. Stay away from aluminum electrolytic as their inductances are an order of magnitude larger than tantalum capacitors.

The noise-bypassing capacitors (placed near the power pins) must be able to supply all the switching current. The minimum capacitance can be calculated by:

$$C = \Delta I * \Delta t / \Delta V$$

The transient voltage drop ΔV in the supply voltage is caused by the transient current ΔI occurring over time Δt. This equation shows that the voltage drop will be minimized as the capacitance is increased. However, using capacitors that are too large should be avoided due to their resonance characteristics.

Since all capacitors have some stray inductance in series with the capacitance, there will be a self-resonance at a certain frequency given by the equation:

$$f = \frac{1}{2\pi\sqrt{LC}}$$

Note that the larger the capacitance (for the same inductance) the lower the resonant frequency. If the capacitor is too large, the self-resonance will be too low to be an effective bypass but if the capacitor is not large enough, there will be insufficient current to supply the transient current during switching. The smallest value capacitor to satisfy the above equations should be used. It is rarely necessary that a capacitor larger than 0.01 µF be used.

Placement of Noise-Bypassing Capacitors

The de-coupling capacitor should be placed as close to the IC power pin as possible reduce the wiring inductance. There are five sources of inductance: the parasitic inductance of the capacitor, the inductance of the wiring between the capacitor and the IC power pin, the power pin lead inductance inside the IC, the ground pin lead inductance inside the IC, and the ground inductance between the IC pin and ground. The capacitor inductance is negligible if the correct capacitor is used. There is no control over the lead frame inductance. To keep the inductance low, both the power lead and the ground lead should be keep as short as possible (less than 1.5 inches). The inductance for a trace is given by:

$$L = 0.005 \log^{-1}\left(2\pi\frac{h}{w}\right) \ \mu\text{H/inch}$$

where **h** is the height between the power or ground lead and the ground plane and **w** is the width of the power or ground lead. Note that doubling the width of the trace or reducing **h** will only decrease **L** approximately by 20%, but decreasing the length by 50% will decrease the inductance by 50%. A typical PCB trace has about 15nH of inductance per inch.

Ferrite Beads

Ferrite beads are mainly used on power rails to pass DC current but to attenuate the higher frequency noise that is riding on the DC rail. The impedance of ferrite beads increases with frequency; at DC the ferrite bead is like a short but at higher frequency the impedance of a ferrite bead can increase to over 100 ohms (depending on the bead and frequency). Ferrite beads attenuate high frequency noise from the power supply from getting into a circuit, but they also stop high frequency switching currents required by digital ICs. It is important, therefore, to use proper noise bypass capacitors when using ferrite beads to provide a local source of switching current.

Ferrite beads should be avoided on CMOS I/O power pins as the high current switching of the CMOS circuits causes a $\Delta I/\Delta t$ noise to be introduced into the power rail. This noise is induced because the ferrite beads "starve" the digital circuitry, causing the voltage to fluctuate locally. Ferrite beads should also be avoided on the ground bus as this inhibits the return currents.

Unused CMOS Inputs

"Floating" CMOS inputs (those that are left unconnected) may switch unpredictably, causing unwanted noise and power consumption. Therefore, all unused inputs should be connected to their inactive state: to ground or to the power rail (Vcc). Unused bi-directionals should be "pulled" through a series resistor (4.7k or greater) to avoid short-circuits occuring if the bi-directionals are erroneously configured as outputs.

APPENDIX B: MATERIAL LIST

Item No.	Part Name - Value	JEDEC Type	Reference Designator	Qty
1	BANTAM-BASE	BANTAM	J91-J98	8
2	CAP-.68UF,METAL POLY	CAP200	C37-C40	4
3	CAP-0.001UF	SMDCAP805	C8,C9,C27,C29, C30	5
4	CAP-100000PF	SMDCAP120	C12-C19, C21-C24,C45-C48	16
5	CAP-10000PF	SMDCAP805	C1-C7,C10, C58-C73	24
6	CAP-1000PF,NPO_805	SMDCAP805	C42-C44,C83	4
7	CAP-47000PF	SMDCAP1206	C53-C56	4
8	CAPACITOR POL-0.47UF, 25V, TANT A	SMDTANCAP_A	C49-C52	4
9	CAPACITOR POL-10UF, 16V,TANT TEH	SMDTANCAP_C	C11,C20,C57	3
10	CONN100-AMP_103911-8	AMP_103911-8	P1,P3	2
11	CONN20-AMP_103911-2	AMP_103911-2	P2	1
12	CY7C136_-BASE	PLCC52	U11	1
13	HEADER3-BASE	JUMPER3	JP1,JP2	2
14	INDUCTOR-FB,50, FAIR RITE	INDUCTOR_FB	L1-L8	8
15	LED10-RED,25MA,2.1V	DIP20_LED	U22	1
16	MPH-BASE	PQFP128	U27	1
17	OSC_TTL_DIP-20.0000MHZ,100 PPMA	CRYS14	U18	1
18	OSC_TTL_DIP-37.056MHZ,32/50 PPA	CRYS14	U13	1
19	PIC16C74-BASE	PIC16C74	U1	1
20	PWRBLOCK_2-BASE	CONN2END	J6	1
21	QDSX-BASE	PQFP128	U24	1
22	RESISTOR-0,5%	SMDRES805	R1,R4,R5,R7	4

APPENDIX C: SCHEMATICS

*** Signal Cross-Reference ***
--- for the entire design --

A <10..0>	1E8 1E9 6D4<
ALE	1F3 2C9<
AR <10..0>	1D4 1E5 1E6 1F3 2B9< 4C10<
BUSYBR	1C4 1D6
CEBR	1C4 1D6
CSB	6C4> 1D9
CSB1	1D3 1F3 2B9<
CSB3	1D3 1F6 4C10<
D <7..0>	6B4<> 1D8 1E8 1E9
DR <7..0>	2B4> 4C3<> 1C4 1D4 1D6 1E6 1F3
INTB	6G4> 1D9
INTB1	2C4> 1D3 1G5 4C10<
INTBR	1C4 1D6
LED1	1D3
LED2	1D3
LED3	1D3
LED4	1D3
MCLK	1D9> 6B4> 1B4<
MCLK_RC	1B3
OEBR	1C4 1D6 1F3 1F6 2C9< 4C10<
OSC1	1D3
RAVD <4..1>	4D1> 4E1> 4G6
RCAMPH	3B5> 1F3 2G9<
RCLKO <4..1>	4D10> 4E10> 1G6 2E10<
RDAT <7..0>	2D4> 3F4> 1G3
RDD <4..1>	4D10> 4E10> 1G6 2D9< 2E9< 2F9<
RFCLK	1G3 2C9< 3E8<
RLCV <4..1>	4D10> 4E10> 1G6 2D9< 2E9< 2F9<
RRA0	1F3 2G9< 3B8<
RRA1	1F3 2G9< 3B8<
RRDMPHB	1F3 2G9< 3B8<
RSOC	2D4> 3E4> 1G3
RSTB	1D3 1E8 1E9 1F3 2B9< 4B10< 6D4<
RWB	6C4> 1D9
RWRB	1C4 1D6 1F3 1F6 2C9< 4C10<
RXPRTY	2D4> 3F4> 1G3
RXRING <4..1>	5A9> 5C9> 5E9> 5G9> 1G8 4D3< 4E3<
RXTIP <4..1>	5B9> 5D9> 5E9> 5G9> 1G8 4D3< 4E3<
TAVD <4..1>	4D1> 4E5 4F5
TCAMPH	1G3 2F9< 3A8<
TCLK	6G4> 1F9 1G5 2G9<
TCLKI <4..1>	2D4> 2E4> 2F4> 1G6 4E10< 4F10<
TDAT <7..0>	3G4> 1G3 2D9<
TDD <4..1>	2D4> 2E4> 2F4> 1G6 4E10< 4F10<
TDN <4..1>	2D4> 2E4> 2F4> 1G6 4E10< 4F10<
TFCLK	3F4> 1G3 2C9<
TFPI	6G4> 1F9 1G5 2G9<
TSOC	3G4> 1G3 2D9<
TWA0	3A5> 1G3 2F9<
TWA1	3A5> 1G3 2F9<
TWRMPHB	3A5> 1G3 2F9<
TXPRTY	3G4> 1G3 2D9<
TXRING <4..1>	4E3> 4F3> 1G8 5B9< 5D9< 5F9< 5G9<
TXTIP <4..1>	4E3> 4F3> 1G8 5C9< 5D9< 5F9< 5H8<
XCLK	1F3 1G6 2C9< 4G10<
XCLK_EXT	1D9> 6B4> 1B7<
XCLK_INT	1B6

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPR

PMC-Sierra, Inc.

DOCUMENT NUMBER: DOCNUM	ISSUE: ISSUE
TITLE: ?T ?T2	DATE: YY/MM/DD
ENGINEER: ?E	PAGE: ?P OF ?T

*** Unit Cross-Reference ***
--- for the entire design --

C1	CAP	2G3
C2	CAP	2G3
C3	CAP	2G3
C4	CAP	2G2
C5	CAP	2G2
C6	CAP	2G2
C7	CAP	2G2
C8	CAP	1C6
C9	CAP	1B5
C10	CAP	1B10
C11	CAPACITOR POL	1B9
C12	CAP	1C7
C13	CAP	1C7
C14	CAP	1A6
C15	CAP	1B7
C16	CAP	1B5
C17	CAP	1C8
C18	CAP	1C7
C19	CAP	1C3
C20	CAPACITOR POL	3G9
C21	CAP	3G9
C22	CAP	3G8
C23	CAP	3G8
C24	CAP	3G7
C27	CAP	1B7
C29	CAP	1C6
C30	CAP	1C6
C37	CAP	5H7
C38	CAP	5F7
C39	CAP	5D7
C40	CAP	5C7
C42	CAP	5H7
C43	CAP	5F7
C44	CAP	5D7
C45	CAP	5G7
C46	CAP	5E7
C47	CAP	5C7
C48	CAP	5A7
C49	CAPACITOR POL	4F5
C50	CAPACITOR POL	4F5
C51	CAPACITOR POL	4E5
C52	CAPACITOR POL	4E5
C53	CAP	4E5
C54	CAP	4D5
C55	CAP	4D5
C56	CAP	4C5
C57	CAPACITOR POL	4G1
C58	CAP	4H2
C59	CAP	4G2
C60	CAP	4G2
C61	CAP	4G2
C62	CAP	4G2
C63	CAP	4G2
C64	CAP	4F2
C65	CAP	4F2
C66	CAP	4E2
C67	CAP	4E2
C68	CAP	4E2
C69	CAP	4E2
C70	CAP	4D2
C71	CAP	4D2
C72	CAP	4D2
C73	CAP	4D2
C83	CAP	5B7
J6	PWRBLOCK_2	1A9
J91	BANTAM	5G4
J92	BANTAM	5F4
J93	BANTAM	5D4
J94	BANTAM	5B4
J95	BANTAM	5G4
J96	BANTAM	5E4
J97	BANTAM	5C4
J98	BANTAM	5B4
JP1	HEADER3	1B6
JP2	HEADER3	1C3
L1	INDUCTOR	4G6
L2	INDUCTOR	4G6
L3	INDUCTOR	4G5
L4	INDUCTOR	4G5
L5	INDUCTOR	4F4
L6	INDUCTOR	4F4
L7	INDUCTOR	4E4
L8	INDUCTOR	4E4
P1	CONN100	6G7
P2	CONN20	3B7
P3	CONN100	3G6
R1	RESISTOR	5G7
R2	RESISTOR	6G6
R3	RESISTOR	1B6
R4	RESISTOR	5F7
R5	RESISTOR	5D7
R6	RESISTOR	1D5
R7	RESISTOR	5B7
R12	RESISTOR	1B6
R13	RESISTOR	1C3
R16	RESISTOR	5H7
R17	RESISTOR	5F7
R20	RESISTOR	5G8
R21	RESISTOR	5E8
R22	RESISTOR	5C8
R23	RESISTOR	5A8
R24	RESISTOR	5C7
R25	RESISTOR	5G7
R26	RESISTOR	5E7
R27	RESISTOR	5B7
R28	RESISTOR	5D7
R29	RESISTOR	5B7
R36	RESISTOR	4E5
R37	RESISTOR	4D5
R38	RESISTOR	4D5
R39	RESISTOR	4C5
RN1	RES_ARRAY_8	2C7 2G7 2G8
RN2	RES_ARRAY_8	1D3
SB1	SOLDER_BRIDGE	3H8
SB2	SOLDER_BRIDGE	3B6
T1	T1008	5H6
T2	T1008	5D6
TP1	TST_PT	1B8
TP2	TST_PT	1B8
TP3	TST_PT	1B8
TP4	TST_PT	1B8
TP5	TST_PT	1B8
TP6	TST_PT	1A8
TP7	TST_PT	1B8
TP8	TST_PT	1B8
TP9	TST_PT	1B8
TP10	TST_PT	1A8
U1	PIC16C74	1D4
U2	QDSX	4C5
U11	CY7C136_	1E7
U13	OSC_TTL	1B6
U18	OSC_TTL	1C4
U22	LED10	1D2
U27	MPH	2H5
U28	74XXX541	2E10

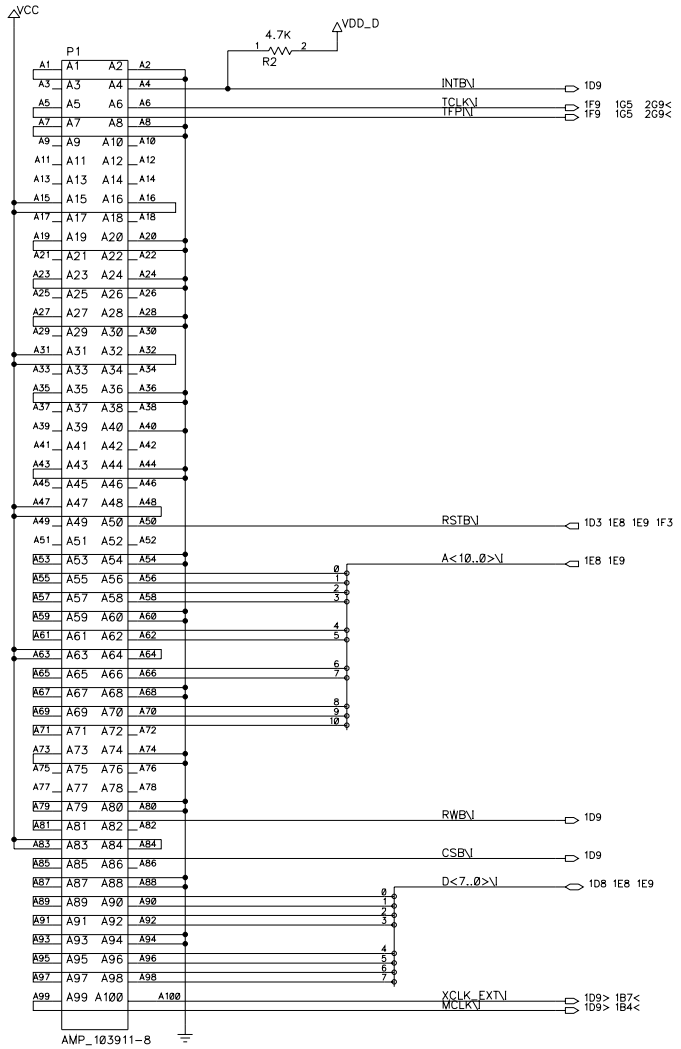
REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPR

PMC-Sierra, Inc.

DOCUMENT NUMBER: DOCNUM	ISSUE: ISSUE
TITLE: ?T ?T2	DATE: YY/MM/DD
ENGINEER: ?E	PAGE: ?P OF ?T

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPR



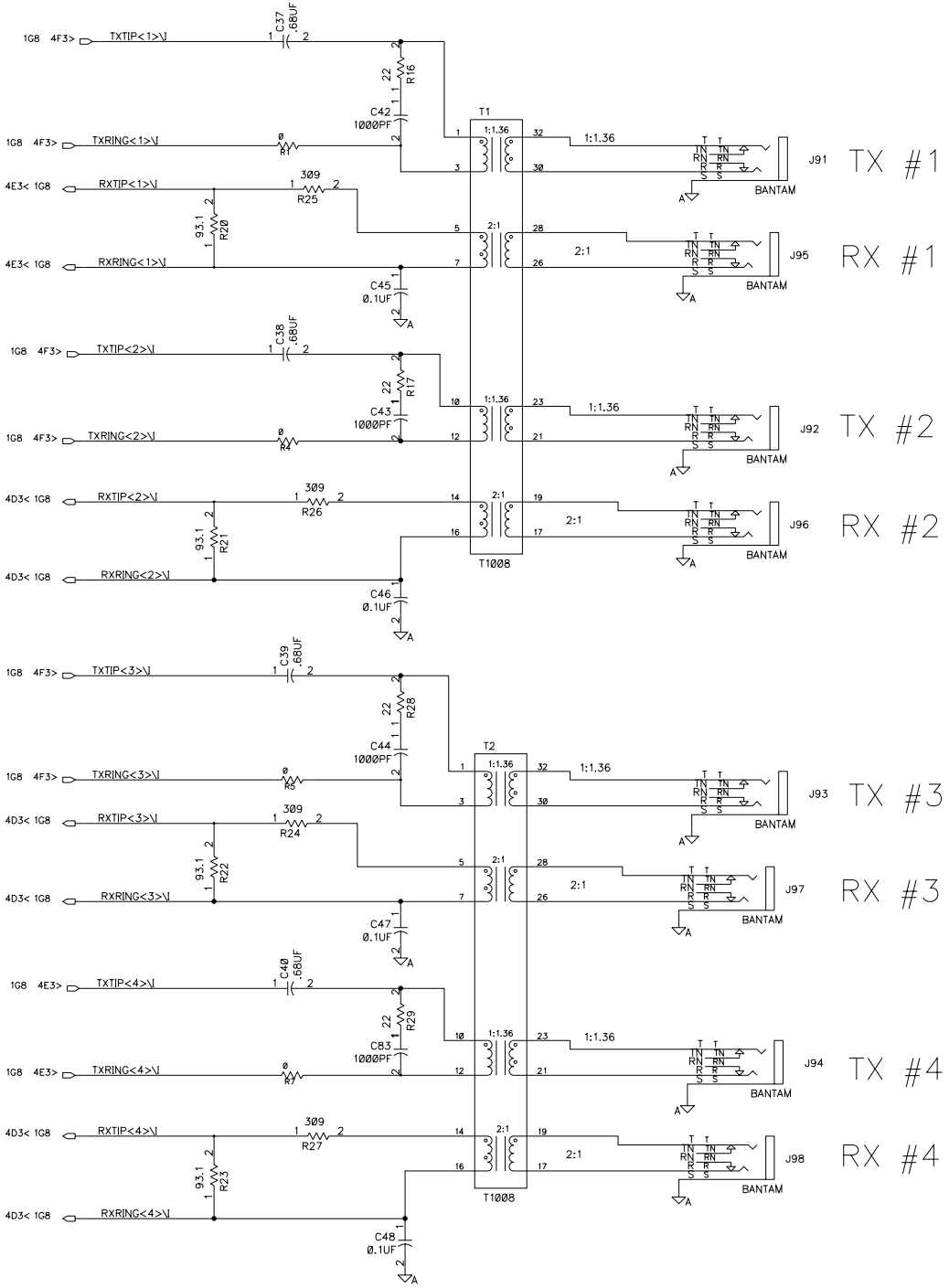
DRAWING

TITLE=HOST_INTERFACE
ABBREV=HOST_INERFACE
LAST_MODIFIED=Tue Jun 24 10:09:01 1997

PMC-Sierra, Inc.

DOCUMENT NUMBER: PMC-960951	ISSUE: 2
TITLE: MPH WITH QDSX REFERENCE DESIGN HOST INTERFACE	DATE: SEP. 30, 96
ENGINEER: AARON GILROY	PAGE: 6 OF 6

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPR



LINE INTERFACE TRANSMIT/RECEIVE

DRAWING
TITLE=LINE_INTERFACE
ABBREV=LIU

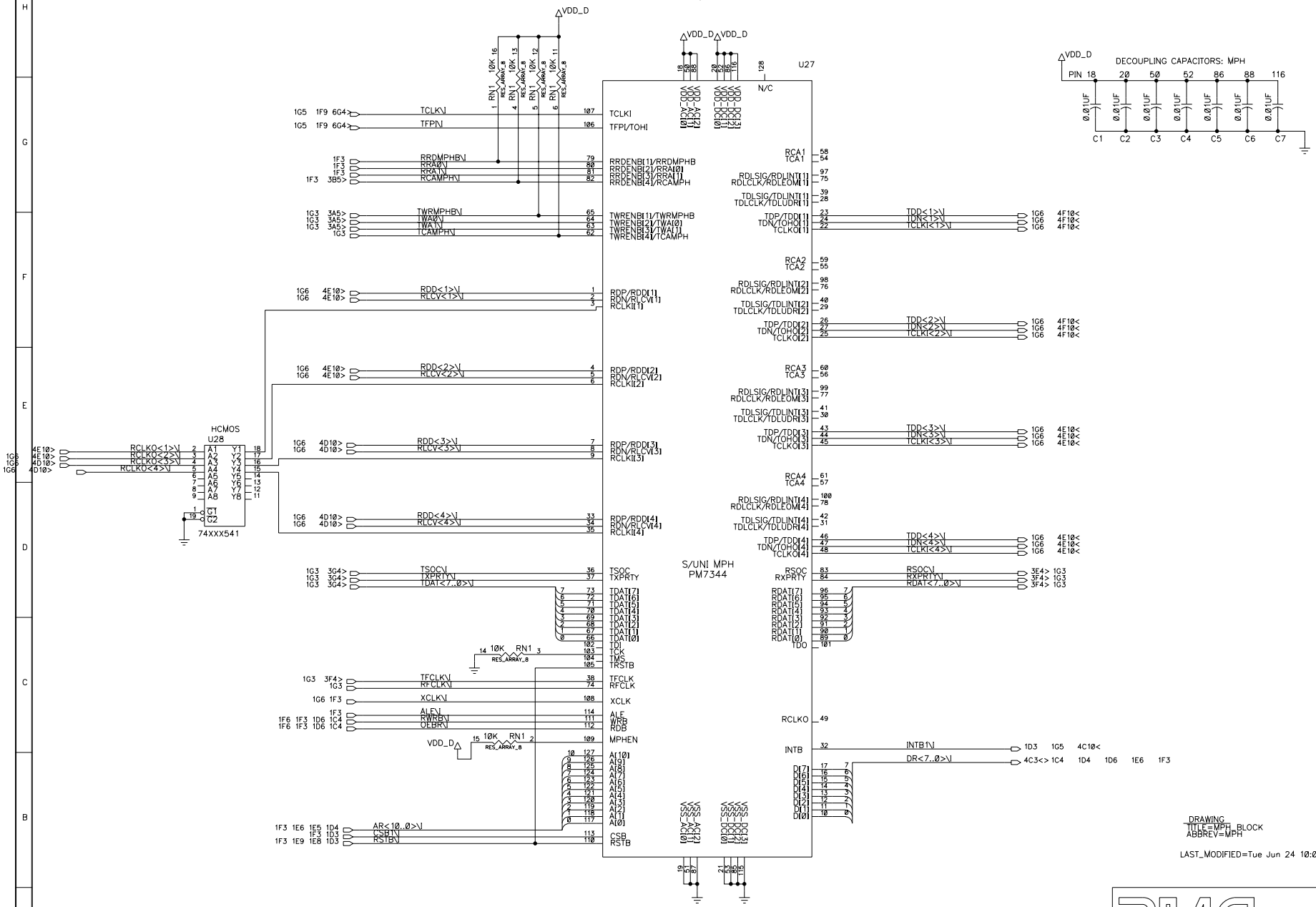
LAST_MODIFIED=Tue Jun 24 10:09:14 1997

PMC-Sierra, Inc.

