PM4344

TQUAD

TQUAD Module of the PM4944 M13 Reference Design

Issue 2: August, 1996

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 4), "PM4344 TQUAD Quadruple T1 Framer", PMC-940910P4
- PMC-Sierra, Reference Design (October 1995), "PM4944 M13 Application Reference Design Utilizing the D3MX and TQUAD", PMC-950715
- PMC-Sierra, Reference Design (October 1995), "D3MX Module of PM4944 M13 Reference Design", PMC-951046

OVERVIEW

This document describes a possible implementation of the TQUAD Modules for the PM4944 M13 reference design.

The PM4944 M13RD (M13 Reference Design) embodies PMC-Sierra's guidelines and suggestions for designing an M13 multiplexer using PMC-Sierra products (such as the PM8313 D3MX and the PM4344 TQUAD). The full M13 reference design consists of two or more types of modules: the D3MX Module (described in PMC-951046) and the TQUAD Module (described in this document). In fact seven of these TQUAD Modules are required per M13 application, since each TQUAD Module processes four of the 28 duplex DS1 tributary serial streams.

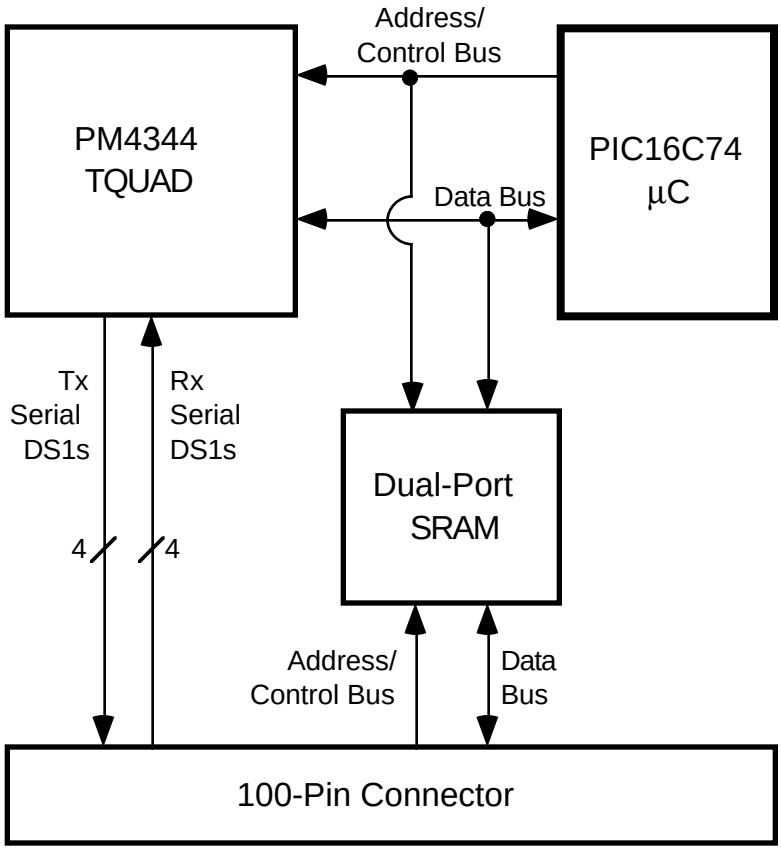
In addition to the PM4344 TQUAD itself, the TQUAD Module incorporates an on-board microcontroller (Microchip PIC16C74) for providing the local maintenance functions — including termination of the ESF datalink in the DS1 overhead.

The TQUAD Module communicates with the 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 TQUAD. Rather, a dual-port RAM, shared by the host and the local PIC16C74, is used to pass control and status information about the TQUAD to and from the host, where the actual register accesses to the TQUAD 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. The host system and PIC16C74 also communicate through the "mailbox" communication channels in the dual-port RAM.

FUNCTIONAL DESCRIPTION

Block Diagram



PM4344 TQUAD

The PM4344 Quadruple T1 Framer (TQUAD) is a feature-rich device suitable for use in many T1 systems with a minimum of external circuitry. Each of the framers and transmitters is independently software-configurable, allowing feature selection without changes to external wiring.

On the receive side, the TQUAD can be configured to frame to any of the common DS-1 signal formats: SF, ESF, T1DM (DDS), or SLC®96. The TQUAD also supports detection of various alarm conditions such as loss of signal, pulse density violation, RED alarm, YELLOW alarm, and AIS alarm. The TQUAD 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 TQUAD also detects the presence of in-band loopback codes, ESF bit-oriented codes, and detects and terminates HDLC messages on the ESF data link. An elastic store for slip buffering and rate adaptation is provided, as is a signaling extractor that supports signaling debounce, signaling freezing, idle code substitution, digital milliwatt tone substitution, data inversion, and signaling bit fixing — all on a per-channel basis. Receive-side data and signaling trunk conditioning is also provided.

On the transmit side, the TQUAD generates framing for SF, ESF, and T1DM (DDS) DS1 formats, or framing can be optionally disabled. The transmission of SLC®96 format is not supported in this design. The TQUAD supports signaling insertion, idle code substitution, digital milliwatt tone substitution, data inversion, and zero code suppression — all on a per-channel basis. The zero code suppression is selectable to Bell (bit 7), GTE, or DDS standards, and can also be disabled. Transmit-side data and signaling trunk conditioning is provided.