DOCUMENT NUMBER: PMC-960951	ISSUE: 2
TITLE: MPH WITH QDSX REFERENCE DESIGN LINE INTERFACE TRANSMIT/RECEIVE	DATE: SEPT., 30, 96
ENGINEER: AARON GILROY	PAGE: 5 OF 6

MPH

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPR



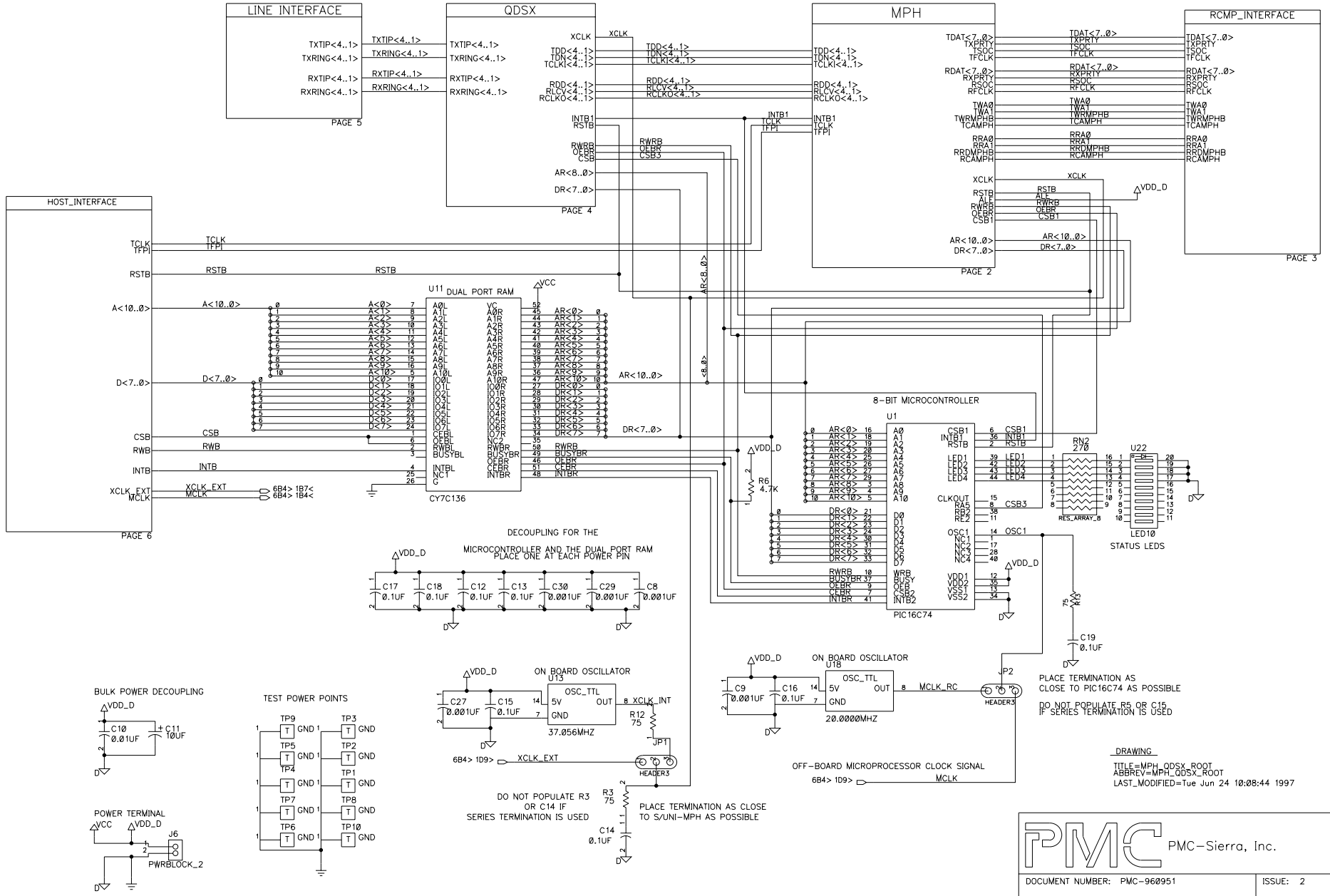
DRAWING
TITLE=MPH_BLOCK
ABBREV=MPH

LAST_MODIFIED=Tue Jun 24 10:09:54 1997

PMC-Sierra, Inc.

DOCUMENT NUMBER: PMC-960951	ISSUE: 2
TITLE: MPH WITH QDSX REFERENCE DESIGN MPH	DATE: SEPT. 30, 96
ENGINEER: AARON GILROY	PAGE: 2 OF 6

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPR
	2.0	SCHEMATICS CREATED	09/22/95	



DRAWING

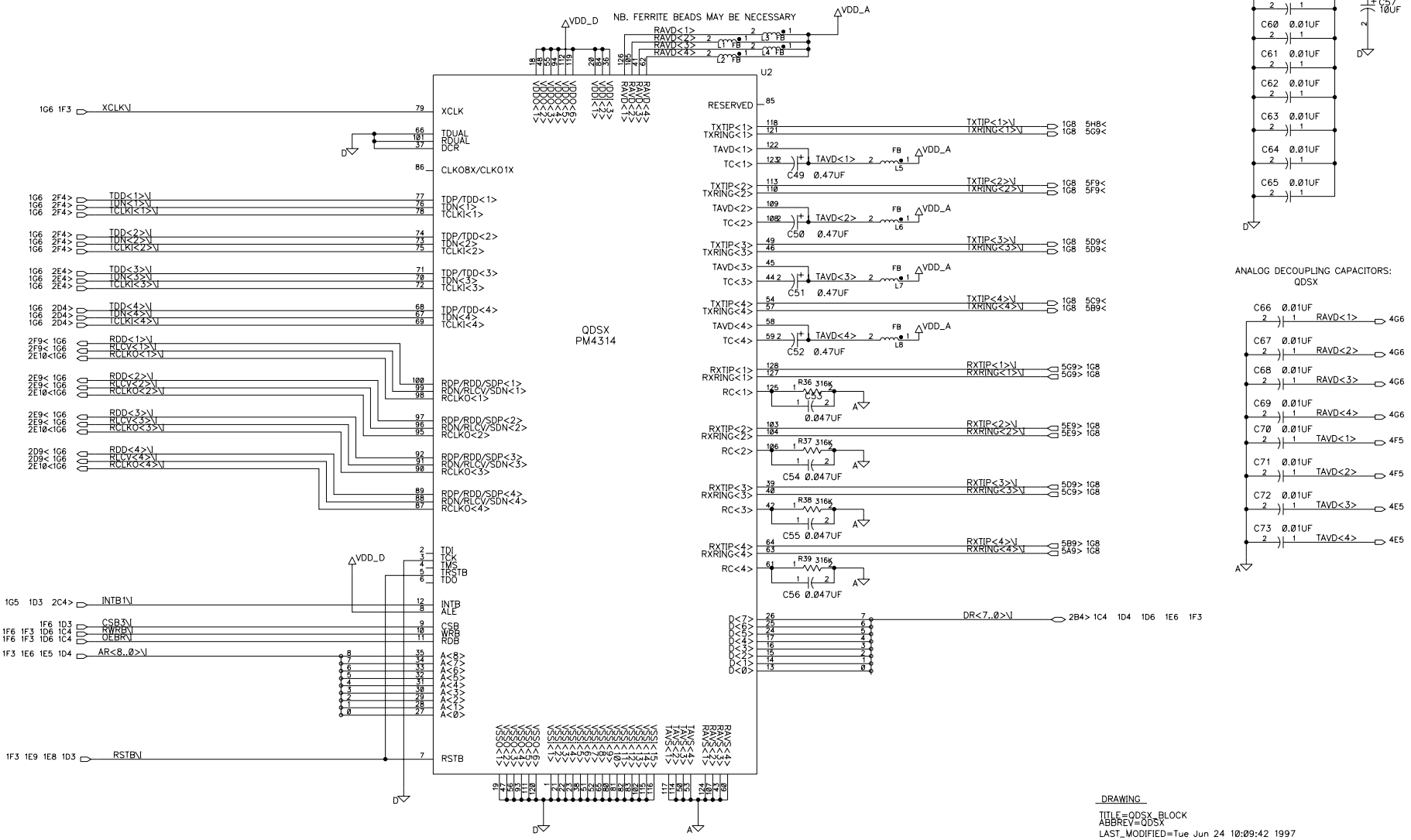
TITLE=MPH_QDSX_ROOT
ABBREV=MPH_QDSX_ROOT
LAST_MODIFIED=Tue Jun 24 10:08:44 1997

PMC-Sierra, Inc.

DOCUMENT NUMBER: PMC-960951	ISSUE: 2
TITLE: MPH WITH QDSX REFERENCE DESIGN ROOT DRAWING	DATE: SEP. 30, 96
ENGINEER: AARON GILROY	PAGE: 1 OF 6

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPR

QDSX



DIIGITAL DECOUPLING CAPACITORS:
QDSX

ANALOG DECOUPLING CAPACITORS:
QDSX

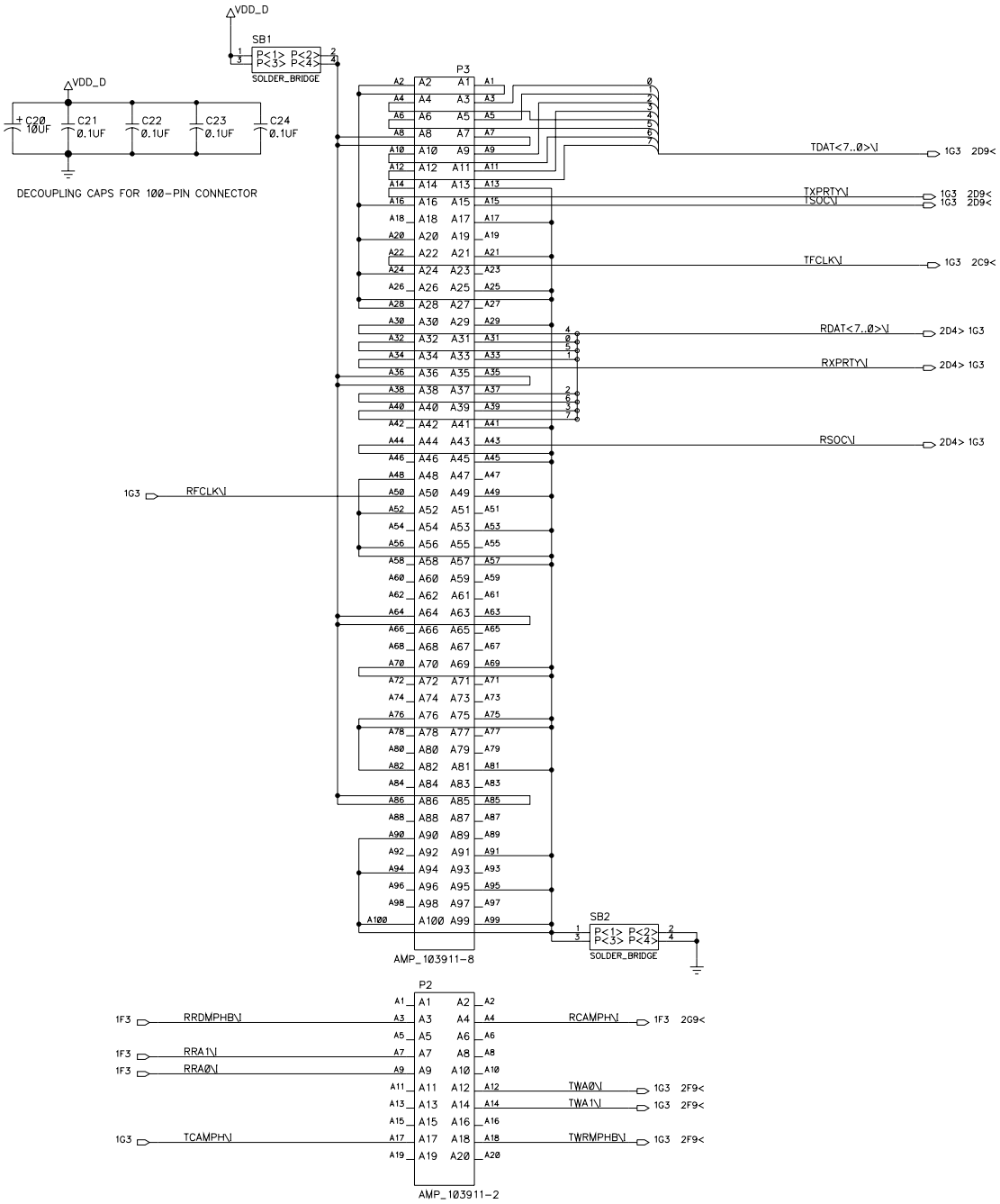
DRAWING

TITLE=QDSX_BLOCK
ABBREV=QDSX
LAST_MODIFIED=Tue Jun 24 10:09:42 1997

PMC-Sierra, Inc.

DOCUMENT NUMBER: PMC-960951	ISSUE: 2
TITLE: MPH WITH QDSX REFERENCE DESIGN QDSX	DATE: SEPT. 30, 96
ENGINEER: AARON GILROY	PAGE: 4 OF 6

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPR



DRAWING

TITLE=RCMP_INTERFACE
ABBREV=RCMP_INTERFACE
LAST_MODIFIED=Tue Jun 24 10:10:05 1997

PMC-Sierra, Inc.

DOCUMENT NUMBER: PMC-960951	ISSUE: 2
TITLE: MPH WITH QDSX REFERENCE DESIGN RCMP INTERFACE	DATE: SEPT. 30, 96
ENGINEER: AARON GILROY	PAGE: 3 OF 6

APPENDIX D: FIRMWARE

This appendix contains the source code for the firmware.

Note: It is the responsibility of the person(s) using or adapting this code to ensure that the resulting system operation complies with both standardized and proprietary requirements.

MACROS.INC file

```
;*****************************************
;* Commonly used general macro definitions *
;*****************************************


; select BANK 0 internal registers
BANK0 MACRO
      BCF   STATUS, RP0       ; select bank 0 (00h-7Fh)
      ENDM

; select BANK 0 internal registers
BANK1 MACRO
      BSF   STATUS, RP0 ; select bank 1 (80h-FFh)
      ENDM

; select program memory PAGE 0
PAGE0 MACRO
      BCF   PCLATH, 3   ; select page 0 (000h-7FFh)
      ENDM

; select program memory PAGE 1
PAGE1 MACRO
      BSF   PCLATH, 3   ; select page 1 (800h-FFFh)
      ENDM

; disable all interrupts
INTSOFF MACRO
      LOCAL CLEAR
CLEAR BCF   INTCON, GIE ; clear global interrupt enable bit
      BTFSC INTCON, GIE ; make sure it was cleared
      GOTO  CLEAR       ; (could have been interrupted)
      ENDM

; enable all interrupts
INTSON      MACRO
      BSF   INTCON, GIE
      ENDM
```

PIC16C74.INC file

```
        LIST
; P16C74.INC  Standard Header File, Version 1.00    Microchip Technology, Inc.
        NOLIST

; This header file defines configurations, registers, and other useful bits of
```

```
; information for the PIC16C74 microcontroller.  These names are taken to match
; the data sheets as closely as possible.

; Note that the processor must be selected before this file is
; included.  The processor may be selected the following ways:

;       1. Command line switch:
;               C:\ MPASM MYFILE.ASM /PIC16C74
;       2. LIST directive in the source file
;               LIST   P=PIC16C74
;       3. Processor Type entry in the MPASM full-screen interface

;==========================================================================
;
;       Revision History
;
;==========================================================================

;Rev:   Date:    Reason:

;1.00   10/31/95 Initial Release

;==========================================================================
;
;       Verify Processor
;
;==========================================================================

        IFNDEF __16C74
           MESSG "Processor-header file mismatch.  Verify selected processor."
        ENDIF

;==========================================================================
;
;       Register Definitions
;
;==========================================================================

W                           EQU     H'0000'
F                           EQU     H'0001'

;----- Register Files------------------------------------------------------

INDF                        EQU     H'0000'
TMR0                        EQU     H'0001'
PCL                         EQU     H'0002'
STATUS                      EQU     H'0003'
FSR                         EQU     H'0004'
PORTA                       EQU     H'0005'
PORTB                       EQU     H'0006'
PORTC                       EQU     H'0007'
PORTD                       EQU     H'0008'
PORTE                       EQU     H'0009'
PCLATH                      EQU     H'000A'
INTCON                      EQU     H'000B'
PIR1                        EQU     H'000C'
PIR2                        EQU     H'000D'
TMR1L                       EQU     H'000E'
TMR1H                       EQU     H'000F'
T1CON                       EQU     H'0010'
```

```
TMR2                            EQU     H'0011'
T2CON                           EQU     H'0012'
SSPBUF                          EQU     H'0013'
SSPCON                          EQU     H'0014'
CCPR1L                          EQU     H'0015'
CCPR1H                          EQU     H'0016'
CCP1CON                         EQU     H'0017'
RCSTA                           EQU     H'0018'
TXREG                           EQU     H'0019'
RCREG                           EQU     H'001A'
CCPR2L                          EQU     H'001B'
CCPR2H                          EQU     H'001C'
CCP2CON                         EQU     H'001D'
ADRES                           EQU     H'001E'
ADCON0                          EQU     H'001F'

OPTION_REG                      EQU     H'0081'
TRISA                           EQU     H'0085'
TRISB                           EQU     H'0086'
TRISC                           EQU     H'0087'
TRISD                           EQU     H'0088'
TRISE                           EQU     H'0089'
PIE1                            EQU     H'008C'
PIE2                            EQU     H'008D'
PCON                            EQU     H'008E'
PR2                             EQU     H'0092'
SSPADD                          EQU     H'0093'
SSPSTAT                         EQU     H'0094'
TXSTA                           EQU     H'0098'
SPBRG                           EQU     H'0099'
ADCON1                          EQU     H'009F'

;----- STATUS Bits --------------------------------------------------------

IRP                             EQU     H'0007'
RP1                             EQU     H'0006'
RP0                             EQU     H'0005'
NOT_TO                          EQU     H'0004'
NOT_PD                          EQU     H'0003'
Z                               EQU     H'0002'
DC                              EQU     H'0001'
C                               EQU     H'0000'

;----- INTCON Bits --------------------------------------------------------

GIE                             EQU     H'0007'
PEIE                            EQU     H'0006'
T0IE                            EQU     H'0005'
INTE                            EQU     H'0004'
RBIE                            EQU     H'0003'
T0IF                            EQU     H'0002'
INTF                            EQU     H'0001'
RBIF                            EQU     H'0000'

;----- PIR1 Bits ----------------------------------------------------------

PSPIF                           EQU     H'0007'
ADIF                            EQU     H'0006'
RCIF                            EQU     H'0005'
TXIF                            EQU     H'0004'
```

```
SSPIF                           EQU     H'0003'
CCP1IF                          EQU     H'0002'
TMR2IF                          EQU     H'0001'
TMR1IF                          EQU     H'0000'

;----- PIR2 Bits --------------------------------------------------------

CCP2IF                          EQU     H'0000'

;----- T1CON Bits -------------------------------------------------------

T1CKPS1                         EQU     H'0005'
T1CKPS0                         EQU     H'0004'
T1OSCEN                         EQU     H'0003'
NOT_T1SYNC                      EQU     H'0002'
T1INSYNC                        EQU     H'0002'    ; Backward compatibility only
TMR1CS                          EQU     H'0001'
TMR1ON                          EQU     H'0000'

;----- T2CON Bits -------------------------------------------------------

TOUTPS3                         EQU     H'0006'
TOUTPS2                         EQU     H'0005'
TOUTPS1                         EQU     H'0004'
TOUTPS0                         EQU     H'0003'
TMR2ON                          EQU     H'0002'
T2CKPS1                         EQU     H'0001'
T2CKPS0                         EQU     H'0000'

;----- SSPCON Bits ------------------------------------------------------

WCOL                            EQU     H'0007'
SSPOV                           EQU     H'0006'
SSPEN                           EQU     H'0005'
CKP                             EQU     H'0004'
SSPM3                           EQU     H'0003'
SSPM2                           EQU     H'0002'
SSPM1                           EQU     H'0001'
SSPM0                           EQU     H'0000'

;----- CCP1CON Bits -----------------------------------------------------

CCP1X                           EQU     H'0005'
CCP1Y                           EQU     H'0004'
CCP1M3                          EQU     H'0003'
CCP1M2                          EQU     H'0002'
CCP1M1                          EQU     H'0001'
CCP1M0                          EQU     H'0000'

;----- RCSTA Bits -------------------------------------------------------

SPEN                            EQU     H'0007'
RX9                             EQU     H'0006'
RC9                             EQU     H'0006'    ; Backward compatibility only
NOT_RC8                         EQU     H'0006'    ; Backward compatibility only
RC8_9                           EQU     H'0006'    ; Backward compatibility only
SREN                            EQU     H'0005'
CREN                            EQU     H'0004'
FERR                            EQU     H'0002'
OERR                            EQU     H'0001'
```

```
RX9D                            EQU     H'0000'
RCD8                            EQU     H'0000'    ; Backward compatibility only

;----- CCP2CON Bits --------------------------------------------------

CCP2X                           EQU     H'0005'
CCP2Y                           EQU     H'0004'
CCP2M3                          EQU     H'0003'
CCP2M2                          EQU     H'0002'
CCP2M1                          EQU     H'0001'
CCP2M0                          EQU     H'0000'

;----- ADCON0 Bits ---------------------------------------------------

ADCS1                       EQU     H'0007'
ADCS0                       EQU     H'0006'
CHS2                        EQU     H'0005'
CHS1                        EQU     H'0004'
CHS0                        EQU     H'0003'
GO                          EQU     H'0002'
NOT_DONE                    EQU     H'0002'
GO_DONE                     EQU     H'0002'
ADON                        EQU     H'0000'

;----- OPTION Bits ---------------------------------------------------

NOT_RBPU                        EQU     H'0007'
INTEDG                          EQU     H'0006'
T0CS                            EQU     H'0005'
T0SE                            EQU     H'0004'
PSA                             EQU     H'0003'
PS2                             EQU     H'0002'
PS1                             EQU     H'0001'
PS0                             EQU     H'0000'

;----- TRISE Bits ----------------------------------------------------

IBF                             EQU     H'0007'
OBF                             EQU     H'0006'
IBOV                            EQU     H'0005'
PSPMODE                         EQU     H'0004'
TRISE2                          EQU     H'0002'
TRISE1                          EQU     H'0001'
TRISE0                          EQU     H'0000'

;----- PIE1 Bits -----------------------------------------------------

PSPIE                           EQU     H'0007'
ADIE                            EQU     H'0006'
RCIE                            EQU     H'0005'
TXIE                            EQU     H'0004'
SSPIE                           EQU     H'0003'
CCP1IE                          EQU     H'0002'
TMR2IE                          EQU     H'0001'
TMR1IE                          EQU     H'0000'

;----- PIE2 Bits -----------------------------------------------------

CCP2IE                          EQU     H'0000'
```

```
;----- PCON Bits --------------------------------------------------------

NOT_POR                         EQU     H'0001'

;----- SSPSTAT Bits -----------------------------------------------------

D                               EQU     H'0005'
I2C_DATA                        EQU     H'0005'
NOT_A                           EQU     H'0005'
NOT_ADDRESS                     EQU     H'0005'
D_A                             EQU     H'0005'
DATA_ADDRESS                    EQU     H'0005'
P                               EQU     H'0004'
I2C_STOP                        EQU     H'0004'
S                               EQU     H'0003'
I2C_START                       EQU     H'0003'
R                               EQU     H'0002'
I2C_READ                        EQU     H'0002'
NOT_W                           EQU     H'0002'
NOT_WRITE                       EQU     H'0002'
R_W                             EQU     H'0002'
READ_WRITE                      EQU     H'0002'
UA                              EQU     H'0001'
BF                              EQU     H'0000'

;----- TXSTA Bits -------------------------------------------------------

CSRC                            EQU     H'0007'
TX9                             EQU     H'0006'
NOT_TX8                         EQU     H'0006'    ; Backward compatibility only
TX8_9                           EQU     H'0006'    ; Backward compatibility only
TXEN                            EQU     H'0005'
SYNC                            EQU     H'0004'
BRGH                            EQU     H'0002'
TRMT                            EQU     H'0001'
TX9D                            EQU     H'0000'
TXD8                            EQU     H'0000'    ; Backward compatibility only

;----- ADCON1 Bits ------------------------------------------------------

PCFG2                       EQU     H'0002'
PCFG1                       EQU     H'0001'
PCFG0                       EQU     H'0000'

;=========================================================================
;
;       RAM Definition
;
;=========================================================================

        __MAXRAM H'FF'
        __BADRAM H'8F'-H'91', H'95'-H'97', H'9A'-H'9E'

;=========================================================================
;
;       Configuration Bits
;
;=========================================================================

_CP_ALL                         EQU     H'3F8F'
```

```
_CP_75                          EQU     H'3F9F'
_CP_50                          EQU     H'3FAF'
_CP_OFF                         EQU     H'3FBF'
_PWRTE_ON                       EQU     H'3FBF'
_PWRTE_OFF                      EQU     H'3FB7'
_WDT_ON                         EQU     H'3FBF'
_WDT_OFF                        EQU     H'3FBB'
_LP_OSC                         EQU     H'3FBC'
_XT_OSC                         EQU     H'3FBD'
_HS_OSC                         EQU     H'3FBE'
_RC_OSC                         EQU     H'3FBF'

        LIST
```