The TQUAD can also generate in-band loopback codes, ESF bit-oriented codes, and transmit HDLC messages on the ESF data link. The TQUAD can generate a low jitter transmit clock and provides a FIFO for transmit jitter attenuation. When not used for jitter attenuation, the full or empty status of this FIFO is made available to facilitate higher order multiplexing applications by controlling bit-stuffing logic.

The TQUAD provides a parallel microprocessor interface for controlling the operation of the TQUAD device. Serial PCM interfaces allow 1.544 Mbit/s or 2.048 Mbit/s backplanes to be directly supported. Tolerance of gapped clocks allows other backplane rates to be supported with a minimum of external logic.

For a complete description of the TQUAD, please refer to PMC-Sierra's PM4344 databook (document number PMC-910910P4).

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 5 define the memory map for the dual-port RAM. The memory map is organized to provide a generic flexible interface to the TQUAD 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 1 (80h)
300	XFDL transmit buffer and status, quadrant 1 (80h)
380	XFDL transmit buffer and status, quadrant 1 (80h)
400	PMON shadow registers, quadrant 1 (6h)
410	PMON shadow registers, quadrant 2 (6h)
420	PMON shadow registers, quadrant 3 (6h)
430	PMON shadow registers, quadrant 4 (6h)
436	SIGX shadow registers, quadrant 1 (18h)
44E	SIGX shadow registers, quadrant 2 (18h)
466	SIGX shadow registers, quadrant 3 (18h)
47E	SIGX shadow registers, quadrant 4 (18h)
496	New signaling value (HOST->PIC)
497	Quadrant to affect with signalling command (HOST->PIC)
498	DS0 channel to affect with signalling 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 6 bytes

Relative Address (hex)	Function
0	Line code violation count LSB
1	Line code violation count MSB
2	Bit-error count LSB
3	Bit-error count MSB
4	Framing error count
5	Out-of-frame count

Table 5. SIGX Memory block of 18H bytes

Relative Address (hex)	Function
00	DS0 channel 1 signaling byte
01	DS0 channel 2 signaling byte
...	...
17	DSO channel 24 signaling byte

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 6 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 6. Host-to-PIC16C74 Mailbox Codes

Function	Code (hex)
Read TQUAD register	01
Write TQUAD register	02
Start FDL packet transmission	03
Start BOC transmission	04
Stop BOC transmission	05
Reserved	06
Reserved	07
Enable DS0 Idle Code insertion	08
Disable DS0 Idle Code insertion	09
Enable DS0 Digital Milliwatt insertion	0A
Disable DS0 Digital Milliwatt insertion	0B
Enable signaling transmission	0C
Disable signaling transmission	0D
Select internal signaling source	0E
Select external signaling source	0F
Change signaling value	10
Change DS0 Idle Code	11

READ TQUAD REGISTER

This function is useful for diagnostic purposes to read the value of a TQUAD register. To read a value from a specific TQUAD 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 TQUAD 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 TQUAD REGISTER

This function is useful for diagnostic purposes to write a TQUAD register with a value. To write a value from a specific TQUAD 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

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

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

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).

ENABLE IDLE CODE INSERTION

To enable Idle Code insertion on a particular DS0 channel of a quadrant perform the following steps:

1. Write the quadrant number (0-3) to RAM location 497.
2. Write the DS0 channel number (01H-18H) to RAM location 498.
3. Write the mailbox command 08 to the PIC16C74 Mailbox (location 7FF).

DISABLE IDLE CODE INSERTION

To disable Idle Code insertion on a particular DS0 channel of a quadrant perform the following steps:

1. Write the quadrant number (0-3) to RAM location 497.
2. Write the DS0 channel number (01H-18H) to RAM location 498.
3. Write the mailbox command 09 to the PIC16C74 Mailbox (location 7FF).

ENABLE DIGITAL MILLIWATT INSERTION

To enable DMW insertion on a particular DS0 channel of a quadrant perform the following steps:

1. Write the quadrant number (0-3) to RAM location 497.
2. Write the DS0 channel number (01H-18H) to RAM location 498.
3. Write the mailbox command 0A to the PIC16C74 Mailbox (location 7FF).

DISABLE DIGITAL MILLIWATT INSERTION

To disable DMW insertion on a particular DS0 channel of a quadrant perform the following steps:

1. Write the quadrant number (0-3) to RAM location 497.
2. Write the DS0 channel number (01H-18H) to RAM location 498.
3. Write the mailbox command 0B to the PIC16C74 Mailbox (location 7FF).

ENABLE SIGNALING TRANSMISSION

To enable the transmission of robbed-bit signaling on a particular DS0 channel of a quadrant perform the following steps:

1. Write the quadrant number (0-3) to RAM location 497.
2. Write the DS0 channel number (01H-18H) to RAM location 498.
3. Write the mailbox command 0C to the PIC16C74 Mailbox (location 7FF).

DISABLE SIGNALING TRANSMISSION

To disable the transmission of robbed-bit signaling on a particular DS0 channel of a quadrant perform the following steps:

1. Write the quadrant number (0-3) to RAM location 497.
2. Write the DS0 channel number (01H-18H) to RAM location 498.
3. Write the mailbox command 0D to the PIC16C74 Mailbox (location 7FF).