MPH.ASM file

```
;*********************************************************
;* S/UNI-MPH Module PIC Firmware                         *
;* Version 0.00 : Octoher 1, 1996                        *
;* Author:      Greg Wynans                              *
;* Revision history:                                     *
;*      960924: Interface routines added for QDSX        *
;*                                                       *
;* Copyright 1996 PMC-Sierra, Inc                        *
;*********************************************************

      TITLE "PIC ASSEMBLY SOURCE CODE FOR MPH MODULE"

      PROCESSOR PIC16C74

      INCLUDE "p16c74.inc"
      INCLUDE "macros.inc"

      ERRORLEVEL 0,-306,-302          ; suppress page-crossing and
                              ; argument out of range messages

;*********************************
;* SPECIAL FEATURES FUSE SETTING *
;*********************************

      __CONFIG _CP_OFF&_PWRTE_ON&_WDT_OFF&_XT_OSC

;************************
;* WR_MPH_L DEFINITIONS *
;************************

;**********************
;* Revision constants *
;**********************

      CONSTANT        VER=0x00        ; version number
      CONSTANT        REV=0x00        ; revision number
      __IDLOCS        0x0000

;****************************************
;* PIC microcontroller register constants *
;****************************************

; define BANK 0 of microcontroller user registers
```

```
REGISTERS
      CBLOCK  20
            DATA_REG        ; data register for micro-bus transfers
            ADDR_MPH_HI     ; MSB of address reg for MPH accesses
            ADDR_MPH_LO     ; LSB of address reg for MPH accesses
            ADDR_RAM_HI     ; MSB of address reg for RAM accesses
            ADDR_RAM_LO     ; LSB of address reg for RAM accesses
            TEMP_W          ; temporary copy of working register
            TEMP_STATUS     ; temporary copy of status register
            TEMP_PCLATH     ; temporary copy of PCLATH register
            TIME_COUNT_H    ; MSB of 1 second timer counter
            TIME_COUNT_L    ; LSB of 1 second timer counter
            SCRATCH         ; used as a macro scratch pad and general reg.
            WORK            ; general working register
            WORK1           ; general working register
            WORK2           ; general working register
                        ; NOT to be used in interrupt handling
            WORK3           ; general working register
                        ; NOT to be used in interrupt handling
            WORK4           ; general working register
                        ; NOT to be used in interrupt handling
            WORK5           ; general working register
                        ; NOT to be used in interrupt handling
                        ; NOTE that SETUP_MADDR uses this register
            TIMEFLAG        ;  bit 0: set every sec. by timer int.

            QUADRANT        ; used to store the interrupting quadrant (0-3)
                        ;  is set at the beginning of the MPH int.
                        ;  handler and is used through the routine
            INT1, INT2      ;  first and second interrupt source
                        ;  from the MPH. INT1 is also used for QDSX
            INT_STATUS      ; interrupt status byte for a block in MPH or QDSX
            TEMP_RFDL_DATA  ; stores data byte read from MPH
            TEMP_RFDL_STATUS        ; stores status byte read from MPH
            INT_STATUS_SP   ; temporary copy of INT_STATUS
            TEMP_DATA_REG   ; temporary copy of DATA_REG
            TEMP_FSR        ; temporary copy of FSR reg
            PRM_COUNTER     ; PMON report message modulo 4 counter
            Q1_FLAGS        ; quadrant 1 bit flags (see bit field constants)
            Q2_FLAGS        ; quadrant 2 bit flags
            Q3_FLAGS        ; quadrant 3 bit flags
            Q4_FLAGS        ; quadrant 4 bit flags
            SEF             ; indicate SEF. bit n set if Qn has SEF (for PMON)
            SEFQUAD         ; indicate quadrant with bit n for Qn
            CLKSTAT     ;indicate status of clock for clock activity monitoring
            ADDR_QDSX_HI     ; MSB of address reg for QDSX accesses
            ADDR_QDSX_LO     ; LSB of address reg for QDSX accesses

      ENDC

      ; Bit field for quadrant bit flags
      CBLOCK 0
            RFDL_ACTIVE     ; is this quadrant's RFDL block active?
            LL_REQ          ; used to store a line loopback request
                        ; until link is idled
            XFDL_BUSY       ; is this quadrant's XFDL busy transmitting
                        ; a packet (user or performance report)?
            XFDL_PENDING    ; is a packet pending (user or performance
                        ; report)?
            XFDL_USR_OR_PRM ; is the currently transmitting packet
```

```
                    ; a user packet or a performance report?
                    ; 0 = user, 1 = PRM
ENDC

CBLOCK 40
      Q1_RFDL_STAT    ; Quadrant 1 RFDL packet status
      Q2_RFDL_STAT    ; Quadrant 2 RFDL packet status
      Q3_RFDL_STAT    ; Quadrant 3 RFDL packet status
      Q4_RFDL_STAT    ; Quadrant 4 RFDL packet status
      Q1_RFDL_PLEN    ; Quadrant 1 RFDL packet length
      Q2_RFDL_PLEN    ; Quadrant 2 RFDL packet length
      Q3_RFDL_PLEN    ; Quadrant 3 RFDL packet length
      Q4_RFDL_PLEN    ; Quadrant 4 RFDL packet length
      Q1_XFDL_PLEN    ; Quadrant 1 XFDL packet length
      Q2_XFDL_PLEN    ; Quadrant 2 XFDL packet length
      Q3_XFDL_PLEN    ; Quadrant 3 XFDL packet length
      Q4_XFDL_PLEN    ; Quadrant 4 XFDL packet length
      Q1_XFDL_RPTR    ; Quadrant 1 XFDL RAM pointer
      Q2_XFDL_RPTR    ; Quadrant 2 XFDL RAM pointer
      Q3_XFDL_RPTR    ; Quadrant 3 XFDL RAM pointer
      Q4_XFDL_RPTR    ; Quadrant 4 XFDL RAM pointer
      Q1_XFDL_TO      ; Quadrant 1 XFDL packet restart timeout
      Q2_XFDL_TO      ; Quadrant 2 XFDL packet restart timeout
      Q3_XFDL_TO      ; Quadrant 3 XFDL packet restart timeout
      Q4_XFDL_TO      ; Quadrant 4 XFDL packet restart timeout
ENDC

CBLOCK 54
      Q1_PRM_PTR      ; pointer to next PRM byte to xmit
      Q2_PRM_PTR      ; pointer to next PRM byte to xmit
      Q3_PRM_PTR      ; pointer to next PRM byte to xmit
      Q4_PRM_PTR      ; pointer to next PRM byte to xmit
ENDC

CBLOCK 60
      Q1_PRM_1B1      ; Quadrant 1 Performance Report 1 (byte 1)
      Q1_PRM_1B2      ; Quadrant 1 Performance Report 1 (byte 2)
      Q1_PRM_2B1      ; Quadrant 1 Performance Report 2 (byte 1)
      Q1_PRM_2B2      ; Quadrant 1 Performance Report 2 (byte 2)
      Q1_PRM_3B1      ; Quadrant 1 Performance Report 3 (byte 1)
      Q1_PRM_3B2      ; Quadrant 1 Performance Report 3 (byte 2)
      Q1_PRM_4B1      ; Quadrant 1 Performance Report 4 (byte 1)
      Q1_PRM_4B2      ; Quadrant 1 Performance Report 4 (byte 2)
      Q2_PRM_1B1      ; Quadrant 2 Performance Report 1 (byte 1)
      Q2_PRM_1B2      ; Quadrant 2 Performance Report 1 (byte 2)
      Q2_PRM_2B1      ; Quadrant 2 Performance Report 2 (byte 1)
      Q2_PRM_2B2      ; Quadrant 2 Performance Report 2 (byte 2)
      Q2_PRM_3B1      ; Quadrant 2 Performance Report 3 (byte 1)
      Q2_PRM_3B2      ; Quadrant 2 Performance Report 3 (byte 2)
      Q2_PRM_4B1      ; Quadrant 2 Performance Report 4 (byte 1)
      Q2_PRM_4B2      ; Quadrant 2 Performance Report 4 (byte 2)
      Q3_PRM_1B1      ; Quadrant 3 Performance Report 1 (byte 1)
      Q3_PRM_1B2      ; Quadrant 3 Performance Report 1 (byte 2)
      Q3_PRM_2B1      ; Quadrant 3 Performance Report 2 (byte 1)
      Q3_PRM_2B2      ; Quadrant 3 Performance Report 2 (byte 2)
      Q3_PRM_3B1      ; Quadrant 3 Performance Report 3 (byte 1)
      Q3_PRM_3B2      ; Quadrant 3 Performance Report 3 (byte 2)
      Q3_PRM_4B1      ; Quadrant 3 Performance Report 4 (byte 1)
      Q3_PRM_4B2      ; Quadrant 3 Performance Report 4 (byte 2)
      Q4_PRM_1B1      ; Quadrant 4 Performance Report 1 (byte 1)
```

```
            Q4_PRM_1B2      ; Quadrant 4 Performance Report 1 (byte 2)
            Q4_PRM_2B1      ; Quadrant 4 Performance Report 2 (byte 1)
            Q4_PRM_2B2      ; Quadrant 4 Performance Report 2 (byte 2)
            Q4_PRM_3B1      ; Quadrant 4 Performance Report 3 (byte 1)
            Q4_PRM_3B2      ; Quadrant 4 Performance Report 3 (byte 2)
            Q4_PRM_4B1      ; Quadrant 4 Performance Report 4 (byte 1)
            Q4_PRM_4B2      ; Quadrant 4 Performance Report 4 (byte 2)
      ENDC

;**************************
;* PORTB bit LED constants *
;**************************

      CONSTANT        LED1=3  ; LED1 bit
      CONSTANT        LED2=5  ; LED2 bit
      CONSTANT        LED3=6  ; LED3 bit
      CONSTANT        LED4=7  ; LED4 bit

;**************************
;* MPH Register constants *
;**************************

; MPH initialization registers

      CONSTANT        RECV_CONFIG=0x00        ; see the MPH data book
      CONSTANT        XMIT_CONFIG=0x01        ;  for more detailed
      CONSTANT        RINT_CONFIG=0x03        ;  information about these
      CONSTANT        XINT_CONFIG=0x04        ;  registers.
      CONSTANT        XCLK_OPTION=0x07
      CONSTANT        CDRC_CONFIG=0x10
      CONSTANT        FRMR_CONFIG=0x1C
      CONSTANT        XFDL_CONFIG=0x34
      CONSTANT        RFDL_CONFIG=0x3C
      CONSTANT        IBCD_CONFIG=0x3C
      CONSTANT        T1TRAN_CONFIG=0x40      ;was XBAS_CONFIG in TQUAD
      CONSTANT        ALMI_CONFIG=0x14
      CONSTANT        RXCP_CONTROL=0x70
      CONSTANT        RXCP_FRAME_CONTROL=0x71
      CONSTANT        TXCP_CONTROL=0x88


; MPH interrupt enable registers
      CONSTANT        CDRC_IE=0x11
      CONSTANT        FRMR_IE=0x1D
      CONSTANT        RBOC_IE=0x30
      CONSTANT        ALMI_IE=0x15
      CONSTANT        IBCD_IE=0x3D
      CONSTANT        RFDL_IE=0x39
      CONSTANT        RXCP_IE=0x72
      CONSTANT        TXCP_IE=0X89

; MPH PMON Registers
      CONSTANT        FER1=0x049
      CONSTANT        FER2=0x149
      CONSTANT        FER3=0x249
      CONSTANT        FER4=0x349

      CONSTANT        FEBEL1=0x04A
      CONSTANT        FEBEL2=0x14A
      CONSTANT        FEBEL3=0x24A
```

```
CONSTANT        FEBEL4=0x34A

CONSTANT        FEBEM1=0x04B
CONSTANT        FEBEM2=0x14B
CONSTANT        FEBEM3=0x24B
CONSTANT        FEBEM4=0x34B

CONSTANT        CRCL1=0x04C
CONSTANT        CRCL2=0x14C
CONSTANT        CRCL3=0x24C
CONSTANT        CRCL4=0x34C

CONSTANT        CRCM1=0x04D
CONSTANT        CRCM2=0x14D
CONSTANT        CRCM3=0x24D
CONSTANT        CRCM4=0x34D

CONSTANT        LCVL1=0x04E
CONSTANT        LCVL2=0x14E
CONSTANT        LCVL3=0x24E
CONSTANT        LCVL4=0x34E

CONSTANT        LCVM1=0x04F
CONSTANT        LCVM2=0x14F
CONSTANT        LCVM3=0x24F
CONSTANT        LCVM4=0x34F

CONSTANT        RXCP_UHCSL1=0x64
CONSTANT        RXCP_UHCSL2=0x164
CONSTANT        RXCP_UHCSL3=0x264
CONSTANT        RXCP_UHCSL4=0x364

CONSTANT        RXCP_UHCSM1=0x65
CONSTANT        RXCP_UHCSM2=0x165
CONSTANT        RXCP_UHCSM3=0x265
CONSTANT        RXCP_UHCSM4=0x365

CONSTANT        RXCP_CHCSL1=0x68
CONSTANT        RXCP_CHCSL2=0x168
CONSTANT        RXCP_CHCSL3=0x268
CONSTANT        RXCP_CHCSL4=0x368

CONSTANT        RXCP_CHCSM1=0x69
CONSTANT        RXCP_CHCSM2=0x169
CONSTANT        RXCP_CHCSM3=0x269
CONSTANT        RXCP_CHCSM4=0x369

CONSTANT        RXCP_IDLEL1=0x6A
CONSTANT        RXCP_IDLEL2=0x16A
CONSTANT        RXCP_IDLEL3=0x26A
CONSTANT        RXCP_IDLEL4=0x36A

CONSTANT        RXCP_IDLEM1=0x6B
CONSTANT        RXCP_IDLEM2=0x16B
CONSTANT        RXCP_IDLEM3=0x26B
CONSTANT        RXCP_IDLEM4=0x36B

CONSTANT        RXCP_RECVL1=0x6C
CONSTANT        RXCP_RECVL2=0x16C
CONSTANT        RXCP_RECVL3=0x26C
```

```
      CONSTANT        RXCP_RECVL4=0x36C

      CONSTANT        RXCP_RECVM1=0x6D
      CONSTANT        RXCP_RECVM2=0x16D
      CONSTANT        RXCP_RECVM3=0x26D
      CONSTANT        RXCP_RECVM4=0x36D

      CONSTANT        TXCP_XMITL1=0x6E
      CONSTANT        TXCP_XMITL2=0x16E
      CONSTANT        TXCP_XMITL3=0x26E
      CONSTANT        TXCP_XMITL4=0x36E

      CONSTANT        TXCP_XMITM1=0x6F
      CONSTANT        TXCP_XMITM2=0x16F
      CONSTANT        TXCP_XMITM3=0x26F
      CONSTANT        TXCP_XMITM4=0x36F


; MPH interrupt status registers
      CONSTANT        INT_ID=0x00D    ; shows which quadrant interrupted
      CONSTANT        INT1_1=0x008    ; interrupt source 1
      CONSTANT        INT2_1=0x009    ; interrupt source 2
      CONSTANT        ALMI_IS=0x016   ; ALMI interrupt status
      CONSTANT        CDRC_IS=0x012   ; CDRI interrupt status
      CONSTANT        FRMR_IS=0x01E   ; FRMR interrupt status
      CONSTANT        RBOC_IS=0x031   ; RBOC interrupt/code status
      CONSTANT        IBCD_IS=0x3D    ; IBCD interrupt status
      CONSTANT        RFDL_IS=0x39    ; RFDL interrupt status
      CONSTANT        XFDL_IS=0x35    ; XFDL interrupt status
;        CONSTANT        ELST_IS=0x1D    ; ELST interrupt enable/status
      CONSTANT        RXCP_IS=0x72    ; RXCP interrupt enable/status
      CONSTANT        TXCP_IS=0x89    ; TXCP interrupt enable/status/control
      CONSTANT        RXCPFC_IS=0x71  ; RXCP framing control (for LCD int)

; other MPH registers
      CONSTANT        T1TRAN_ALM=0x41 ; transmit yellow or AIS alarm
      CONSTANT        CLKAMON=0x0E      ; clock activity monitor
      CONSTANT        GLOBAL_PMON=0x0C      ; global PMON update
      CONSTANT        MSTR_DIAG=0x0A        ; master diagnostic reg.
      CONSTANT        DL_OPT=0x02           ; datalink options reg.
      CONSTANT        RFDL_DATA=0x3B        ; RFDL reveived data reg.
      CONSTANT        RFDL_STATUS=0x3A      ; RFDL received data status reg.
      CONSTANT        XFDL_DATA=0x36        ; XFDL data register
      CONSTANT        XBOC_CODE=0x57        ; XBOC code register
      CONSTANT        IBCD_AC=0x3E          ; IBCD activate code
      CONSTANT        IBCD_DC=0x3F          ; IBCD deactivate code
      CONSTANT        RXCP_IDLEPAT_H1=0x73      ; RXCP Idle/unassigned pattern
H1 octet
      CONSTANT        RXCP_IDLEPAT_H2=0x74      ; RXCP Idle/unassigned pattern
H2 octet
      CONSTANT        RXCP_IDLEPAT_H3=0x75      ; RXCP Idle/unassigned pattern
H3 octet
      CONSTANT        RXCP_IDLEPAT_H4=0x76      ; RXCP Idle/unassigned pattern
H4 octet
      CONSTANT         RXCP_IDLEMASK_H1=0x77       ; RXCP Idle/unassigned mask
H1 octet
      CONSTANT         RXCP_IDLEMASK_H2=0x78       ; RXCP Idle/unassigned mask
H2 octet
```

```
      CONSTANT         RXCP_IDLEMASK_H3=0x79        ; RXCP Idle/unassigned mask
H3 octet
      CONSTANT         RXCP_IDLEMASK_H4=0x7A        ; RXCP Idle/unassigned mask
H4 octet
      CONSTANT        RXCP_USERPAT_H1=0x7B       ; RXCP user pattern H1 octet
      CONSTANT        RXCP_USERPAT_H2=0x7C       ; RXCP user pattern H2 octet
      CONSTANT        RXCP_USERPAT_H3=0x7D       ; RXCP user pattern H3 octet
      CONSTANT        RXCP_USERPAT_H4=0x7E       ; RXCP user pattern H4 octet
      CONSTANT        RXCP_USERMASK_H1=0x7F       ; RXCP user mask H1 octet
      CONSTANT        RXCP_USERMASK_H2=0x80       ; RXCP user mask H2 octet
      CONSTANT        RXCP_USERMASK_H3=0x81       ; RXCP user mask H3 octet
      CONSTANT        RXCP_USERMASK_H4=0x82       ; RXCP user mask H4 octet
      CONSTANT        TXCP_IDLEPAT_H1=0x8A       ; TXCP Idle/unassigned pattern
H1 octet
      CONSTANT        TXCP_IDLEPAT_H2=0x8B       ; TXCP Idle/unassigned pattern
H2 octet
      CONSTANT        TXCP_IDLEPAT_H3=0x8C       ; TXCP Idle/unassigned pattern
H3 octet
      CONSTANT        TXCP_IDLEPAT_H4=0x8D       ; TXCP Idle/unassigned pattern
H4 octet
      CONSTANT        TXCP_IDLEPAT_H5=0x8E       ; TXCP Idle/unassigned pattern
H5 octet
      CONSTANT         TXCP_IDLEPAY=0x8F           ; TXCP Idle/unassigned cell
payload


;*****************************
;* QDSX Register constants *
;*****************************
; Constants are suffixed with a Q to differentiate from MPH counterparts
; QDSX initialization registers

      CONSTANT        RECV_CONFIGQ=0x00        ; see the QDSX data book
      CONSTANT        XMIT_CONFIGQ=0x01        ;  for more detailed
      CONSTANT        TOPS_CONFIGQ=0x09        ;  information about these
      CONSTANT        TOPS_TM_CONFIGQ=0x0A     ;  registers.
      CONSTANT        LCODE_CONFIGQ=0x0B
      CONSTANT        CDRC_CONFIGQ=0x10
      CONSTANT        DJAT_CONFIGQ=0x1F
      CONSTANT        IBCD_CONFIGQ=0x20
      CONSTANT        XIBC_CONTROLQ=0x24
      CONSTANT        PRSG_CONFIGQ=0x27
      CONSTANT        PRSM_CONTROLQ=0x29
      CONSTANT        XPLS_CONFIGQ=0x2C
      CONSTANT        XPLS_CONTROLQ=0x2D
      CONSTANT        RSLC_CONFIGQ=0x30

; QDSX interrupt enable registers
      CONSTANT        CDRC_IEQ=0x11
      CONSTANT        LCV_PMON_IEQ=0x14
      CONSTANT        IBCD_IEQ=0x21
      CONSTANT        RSLC_IEQ=0x31

; QDSX PMON Registers

      CONSTANT        GLOBAL_PMONQ=0x007
      CONSTANT        DIAGQ=0x005
      CONSTANT        MTESTQ=0x006

      CONSTANT        LCVL1Q=0x01A
```

```
      CONSTANT        LCVL2Q=0x05A
      CONSTANT        LCVL3Q=0x09A
      CONSTANT        LCVL4Q=0x0DA

      CONSTANT        LCVM1Q=0x01B
      CONSTANT        LCVM2Q=0x05B
      CONSTANT        LCVM3Q=0x09B
      CONSTANT        LCVM4Q=0x0DB

      CONSTANT        PRSML1Q=0x02A
      CONSTANT        PRSML2Q=0x06A
      CONSTANT        PRSML3Q=0x0AA
      CONSTANT        PRSML4Q=0x0EA

      CONSTANT        PRSMM1Q=0x02B
      CONSTANT        PRSMM2Q=0x06B
      CONSTANT        PRSMM3Q=0x0AB
      CONSTANT        PRSMM4Q=0x0EB

; QDSX interrupt status registers
      CONSTANT        INT_IDQ=0x008    ; shows which quadrant interrupted
      CONSTANT        INTQ=0x003    ; interrupt source
      CONSTANT        CDRC_ISQ=0x012   ; CDRC interrupt status
      CONSTANT        LCV_PMON_ISQ=0x014   ; LCV_PMON interrupt status
      CONSTANT        DJAT_ISQ=0x01C   ; RBOC interrupt/code status
      CONSTANT        IBCD_ISQ=0x21    ; IBCD interrupt status
      CONSTANT        PRSM_ISQ=0x29    ; PRSM interrupt status
      CONSTANT        XPLS_ISQ=0x2D    ; XPLS interrupt status
      CONSTANT        RLSC_ISQ=0x31    ; RLSC interrupt status

; Other QDSX registers
      CONSTANT        IBCD_ACQ=0x22    ; IBCD activate code for QDSX
      CONSTANT        IBCD_DCQ=0x23    ; IBCD deactivate code for QDSX

;*****************************************
;* RAM memory location/register constants *
;*****************************************

      ; PMON Registers
      ; these memory locations reflect their MPH counterparts
      ; see the associated MPH register description for more info.


      CONSTANT        FER1_RAM=0x400
      CONSTANT        FER2_RAM=0x413
      CONSTANT        FER3_RAM=0x426
      CONSTANT        FER4_RAM=0x439

      CONSTANT        FEBEL1_RAM=0x401
      CONSTANT        FEBEL2_RAM=0x414
      CONSTANT        FEBEL3_RAM=0x427
      CONSTANT        FEBEL4_RAM=0x43A

      CONSTANT        FEBEM1_RAM=0x402
      CONSTANT        FEBEM2_RAM=0x415
      CONSTANT        FEBEM3_RAM=0x428
      CONSTANT        FEBEM4_RAM=0x43B

      CONSTANT        CRCL1_RAM=0x403
      CONSTANT        CRCL2_RAM=0x416
```

```
CONSTANT        CRCL3_RAM=0x429
CONSTANT        CRCL4_RAM=0x43C

CONSTANT        CRCM1_RAM=0x404
CONSTANT        CRCM2_RAM=0x417
CONSTANT        CRCM3_RAM=0x42A
CONSTANT        CRCM4_RAM=0x43D

CONSTANT        LCVL1_RAM=0x405
CONSTANT        LCVL2_RAM=0x418
CONSTANT        LCVL3_RAM=0x42B
CONSTANT        LCVL4_RAM=0x43E

CONSTANT        LCVM1_RAM=0x406
CONSTANT        LCVM2_RAM=0x419
CONSTANT        LCVM3_RAM=0x42C
CONSTANT        LCVM4_RAM=0x43F

CONSTANT        RXCP_UHCSL1_RAM=0x407
CONSTANT        RXCP_UHCSL2_RAM=0x41A
CONSTANT        RXCP_UHCSL3_RAM=0x42D
CONSTANT        RXCP_UHCSL4_RAM=0x440

CONSTANT        RXCP_UHCSM1_RAM=0x408
CONSTANT        RXCP_UHCSM2_RAM=0x41B
CONSTANT        RXCP_UHCSM3_RAM=0x42E
CONSTANT        RXCP_UHCSM4_RAM=0x441

CONSTANT        RXCP_CHCSL1_RAM=0x409
CONSTANT        RXCP_CHCSL2_RAM=0x41C
CONSTANT        RXCP_CHCSL3_RAM=0x42F
CONSTANT        RXCP_CHCSL4_RAM=0x442

CONSTANT        RXCP_CHCSM1_RAM=0x40A
CONSTANT        RXCP_CHCSM2_RAM=0x41D
CONSTANT        RXCP_CHCSM3_RAM=0x430
CONSTANT        RXCP_CHCSM4_RAM=0x443

CONSTANT        RXCP_IDLEL1_RAM=0x40B
CONSTANT        RXCP_IDLEL2_RAM=0x41E
CONSTANT        RXCP_IDLEL3_RAM=0x431
CONSTANT        RXCP_IDLEL4_RAM=0x444

CONSTANT        RXCP_IDLEM1_RAM=0x40C
CONSTANT        RXCP_IDLEM2_RAM=0x41F
CONSTANT        RXCP_IDLEM3_RAM=0x432
CONSTANT        RXCP_IDLEM4_RAM=0x445

CONSTANT        RXCP_RECVL1_RAM=0x40D
CONSTANT        RXCP_RECVL2_RAM=0x420
CONSTANT        RXCP_RECVL3_RAM=0x433
CONSTANT        RXCP_RECVL4_RAM=0x446

CONSTANT        RXCP_RECVM1_RAM=0x40E
CONSTANT        RXCP_RECVM2_RAM=0x421
CONSTANT        RXCP_RECVM3_RAM=0x434
CONSTANT        RXCP_RECVM4_RAM=0x447

CONSTANT        TXCP_XMITL1_RAM=0x40F
CONSTANT        TXCP_XMITL2_RAM=0x422
```

```
CONSTANT        TXCP_XMITL3_RAM=0x435
CONSTANT        TXCP_XMITL4_RAM=0x448

CONSTANT        TXCP_XMITM1_RAM=0x410
CONSTANT        TXCP_XMITM2_RAM=0x423
CONSTANT        TXCP_XMITM3_RAM=0x436
CONSTANT        TXCP_XMITM4_RAM=0x449


; RAM RXDL/XFDL register
CONSTANT        RAM_CHAN_STAT=0x7E      ; FDL channel status
CONSTANT        RAM_PCT_LEN=0x7F        ; FDL packet length

; Version/Revision number
CONSTANT        VER_ADDR=0x7DE  ; address in RAM to store version #
CONSTANT        REV_ADDR=0x7DF  ; address in RAM to store revision #

; Ram comm. registers
CONSTANT        MAILIN=7FF      ; RAM mailbox reg. (HOST->PIC)
CONSTANT        MAILOUT=7FE     ; RAM mailbox reg. (PIC->HOST)
CONSTANT        INT_QUAD=7F0    ; this register reflects the
                        ;  the interrupting quadrant
                        ;  from the PIC->HOST
CONSTANT        RAM_RBOC=7F1    ; storage location BOCs PIC->HOST
CONSTANT        RAM_XBOC=7E0    ; storage location BOCs HOST->PIC
CONSTANT        RFDL_QUAD=7F8   ; current RFDL interrupting quad.
CONSTANT        XFDL_QUAD=7F9   ; current XFDL interrupting quad.
CONSTANT        SERVICE_QUAD=7FA        ; this is the RAM location
                        ; used to communicate from HOST->PIC
                        ; for send boc and idle boc commands

; Register access. registers
; these are the registers in RAM that are used to communicate
; between the MPH and the host when performing individual
; register reads and writes.
CONSTANT        REG_DATA=0x7FB          ; data xfer reg.
CONSTANT        REG_ADR_LO=0x7FC        ; low reg. address
CONSTANT        REG_ADR_HI=0x7FD        ; hi reg. address

; Timer interrupt constants (for 1 second period)
CONSTANT        ONE_SEC_L=0x01
CONSTANT        ONE_SEC_H=0x00

; FDL Constants
CONSTANT        MAX_FDL_PLEN=0x7C       ; max packet length
CONSTANT        XFDL_MAX_RESTARTS=0x0A  ; max packet Tx restarts

; MailBox communication constants: PIC -> HOST
; the _A postfix indicates an assertion signal
; the _C postfix indicates a clear signal
; eg. LOS_A is the code for LOS asserted
;     LOS_C is the code for LOS cleared
CONSTANT        PDV=0x01        ; pulse density violation
CONSTANT        LOS_A=0x02      ; LOS asserted
CONSTANT        LOS_C=0x03      ; LOS cleared
CONSTANT        INFR_A=0x04     ; INFR asserted
CONSTANT        INFR_C=0x05     ; INFR cleared
CONSTANT        AIS_A=0x06      ; AIS asserted
CONSTANT        AIS_C=0x07      ; AIS cleared
CONSTANT        YEL_A=0x08      ; YELLOW alarm asserted
```

```
CONSTANT        YEL_C=0x09      ; YELLOW alarm cleared
CONSTANT        FER=0x0A        ; Framing error
CONSTANT        SFER=0x0B       ; Severe framing error
CONSTANT        RED_A=0x0C      ; RED alarm asserted
CONSTANT        RED_C=0x0D      ; RED alarm cleared
CONSTANT        BOC_VALID=0x0E  ; indicates a valid BOC
CONSTANT        BOC_IDLE=0x0F   ; indicates a return to idle code
CONSTANT        RFDL_NEW_Q1=0x10        ; indicates a new packet
                        ; has arrived on quadrant 1
CONSTANT        RFDL_NEW_Q2=0x11        ; indicates a new packet
                        ; has arrived on quadrant 2
CONSTANT        RFDL_NEW_Q3=0x12        ; indicates a new packet
                        ; has arrived on quadrant 3
CONSTANT        RFDL_NEW_Q4=0x13        ; indicates a new packet
                        ; has arrived on quadrant 4
CONSTANT        IBC_LLA=0x14            ; indicates line loopback
                        ; activated because in-band
                        ; code detected
CONSTANT        IBC_LLD=0x15            ; indicates line loopback
                        ; deactivated because in-band
                        ; code detected
CONSTANT        XFDL_DONE_Q1=0x20       ; indicates an XFDL has been
                ; successfully xmitted from quadrant 1
CONSTANT        XFDL_DONE_Q2=0x21       ; indicates an XFDL has been
                ; successfully xmitted from quadrant 2
CONSTANT        XFDL_DONE_Q3=0x22       ; indicates an XFDL has been
                ; successfully xmitted from quadrant 3
CONSTANT        XFDL_DONE_Q4=0x23       ; indicates an XFDL has been
                ; successfully xmitted from quadrant 4
CONSTANT        LOCA_A=0x25       ;loss of clock activity asserted
CONSTANT        LOCA_C=0x26       ;loss of clock activity cleared
CONSTANT        LCD_A=0x27       ;loss of cell delineation asserted
CONSTANT        LCD_C=0x28       ;loss of cell delineation cleared
CONSTANT        XFFO_A=0x29       ;transmit FIFO overrun asserted
CONSTANT        RFFO_A=0x2A       ;recieve FIFO overrun asserted
CONSTANT        COCA_A=0x2B       ;change of cell allignment
CONSTANT        PMON_UPDATE=0x80  ; PMON counters updated
CONSTANT        RW_DONE=0xFF    ; reg. R/W complete


; MailBox communication constants: HOST -> PIC
CONSTANT        READ_REG=0x01   ; MPH Reg Read Command
CONSTANT        WRITE_REG=0x02  ; MPH Reg Write Command
CONSTANT        XFDL_START=0x03 ; XFDL packet ready to send
CONSTANT        XBOC_START=0x04 ; begin sending a BOC
CONSTANT        XBOC_STOP=0x05  ; stop sending BOC
CONSTANT        TIMER_ON=0x06   ; enable timer ints
CONSTANT        TIMER_OFF=0x07  ; disable timer ints

CONSTANT        LLBA=0x0C          ; line loopback activate
CONSTANT        LLBD=0x0D          ; line loopback deactivate
CONSTANT        PLBA=0x0E          ; payload loopback activate
CONSTANT        PLBD=0x0F          ; payload loopback deactivate
CONSTANT        DLBA=0x12          ; diagnostic loopback activate
CONSTANT        DLBD=0x13          ; diagnostic loopback deactivate
CONSTANT        XAIS_START=0x14          ; start transmit AIS signal
CONSTANT        XAIS_STOP=0x15          ; stop transmit AIS signal
CONSTANT        READ_REG_QDSX=0x16     ; QDSX read command
CONSTANT        WRITE_REG_QDSX=0x17     ; QDSX write command
```

```
      ; Bit Oriented Code (BOC) constants
      CONSTANT        BOC_LLA=0x07    ; Line loopback activate
      CONSTANT        BOC_LLD=0x1C    ; Line loopback deactivate
      CONSTANT        BOC_PLA=0x0A    ; Payload loopback activate
      CONSTANT        BOC_PLD=0x19    ; Payload loopback deactivate
      CONSTANT        BOC_ULD=0x12    ; Universal loopback deactivate
      CONSTANT        BOC_YELI=0x00   ; YELLOW alarm indication
      CONSTANT        BOC_TERMINATE=0xFF  ; terminate code
                              ;  used to tell MPH it should stop
                              ;  sending current BOC

      ; Performance Report Message Bitmap
      CBLOCK 0
            PRM_G6, PRM_SL, PRM_G5, PRM_U2, PRM_U1, PRM_G4, PRM_LV, PRM_G3
      ENDC

      CBLOCK 0
            PRM_NL, PRM_NM, PRM_G2, PRM_R, PRM_G1, PRM_LB, PRM_SE, PRM_FE
      ENDC

      ; LAPD Address and Control bytes for PRMs

      CONSTANT        LAPD_ADDR_BYTE1=0x38    ; address byte 1,
                                    ; SAPI=14, C/R=0 (CI), EA=0
      CONSTANT        LAPD_ADDR_BYTE2=0x01    ; TEI=0, EA=1
      CONSTANT        LAPD_CNTRL_BYTE=03

;*************************
;* I/O MACRO DEFINITIONS *
;*************************

; write to RAM
; takes address and value
WR_RAM  MACRO   ADDRESS, VALUE

      BANK0
      MOVLW   VALUE
      MOVWF   DATA_REG        ; setup data register
      MOVLW   LOW ADDRESS
      MOVWF   ADDR_RAM_LO     ; setup address registers
      MOVLW   HIGH ADDRESS
      MOVWF   ADDR_RAM_HI
      PAGE0
      CALL    WRITE_RAM       ; perform write

      ENDM                    ; end of macro

; write to quadrant-specific RAM area
; Only changes the lower 7 address bits
; the upper 2 address bits are assumed valid
WR_RAM_L  MACRO  REGISTER, VALUE

      BANK0
      MOVLW   VALUE
      MOVWF   DATA_REG        ; setup data register
      MOVLW   0x80            ; setup address registers
      ANDWF   ADDR_RAM_LO, 1
      MOVLW   (LOW REGISTER) & 0x7F
      IORWF   ADDR_RAM_LO, 1
```

```
      PAGE0
      CALL    WRITE_RAM       ; perform write

      ENDM                    ; end of macro

; write DATA_REG to ram
; takes RAM address
WR_RAM_D  MACRO  ADDRESS

      BANK0
      MOVLW   LOW ADDRESS     ; setup address registers
      MOVWF   ADDR_RAM_LO
      MOVLW   HIGH ADDRESS
      MOVWF   ADDR_RAM_HI
      PAGE0
      CALL    WRITE_RAM       ; perform write

      ENDM                    ; end of macro

; write DATA_REG to quadrant-specific RAM register
; Only changes the lower 7 address bits
; the upper 2 address bits are assumed valid
WR_RAM_DL  MACRO  REGISTER

      BANK0
      MOVLW   0x80            ; setup address registers
      ANDWF   ADDR_RAM_LO, 1
      MOVLW   (LOW REGISTER) & 0x7F
      IORWF   ADDR_RAM_LO, 1
      PAGE0
      CALL    WRITE_RAM       ; perform write

      ENDM                    ; end of macro

; write DATA_REG to quadrant-specific PMON RAM register
; PRE:  WORK2 contains the quadrant to access
WR_RAM_DP  MACRO  ADDRESS

      BANK0
      MOVLW   LOW ADDRESS     ; setup address registers
      MOVWF   ADDR_RAM_LO
      MOVLW   HIGH ADDRESS
      MOVWF   ADDR_RAM_HI
      MOVF    WORK2, W
      PAGE0
      CALL    SETUP_RADDR_PMON
      CALL    WRITE_RAM

      ENDM                    ; end of macro

; write to MPH direct register
WR_MPH  MACRO  REGISTER, VALUE

      BANK0
      MOVLW   VALUE           ; setup data register
      MOVWF   DATA_REG
      MOVLW   LOW REGISTER    ; setup address registers
      MOVWF   ADDR_MPH_LO
      MOVLW   HIGH REGISTER
      MOVWF   ADDR_MPH_HI
```

```
      PAGE0
      CALL    WRITE_MPH       ; perform write

      ENDM                    ; end of macro

; write to MPH direct register
; takes register and value
; Only changes the lower 8 address bits
; the upper 3 address bits are assumed valid
WR_MPH_L  MACRO  REGISTER, VALUE

      BANK0
      MOVLW   VALUE           ; setup data register
      MOVWF   DATA_REG
      MOVLW   LOW REGISTER
      MOVWF   ADDR_MPH_LO
      PAGE0
      CALL    WRITE_MPH       ; perform write

      ENDM                    ; end of macro

; write DATA_REG to MPH direct register
; Only changes the lower 7 address bits
; the upper 2 address bits are assumed valid
WR_MPH_DL  MACRO  REGISTER

      BANK0
      MOVLW   LOW REGISTER
      MOVWF   ADDR_MPH_LO
      PAGE0
      CALL    WRITE_MPH       ; perform write

      ENDM                    ; end of macro

; read from RAM
; takes address and returns contents in both W and DATA_REG
RD_RAM  MACRO   ADDRESS

      BANK0
      MOVLW   LOW ADDRESS     ; setup address registers
      MOVWF   ADDR_RAM_LO
      MOVLW   HIGH ADDRESS
      MOVWF   ADDR_RAM_HI
      PAGE0
      CALL    READ_RAM        ; perform read

      ENDM                    ; end of macro

; read from quadrant-specific RAM area
; Only changes the lower 7 address bits
; the upper 2 address bits are assumed valid
RD_RAM_L  MACRO  REGISTER

      BANK0
      MOVLW   0x80            ; setup address registers
      ANDWF   ADDR_RAM_LO, 1
      MOVLW   (LOW REGISTER) & 0x7F
      IORWF   ADDR_RAM_LO, 1
      PAGE0
      CALL    READ_RAM        ; perform read
```

```
      ENDM                    ; end of macro

; read from MPH
; takes register address and returns contents in both W and DATA_REG
RD_MPH  MACRO  REGISTER

      BANK0
      MOVLW   LOW REGISTER    ; setup address registers
      MOVWF   ADDR_MPH_LO
      MOVLW   HIGH REGISTER
      MOVWF   ADDR_MPH_HI
      PAGE0
      CALL    READ_MPH        ; perform read

      ENDM                    ; end of macro

; read from MPH
; takes register address and returns contents in both W and DATA_REG
; Only changes the lower 8 address bits
; the upper 3 address bits are assumed valid
RD_MPH_L  MACRO  REGISTER

      BANK0
      MOVLW   LOW REGISTER
      MOVWF   ADDR_MPH_LO
      PAGE0
      CALL    READ_MPH        ; perform read

      ENDM                    ; end of macro


; write to QDSX direct register
WR_QDSX    MACRO  REGISTER, VALUE

      BANK0
      MOVLW   VALUE           ; setup data register
      MOVWF   DATA_REG
      MOVLW   LOW REGISTER    ; setup address registers
      MOVWF   ADDR_QDSX_LO
      MOVLW   HIGH REGISTER
      MOVWF   ADDR_QDSX_HI
      PAGE0
      CALL    WRITE_QDSX       ; perform write

      ENDM                    ; end of macro

; write to QDSX direct register
; takes register and value
; Only changes the lower 6 address bits
; the upper 2 address bits are assumed valid
WR_QDSX_L  MACRO  REGISTER, VALUE

      BANK0
      MOVLW   VALUE           ; setup data register
      MOVWF   DATA_REG
      MOVLW   0xC0
      ANDWF   ADDR_QDSX_LO, 1  ; setup address registers
      MOVLW   (LOW REGISTER) & 0x3F
      IORWF   ADDR_QDSX_LO, 1
```

```
      PAGE0
      CALL    WRITE_QDSX       ; perform write

      ENDM                     ; end of macro

; write DATA_REG to QDSX direct register
; Only changes the lower 6 address bits
; the upper 2 address bits are assumed valid
WR_QDSX_DL  MACRO  REGISTER

      BANK0
      MOVLW   0xC0             ; setup address registers
      ANDWF   ADDR_QDSX_LO, 1
      MOVLW   (LOW REGISTER) & 0x3F
      IORWF   ADDR_QDSX_LO, 1
      PAGE0
      CALL    WRITE_QDSX       ; perform write

      ENDM                     ; end of macro


; read from QDSX
; takes register address and returns contents in both W and DATA_REG
RD_QDSX  MACRO  REGISTER

      BANK0
      MOVLW   LOW REGISTER     ; setup address registers
      MOVWF   ADDR_QDSX_LO
      MOVLW   HIGH REGISTER
      MOVWF   ADDR_QDSX_HI
      PAGE0
      CALL    READ_QDSX        ; perform read

      ENDM                     ; end of macro

; read from QDSX
; takes register address and returns contents in both W and DATA_REG
; Only changes the lower 6 address bits
; the upper two address bits are assumed valid
RD_QDSX_L  MACRO  REGISTER

      BANK0
      MOVLW   0xC0             ; setup address registers
      ANDWF   ADDR_QDSX_LO, 1
      MOVLW   (LOW REGISTER) & 0x3F
      IORWF   ADDR_QDSX_LO, 1
      PAGE0
      CALL    READ_QDSX        ; perform read

      ENDM                     ; end of macro


;****************************
;* Quadrant bit flag macros *
;****************************

BS_QUAD_BIT     MACRO  QUAD, BIT

      MOVLW   Q1_FLAGS
      ADDWF   QUAD, W
```

```
      MOVWF   FSR
      BSF     INDF, BIT

      ENDM                    ; end of macro

BC_QUAD_BIT     MACRO  QUAD, BIT

      MOVLW   Q1_FLAGS
      ADDWF   QUAD, W
      MOVWF   FSR
      BCF     INDF, BIT

      ENDM                    ; end of macro

BTSS_QUAD_BIT   MACRO  QUAD, BIT

      MOVLW   Q1_FLAGS
      ADDWF   QUAD, W
      MOVWF   FSR
      BTFSS   INDF, BIT

      ENDM                    ; end of macro

BTSC_QUAD_BIT   MACRO  QUAD, BIT

      MOVLW   Q1_FLAGS
      ADDWF   QUAD, W
      MOVWF   FSR
      BTFSC   INDF, BIT

      ENDM                    ; end of macro

;******************
;* SET UP VECTORS *
;******************

; RESET VECTOR
      ORG     0000
      GOTO    INIT            ; initialize on reset

; INTERRUPT VECTOR
      ORG     0004

      MOVWF   TEMP_W          ; save W and STATUS
      SWAPF   STATUS, W       ; (order of commands is important)
      BCF     STATUS, RP0     ; switch to bank 0
      MOVWF   TEMP_STATUS
      MOVF    PCLATH, W       ; save PCLATH
      MOVWF   TEMP_PCLATH
      MOVF    INT_STATUS, W   ; save INT_STATUS
      MOVWF   INT_STATUS_SP
      MOVF    DATA_REG, W     ; save DATA_REG
      MOVWF   TEMP_DATA_REG