SELECT INTERNAL SIGNALING SOURCE

Once signaling transmission is enabled for a particular DS0 channel, it can either be sourced from TQUAD internal registers or from the TQUAD's BTSIG[x] input pins. To select internally sourced signaling on a particular DS0 channel of a quadrant perform the following steps:

1. Write the quadrant number (0-3) to RAM location 497.
2. Write the DS0 channel number (01H-18H) to RAM location 498.
3. Write the mailbox command 0E to the PIC16C74 Mailbox (location 7FF).

SELECT EXTERNAL SIGNALING SOURCE

Once signaling transmission is enabled for a particular DS0 channel, it can either be sourced from TQUAD internal registers or from the TQUAD's BTSIG[x] input pins. To select externally sourced signaling on a particular DS0 channel of a quadrant perform the following steps:

1. Write the quadrant number (0-3) to RAM location 497.
2. Write the DS0 channel number (01H-18H) to RAM location 498.
3. Write the mailbox command 0F to the PIC16C74 Mailbox (location 7FF).

CHANGE SIGNALING VALUE

If a transmitted DS0 channel's signaling is being sourced internally, the signaling value can be changed. To change the signaling value on a particular DS0 channel of a quadrant perform the following steps:

1. Write the quadrant number (0-3) to RAM location 497.
2. Write the DS0 channel number (01H-18H) to RAM location 498.
3. Write the new signaling value, ABCD, in the least significant nybble of RAM location 496. For four-state signaling (e.g. in SF format), the AB bits should be repeated, ABAB, in the least significant nybble of location 496. For two-state signaling, the A bit should be repeated in all four bits, AAAA, of the least significant nybble of location 496.
3. Write the mailbox command 10 to the PIC16C74 Mailbox (location 7FF).

CHANGE IDLE CODE

To change the transmitted DS0 Idle Code on a particular DS0 channel of a quadrant perform the following steps:

1. Write the quadrant number (0-3) to RAM location 497.
2. Write the DS0 channel number (01H-18H) to RAM location 498.
3. Write the new Idle Code to RAM location 499.
3. Write the mailbox command 11 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 7 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 7. 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	RED alarm asserted
0D	RED alarm cleared
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)
23	FDL packet has been successfully transmitted (Q4)
FF	Requested TQUAD register I/O completed
80	PMON statistics updated

AIS ASSERTED (06)

This message is sent when the TQUAD 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 TQUAD 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 TQUAD 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 TQUAD 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.

RED ALARM ASSERTED (0C)

This message is sent when the TQUAD detects an out-of-frame condition for a sufficient duration to declare a RED alarm. The originating quadrant can be determined by reading RAM location 7F0. The RED declaration will take 2.5 seconds to integrate under a continuous out-of-frame condition. Note that since an AIS pattern is an un-framed signal, a RED alarm will be declared during reception of AIS.

RED ALARM CLEARED (0D)

This message is sent when the TQUAD detects an in-frame condition for a sufficient duration to clear the RED alarm. The originating quadrant can be determined by reading RAM location 7F0. The RED clear will take 16.6 seconds to integrate under a continuous in-frame condition.

VALID BOC DETECTED (0E)

This message is sent when the TQUAD 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 (F)

This message is sent when the TQUAD 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 TQUAD 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 TQUAD 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 TQUAD 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 TQUAD 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.

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 TQUAD 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 PM4344 TQUAD.

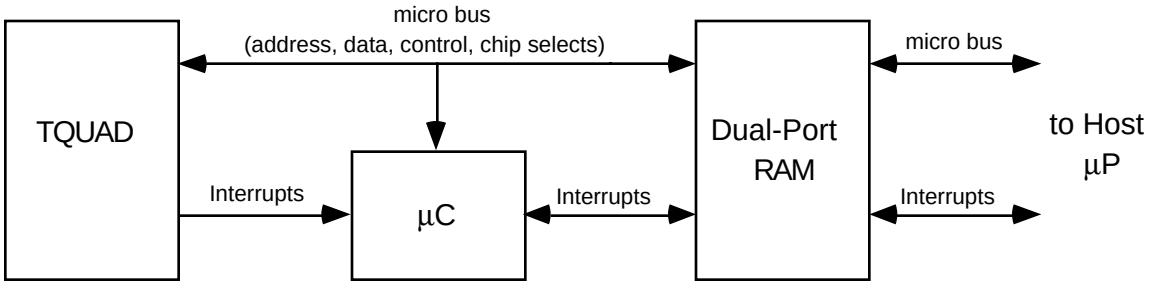
Firmware

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 TQUAD 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 two external devices, it might as well generate 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 TQUAD 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 four general access routines: RD_TQUAD, RD_RAM, WR_TQUAD, and WR_RAM. All four routines share a common Data register so that data transfers could be optimized.

Initialization

The firmware initializes the TQUAD 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 TQUAD is initialized for Extended Superframe Format (ESF) in all quadrants.

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.

Lastly, 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 PM4344 TQUAD 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 TQUAD 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. In SF format, the YELLOW Alarm consists of forcing Bit 2 of every DS0 channel to a logic 0; in ESF format, the YELLOW Alarm consists of sending eight ones followed by eight zeros repeatedly over the facility datalink channel.