      MOVF    FSR, W          ; save FSR
      MOVWF   TEMP_FSR
      PAGE0                   ; very important, as this ISR may be
                        ; called from anywhere
      GOTO    INTHDLR         ; jump to INTHDLR

;******************
```

```
;* INITIALIZATION *
;******************

INIT
      CALL    INIT_MICRO      ; initialize the PIC microcontroller
      WR_QDSX GLOBAL_PMONQ, 0x80 ; reset QDSX
      WR_QDSX GLOBAL_PMONQ, 0x00
      WR_MPH GLOBAL_PMON, 0x80 ; reset MPH
      WR_MPH GLOBAL_PMON, 0x00
      CALL    INIT_RAM        ; initialize the RAM
      MOVLW   0x00            ; init all 4 quadrants of MPH
      CALL    INIT_MPH
      MOVLW   0x01
      CALL    INIT_MPH
      MOVLW   0x02
      CALL    INIT_MPH
      MOVLW   0x03
      CALL    INIT_MPH
      MOVLW   0x00            ; init all 4 quadrants of QDSX
      CALL    INIT_QDSX
      MOVLW   0x01
      CALL    INIT_QDSX
      MOVLW   0x02
      CALL    INIT_QDSX
      MOVLW   0x03
      CALL    INIT_QDSX

      CALL    ENABLE_INTS     ; enable PIC interrupts
      GOTO    MAIN_LOOP       ; go to main loop

;*************
;* MAIN LOOP *
;*************


MAIN_LOOP
      BTFSC   TIMEFLAG, 0     ; has 1s interrupt occured?
      GOTO    CALL_PMON       ; if so, call PMON subroutine
      INTSOFF
      BCF     CLKSTAT, 0      ; clear bit (assume clock is ok)
      RD_MPH_L    CLKAMON
      BTFSC   STATUS, Z       ; Check if clock activity monitor is bad
      BSF     CLKSTAT, 0      ; yes -> set bit
      BTFSC   CLKSTAT, 0
      CALL    NOCLK
      BTFSS   CLKSTAT, 0
      CALL    CLKOK
      INTSON
      CLRWDT
      GOTO    MAIN_LOOP

CALL_PMON
      PAGE1
      CALL    PMON            ; make cross-page call
      PAGE0
      GOTO    MAIN_LOOP

NOCLK
      BTFSC   CLKSTAT, 1      ; if host has already been informed
      RETURN                  ; no need to tell it again
```

```
      BSF     CLKSTAT, 1
      BCF     CLKSTAT, 2
      WR_RAM  MAILOUT, LOCA_A ; inform host of loss of clock activity
      RETURN
CLKOK
      BTFSC   CLKSTAT, 2      ; if host has already been informed
      RETURN                  ; no need to tell it again
      BSF     CLKSTAT, 2
      BCF     CLKSTAT, 1
      WR_RAM   MAILOUT,  LOCA_C  ;  inform  host  that  clock  activity  has  been
regained.
      RETURN


;*****************************
;* READ SUBROUTINE FOR MPH *
;*****************************

READ_MPH
      ; setup address bus
      BANK0
      MOVF    ADDR_MPH_LO, 0  ; transfer MPH address to latch
      MOVWF   PORTC
      MOVLW   0xF8
      ANDWF   PORTA, 1                ; clear A8-A10
      MOVF    ADDR_MPH_HI, 0  ; load W with A8-A10
      ANDLW   0x07                    ; mask off unused bits
      IORWF   PORTA, 1                ; insert A8-A10 into PORTA

      ; perform read by toggling RWB and CSB
      MOVLW   0xF7
      ANDWF   PORTA, 1        ; activate CSB
      MOVLW   0xFE
      ANDWF   PORTE, 1        ; activate RDB
      MOVF    PORTD, 0        ; latch data
      MOVWF   DATA_REG        ; save data
      MOVLW   0x01
      IORWF   PORTE, 1        ; deactivate RDB
      MOVLW   0x08
      IORWF   PORTA, 1        ; deactivate CSB and complete read
      BANK0

      MOVF    DATA_REG, 0     ; place result into W reg
      RETURN                  ; end of subroutine

;*****************************
;* READ SUBROUTINE FOR QDSX *
;*****************************

READ_QDSX
      ; setup address bus
      BANK0
      MOVF    ADDR_QDSX_LO, 0  ; transfer QDSX address to latch
      MOVWF   PORTC
      MOVLW   0xF8
      ANDWF   PORTA, 1                ; clear A8-A10
      MOVF    ADDR_QDSX_HI, 0  ; load W with A8-A10
      ANDLW   0x07                    ; mask off unused bits
      IORWF   PORTA, 1                ; insert A8-A10 into PORTA
```

```
      ; perform read by toggling RWB and CSB3
      MOVLW   0xDF
      ANDWF   PORTA, 1        ; activate CSB3
      MOVLW   0xFE
      ANDWF   PORTE, 1        ; activate RDB
      MOVF    PORTD, 0        ; latch data
      MOVWF   DATA_REG        ; save data
      MOVLW   0x01
      IORWF   PORTE, 1        ; deactivate RDB
      MOVLW   0x20
      IORWF   PORTA, 1        ; deactivate CSB and complete read
      BANK0

      MOVF    DATA_REG, 0     ; place result into W reg
      RETURN                  ; end of subroutine

;**************************
;* READ SUBROUTINE FOR RAM *
;**************************

READ_RAM
      ; setup address bus
      BANK0
      MOVF    ADDR_RAM_LO, 0  ; transfer RAM address to latch
      MOVWF   PORTC
      MOVLW   0xF8
      ANDWF   PORTA, 1        ; clear A8-A10
      MOVF    ADDR_RAM_HI, 0  ; load W with A8-A10
      ANDLW   0x07            ; mask off unused bits
      IORWF   PORTA, 1        ; insert A8-A10 into PORTA

      ; perform read by toggling RWB and CSB
      MOVLW   0xEF
      ANDWF   PORTA, 1        ; activate CSB
      MOVLW   0xFE
      ANDWF   PORTE, 1        ; activate OEB
      MOVF    PORTD, 0        ; latch data
      MOVWF   DATA_REG        ; save data
      MOVLW   0x01
      IORWF   PORTE, 1        ; deactiveate OEB
      MOVLW   0x10
      IORWF   PORTA, 1        ; deactivate CSB and complete read

      BANK0
      MOVF    DATA_REG, 0     ; place result into W reg.
      RETURN                  ; end of subroutine

;*****************************
;* WRITE SUBROUTINE FOR MPH *
;*****************************

WRITE_MPH
      BANK1
      MOVLW   0x00            ; config data bus as output
      MOVWF   TRISD

      ; setup address bus
      BANK0
      MOVF    ADDR_MPH_LO, 0  ; transfer MPH address to latch
      MOVWF   PORTC
```

```
      MOVLW   0xF8
      ANDWF   PORTA, 1                ; clear A8-A10
      MOVF    ADDR_MPH_HI, 0  ; load W with A8-A10
      ANDLW   0x07                    ; mask off unused bits
      IORWF   PORTA, 1                ; insert A8-A10 into PORTA

      ; setup data bus
      MOVF    DATA_REG, 0             ; transfer DATA_REG to latch
      MOVWF   PORTD

      ; perform write by toggling WRB and CSB
      MOVLW   0xF7
      ANDWF   PORTA, 1        ; activate CSB
      MOVLW   0xFD
      ANDWF   PORTE, 1        ; activate WRB
      MOVLW   0x02
      IORWF   PORTE, 1        ; deactivate WRB
      MOVLW   0x08
      IORWF   PORTA, 1        ; deactivate CSB and complete write

      ; tristate data bus
      BANK1
      MOVLW   0xFF    ;
      MOVWF   TRISD   ;

      BANK0           ; select as default
      RETURN          ; end of subroutine

;*****************************
;* WRITE SUBROUTINE FOR QDSX *
;*****************************

WRITE_QDSX
      BANK1
      MOVLW   0x00            ; config data bus as output
      MOVWF   TRISD

      ; setup address bus
      BANK0
      MOVF    ADDR_QDSX_LO, 0  ; transfer QDSX address to latch
      MOVWF   PORTC
      MOVLW   0xDF
      ANDWF   PORTA, 1                ; clear A8-A10
      MOVF    ADDR_QDSX_HI, 0  ; load W with A8-A10
      ANDLW   0x07                    ; mask off unused bits
      IORWF   PORTA, 1                ; insert A8-A10 into PORTA

      ; setup data bus
      MOVF    DATA_REG, 0             ; transfer DATA_REG to latch
      MOVWF   PORTD

      ; perform write by toggling WRB and CSB
      MOVLW   0xDF
      ANDWF   PORTA, 1        ; activate CSB
      MOVLW   0xFD
      ANDWF   PORTE, 1        ; activate WRB
      MOVLW   0x02
      IORWF   PORTE, 1        ; deactivate WRB
      MOVLW   0x20
      IORWF   PORTA, 1        ; deactivate CSB and complete write
```

```
      ; tristate data bus
      BANK1
      MOVLW   0xFF    ;
      MOVWF   TRISD   ;

      BANK0           ; select as default
      RETURN          ; end of subroutine

;****************************
;* WRITE SUBROUTINE FOR RAM *
;****************************

WRITE_RAM
      ; activate data bus as output
      BANK1
      MOVLW   0x00
      MOVWF   TRISD

      ; setup address bus
      BANK0
      MOVF    ADDR_RAM_LO, 0  ; transfer RAM address to latch
      MOVWF   PORTC
      MOVLW   0xF8
      ANDWF   PORTA, 1        ; clear A8-A10
      MOVF    ADDR_RAM_HI, 0  ; load W with A8-A10
      ANDLW   0x07            ; mask off unused bits
      IORWF   PORTA, 1        ; insert A8-A10 into PORTA

      ; setup data bus
      MOVF    DATA_REG, 0     ; transfer DATA_REG to latch
      MOVWF   PORTD

      ; perform write by toggling WRB and CSB
      MOVLW   0xEF
      ANDWF   PORTA, 1        ; activate CSB
      MOVLW   0xFD
      ANDWF   PORTE, 1        ; activate WRB
      MOVLW   0x02
      IORWF   PORTE, 1        ; deactivate WRB
      MOVLW   0x10
      IORWF   PORTA, 1        ; deactivate CSB and complete write

      ; tristate data bus
      BANK1
      MOVLW   0xFF
      MOVWF   TRISD

      BANK0                   ; select as default
      RETURN                  ; end of subroutine

;*****************************
;* INIT SUBROUTINE FOR MICRO *
;*****************************

INIT_MICRO      BANK0
      MOVLW   0x18            ; initialize ports
      MOVWF   PORTA           ; make RAM CSB and MPH CSB pins inactive
      MOVLW   0x04            ; turn off LEDs, force unused pin high
      MOVWF   PORTB           ;
```

```
      CLRF    PORTC           ; clear A0-A7
      CLRF    PORTD           ; clear D0-D7
      MOVLW   0x03
      MOVWF   PORTE           ; make RDB (OEBR) and WRB (RWBR) inactive
      MOVLW   0x7F
      MOVWF   FSR
CLEAR_USER_REGS
      CLRF    INDF
      DECF    FSR, F
      MOVF    FSR, W
      XORLW   0x1F
      BTFSS   STATUS, Z
      GOTO    CLEAR_USER_REGS
      BSF     CLKSTAT, 2
      MOVLW   0xFF
      MOVWF   PRM_COUNTER     ; set the performance report counter
                        ; to FF so that it will roll over to 00
                        ; on very first generation of PRMs

      MOVLW   ONE_SEC_L       ; initialize timer counter bytes
      MOVWF   TIME_COUNT_L
      MOVLW   ONE_SEC_H
      MOVWF   TIME_COUNT_H

      BANK1

      ; I/O port configuration: see the PIC databook for detailed information
      ;  regarding the configuration of the ports
      MOVLW   0x07            ; configure PORTA as digital
      MOVWF   ADCON1
      MOVLW   0x05            ; set OPT and TMR0 prescalar TO 1:64
      MOVWF   OPTION_REG
      MOVLW   0x20            ; tristate LOCKIN input
      MOVWF   TRISA
      MOVLW   0x13            ; LED outputs, INT/BUSY inputs,
                        ; unused pin output
      MOVWF   TRISB
      MOVLW   0x00            ; configure address bus as output
      MOVWF   TRISC
      MOVLW   0xFF            ; configure data bus as input
      MOVWF   TRISD
      MOVLW   0x00            ; configure control bus as output
      MOVWF   TRISE
      BANK0                   ; should switch back to bank 0 by default
      MOVF    PORTB,0 ; read PORTB to initialize latch value for RBIF
                  ;  this is necessary because of the method that the PIC
                  ;  uses for interrupt detection on some of the PORTB
                  ;  pins: an interrupt is generated when a difference
                  ;  between the current value and the last read value
                  ;  is detected.

      RETURN                  ; end of subroutine

;**************************
;* INIT SUBROUTINE FOR RAM *
;**************************

INIT_RAM
      RD_RAM  MAILIN          ; read mailbox to clear any interrupts
      WR_RAM  VER_ADDR, VER   ; write version number to RAM
```

```
      WR_RAM  REV_ADDR, REV   ; write revision number to RAM
      RETURN                  ; end of subroutine

;*****************************
;* INIT SUBROUTINE FOR MPH *
;*****************************
;
; Input:        Quadrant to be initialized in W reg (0-3)
; Results:      The associated quadrant is initialized
;  this routine is provided as a generic routine which initializes
;  the specified MPH quadrant. A call to SETUP_MADDR is made to setup the
;  upper address lines and from this call forward, special read/write
;  macros are used which do not alter the upper address lines.


INIT_MPH

      CALL    SETUP_MADDR     ; setup address regs. for MPH accesses

; MPH initialization registers

      WR_MPH_L        RECV_CONFIG, 0x00       ; 1.544 Mbit/s T1 ATM UNI
      WR_MPH_L        XMIT_CONFIG, 0x10       ; 1.544 Mbit/s T1 ATM UNI
      WR_MPH_L        RINT_CONFIG, 0x04       ; single rail
      WR_MPH_L        XINT_CONFIG, 0x08       ; single rail, NRZ
      WR_MPH_L        XCLK_OPTION, 0x00       ; 24xT1 rate, TCLK sources TCLKO
      WR_MPH_L        CDRC_CONFIG, 0x00       ; B8ZS, RZ
      WR_MPH_L        FRMR_CONFIG, 0x10       ; ESF, 4kbit/s FDL
      WR_MPH_L        XFDL_CONFIG, 0x03       ; Enable XFDL block/CRC append
      WR_MPH_L        RFDL_CONFIG, 0x01       ; Enable RFDL block
      WR_MPH_L        IBCD_CONFIG, 0x04       ; x bit code length
      WR_MPH_L        T1TRAN_CONFIG, 0x30     ; ESF, 4kbit/s FDl, B8ZS
      WR_MPH_L        ALMI_CONFIG, 0x10       ; ESF, 4kbit/s FDL
      WR_MPH_L        RXCP_CONTROL, 0x2C      ; HCSADD, BLOCK
      WR_MPH_L        RXCP_FRAME_CONTROL, 0x40        ; LCDE
      WR_MPH_L        TXCP_CONTROL, 0xA4      ; HCINS, HCSADD

; MPH interrupt enable registers
      WR_MPH_L        CDRC_IE, 0x00   ; No LOS interrupt yet
      WR_MPH_L        FRMR_IE, 0x01   ; Enable INFR
      WR_MPH_L        RBOC_IE, 0x05   ; Enable IDLE, BOCE
      WR_MPH_L        ALMI_IE, 0x07   ; Enable YEL, RED, AIS
      WR_MPH_L        IBCD_IE, 0x30   ; Enable LBAE, LBDE3D
      WR_MPH_L        RFDL_IE, 0x02   ; INTC039
      WR_MPH_L        RXCP_IE, 0x20   ; FIFOE
      WR_MPH_L        TXCP_IE, 0X40   ; FIFOE


; other MPH registers
      WR_MPH_L        DL_OPT, 0x0F            ; RDLINTE, RDLEOME,
                                      ; TDLINTE, TDLUDRE
      WR_MPH_L        IBCD_AC, 0x08            ; IBCD activate code
      WR_MPH_L        IBCD_DC, 0x24            ; IBCD deactivate code
      WR_MPH_L        RXCP_IDLEMASK_H1, 0xFF       ; Header mask
      WR_MPH_L        RXCP_IDLEMASK_H2, 0xFF       ; Header mask
      WR_MPH_L        RXCP_IDLEMASK_H3, 0xFF       ; Header mask
      WR_MPH_L        RXCP_IDLEMASK_H4, 0xFF       ; Header mask
      WR_MPH_L        RXCP_USERMASK_H1, 0xFF       ; User mask
      WR_MPH_L        RXCP_USERMASK_H2, 0xFF       ; User mask
      WR_MPH_L        RXCP_USERMASK_H3, 0xFF       ; User mask
      WR_MPH_L        RXCP_USERMASK_H4, 0xFF       ; User mask
```

```
      WR_MPH_L         TXCP_IDLEPAT_H5, 0x55        ; Header Check sequence for
idle cells
      WR_MPH_L          TXCP_IDLEPAY, 0x6A            ; Payload for unassigned
cells
      RETURN                                        ; end of subroutine


;*****************************
;* INIT SUBROUTINE FOR QDSX *
;*****************************
; Input:        Quadrant to be initialized in W reg (0-3)
; Results:      The associated quadrant is initialized
;  this routine is provided as a generic routine which initializes
;  the specified QDSX quadrant. A call to SETUP_QADDR is made to setup the
;  upper address lines and from this call forward, special read/write
;  macros are used which do not alter the upper address lines.

INIT_QDSX
      CALL    SETUP_QADDR     ; setup address regs. for QDSX accesses

; QDSX initialization registers

      WR_QDSX_L        CDRC_CONFIGQ, 0x64
      WR_QDSX_L        IBCD_CONFIGQ, 0x04
      WR_QDSX_L        XPLS_CONFIGQ, 0x40
      WR_QDSX_L        IBCD_ACQ, 0x08
      WR_QDSX_L        IBCD_DCQ, 0x24
      RETURN           ; end of subroutine

;************************************
;* SUBROUTINE FOR ENABLING INTERRUPTS *
;************************************

ENABLE_INTS
      MOVLW   0xB8    ; Enable TMR0, INT, RBI
      MOVWF   INTCON
      RETURN          ; end of subroutine

;*****************************
;* INTERRUPT HANDLING ROUTINE *
;*****************************

INTHDLR

DET_SOURCE
      BTFSC   INTCON, INTF    ; check for external (MPH or QDSX) Interrupt
      CALL    EXT_INT
      BTFSC   INTCON, RBIF    ; check for RAM Interrupt
      CALL    CALL_RAM_INT
      BTFSS   PORTB, 4        ; check for missed RAM Interrupt
                        ; it's neccesary to poll the RAM interrupt
                        ; pin because of a flaw in the PIC micro's
                        ; interrupt handling.  Basically, it
                        ; occassionaly misses interrupts asserted
                        ; by the interrupt-on-change pins (bits 4-7)
                        ; of PORTA.
      CALL    CALL_RAM_INT
      BTFSS   INTCON, T0IE    ; check that the TIMER int is not disabled
      GOTO    SKIP            ;
      BTFSC   INTCON, T0IF    ; check for TIMER Interrupt
```

```
      CALL    CALL_TIMER_INT
SKIP                          ;
      ; Check for pending external interrupt
      ; this check is here to minimize interrupt latency when
      ; there are multiple external interrupts pending, and to catch
      ; an external interrupt that may have occurred while another was
      ; being serviced

      BTFSS   PORTB, 0        ; skip if interrupt is not pending
TINT_LOOP
      CALL    EXT_INT ; if there is still a pending external
                      ;  int, then service it now
      BTFSS   PORTB, 0        ; loop until no MPH interrupts
      GOTO    TINT_LOOP

CLEAN_UP
      BANK0
      MOVF    TEMP_PCLATH, W  ; restore PCLATH
      MOVWF   PCLATH
      MOVF    INT_STATUS_SP, W  ; restore state of INT_STATUS
      MOVWF   INT_STATUS
      MOVF    TEMP_DATA_REG, W  ; restore DATA_REG
      MOVWF   DATA_REG
      MOVF    TEMP_FSR, W     ; restore FSR
      MOVWF   FSR
      SWAPF   TEMP_STATUS, W  ; restore W and STATUS registers
      MOVWF   STATUS
      SWAPF   TEMP_W, F
      SWAPF   TEMP_W, W
      RETFIE                  ; return from interrupt

EXT_INT
      RD_MPH  INT_ID          ; check intid for MPH
      ANDLW   0xF0            ; mask off unused bits
      BTFSC   STATUS, Z       ; test if MPH caused interupt
      GOTO    MPH_INT
      RD_QDSX INT_IDQ         ; check intid for QDSX
      ANDLW   0xF0
      BTFSC   STATUS, Z       ; test if QDSX caused interupt
      GOTO    QDSX_INT
      RETURN

; the following routines are required for cross-page calling
; they are associated with the code directly above