The firmware automatically transmits a YELLOW Alarm for quadrants which are in a RED Alarm state. It also 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.

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 TQUAD.

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 TQUAD 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 TQUAD 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 performance monitoring parameters required by ANSI T1.231 are available within registers of the TQUAD. 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.

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 8.

Table 8. 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 9.

Table 9. 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 Line Code Violation Events occurred during the interval,
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

Channel-Associated Signaling

The TQUAD extracts the channel-associated (robbed-bit) signaling information for each DS0 channel. The firmware transfers this information into the dual-port RAM so it can be polled by the host.

In the transmit direction, the signaling for each channel is transmitted if it is enabled with the Enable Signaling Transmission command. Each DS0 can be independently controlled. Once enabled, the signaling data for each DS0 can be sourced from either internal TQUAD registers or from the BTSIG[x] input pins of the TQUAD. If enabled and internally sourced, the signaling data for a particular DS0 will remain constant until it is changed with a Change Signaling Value command. The value of each channel's signaling data can be independently controlled.

IMPLEMENTATION DESCRIPTION

The schematic diagram of the TQUAD Module is contained in Appendix C. This section explains that schematic diagram.

Hardware

The TQUAD Module schematic shows three main functional blocks: a PIC16C74 microcontroller, dual-port RAM, and a PM4344 TQUAD. 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

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

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

Each TQUAD Module has been given 2 kB of space because they must accumulate information on *four* TQUAD 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 TQUAD).

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 TQUAD even though it is not directly connected to the TQUAD's microprocessor port,
- 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.

TQUAD Functional Block

The TQUAD is a full-featured quadruple DS1 framer which provides most standard DS1 framing functions and performance monitoring.

The full register set of the TQUAD, including Test Mode registers are accessible to the PIC16C74 block. For a full description of these registers, refer to the TQUAD Data Book.

The backplane signals of the TQUAD is connected to header blocks. In a real application, these signals would be further processes. The header blocks are arranged to allow easy physical (payload) loopbacks on the TQUAD Module.

Timing Distribution

The high-speed and backplane timing to the TQUAD Module are sourced from the D3MX Module through the 100-pin connector.

The TQUAD requires a 37.056MHz high-speed clock. The PIC16C74 requires a high-speed clock, with 20MHz being the maximum frequency. The PIC16C74 has a built-in oscillator which can operate with a simple external RC network, although the frequency will not be tightly controlled. It is recommended that all clocks be sourced from the D3MX Module since that provides the lowest cost solution.

There are also two backplane timing signals (clock and frame sync pulse) coming from the D3MX Module over the 100-pin connector. Via the header blocks, these signals can be used to synchronize the framer backplanes for synchronous switching applications.

Header Blocks

To keep this reference design general, the backplane signals of the TQUAD are not further processed. However, these signals are brought to header blocks, so that they can be observed and inserted as appropriate. The following signals are brought to headers:

- BRPCM[4:1] are the four Backplane Receive PCM serial data streams.
- BRSIG[4:1] are the four Backplane Receive Signaling serial data streams which indicate the extracted robbed-bit signaling information of each PCM timeslot on BRPCM.
- BRFPO{4:1] are the four Backplane Receive Frame Pulse Outputs which indicate the frame alignment within the associated BRPCM and BRSIG streams.
- RCLKO[4:1] are the four Receive Clock Outputs. If an Elastic Store (ELST) within the TQUAD is bypassed, then the associated BRPCM and BRSIG signals are timed to this clock.
- BRCLK_EXT is the Backplane Receive Clock. This clock is provided across the 100-pin connector. If an Elastic Store (ELST) within the TQUAD is enabled, then the associated BRPCM and BRSIG signals are timed to this clock. This clock is common to all the quadrants within the TQUAD.

For more detailed information on the use of these signals, refer to the TQUAD data book.

The headers are physically arranged to facilitate physical loopbacks. Each backplane receive PCM stream can be looped back by physically connecting the BRPCM[n] and BRSIG[n] signals to BTPCM[n] and BRSIG[n] respectively. Additionally, either the RCLKO[n] or the BRCLK_EXT signal should be connected to BTCLK[n] (depending whether the ELST is bypassed or not).

100-Pin Connector

The 100-pin connector is used to connect the TQUAD Module to a D3MX Module. This connector carries four duplex DS1 signals (clock and data), a microprocessor control, address, and data bus, as well as power to the TQUAD Module. The signals are defined in the following table.