CALL_RAM_INT
      PAGE1
      CALL    RAM_INT
      PAGE0
      MOVF    PORTB, 1        ; Read PORTB onto itself to clear
                      ; interrupt on change logic
      BCF     INTCON, RBIF    ; Clear PORTB interrupt flag
      RETURN                  ; end of subroutine

CALL_TIMER_INT
      PAGE1
      CALL    TIMER_INT
      PAGE0
      RETURN                  ; end of subroutine
```

```
;**********************************
;* SUBROUTINE FOR MPH INTERRUPTS *
;**********************************

MPH_INT
      BCF     INTCON, INTF    ; clear interrupt flag

      ; determine interrupting quadrant and set up MPH address regs.
      RD_MPH  INT_ID  ; which quadrant interrupted?
      MOVWF   WORK    ; save in reg for testing
      MOVLW   0x80    ; Initialize W to test for invalid quadrant
                  ;  after case statement
      BTFSC   WORK, 4 ; quadrant 1?
      MOVLW   00      ; quadrant 1
      BTFSC   WORK, 5 ; quadrant 2?
      MOVLW   01      ; quadrant 2
      BTFSC   WORK, 6 ; quadrant 3?
      MOVLW   02      ; quadrant 3
      BTFSC   WORK, 7 ; quadrant 4?
      MOVLW   03      ; quadrant 4
      MOVWF   QUADRANT        ; save quadrant
      BTFSC   QUADRANT, 7     ; if MSB set, then no valid quadrant
      RETURN                  ; end of subroutine
      MOVWF   DATA_REG        ; prepare for write to RAM

      MOVF    QUADRANT, W
      CALL    SETUP_MADDR     ; setup the upper address lines
                        ;  to address interrupting quadrant
                        ;  of the MPH

      ; read in MPH int source registers for interrupting quadrant
      RD_MPH_L        INT1_1  ; read register
      MOVWF           INT1    ; save value
      RD_MPH_L        INT2_1  ; read register
      MOVWF           INT2    ; save value



      ; test for interrupt source
      BTFSC   INT1, 0         ; ALMI block?
      GOTO    ALMI
      BTFSC   INT2, 0         ; CDRC block?
      GOTO    CDRC
      BTFSC   INT1, 5         ; FRMR block?
      GOTO    FRMR
      BTFSC   INT1, 1         ; RBOC block?
      GOTO    RBOC
      BTFSC   INT1, 6         ; IBCD block?
      GOTO    IBCD
      BTFSC   INT1, 2         ; RFDL block?
      GOTO    RFDL
      BTFSC   INT2, 1         ; XFDL block?
      GOTO    XFDL
      BTFSC   INT2, 2         ; TXCP block?
      GOTO    TXCP
      BTFSC   INT2, 3         ; RXCP block?
      GOTO    RXCP
      RETURN                  ; end of subroutine

; Alarm Integrator block
```

```
ALMI
      RD_MPH_L        ALMI_IS
      MOVWF   INT_STATUS
      BTFSC   INT_STATUS, 3   ; test for AISI
      CALL    AIS
      BTFSC   INT_STATUS, 5   ; test for YELI
      CALL    YEL
      BTFSC   INT_STATUS, 4   ; test for REDI
      CALL    RED
      RETURN                  ; return from subroutine

; Clock and Data Recovery block
CDRC
      RD_MPH_L        CDRC_IS
      MOVWF   INT_STATUS
      BTFSC   INT_STATUS, 6   ; test for LOSI
      CALL    LOS
      BTFSC   INT_STATUS, 3   ; test for Z16DI
      CALL    Z16DI
      RETURN                  ; return from subroutine

; Framer block
FRMR
      RD_MPH_L        FRMR_IS
      MOVWF   INT_STATUS
      BTFSC   INT_STATUS, 2   ; test for INFRI
      CALL    INFR
      BTFSC   INT_STATUS, 6   ; test for FERI
      CALL    FERI
      BTFSC   INT_STATUS, 4   ; test for SFEI
      CALL    SFEI
      RETURN                  ; return from subroutine

; Bit Oriented Code Detector block
RBOC
      RD_MPH_L        RBOC_IS
      MOVWF   INT_STATUS
      ANDLW   0x3F            ; mask out 6-bit BOC
      MOVWF   WORK            ; save BOC in work
      BTFSC   INT_STATUS, 6   ; test for BOCI
      CALL    BOCI
      BTFSC   INT_STATUS, 7   ; test for IDLEI
      CALL    IDLEI
      RETURN                  ; return from subroutine

BOCI
      MOVLW   BOC_YELI
      XORWF   WORK, W         ; test for YELLOW alarm
      BTFSC   STATUS, Z       ;  if it is, exit now (will be dealt with
                        ;  by alarm integrator block)
      RETURN                  ; return from subroutine
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      MOVF    WORK, W         ; re-load BOC
      MOVWF   DATA_REG        ; place in RAM for host
      WR_RAM_D        RAM_RBOC
      WR_RAM  MAILOUT, BOC_VALID      ; indicate a valid BOC detected
      BC_QUAD_BIT     QUADRANT, LL_REQ        ; clear loopback request bit
      MOVF    WORK, W         ; re-load BOC
```

```
      XORLW   BOC_LLA         ; test for line loopback activate
      BTFSC   STATUS, Z
      GOTO    LLA
      MOVF    WORK, W         ; re-load BOC
      XORLW   BOC_LLD         ; test for line loopback deactivate
      BTFSC   STATUS, Z
      GOTO    LLD
      MOVF    WORK, W         ; re-load BOC
      XORLW   BOC_PLA         ; test for payload loopback act.
      BTFSC   STATUS, Z
      GOTO    PLA
      MOVF    WORK, W         ; re-load BOC
      XORLW   BOC_PLD         ; test for payload looback deact.
      BTFSC   STATUS, Z
      GOTO    PLD
      MOVF    WORK, W         ; re-load BOC
      XORLW   BOC_ULD         ; test for universal loopback dis.
      BTFSC   STATUS, Z
      GOTO    ULD
      RETURN                  ; if none of these, then done
LLA
      BS_QUAD_BIT     QUADRANT, LL_REQ        ; set loopback request
      RETURN                                  ; return from subroutine
LLD
      RD_MPH_L        MSTR_DIAG       ; read in diagnostic register
      BCF     DATA_REG, 4             ; clear line loopback bit
      WR_MPH_DL       MSTR_DIAG       ; update register
      RETURN                  ; return from subroutine
PLA
      RD_MPH_L        MSTR_DIAG
      BSF     DATA_REG, 5             ; activate payload loopback
      WR_MPH_DL       MSTR_DIAG       ; update register
      RETURN                  ; return from subroutine
PLD
      RD_MPH_L        MSTR_DIAG
      BCF     DATA_REG, 5             ; deactivate payload loopback
      WR_MPH_DL       MSTR_DIAG       ; update register
      RETURN                  ; return from subroutine
ULD
      RD_MPH_L        MSTR_DIAG       ; universal loopback deactivate
      BCF     DATA_REG, 4             ; clear all loopbacks
      BCF     DATA_REG, 5
      WR_MPH_DL       MSTR_DIAG
      RETURN                  ; return from subroutine

IDLEI
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      WR_RAM  MAILOUT, BOC_IDLE       ; indicate idle BOC to host
      BTSS_QUAD_BIT   QUADRANT, LL_REQ        ; test for ll request
      RETURN
      RD_MPH_L        MSTR_DIAG
      BSF     DATA_REG, 4             ; activate line loopback
      WR_MPH_DL       MSTR_DIAG
      RETURN                  ; return from subroutine

; Inband Loopback Code Detector block
IBCD
      RD_MPH_L        IBCD_IS
```

```
      MOVWF   INT_STATUS
      BTFSC   INT_STATUS, 1           ; test for in-band code act.
      GOTO    IBCD_ACTIVATE
      BTFSC   INT_STATUS, 0           ; test for in-band code deact.
      GOTO    IBCD_DEACTIVATE
      RETURN                  ; return from subroutine
IBCD_ACTIVATE
      RD_MPH_L        MSTR_DIAG
      BSF     DATA_REG, 4             ; activate line loopback
      WR_MPH_DL       MSTR_DIAG
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      WR_RAM  MAILOUT, IBC_LLA        ; indicate IBC ll activate to host
      RETURN                  ; return from subroutine
IBCD_DEACTIVATE
      RD_MPH_L        MSTR_DIAG
      BCF     DATA_REG, 4             ; deactivate line loopback
      WR_MPH_DL       MSTR_DIAG
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      WR_RAM  MAILOUT, IBC_LLD        ; indicate IBC ll deactivate to host
      RETURN                  ; return from subroutine

; HDLC Transmitter block
; *** N.B. the timing with which the last byte of a packet and the
;     end of message (EOM) bit is set is very important.  The last byte is
;     written, and then not until the XFDL block interrupts for another byte
;     is the EOM bit set.  This is because if the EOM bit is set right after
;     writing the last byte of the packet, it is possible that the EOM bit
;     will be erroneously cleared by the state machine in the XFDL block.
;     Doing things in the sequence as in the code below prevents any
;     possibility of this occuring.

XFDL
      BTSC_QUAD_BIT   QUADRANT, XFDL_USR_OR_PRM ; are we sending a user packet
      GOTO    PRM_XMIT                ; or a performance report message?
      RD_MPH_L        XFDL_IS         ; read MPH int. status register
      BTFSC   DATA_REG, 0             ; re-send packet if under-run
      GOTO    XFDL_RESTART
      BTFSS   DATA_REG, 1             ; confirm an XFDL interrupt
      RETURN                          ; return from subroutine
      CALL    SETUP_RADDR_XFDL        ; setup RAM address to XFDL range
      MOVF    QUADRANT, W             ; access local packet length value
      ADDLW   Q1_XFDL_PLEN
      MOVWF   FSR
      MOVF    INDF, W
      MOVWF   WORK1
      MOVF    QUADRANT, W             ; access data pointer value
      ADDLW   Q1_XFDL_RPTR
      MOVWF   FSR
      MOVF    INDF, W
      XORWF   WORK1, W                ; have we reached the end of pkt.?
      BTFSC   STATUS, Z               ; if so go to end of message routine
      GOTO    XFDL_EOM
      MOVLW   0x80                    ; zero lower 7 RAM address bits
      ANDLW   ADDR_RAM_LO
      MOVF    INDF, W                 ; get RAM pointer for this quadrant
      IORWF   ADDR_RAM_LO, 1          ; place it in RAM address reg.
```

```
      CALL    READ_RAM                ; read the RAM byte
      WR_MPH_DL       XFDL_DATA       ; write it to the MPH
      INCF    INDF, 1                 ; increment the RAM pointer
      MOVF    INDF, W
      RETURN                          ; end of subroutine
XFDL_EOM
      BTSC_QUAD_BIT   QUADRANT, XFDL_PENDING
      GOTO    SEND_PENDING_PRM
      RD_MPH_L        XFDL_CONFIG     ; we are finished, so
      BSF     DATA_REG, 4             ; set EOM bit in XFDL
      BCF     DATA_REG, 3             ; turn off XFDL interrupts
      WR_MPH_DL       XFDL_CONFIG
      BC_QUAD_BIT     QUADRANT, XFDL_BUSY     ; clear XFDL busy bit
      MOVLW   XFDL_DONE_Q1            ; signal done
      ADDWF   QUADRANT, W
      MOVWF   DATA_REG
      WR_RAM_D        MAILOUT
      RETURN                          ; done this packet
SEND_PENDING_PRM
      RD_MPH_L        XFDL_CONFIG     ; we are finished send user packet, so
      BSF     DATA_REG, 4             ; set EOM bit in XFDL
      WR_MPH_DL       XFDL_CONFIG
      BC_QUAD_BIT     QUADRANT, XFDL_PENDING  ; clear XFDL pending bit
      BS_QUAD_BIT     QUADRANT, XFDL_USR_OR_PRM ; set user/PRM bit to PRM
      MOVLW   XFDL_DONE_Q1            ; signal done
      ADDWF   QUADRANT, W
      MOVWF   DATA_REG
      WR_RAM_D        MAILOUT
      RETURN                          ; done this packet

XFDL_RESTART
      BCF     DATA_REG, 0             ; clear under-run bit
      WR_MPH_DL       XFDL_IS
      MOVF    QUADRANT, W             ; setup for access to timeout counter
      ADDLW   Q1_XFDL_TO              ; add base offset
      MOVWF   FSR                     ; place result in indirect mem. ptr.
      MOVF    INDF, W                 ; check we haven't timed out
      BTFSC   STATUS, Z               ; on packet restarts
      GOTO    XFDL_GIVEUP             ; giveup if so
      DECF    INDF, F                 ; decrement counter
      GOTO    XFDL_PKT_RESTART        ; call routine to setup for
                              ; the transmission of an XFDL packet
XFDL_GIVEUP
      BTSC_QUAD_BIT   QUADRANT, XFDL_PENDING
      GOTO    SEND_PENDING_PRM2
      RD_MPH_L        XFDL_CONFIG
      BCF     DATA_REG, 3             ; turn off XFDL interrupts
      WR_MPH_DL       XFDL_CONFIG
      BC_QUAD_BIT     QUADRANT, XFDL_BUSY ; clear FDL busy bit
      RETURN                          ; end of subroutine
SEND_PENDING_PRM2
      BC_QUAD_BIT     QUADRANT, XFDL_PENDING  ; clear XFDL pending bit
      BS_QUAD_BIT     QUADRANT, XFDL_USR_OR_PRM ; set user/PRM bit to PRM
      RETURN                                  ; done this packet

XFDL_PKT_START
      RD_RAM  XFDL_QUAD       ; which quadrant's XFDL block?
      ANDLW   0x03            ; confine quadrant range
      MOVWF   QUADRANT        ; save in QUADRANT
      ; check if a user HDLC packet is already in progress, return if so
```

```
      BTSS_QUAD_BIT   QUADRANT, XFDL_BUSY
      GOTO    XFDL_PKT_START_CONT
      BTSS_QUAD_BIT   QUADRANT, XFDL_USR_OR_PRM
      RETURN
XFDL_PKT_START_CONT
      MOVF    QUADRANT, W     ; reset timeout counter
      ADDLW   Q1_XFDL_TO
      MOVWF   FSR
      MOVLW   XFDL_MAX_RESTARTS
      MOVWF   INDF
      MOVF    QUADRANT, W     ; clear the RAM pkt. pointer to 0
      ADDLW   Q1_XFDL_RPTR
      MOVWF   FSR
      CLRF    INDF
      MOVF    QUADRANT, W     ; make local copy of packet length
      ADDLW   Q1_XFDL_PLEN
      MOVWF   FSR
      CALL    SETUP_RADDR_XFDL
      RD_RAM_L        RAM_PCT_LEN     ; read in packet length
      MOVF    DATA_REG, W     ; store in local length register
      MOVWF   INDF
      SUBLW   MAX_FDL_PLEN    ; make sure packet length < maxlength
      BTFSS   STATUS, C
      RETURN                  ; abort if not so
      BTSC_QUAD_BIT   QUADRANT, XFDL_BUSY
      GOTO    XFDL_MAKE_PEND
      BC_QUAD_BIT     QUADRANT, XFDL_USR_OR_PRM ; set user/PRM bit to user
      BS_QUAD_BIT     QUADRANT, XFDL_BUSY ; set FDL busy bit
      MOVF    QUADRANT, W
      CALL    SETUP_MADDR
      RD_MPH_L        XFDL_CONFIG
      BSF     DATA_REG, 3     ; enable XFDL interrupts
      WR_MPH_DL       XFDL_CONFIG
      RETURN                  ; return from subroutine
XFDL_MAKE_PEND
      BS_QUAD_BIT     QUADRANT, XFDL_PENDING  ; set XFDL pending bit
      RETURN

XFDL_PKT_RESTART
      MOVF    QUADRANT, W     ; clear the RAM pkt. pointer to 0
      ADDLW   Q1_XFDL_RPTR
      MOVWF   FSR
      CLRF    INDF
      MOVF    QUADRANT, W     ; make local copy of packet length
      ADDLW   Q1_XFDL_PLEN
      MOVWF   FSR
      CALL    SETUP_RADDR_XFDL
      RD_RAM_L        RAM_PCT_LEN     ; read in packet length
      MOVF    DATA_REG, W     ; store in local length register
      MOVWF   INDF
      RETURN

PRM_XMIT
      RD_MPH_L        XFDL_IS         ; read MPH int. status register
      BTFSC   DATA_REG, 0             ; abort PRM if under-run
      GOTO    PRM_ABORT
      BTFSS   DATA_REG, 1             ; confirm an XFDL interrupt
      RETURN                          ; return from subroutine
      MOVLW   Q1_PRM_PTR              ; check which byte we need to send next
      MOVWF   FSR
```

```
      MOVF    QUADRANT, W
      ADDWF   FSR, F
      MOVF    INDF, W
      BTFSC   STATUS, Z               ; address byte #1
      GOTO    PRM_BYTE1
      XORLW   0x01                    ; address byte #2
      BTFSC   STATUS, Z
      GOTO    PRM_BYTE2
      MOVF    INDF, W
      XORLW   0x02                    ; control byte
      BTFSC   STATUS, Z
      GOTO    PRM_BYTE3
      MOVF    INDF, W
      XORLW   0x0B                    ; have we finished?
      BTFSC   STATUS, Z
      GOTO    END_PRM

      MOVLW   0x03                    ; point FSR to next PRM byte
      SUBWF   INDF, W                 ; rolling over if neccesary
      INCF    INDF, F                 ; increment PRM pointer here
      ADDLW   0x06
      MOVWF   WORK1
      MOVF    PRM_COUNTER, W
      SUBWF   WORK1, F
      SUBWF   WORK1, W
      ANDLW   0x07
      ADDLW   Q1_PRM_1B1
      MOVWF   FSR
      MOVF    QUADRANT, W
      MOVWF   WORK
      BCF     STATUS, C
      RLF     WORK, F
      RLF     WORK, F
      RLF     WORK, W
      ADDWF   FSR, F
      MOVF    INDF, W
      MOVWF   DATA_REG
COPY_PRM_BYTE
      WR_MPH_DL       XFDL_DATA
      RETURN

PRM_BYTE1
      INCF    INDF, F                 ; address byte #1
      MOVLW   LAPD_ADDR_BYTE1
      MOVWF   DATA_REG
      GOTO    COPY_PRM_BYTE
PRM_BYTE2
      INCF    INDF, F                 ; address byte #2
      MOVLW   LAPD_ADDR_BYTE2
      MOVWF   DATA_REG
      GOTO    COPY_PRM_BYTE
PRM_BYTE3
      INCF    INDF, F                 ; control byte
      MOVLW   LAPD_CNTRL_BYTE
      MOVWF   DATA_REG
      GOTO    COPY_PRM_BYTE

PRM_ABORT
      BCF     DATA_REG, 0             ; clear under-run bit
      WR_MPH_DL       XFDL_IS
```

```
      ; fall through to END_PRM, setting EOM bit doesn't matter
END_PRM
      BTSC_QUAD_BIT   QUADRANT, XFDL_PENDING
      GOTO    SEND_PENDING_USR
      RD_MPH_L        XFDL_CONFIG     ; we are finished, so
      BSF     DATA_REG, 4             ; set EOM bit in XFDL
      BCF     DATA_REG, 3             ; turn off XFDL interrupts
      WR_MPH_DL       XFDL_CONFIG
      BC_QUAD_BIT     QUADRANT, XFDL_BUSY ; clear FDL busy bit
      RETURN
SEND_PENDING_USR
      RD_MPH_L        XFDL_CONFIG     ; we are finished, so
      BSF     DATA_REG, 4             ; set EOM bit in XFDL
      WR_MPH_DL       XFDL_CONFIG
      BC_QUAD_BIT     QUADRANT, XFDL_PENDING ; clear FDL pending bit
      BC_QUAD_BIT     QUADRANT, XFDL_USR_OR_PRM ; set user/PRM bit to USR
      RETURN

; HDLC Receiver block
RFDL
      MOVF    QUADRANT, 0     ; update RAM indicating to which
      MOVWF   DATA_REG        ;  quadrant current RFDL operation
      WR_RAM_D        RFDL_QUAD       ;  pertains
      CALL    SETUP_RADDR_RFDL; setup address
      MOVF    QUADRANT, 0     ; setup FSR register to access
      ADDLW   Q1_RFDL_STAT    ;  RFDL status reg. per quad.
      MOVWF   FSR
RFDL_REPEAT
      BCF     FSR, 2                  ; explicitly point to status regs
      RD_MPH_L        RFDL_DATA       ; read data byte
      MOVWF   TEMP_RFDL_DATA          ; save the read data
      RD_MPH_L        RFDL_STATUS     ; read status
      MOVWF   TEMP_RFDL_STATUS
      BTFSC   TEMP_RFDL_STATUS, 6     ; overrun?
      GOTO    OVR
      BTFSS   TEMP_RFDL_STATUS, 5     ; FLG set to 1?
      GOTO    FLG_0                   ; must be 0
FLG_1
      MOVF    FSR, W          ; save FSR
      MOVWF   WORK1
      BTSC_QUAD_BIT   QUADRANT, RFDL_ACTIVE
      GOTO    NEW_BYTE
      BS_QUAD_BIT     QUADRANT, RFDL_ACTIVE   ; yes, then mark active
      MOVF    WORK1, W        ; restore FSR
      MOVWF   FSR
      BSF     FSR, 2          ; set to access counters