In the 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 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).
BFPI	I	A5	Backplane frame pulse. This is an 8kHz signal used to synchronize the frame alignment of the framer backplanes for synchronous switching applications.
BCLK	I	A6	Backplane clock. 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
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
RCLKI[1]	I	A21	Receive Clock for tributary #1
RDD[1]	I	A22	Receive Digital Data for tributary #1
GND	GND	A23, A24	Ground
RCLKI[2]	I	A25	Receive Clock for tributary #2
RDD[2]	I	A26	Receive Digital Data for tributary #2
GND	GND	A27, A28	Ground
RCLKI[3]	I	A29	Receive Clock for tributary #3
RDD[3]	I	A30	Receive Digital Data for tributary #3
GND	GND	A31, A32	Ground
RCLKI[4]	I	A33	Receive Clock for tributary #4
RDD[4]	I	A34	Receive Digital Data for tributary #4
GND	GND	A35, A36	Ground
TCLKO[1]	O	A37	Transmit Clock for tributary #1
TDD[1]	O	A38	Transmit Digital Data for tributary #1
GND	GND	A39, A40	Ground
TCLKO[2]	O	A41	Transmit Clock for tributary #2
TDD[2]	O	A42	Transmit Digital Data for tributary #2
GND	GND	A43, A44	Ground
TCLKO[3]	O	A45	Transmit Clock for tributary #3
TDD[3]	O	A46	Transmit Digital Data for tributary #3
VDD	PWR	A47, A48	+5V
N/C	N/C	A49	No connection
RSTB	I	A50	Hardware Reset (active low)
TCLKO[4]	O	A51	Transmit Clock for tributary #4
TDD[4]	O	A52	Transmit Digital Data for tributary #4
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 TQUAD Module for microprocessor access.
N/C	N/C	A86	No connection
GND	GND	A87, A88	Ground
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]
TQCLK	I	A99	High-Speed Clock (37.056MHz) for TQUAD
MCLK	I	A100	High-Speed Clock (20.000MHz) for PIC16C74

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 PM4344 TQUAD device. *It is the responsibility of the person 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 TQUAD 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 TQUAD or the dual-port RAM that interrupted. If it was the TQUAD, then the TQUAD registers 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 11 describes the macros defined in the *macros.inc* file.

Table 11. 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 *TQUAD.ASM* File

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

CONSTANTS

The first portion of the *tquad.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.
- TQUAD direct access registers.
- TQUAD SIGX and TPSC 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 12 describes the macros used to perform I/O operations with the TQUAD and the dual-port RAM.

Table 12. Description of Macros in *TQUAD.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_TQUAD	*register*, *value*	Writes *value* (byte) to the *register* in the TQUAD. Only the lower 9 bits of the *register* argument are used.
WR_TQUAD_L	*register*, *value*	Writes *value* (byte) to a TQUAD *register* in a particular quadrant. Only the lower 7 bits are used, the upper 2 bits in the TQUAD address register (bit 7 of ADDR_TQUAD_LO and bit 0 of ADDR_TQUAD_HI) must be setup prior to invoking this macro.
WR_TQUAD_DL	*register*	Writes the value in the data register to a TQUAD *register* in a particular quadrant. Only the lower 7 bits are used, the upper 2 bits in the TQUAD address register 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_TQUAD	*register*	Reads from the *register* in the TQUAD. Only the lower 9 bits of the *register* argument are used. The value is returned in the data register and the W register.
RD_TQUAD_L	*register*	Reads from a TQUAD *register* in a particular quadrant. Only the lower 7 bits are used, the upper 2 bits in the TQUAD 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 *tquad.asm* file.

Main Loop

The processor loops infinitely waiting for the 3ms or 1s time flags to be set. When the 3ms flag is set the processor executes the SIGX subroutine, updating signaling shadow registers in RAM, and returns to the main loop. When the 1s flag is set the processor executes the PMON subroutine, updating performance monitoring 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 TQUAD to trigger the update of performance statistics. The routine then delays for 6 µs (to allow for update latency of the TQUAD) then reads the performance statistic data from the TQUAD PMON 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 TQUAD PMON registers and constructs the octet pair for the last second accordingly. Besides the TQUAD PMON data, the GENERATE_PRM subroutine also reads the ELST Interrupt Status register (to determine if slips have occurred in the last second) and 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.

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).

Signaling Extraction

Shadowing of signaling data in the dual-port RAM is done by the SIGX subroutine. Signaling data is copied out for each DS0 channel of each quadrant through indirect register accesses.

Read Access to TQUAD

The READ_TQUAD subroutine is used by the RD_TQUAD and RD_TQUAD_L macros. The address for the TQUAD 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 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 TQUAD

The WRITE_TQUAD subroutine is used by the WR_TQUAD, WR_TQUADL and WR_TQUAD_DL macros. The address for the TQUAD 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 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 TQUAD

The INIT_TQUAD routine is called four times on start-up, once per quadrant. The framer for the particular quadrant is reset. Table 13 shows the TQUAD registers initialized and the value to which they are set. The SIGX and TPSC indirect access registers are also initialized here.

The TQUAD is initialized for the Extended Superframe Format (ESF) for all channels.

Table 13. TQUAD Initialization Register Values

Register	Location	Value
Data Link Options	02	0F
Receive DS1 Configuration	03	04
Transmit DS1 Configuration	04	08
CDRC Configuration	10	84
CDRC Interrupt Enable	11	40
FRMR Configuration	20	30
FRMR Interrupt Enable	21	01
RBOC Enable	2A	05
ALMI Configuration	2C	10
ALMI Interrupt Enable	2D	07
TPSC Configuration	30	03
XFDL Configuration	34	03
RFDL Configuration	38	01
RFDL Interrupt Enable/Status	39	02
IBCD Configuration	3C	04
IBCD Interrupt Enable/Status	3D	30
IBCD Activate Code	3E	08
IBCD Deactivate Code	3F	24
SIGX Configuration	40	13
XBAS Configuration	44	10

Interrupt Service

The INTHDLR routine deals with interrupts asserted by the TQUAD, 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 TQUAD interrupt is serviced as long as it is asserted to minimize latency when there are multiple pending TQUAD interrupts. This is important because the interrupts are edge-triggered.

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

TQUAD Interrupt Service

The TQUAD_INT routine determines the source of a TQUAD 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.

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.

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.

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 TQUAD 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 $\Delta \mathbf{I}/\Delta \mathbf{t}$ 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	CAP-0.001UF	SMDCAP805	C1-C4,C26-C33	12
2	CAP-100000PF	SMDCAP1206	C5-C9,C12-C14, C17,C18,C21-C24	14
3	CAP-10000PF	SMDCAP805	C10	1
4	CAPACITOR POL-10UF, 16V,TANT TEH	SMDTANCAP_C	C19	1
5	CONN100-AMP_103911-8	AMP_103911-8	P1	1
6	CY7C136_-BASE	PLCC52	U3	1
7	HEADER8-BASE	SIP8	J1-J5	5
8	LED10-RED,25MA,2.1V	DIP20_LED	U2	1
9	PIC16C74-BASE	PIC16C74	U1	1
10	PWRBLOCK_2-BASE	CONN2END	J6	1
11	RESISTOR-10K,5%	SMDRES805	R12-R15	4
12	RESISTOR-4.7K,5%	SMDRES805	R1,R2,R6,R8-R11	7
13	RESISTOR-75,1%	SMDRES805	R3	1
14	RES_ARRAY_8_SMD-10K	SOIC16	RN1,RN3	2
15	RES_ARRAY_8_SMD-270	SOIC16	RN2	1
16	TQUAD-BASE	PQFP128	U7	1
17	TST_PT-BASE	TST_PT_1	TP1-TP10	10

APPENDIX C: SCHEMATICS

QUAD T1 FRAMER

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPR
E3, D6	2.0	PULL UP RESISOTRS ADDED	06/13/96	

NOTE: PLACE ALL DECOUPLING CAPS NEAR POWER PINS

DO NOT POPULATE R3 OR C14 IF SERIES TERMINATION IS USED

PLACE TERMINATION AS CLOSE TO TQUAD AS POSSIBLE

DECOUPLING FOR THE MICROCONTROLLER AND THE DUAL PORT RAM PLACE ONE AT EACH POWER PIN

HEADER BLOCK FOR TESTING PURPOSES

TEST POWER POINTS

POWER TERMINAL

BULK POWER DECOUPLING

STATUS LEDS

DRAWING

TITLE=TQUAD_ROOT
ABBREV=TQUAD
LAST_MODIFIED=Thu Sep 5 15:01:56 1996

PMC-Sierra, Inc.	
DOCUMENT NUMBER: PMC-951003	ISSUE: 2
TITLE: M13 REFERENCE DESIGN TQUAD MODULE	DATE: JUNE 13, 96
ENGINEER: AARON GILROY	PAGE: 1 OF 1

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
```

TQUAD.ASM file

```
;*******************************************************
;* TQUAD Module PIC Firmware                           *
;* Version 1.03 : August 26, 1996                      *
;* Authors: Andrew Jones                               *
;*          Sean Puttergill                            *
;*******************************************************

         TITLE "PIC ASSEMBLY SOURCE CODE FOR TQUAD 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

;************************
;* CONSTANT DEFINITIONS *
;************************

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

         CONSTANT VER=0x01 ; version number
         CONSTANT REV=0x03 ; revision number
         __IDLOCS          0x0103

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

; define BANK 0 of microcontroller user registers
REGISTERS
         CBLOCK  20
                  DATA_REG ; data register for micro-bus transfers
                  ADDR_TQUAD_HI     ; MSB of address reg for TQUAD accesses
                  ADDR_TQUAD_LO     ; LSB of address reg for TQUAD 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_TADDR uses this register
            TIMEFLAG ; usage breaks down as follows:
                             ;  bit 0: set every sec. by timer int.
                             ;  bit 1: set every ~3ms by timer int.
            QUADRANT ; used to store the interrupting quadrant (0-3)
                             ;  is set at the beginning of the TQUAD int.
                             ;  handler and is used through the routine
            INT1, INT2       ;  first and second interrupt source
                             ;  from the TQUAD.
            INT_STATUS       ; interrupt status byte for a block in TQUAD
            TEMP_RFDL_DATA   ; stores data byte read from TQUAD
            TEMP_RFDL_STATUS ; stores status byte read from TQUAD
            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
        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

;**************************
;* TQUAD Register constants *
;**************************

; TQUAD initialization registers
        CONSTANT CDRC_CONFIG=0x10  ; see the TQUAD data book
        CONSTANT RDS1_CONFIG=0x03  ;  for more detailed
        CONSTANT TDS1_CONFIG=0x04  ;  information about these
        CONSTANT XBAS_CONFIG=0x44  ;  registers.
        CONSTANT FRMR_CONFIG=0x20
        CONSTANT ALMI_CONFIG=0x2C
        CONSTANT SIGX_CONFIG=0x40
        CONSTANT IBCD_CONFIG=0x3C
        CONSTANT RFDL_CONFIG=0x38
        CONSTANT XFDL_CONFIG=0x34
        CONSTANT TPSC_CONFIG=0x30
        CONSTANT IBCD_AC=0x3E
        CONSTANT IBCD_DC=0x3F