      CLRF    INDF            ; clear counter
      RETURN                  ; return from subroutine
NEW_BYTE
      MOVF    WORK1, W        ; restore FSR
      MOVWF   FSR
      MOVF    TEMP_RFDL_DATA, 0       ; re-load data
      MOVWF   DATA_REG                ; get ready for write
      BSF     FSR, 2                  ; set to access counter
      MOVLW   0x80
      ANDWF   ADDR_RAM_LO, 1          ; mask off lower address lines
      MOVF    INDF, 0                 ; load pointer
      IORWF   ADDR_RAM_LO, 1          ; setup address
      CALL    WRITE_RAM
      INCF    INDF, 1                 ; increment pointer
```

```
      BTFSC   TEMP_RFDL_STATUS, 4     ; end of message?
      GOTO    RFDL_EOM
      BTFSC   TEMP_RFDL_STATUS, 7     ; is there another byte
                                ;  waiting?
      RETURN                  ; return from subroutine
      GOTO    RFDL_REPEAT
FLG_0
      MOVF    FSR, W  ; save FSR
      MOVWF   WORK1
      BTSC_QUAD_BIT   QUADRANT, RFDL_ACTIVE
      GOTO    RFDL_ABORT
      RETURN                  ; return from subroutine
RFDL_ABORT
      BC_QUAD_BIT     QUADRANT, RFDL_ACTIVE ; yes, then set inactive
      MOVF    WORK1, W        ; restore FSR
      MOVWF   FSR
      BCF     FSR, 2                  ; access status register
      CLRF    INDF
      BSF     FSR, 2                  ; access counter register
      CLRF    INDF
      RETURN                  ; return from subroutine
OVR
      BCF     FSR, 2                  ; choose status registers
      BSF     INDF, 6                 ; set OVERRUN bit
      RETURN                  ; return from subroutine
RFDL_EOM
      ; update channel status in local (PIC) registers
      BCF     FSR, 2                  ; choose status registers
      BTFSC   TEMP_RFDL_STATUS, 3
      BSF     INDF, 3                 ; set CRC bit
      BTFSC   TEMP_RFDL_STATUS, 0
      BSF     INDF, 0                 ; NVB0
      BTFSC   TEMP_RFDL_STATUS, 1
      BSF     INDF, 1                 ; NVB1
      BTFSC   TEMP_RFDL_STATUS, 2
      BSF     INDF, 2                 ; NVB2
      ; transfer registers to ram
      MOVF    INDF, 0
      MOVWF   DATA_REG
      WR_RAM_DL       RAM_CHAN_STAT   ; write channel status
      CLRF    INDF
      BSF     FSR, 2                  ; choose counter register
      MOVF    INDF, 0
      MOVWF   DATA_REG
      WR_RAM_DL       RAM_PCT_LEN
      CLRF    INDF
      MOVF    DATA_REG, W             ; check if packet length >= 3
      SUBLW   0x02
      BTFSC   STATUS, C
      RETURN                  ; return from subroutine
      MOVLW   RFDL_NEW_Q1             ; signal end of new packet
      ADDWF   QUADRANT, W
      MOVWF   DATA_REG
      WR_RAM_D        MAILOUT
      RETURN                  ; return from subroutine

; TXCP Block
TXCP
      RD_MPH_L  TXCP_IS
      MOVWF   INT_STATUS
```

```
      BTFSC   INT_STATUS, 2   ; Change of cell alignment?
      GOTO    COCA
      BTFSC   INT_STATUS, 1   ; Transmit FIFO overrun?
      GOTO    FOVR
      RETURN

COCA                            ; indicate to host there has been a change of
      MOVF    QUADRANT, W      ; cell alignment
      MOVWF   DATA_REG         ; indicate which quadrant
      WR_RAM_D  INT_QUAD
      WR_RAM  MAILOUT, COCA_A  ; send mailbox code
      RETURN
FOVR                             ;indicate to host there has been a transmit
      MOVF    QUADRANT, W      ; FIFO overrun
      MOVWF   DATA_REG         ; indicate which quadrant
      WR_RAM_D  INT_QUAD
      WR_RAM  MAILOUT, XFFO_A  ; send mailbox code
      RETURN

; RXCP Block
RXCP
      RD_MPH_L  RXCP_IS
      MOVWF   INT_STATUS
      BTFSS   INT_STATUS, 1    ; recieve FIFO overrun?
      GOTO    LCD              ; if not may be loss of cell delineation
      MOVF    QUADRANT, W
      MOVWF   DATA_REG
      WR_RAM_D   INT_QUAD      ; indicate quadrant
      WR_RAM  MAILOUT, RFFO_A  ; send mailbox code
      RETURN
LCD
      RD_MPH_L  RXCPFC_IS
      MOVWF   INT_STATUS       ; loss of cell delineation?
      BTFSS   INT_STATUS, 5
      RETURN
      BTFSS   INT_STATUS, 4    ; send appropriate status
      GOTO    LCD_CLEAR
      MOVF    QUADRANT, W
      MOVWF   DATA_REG
      WR_RAM_D   INT_QUAD      ; indicate quadrant
      WR_RAM  MAILOUT, LCD_A  ; send mailbox code
      RETURN
LCD_CLEAR
      MOVF    QUADRANT, W
      MOVWF   DATA_REG
      WR_RAM_D   INT_QUAD      ; indicate quadrant
      WR_RAM  MAILOUT, LCD_C  ; send mailbox code
      RETURN

AIS
      BTFSC   INT_STATUS, 0          ; is AIS set?
      GOTO    AIS_SET
      MOVF    QUADRANT, W            ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      WR_RAM  MAILOUT, AIS_C         ; send clear message
      RETURN                         ; return from subroutine
AIS_SET
      MOVF    QUADRANT, W            ; indicate which quadrant
      MOVWF   DATA_REG
```

```
      WR_RAM_D        INT_QUAD
      WR_RAM  MAILOUT, AIS_A          ; send asserted message
      RD_MPH_L        MSTR_DIAG       ; clear all loopbacks
      BCF     DATA_REG, 4             ; clear line loopback
      BCF     DATA_REG, 5             ; clear payload loopback
      WR_MPH_DL       MSTR_DIAG       ; update MPH
      RETURN                          ; return from subroutine

LOS
      BTFSC   INT_STATUS, 0           ; is LOS set?
      GOTO    LOS_SET
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      WR_RAM  MAILOUT, LOS_A          ; send clear message
      GOTO    LOS_END
LOS_SET
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      WR_RAM  MAILOUT, LOS_C          ; send asserted message
LOS_END
      RETURN                  ; return from subroutine

; In-Frame indication is commented out
INFR
;       BTFSC   INT_STATUS, 0           ; is INFR set?
;       GOTO    INFR_SET
;       MOVF    QUADRANT, W             ; indicate which quadrant
;       MOVWF   DATA_REG
;       WR_RAM_D        INT_QUAD
;       WR_RAM  MAILOUT, INFR_C         ; send clear message
;       GOTO    INFR_END
INFR_SET
;       MOVF    QUADRANT, W             ; indicate which quadrant
;       MOVWF   DATA_REG
;       WR_RAM_D        INT_QUAD
;       WR_RAM          MAILOUT, INFR_A ; send asserted message
INFR_END
      RETURN                  ; return from subroutine

YEL
      BTFSC   INT_STATUS, 2           ; is YEL set?
      GOTO    YEL_SET
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      WR_RAM  MAILOUT, YEL_C          ; send clear message
      GOTO    YEL_END
YEL_SET
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      WR_RAM  MAILOUT, YEL_A          ; send asserted message
YEL_END
      RETURN                  ; return from subroutine
RED
      BTFSC   INT_STATUS, 1           ; is RED set?
      GOTO    RED_SET
      MOVF    QUADRANT, W             ; indicate which quadrant
```

```
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      WR_RAM  MAILOUT, RED_C          ; send clear message
      GOTO    RED_END
RED_SET
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      WR_RAM  MAILOUT, RED_A          ; send asserted message
RED_END
      RETURN                  ; return from subroutine

; Pulse Density Violation indication is commented out
Z16DI
;       MOVF    QUADRANT, W             ; indicate which quadrant
;       MOVWF   DATA_REG
;       WR_RAM_D        INT_QUAD
;       WR_RAM  MAILOUT, PDV            ; signal a pulse density voilation
      RETURN                  ; return from subroutine

; Framing Error indication is commented out
FERI
;       MOVF    QUADRANT, W             ; indicate which quadrant
;       MOVWF   DATA_REG
;       WR_RAM_D        INT_QUAD
;       WR_RAM  MAILOUT, FER            ; signal a framing error
      RETURN                  ; return from subroutine

; Severe Framing Error indication is commented out
SFEI
      BSF     SEF, QUADRANT   ; set bit for PMON SEF bit
;       MOVF    QUADRANT, W             ; indicate which quadrant
;       MOVWF   DATA_REG
;       WR_RAM_D        INT_QUAD
;       WR_RAM  MAILOUT, SFER           ; signal a severe framing error
      RETURN                  ; return from subroutine

;**********************************
;* SUBROUTINE FOR QDSX INTERRUPTS *
;**********************************

QDSX_INT
      BCF     INTCON, INTF    ; clear interrupt flag

      ; determine interrupting quadrant and set up QDSX address regs.
      RD_QDSX  INT_IDQ  ; which quadrant interrupted?
      MOVWF   WORK    ; save in reg for testing
      MOVLW   0x80    ; Initialize W to test for invalid quadrant
                  ;  after case statement
      BTFSC   WORK, 4 ; quadrant 1?
      MOVLW   00      ; quadrant 1
      BTFSC   WORK, 5 ; quadrant 2?
      MOVLW   01      ; quadrant 2
      BTFSC   WORK, 6 ; quadrant 3?
      MOVLW   02      ; quadrant 3
      BTFSC   WORK, 7 ; quadrant 4?
      MOVLW   03      ; quadrant 4
      MOVWF   QUADRANT        ; save quadrant
      BTFSC   QUADRANT, 7     ; if MSB set, then no valid quadrant
      RETURN                  ; end of subroutine
```

```
      MOVWF   DATA_REG        ; prepare for write to RAM

      MOVF    QUADRANT, W
      CALL    SETUP_QADDR     ; setup the upper address lines
                      ;  to address interrupting quadrant
                      ;  of the QDSX

      ; read in QDSX int source register for interrupting quadrant
      RD_QDSX_L        INTQ  ; read register
      MOVWF           INT1    ; save value



      ; test for interrupt source
      BTFSC   INT1, 0
      GOTO    DJAT_Q          ; DJAT ?
      BTFSC   INT1, 1
      GOTO    CDRC_Q          ; CDRC ?
      BTFSC   INT1, 3
      GOTO    LCV_PMON_Q      ; LCV_PMON ?
      BTFSC   INT1, 4
      GOTO    PRSM_Q          ; PRSM ?
      BTFSC   INT1, 5
      GOTO    IBCD_Q          ; IBCD ?
      BTFSC   INT1, 6
      GOTO    XPLS_Q          ; XPLS ?
      BTFSC   INT1, 7
      GOTO    RLSC_Q          ; RLSC ?
      RETURN                  ; end of subroutine

DJAT_Q
      RD_QDSX_L   DJAT_ISQ    ; read the interupt to clear it
      RETURN  ;do nothing
CDRC_Q
      RD_QDSX_L   CDRC_ISQ    ; read the interupt to clear it
      RETURN  ;do nothing
LCV_PMON_Q
      RD_QDSX_L   LCV_PMON_ISQ   ; read the interupt to clear it
      RETURN  ;do nothing
PRSM_Q
      RD_QDSX_L   PRSM_ISQ    ; read the interupt to clear it
      RETURN  ;do nothing
IBCD_Q
      RD_QDSX_L   IBCD_ISQ    ; read the interupt to clear it
      RETURN  ;do nothing
XPLS_Q
      RD_QDSX_L   XPLS_ISQ    ; read the interupt to clear it
      RETURN  ;do nothing
RLSC_Q
      RD_QDSX_L   RLSC_ISQ    ; read the interupt to clear it
      RETURN  ;do nothing

; This routine sets the upper 2 bits of the MPH address register
; according to the quadrant to be accessed
; Input:        W register contains the quadrant
; Results:      ADDR_MPH_HI and ADDR_MPH_LO are set up

SETUP_MADDR
      MOVWF   WORK5
      CLRF    ADDR_MPH_LO             ; clear address registers
```

```
      CLRF    ADDR_MPH_HI
      IORWF   ADDR_MPH_HI, F
      RETURN                  ; return from subroutine

; This routine sets the upper 2 bits of the QDSX address register
; according to the quadrant to be accessed
; Input:        W register contains the quadrant
; Results:      ADDR_QDSX_HI and ADDR_QDSX_LO are set up
; **NB** WORK5 is affected by this routine !!!
SETUP_QADDR
      MOVWF   WORK5
      CLRF    ADDR_QDSX_LO             ; clear address registers
      CLRF    ADDR_QDSX_HI
      BSF     ADDR_QDSX_LO, 6  ; copy bit 0 of quadrant (W=WORK5) to
      BSF     ADDR_QDSX_LO, 7  ; bit 6 of LSB of QDSX address reg
      BTFSS   WORK5, 0                ; and bit 1 of quadrant to
      BCF     ADDR_QDSX_LO, 6  ; bit 7 of LSB of QDSX address reg
      BTFSS   WORK5, 1
      BCF     ADDR_QDSX_LO, 7
      RETURN                  ; return from subroutine

; This routines sets up bits 4 and 5 of the RAM address register
; according to quadrant specified in the W register to access
; PMON RAM locations
; **NB** WORK5 is affected by this routine !!!
SETUP_RADDR_PMON
      MOVWF   WORK5
      BSF     ADDR_RAM_LO, 4  ; copy bit 0 of quadrant (W) to
      BSF     ADDR_RAM_LO, 5  ; bit 4 of LSB of RAM address reg
      BTFSS   WORK5, 0                ; and bit 1 of quadrant (W) to
      BCF     ADDR_RAM_LO, 4  ; bit 5 of LSB of RAM address reg
      BTFSS   WORK5, 1
      BCF     ADDR_RAM_LO, 5
      RETURN                  ; end of SETUP_RADDR_PMON

; This routine sets up the RAM address register to access the RFDL
; memory of a particular quadrant
; Input:        QUADRANT register contains the quadrant
; Results:      ADDR_RAM_HI and ADDR_RAM_LO are set up to point to the
;               beginning of the RFDL data area for the correct quad.
SETUP_RADDR_RFDL
      CLRF    ADDR_RAM_LO     ; clear address registers
      CLRF    ADDR_RAM_HI
      BSF     ADDR_RAM_LO, 7  ; copy bit 0 of QUADRANT to
      BSF     ADDR_RAM_HI, 0  ; bit 7 of LSB of RAM address reg
      BTFSS   QUADRANT, 0     ; and bit 1 of quadrant QUADRANT to
      BCF     ADDR_RAM_LO, 7  ; bit 0 of MSB of RAM address reg
      BTFSS   QUADRANT, 1
      BCF     ADDR_RAM_HI, 0
      RETURN                  ; end of SETUP_RADDR_RFDL

; This routine sets up the RAM address register to access the XFDL
; memory of a particular quadrant
; Input:        QUADRANT register contains the quadrant
; Results:      ADDR_RAM_HI and ADDR_RAM_LO are set up to point to the
;               beginning of the XFDL data area for the correct quad.
SETUP_RADDR_XFDL
      CLRF    ADDR_RAM_LO     ; clear address registers
      CLRF    ADDR_RAM_HI
      BSF     ADDR_RAM_HI, 1  ; set bit 1 of MSB of RAM address reg
```

```
      BSF     ADDR_RAM_LO, 7  ; copy bit 0 of QUADRANT to
      BSF     ADDR_RAM_HI, 0  ; bit 7 of LSB of RAM address reg
      BTFSS   QUADRANT, 0     ; and bit 1 of quadrant QUADRANT to
      BCF     ADDR_RAM_LO, 7  ; bit 0 of MSB of RAM address reg
      BTFSS   QUADRANT, 1
      BCF     ADDR_RAM_HI, 0
      RETURN                  ; return from subroutine


; !!!!!!!!!!! Watch for this code bumping into the 0x800 boundary
; !!!!!!!!!!! Currently at ~0x794

;*********************************                    PAGE                    1
*********************************
;*****************************************************************************
;
*

      ORG             0x800           ; start of page 1

;*****************************
;
;* HANDLER FOR RAM INTERRUPTS *
;*****************************
;


; Read mailbox and process
RAM_INT
      RD_RAM  MAILIN          ; read mailbox
      PAGE1
      ; The following is one big CASE statement
      MOVWF   SCRATCH         ; save for comparisons
      XORLW   READ_REG        ; check for MPH reg read
      BTFSC   STATUS, Z
      GOTO    REG_READ
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   WRITE_REG       ; check for MPH reg write
      BTFSC   STATUS, Z
      GOTO    REG_WRITE
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   XFDL_START      ; check for start of XFDL packet
      BTFSC   STATUS, Z
      GOTO    CALL_XFDL_PKT_START
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   XBOC_START      ; check for transmit BOC command
      BTFSC   STATUS, Z
      GOTO    SETUP_BOC
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   XBOC_STOP       ; check for idle BOC command
      BTFSC   STATUS, Z
      GOTO    FINISH_BOC
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   TIMER_ON        ; check for enable timer ints command
      BTFSC   STATUS, Z
      BSF     INTCON, T0IE
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   TIMER_OFF       ; check for disable timer ints command
      BTFSC   STATUS, Z
      BCF     INTCON, T0IE
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   LLBA            ; check for line loopback act. command
```

```
      BTFSC   STATUS, Z
      GOTO    LLBA_REQ
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   LLBD            ; check for line loopback deact. command
      BTFSC   STATUS, Z
      GOTO    LLBD_REQ
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   PLBA            ; check for payload loopback act. command
      BTFSC   STATUS, Z
      GOTO    PLBA_REQ
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   PLBD            ; check for payload loopback deact. command
      BTFSC   STATUS, Z
      GOTO    PLBD_REQ
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   DLBA            ; check for diag. loopback act. command
      BTFSC   STATUS, Z
      GOTO    DLBA_REQ
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   DLBD            ; check for line loopback deact. command
      BTFSC   STATUS, Z
      GOTO    DLBD_REQ
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   XAIS_START     ; check for start xmit ais command
      BTFSC   STATUS, Z
      GOTO    XAIS_START_REQ
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   XAIS_STOP       ; check for stop xmit ais command
      BTFSC   STATUS, Z
      GOTO    XAIS_STOP_REQ
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   READ_REG_QDSX    ; check for read QDSX  command
      BTFSC   STATUS, Z
      GOTO    READ_QDSX_REQ
      MOVF    SCRATCH, 0      ; restore W for next compare
      XORLW   WRITE_REG_QDSX    ; check for write QDSX  command
      BTFSC   STATUS, Z
      GOTO    WRITE_QDSX_REQ
      RETURN                  ; return from subroutine

CALL_XFDL_PKT_START
      PAGE0                   ; XFDL_PKT_START is in page 0
      CALL    XFDL_PKT_START
      PAGE1
      RETURN                  ; return from subroutine
REG_READ
      RD_RAM  REG_ADR_LO              ; transfer address
      MOVWF   ADDR_MPH_LO
      RD_RAM  REG_ADR_HI
      MOVWF   ADDR_MPH_HI
      PAGE0
      CALL    READ_MPH
      PAGE1
      WR_RAM_D REG_DATA
      WR_RAM  MAILOUT, RW_DONE        ; signal end of access
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      RETURN                  ; return from subroutine
REG_WRITE
```

```
      RD_RAM  REG_ADR_LO              ; transfer address
      MOVWF   ADDR_MPH_LO
      RD_RAM  REG_ADR_HI
      MOVWF   ADDR_MPH_HI
      RD_RAM  REG_DATA
      MOVWF   DATA_REG
      PAGE0
      CALL    WRITE_MPH
      PAGE1
      WR_RAM  MAILOUT, RW_DONE        ; signal end of access