; TQUAD interrupt enable registers
        CONSTANT CDRC_IE=0x11
        CONSTANT FRMR_IE=0x21
        CONSTANT RBOC_IE=0x2A
        CONSTANT ALMI_IE=0x2D
        CONSTANT IBCD_IE=0x3D
        CONSTANT RFDL_IE=0x39

; TQUAD PMON Registers
        CONSTANT LCVL1=0x04A
        CONSTANT LCVL2=0x0CA
        CONSTANT LCVL3=0x14A
        CONSTANT LCVL4=0x1CA

        CONSTANT LCVM1=0x04B
        CONSTANT LCVM2=0x0CB
        CONSTANT LCVM3=0x14B
        CONSTANT LCVM4=0x1CB

        CONSTANT BEEL1=0x04C
        CONSTANT BEEL2=0x0CC
        CONSTANT BEEL3=0x14C
        CONSTANT BEEL4=0x1CC

        CONSTANT BEEM1=0x04D
```

```
          CONSTANT BEEM2=0x0CD
          CONSTANT BEEM3=0x14D
          CONSTANT BEEM4=0x1CD

          CONSTANT FER1=0x04E
          CONSTANT FER2=0x0CE
          CONSTANT FER3=0x14E
          CONSTANT FER4=0x1CE

          CONSTANT OOF1=0x04F
          CONSTANT OOF2=0x0CF
          CONSTANT OOF3=0x14F
          CONSTANT OOF4=0x1CF

; TQUAD interrupt status registers
          CONSTANT INT_ID=0x027      ; shows which quadrant interrupted
          CONSTANT INT1_1=0x008      ; interrupt source 1
          CONSTANT INT2_1=0x009      ; interrupt source 2
          CONSTANT ALMI_IS=0x02E     ; ALMI interrupt status
          CONSTANT CDRC_IS=0x012     ; CDRI interrupt status
          CONSTANT FRMR_IS=0x022     ; FRMR interrupt status
          CONSTANT RBOC_IS=0x02B     ; 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

; other TQUAD registers
          CONSTANT FRMR_RST=0x0D             ; framer reset
          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 TPSC_ADDR=0x32            ; TPSC channel indirect
                                             ; address/control reg.
          CONSTANT TPSC_DATA=0x33            ; TPSC channel indirect
                                             ; data reg.
          CONSTANT SIGX_ADDR=0x42            ; SIGX channel indirect
                                             ; address/control reg.
          CONSTANT SIGX_DATA=0x43            ; SIGX channel indirect
                                             ; data reg.

; SIGX internal registers
          CONSTANT SIGX_PCH_DATA=0x81        ; SIGX per channel
                                             ; signalling data (RWB set)
          CONSTANT SIGX_PCH_CONFIG=0x21      ; SIGX per channel
                                             ; configuration data
          ; SIGX bit constants
          CBLOCK 0
                    DEB, POL, FIX, INV          ; bits in per channel config
          ENDC
```

```
        ; TPSC internal registers
        CONSTANT TPSC_PCH_PCM=0x01 ; TPSC per channel
                                          ; PCM data control byte
        CONSTANT TPSC_PCH_IDLE=0x19       ; TPSC per channel
                                          ; IDLE byte
        CONSTANT TPSC_PCH_SIG=0x31 ; TPSC per channel
                                          ; SIGNALLING control byte

        ; TPSC bit constants
        CBLOCK 0
                ZCS0, ZCS1, ZCS2          ; bits in PCM control byte
        ENDC

        CBLOCK 4
                SIGNINV, DMW, IDLC, INVERT ;
        ENDC

        CBLOCK 6
                SIGC0, SIGC1              ; bits in signalling
        ENDC

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

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

        CONSTANT LCVL1_RAM=0x400
        CONSTANT LCVL2_RAM=0x410
        CONSTANT LCVL3_RAM=0x420
        CONSTANT LCVL4_RAM=0x430

        CONSTANT LCVM1_RAM=0x401
        CONSTANT LCVM2_RAM=0x411
        CONSTANT LCVM3_RAM=0x421
        CONSTANT LCVM4_RAM=0x431

        CONSTANT BEEL1_RAM=0x402
        CONSTANT BEEL2_RAM=0x412
        CONSTANT BEEL3_RAM=0x422
        CONSTANT BEEL4_RAM=0x432

        CONSTANT BEEM1_RAM=0x403
        CONSTANT BEEM2_RAM=0x413
        CONSTANT BEEM3_RAM=0x423
        CONSTANT BEEM4_RAM=0x433

        CONSTANT FER1_RAM=0x404
        CONSTANT FER2_RAM=0x414
        CONSTANT FER3_RAM=0x424
        CONSTANT FER4_RAM=0x434

        CONSTANT OOF1_RAM=0x405
```

```
CONSTANT OOF2_RAM=0x415
CONSTANT OOF3_RAM=0x425
CONSTANT OOF4_RAM=0x435

; SIGNALLING Registers
CONSTANT SIGDATA1_RX_RAM=0x436      ; signalling data for q1
CONSTANT SIGDATA2_RX_RAM=0x44E      ; signalling data for q2
CONSTANT SIGDATA3_RX_RAM=0x466      ; signalling data for q3
CONSTANT SIGDATA4_RX_RAM=0x47E      ; signalling data for q4
CONSTANT SIGDATA_TX_RAM=0x496       ; new Tx signalling value
CONSTANT SIG_QUAD=0x497             ; quadrant to affect
                                    ; with next signalling command
CONSTANT SIG_CHAN=0x498             ; DS0 channel to affect
                                    ; with next signalling command
CONSTANT IDLE_CODE_RAM=0x499        ; new idle code value