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      PAGE1
      RETURN                  ; return from subroutine

SETUP_BOC
      RD_RAM  SERVICE_QUAD
      PAGE0                           ; located in PAGE 0
      MOVF    DATA_REG, W
      CALL    SETUP_MADDR
      PAGE1
      RD_RAM  RAM_XBOC                ; read in BOC to transmit
      WR_MPH_DL       XBOC_CODE       ; start BOC transmission
      PAGE1
      RETURN                  ; return from subroutine

FINISH_BOC
      RD_RAM  SERVICE_QUAD
      PAGE0                   ; located in PAGE 0
      MOVF    DATA_REG, W
      CALL    SETUP_MADDR
      PAGE1
      WR_MPH_L        XBOC_CODE, BOC_TERMINATE        ; end BOC transmission
      PAGE1
      RETURN                  ; return from subroutine

LLBA_REQ        ; handle line loopback activate request
      RD_RAM  SERVICE_QUAD
      PAGE0                           ; located in PAGE 0
      MOVF    DATA_REG, W
      CALL    SETUP_MADDR
      PAGE1
      RD_RAM  SERVICE_QUAD
      PAGE0                           ; located in PAGE 0
      MOVF    DATA_REG, W
      MOVWF   QUADRANT
      CALL    LLA
      PAGE1
      RETURN                  ; return from subroutine


LLBD_REQ        ; handle line loopback deactivate request
      RD_RAM  SERVICE_QUAD
      PAGE0                           ; located in PAGE 0
      MOVF    DATA_REG, W
      CALL    SETUP_MADDR
      PAGE1
      RD_RAM  SERVICE_QUAD
      PAGE0                           ; located in PAGE 0
```

```
      MOVF    DATA_REG, W
      MOVWF   QUADRANT
      CALL    LLD
      PAGE1
      RETURN                  ; return from subroutine

PLBA_REQ        ; handle payload loopback activate request
      RD_RAM  SERVICE_QUAD
      PAGE0                             ; located in PAGE 0
      MOVF    DATA_REG, W
      CALL    SETUP_MADDR
      PAGE1
      RD_RAM  SERVICE_QUAD
      PAGE0                             ; located in PAGE 0
      MOVF    DATA_REG, W
      MOVWF   QUADRANT
      CALL    PLA
      PAGE1
      RETURN                  ; return from subroutine

PLBD_REQ        ; handle payload loopback deactivate request
      RD_RAM  SERVICE_QUAD
      PAGE0                             ; located in PAGE 0
      MOVF    DATA_REG, W
      CALL    SETUP_MADDR
      PAGE1
      RD_RAM  SERVICE_QUAD
      PAGE0                             ; located in PAGE 0
      MOVF    DATA_REG, W
      MOVWF   QUADRANT
      CALL    PLD
      PAGE1
      RETURN                  ; return from subroutine

DLBA_REQ                        ; deactivate diagnostic loopback
      RD_RAM  SERVICE_QUAD    ; setup for quadrant
      PAGE0
      MOVF    DATA_REG, W
      CALL    SETUP_MADDR
      PAGE0
      RD_MPH_L  MSTR_DIAG     ; read register
      BSF     DATA_REG, 2     ; change register
      WR_MPH_DL MSTR_DIAG     ; save register
      PAGE1
      RETURN

DLBD_REQ                        ; deactivate diagnostic loopback
      RD_RAM  SERVICE_QUAD    ; setup for quadrant
      PAGE0
      MOVF    DATA_REG, W
      CALL    SETUP_MADDR
      PAGE0
      RD_MPH_L  MSTR_DIAG     ; read register
      BCF     DATA_REG, 2     ; change register
      WR_MPH_DL MSTR_DIAG     ; save register
      PAGE1
      RETURN

XAIS_START_REQ                          ; Start transmit AIS
      RD_RAM  SERVICE_QUAD    ; setup for quadrant
```

```
      PAGE0
      MOVF    DATA_REG, W
      CALL    SETUP_MADDR
      PAGE0
      RD_MPH_L  T1TRAN_ALM     ; read register
      BSF     DATA_REG, 0     ; change register
      WR_MPH_DL T1TRAN_ALM     ; save register
      PAGE1
      RETURN
XAIS_STOP_REQ                        ; Stop transmit AIS
      RD_RAM  SERVICE_QUAD    ; setup for quadrant
      PAGE0
      MOVF    DATA_REG, W
      CALL    SETUP_MADDR
      PAGE0
      RD_MPH_L  T1TRAN_ALM     ; read register
      BCF     DATA_REG, 0     ; change register
      WR_MPH_DL T1TRAN_ALM     ; save register
      PAGE1
      RETURN
READ_QDSX_REQ
      RD_RAM  REG_ADR_LO              ; transfer address
      MOVWF   ADDR_QDSX_LO
      RD_RAM  REG_ADR_HI
      MOVWF   ADDR_QDSX_HI
      PAGE0
      CALL    READ_QDSX
      PAGE1
      WR_RAM_D REG_DATA
      WR_RAM  MAILOUT, RW_DONE        ; signal end of access
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      RETURN                  ; return from subroutine
WRITE_QDSX_REQ
      RD_RAM  REG_ADR_LO              ; transfer address
      MOVWF   ADDR_QDSX_LO
      RD_RAM  REG_ADR_HI
      MOVWF   ADDR_QDSX_HI
      RD_RAM  REG_DATA
      MOVWF   DATA_REG
      PAGE0
      CALL    WRITE_QDSX
      PAGE1
      WR_RAM  MAILOUT, RW_DONE        ; signal end of access
      MOVF    QUADRANT, W             ; indicate which quadrant
      MOVWF   DATA_REG
      WR_RAM_D        INT_QUAD
      PAGE1
      RETURN                  ; return from subroutine


;*******************************
;* HANDLER FOR TIMER INTERRUPTS *
;*******************************


TIMER_INT
      BANK0
      BCF     INTCON, T0IF    ; clear interrupt flag
```

```
      BSF     TIMEFLAG, 1     ;  set 3.2768mS flag
      DECFSZ  TIME_COUNT_L, 1 ; decrement low byte of counter
      RETURN                  ; return from subroutine
      MOVF    TIME_COUNT_H, 0 ; affect Z flag
      BTFSS   STATUS, Z
      GOTO    DECREMENT_COUNTER
      MOVLW   ONE_SEC_L       ; reset counter (1SEC=200NS*64*256*305)
      MOVWF   TIME_COUNT_L
      MOVLW   ONE_SEC_H
      MOVWF   TIME_COUNT_H
      BSF     TIMEFLAG, 0     ; set flag bit

      MOVLW   0x80            ; toggle LED4
      XORWF   PORTB, 1

      RETURN                  ; return from subroutine

DECREMENT_COUNTER
      DECF    TIME_COUNT_H, 1         ; decrement counter
      RETURN                  ; return from subroutine



;*****************************************
;* PERFORMANCE REPORT GENERATION SUBROUTINE *
;*****************************************

GENERATE_PRM
      MOVWF   WORK2                   ; save quadrant to WORK2
      MOVWF   WORK3
      BCF     STATUS, C
      RLF     WORK3, F
      RLF     WORK3, F
      RLF     WORK3, W
      ADDLW   0x06
      MOVWF   FSR
      MOVLW   Q1_PRM_1B1              ; setup to access performance
      ADDWF   FSR, F                  ; report register according
      MOVF    PRM_COUNTER, W          ; to quadrant and PRM counter
      SUBWF   FSR, F                  ; in order for the history
      SUBWF   FSR, F                  ; to work properly, we must write
                              ; in the opposite direction that
                              ; we read, hence we start with
                              ; Q1_PRM_4B1 and subtract PRM_COUNTER
      CLRF    INDF                    ; clear performance report byte 1
      INCF    FSR, F                  ; copy performance report counter
      MOVF    PRM_COUNTER, W          ; bits into performance report byte 2
      MOVWF   INDF                    ; this is ok because bits 2-7 = 0
      DECF    FSR, F

RTXCP                           ;Shadow RXCP and TXCP counters to RAM
      INTSOFF
      PAGE0
      MOVF    WORK2, W
      CALL     SETUP_MADDR
      RD_MPH_L        RXCP_UHCSL1     ; read uncorrectable HCS error count
      MOVLW   0xFF                    ; LSB
      ANDWF   DATA_REG, F             ; Mask off unused bits
      WR_RAM_DP       RXCP_UHCSL1_RAM ; copy to ram
      INTSON
```

```
INTSOFF
PAGE0
MOVF    WORK2, W
CALL     SETUP_MADDR
RD_MPH_L        RXCP_UHCSM1     ; read uncorrectable HCS error count
MOVLW   0x0F                    ; MSB
ANDWF   DATA_REG, F             ; Mask off unused bits
WR_RAM_DP       RXCP_UHCSM1_RAM ; copy to ram
INTSON

INTSOFF
PAGE0
MOVF    WORK2, W
CALL     SETUP_MADDR
RD_MPH_L        RXCP_CHCSL1     ; read correctable HCS error count
MOVLW   0xFF                    ; LSB
ANDWF   DATA_REG, F             ; Mask off unused bits
WR_RAM_DP       RXCP_CHCSL1_RAM ; copy to ram
INTSON

INTSOFF
PAGE0
MOVF    WORK2, W
CALL     SETUP_MADDR
RD_MPH_L        RXCP_CHCSM1     ; read uncorrectable HCS error count
MOVLW   0x0F                    ; LSB
ANDWF   DATA_REG, F             ; Mask off unused bits
WR_RAM_DP       RXCP_CHCSM1_RAM ; copy to ram
INTSON

INTSOFF
PAGE0
MOVF    WORK2, W
CALL     SETUP_MADDR
RD_MPH_L        RXCP_IDLEL1     ; read Idle/unassigned cell count
MOVLW   0xFF                    ; LSB
ANDWF   DATA_REG, F             ; Mask off unused bits
WR_RAM_DP       RXCP_IDLEL1_RAM ; copy to ram
INTSON

INTSOFF
PAGE0
MOVF    WORK2, W
CALL     SETUP_MADDR
RD_MPH_L        RXCP_IDLEM1     ; read Idle/unassigned cell count
MOVLW   0xFF                    ; MSB
ANDWF   DATA_REG, F             ; Mask off unused bits
WR_RAM_DP       RXCP_IDLEM1_RAM ; copy to ram
INTSON

INTSOFF
PAGE0
MOVF    WORK2, W
CALL     SETUP_MADDR
RD_MPH_L        RXCP_RECVL1     ; read RXCP recieive cell count
MOVLW   0xFF                    ; LSB
ANDWF   DATA_REG, F             ; Mask off unused bits
WR_RAM_DP       RXCP_RECVL1_RAM ; copy to ram
INTSON
```

```
      INTSOFF
      PAGE0
      MOVF    WORK2, W
      CALL     SETUP_MADDR
      RD_MPH_L        RXCP_RECVM1     ; read RXCP recieive cell count
      MOVLW   0xFF                    ; MSB
      ANDWF   DATA_REG, F             ; Mask off unused bits
      WR_RAM_DP       RXCP_RECVM1_RAM ; copy to ram
      INTSON

      INTSOFF
      PAGE0
      MOVF    WORK2, W
      CALL     SETUP_MADDR
      RD_MPH_L        TXCP_XMITL1     ; read TXCP transmit cell count
      MOVLW   0xFF                    ; LSB
      ANDWF   DATA_REG, F             ; Mask off unused bits
      WR_RAM_DP       TXCP_XMITL1_RAM ; copy to ram
      INTSON

      INTSOFF
      PAGE0
      MOVF    WORK2, W
      CALL     SETUP_MADDR
      RD_MPH_L        TXCP_XMITM1     ; read TXCP transmit cell count
      MOVLW   0xFF                    ; MSB
      ANDWF   DATA_REG, F             ; Mask off unused bits
      WR_RAM_DP       TXCP_XMITM1_RAM ; copy to ram
      INTSON


G_BITS
      INTSOFF
      PAGE0
      MOVF    WORK2, W
      CALL    SETUP_MADDR
      RD_MPH_L        CRCM1           ; read MSB of CRC count
      MOVLW   0x02                    ; mask off unused bits
      ANDWF   DATA_REG, F
      WR_RAM_DP       CRCM1_RAM       ; copy it to RAM
      MOVF    DATA_REG, W
      INTSON
      PAGE1
      BTFSS   STATUS, Z               ; is MSB of CRC count > 0 ?
      GOTO    BE_256PLUS
      INTSOFF
      PAGE0
      MOVF    WORK2, W
      CALL    SETUP_MADDR
      RD_MPH_L        CRCL1           ; read LSB of CRC count
      WR_RAM_DP       CRCL1_RAM       ; copy it to RAM
      MOVF    DATA_REG, W
      INTSON
      PAGE1
      BTFSC   STATUS, Z               ; is LSB of CRC count = 0 ?
      GOTO    SEF_BIT                 ; yes then no need to set any G bits
      XORLW   0x01
      BTFSS   STATUS, Z               ; is LSB of CRC count = 1 ?
```

```
      GOTO    BE_2PLUS
      INCF    FSR, F                  ; yes then
      BSF     INDF, PRM_G1            ; set G1
      DECF    FSR, F
      GOTO    SEF_BIT
BE_2PLUS
      MOVF    DATA_REG, W
      SUBLW   0x05
      BTFSS   STATUS, C               ; is LSB of CRC count < 6 ?
      GOTO    BE_6PLUS
      INCF    FSR, F                  ; yes then
      BSF     INDF, PRM_G2            ; set G2
      DECF    FSR, F
      GOTO    SEF_BIT
BE_6PLUS
      MOVF    DATA_REG, W
      SUBLW   0x0A
      BTFSS   STATUS, C               ; is LSB of CRC count < 11 ?
      GOTO    BE_11PLUS
      BSF     INDF, PRM_G3            ; yes then set G3
      GOTO    SEF_BIT
BE_11PLUS
      MOVF    DATA_REG, W
      SUBLW   0x64
      BTFSS   STATUS, C               ; is LSB of CRC count < 101 ?
      GOTO    BE_101PLUS
      BSF     INDF, PRM_G4            ; yes then set G4
      GOTO    SEF_BIT
BE_101PLUS
      BSF     INDF, PRM_G5            ; CRC count must be 100<x<256
                              ; so set G5
      GOTO    SEF_BIT
BE_256PLUS
      INTSOFF
      PAGE0
      MOVF    WORK2, W
      CALL    SETUP_MADDR
      RD_MPH_L        CRCL1
      WR_RAM_DP       CRCL1_RAM
      MOVF    DATA_REG, W
      INTSON
      PAGE1
      SUBLW   0x3F
      BTFSS   STATUS, C               ; is LSB of CRC count < 64 ?
      GOTO    BE_320PLUS
      BSF     INDF, PRM_G5            ; yes then set G5
      GOTO    SEF_BIT
BE_320PLUS
      BSF     INDF, PRM_G6            ; CRC count must be x>319
                              ; so set G6
SEF_BIT
      INTSOFF                         ; need to copy framing error count
      PAGE0
      MOVF    WORK2, W                ; anyway
      CALL    SETUP_MADDR
      RD_MPH_L        FER1            ; read framing error count
      MOVLW   0x1F                    ; mask off unused bits
      ANDWF   DATA_REG, F
      WR_RAM_DP       FER1_RAM        ; copy it to RAM
      INTSOFF
```

```
      PAGE0
      MOVF    WORK2, W
      CALL    SETUP_MADDR
      INTSON
      PAGE1
      BTFSC   SEF, WORK2              ; is severely errored frame count >= 1 ?
      GOTO    FE_BIT
      INCF    FSR, F                  ; yes then
      BSF     INDF, PRM_SE            ; set SE
      DECF    FSR, F
      PAGE1
      GOTO    LV_BIT
FE_BIT
      MOVF    DATA_REG, W
      INTSON
      PAGE1
      BTFSC   STATUS, Z               ; is framing error count >= 1 ?
      GOTO    LV_BIT
      INCF    FSR, F                  ; yes then
      BSF     INDF, PRM_FE            ; set FE
      DECF    FSR, F
LV_BIT
      INTSOFF
      PAGE0
      MOVF    WORK2, W
      CALL    SETUP_MADDR
      RD_MPH_L        LCVL1           ; read LSB of line code violation count
      WR_RAM_DP       LCVL1_RAM       ; copy it to RAM
      MOVF    DATA_REG, W
      INTSON
      PAGE1
      BTFSC   STATUS, Z               ; is line code violation LSB >= 1 ?
      GOTO    LCVL_ZERO
      BSF     INDF, PRM_LV            ; yes then set LV
                              ; but have to read/copy MSB of line
                              ; code violation anyway
LCVL_ZERO
      INTSOFF
      PAGE0
      MOVF    WORK2, W
      CALL    SETUP_MADDR
      RD_MPH_L        LCVM1           ; read MSB of line code violation count
      MOVLW   0x0F                    ; mask off unused bits
      ANDWF   DATA_REG, F
      WR_RAM_DP       LCVM1_RAM       ; copy it to RAM
      MOVF    DATA_REG, W
      INTSON
      PAGE1
      BTFSC   STATUS, Z               ; is line code violation MSB >= 1 ?
      GOTO    SL_BIT
      BSF     INDF, PRM_LV            ; yes then set LV
SL_BIT
      BCF     INDF, PRM_SL            ; slip bit is always zero

LB_BIT
      INCF    FSR, F                  ; set LB if this quadrant is in
                              ; payload loopback
      INTSOFF
      PAGE0
      MOVF    WORK2, W
```

```
      CALL    SETUP_MADDR
      RD_MPH_L        MSTR_DIAG       ; read master diagnostic register
                              ; to check for payload loopback
      INTSON
      PAGE1
      BSF     INDF, PRM_LB
      BTFSS   DATA_REG, 5
      BCF     INDF, PRM_LB

      MOVLW   Q1_PRM_PTR              ; clear the PRM pointer
      MOVWF   FSR
      MOVF    WORK2, W
      ADDWF   FSR, F
      CLRF    INDF

      BTSS_QUAD_BIT   WORK2, XFDL_BUSY
      GOTO    PRM_ENBL_XFDL
      BS_QUAD_BIT     WORK2, XFDL_PENDING
      RETURN

PRM_ENBL_XFDL
      BS_QUAD_BIT     WORK2, XFDL_BUSY        ; set XFDL busy bit
      BS_QUAD_BIT     WORK2, XFDL_USR_OR_PRM  ; set user/PRM bit to PRM
      MOVF    WORK2, W
      INTSOFF
      PAGE0
      CALL    SETUP_MADDR
      RD_MPH_L        XFDL_CONFIG
      BSF     DATA_REG, 3             ; enable XFDL interrupts
      WR_MPH_DL       XFDL_CONFIG
      PAGE1
      INTSON
      RETURN                          ; end of GENERATE_PRM

;***********************************
;* PERFORMANCE MONITORING SUBROUTINE *
;***********************************

PMON
      BCF     TIMEFLAG, 0             ; clear the 1 sec. flag
      ; Write to MPH to update all PMON counters
      INTSOFF
      WR_MPH   GLOBAL_PMON, 00
      INTSON
      PAGE1

PMON_LOOP
      RD_MPH  GLOBAL_PMON
      PAGE1
      BTFSC   DATA_REG, 3      ; tight loop until PMON registers are updated
      GOTO    PMON_LOOP

      INCF    PRM_COUNTER, W  ; increment performance report counter
      ANDLW   0x03            ; and take modulus 4
      MOVWF   PRM_COUNTER

      MOVLW   0x00                    ; create performance report messages
      CALL    GENERATE_PRM            ; and make RAM copies of performance
      MOVLW   0x01                    ; monitoring statistics for each
      CALL    GENERATE_PRM            ; quadrant
```

```
MOVLW   0x02
CALL    GENERATE_PRM
MOVLW   0x03
CALL    GENERATE_PRM

INTSOFF
WR_RAM  MAILOUT, PMON_UPDATE    ; signal update to host
MOVF    QUADRANT, W             ; indicate which quadrant
MOVWF   DATA_REG
WR_RAM_D        INT_QUAD
INTSON

RETURN                  ; return from subroutine

END             ;** END OF SOURCE FILE **
```

CONTACTING PMC-SIERRA

PMC-Sierra, Inc.
105 - 8555 Baxter Place
Burnaby, B.C.
Canada V5A 4V7

Telephone: 604-415-6000
Facsimile: 604-415-6200

Product Information: info@pmc-sierra.bc.ca
Applications information: apps@pmc-sierra.bc.ca

World Wide Web Site: http://www.pmc-sierra.com

NOTES

Seller will have no obligation or liability in respect of defects or damage caused by unauthorized use, mis-use, accident, external cause, installation error, or normal wear and tear. There are no warranties, representations or guarantees of any kind, either express or implied by law or custom, regarding the product or its performance, including those regarding quality, merchantability, fitness for purpose, condition, design, title, infringement of third-party rights, or conformance with sample. Seller shall not be responsible for any loss or damage of whatever nature resulting from the use of, or reliance upon, the information contained in this document. In no event will Seller be liable to Buyer or to any other party for loss of profits, loss of savings, or punitive, exemplary, incidental, consequential or special damages, even if Seller has knowledge of the possibility of such potential loss or damage and even if caused by Seller's negligence.

© 1997 PMC-Sierra, Inc.

PMC-960951 Issue date: June, 1997

Printed in Canada