; 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 TQUAD 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=0x30
CONSTANT ONE_SEC_H=0x01

; 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 RW_DONE=0xFF      ; reg. R/W complete
 CONSTANT PMON_UPDATE=0x80  ; PMON counters updated
 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

 ; MailBox communication constants: HOST -> PIC
 CONSTANT READ_REG=0x01     ; TQUAD Reg Read Command
 CONSTANT WRITE_REG=0x02    ; TQUAD 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
 ; **NOTE** all signalling related commands must retain
 ; the following mailbox codes because of some neccesary
 ; hardcoding
```

```
         CONSTANT SIG_IDLE_ON=0x08  ; turn idle code insertion on
         CONSTANT SIG_IDLE_OFF=0x09 ; turn idle code insertion off
         CONSTANT SIG_DMW_ON=0x0A           ; turn DMW code insertion on
         CONSTANT SIG_DMW_OFF=0x0B  ; turn DMW code insertion off
         CONSTANT SIG_ON=0x0C               ; turn signal tx on
         CONSTANT SIG_OFF=0x0D              ; turn signal tx off
         CONSTANT SIG_SRC_INT=0x0E  ; internal signalling source
         CONSTANT SIG_SRC_EXT=0x0F  ; external signalling source
         CONSTANT SIG_VAL=0x10              ; change signalling value
        CONSTANT  IDLE_VAL=0x11             ; change idle code

         ; 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 TQUAD 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 TQUAD direct register
WR_TQUAD  MACRO  REGISTER, VALUE

          BANK0
          MOVLW     VALUE             ; setup data register
          MOVWF     DATA_REG
          MOVLW     LOW REGISTER      ; setup address registers
          MOVWF     ADDR_TQUAD_LO
          MOVLW     HIGH REGISTER
          MOVWF     ADDR_TQUAD_HI
          PAGE0
          CALL      WRITE_TQUAD       ; perform write

          ENDM              ; end of macro

; write to TQUAD direct register
; takes register and value
; Only changes the lower 7 address bits
; the upper 2 address bits are assumed valid
WR_TQUAD_L  MACRO  REGISTER, VALUE

          BANK0
          MOVLW     VALUE             ; setup data register
          MOVWF     DATA_REG
          MOVLW     0x80
          ANDWF     ADDR_TQUAD_LO, 1  ; setup address registers
          MOVLW     (LOW REGISTER) & 0x7F
          IORWF     ADDR_TQUAD_LO, 1
          PAGE0
          CALL      WRITE_TQUAD       ; perform write

          ENDM                        ; end of macro

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

          BANK0
          MOVLW     0x80              ; setup address registers
          ANDWF     ADDR_TQUAD_LO, 1
          MOVLW     (LOW REGISTER) & 0x7F
          IORWF     ADDR_TQUAD_LO, 1
          PAGE0
          CALL      WRITE_TQUAD       ; 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 TQUAD
; takes register address and returns contents in both W and DATA_REG
RD_TQUAD  MACRO  REGISTER

         BANK0
         MOVLW    LOW REGISTER      ; setup address registers
         MOVWF    ADDR_TQUAD_LO
         MOVLW    HIGH REGISTER
         MOVWF    ADDR_TQUAD_HI
         PAGE0
         CALL     READ_TQUAD        ; perform read

         ENDM                       ; end of macro

; read from TQUAD
; takes register address and returns contents in both W and DATA_REG
; Only changes the lower 7 address bits
; the upper two address bits are assumed valid
RD_TQUAD_L  MACRO  REGISTER

         BANK0
         MOVLW    0x80              ; setup address registers
         ANDWF    ADDR_TQUAD_LO, 1
```

```
         MOVLW    (LOW REGISTER) & 0x7F
         IORWF    ADDR_TQUAD_LO, 1
         PAGE0
         CALL     READ_TQUAD        ; 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
         CALL     INIT_RAM ; initialize the RAM
         MOVLW    0x00              ; init all 4 quadrants of TQUAD
         CALL     INIT_TQUAD
         MOVLW    0x01
         CALL     INIT_TQUAD
         MOVLW    0x02
         CALL     INIT_TQUAD
         MOVLW    0x03
         CALL     INIT_TQUAD
         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
         BTFSC    TIMEFLAG, 1       ; has ~3mS interrupt occured?
         GOTO     CALL_SIGX         ; if so, call SIGX subroutine
         CLRWDT                     ; clear watchdog timer
         GOTO     MAIN_LOOP         ; infinite loop

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

```
CALL_SIGX
         PAGE1
         CALL     SIGX              ; make cross-page call
         PAGE0
         GOTO     MAIN_LOOP


;*****************************
;* READ SUBROUTINE FOR TQUAD *
;*****************************

READ_TQUAD
         ; setup address bus
         BANK0
         MOVF     ADDR_TQUAD_LO, 0  ; transfer TQUAD address to latch
         MOVWF    PORTC
         MOVLW    0xF8
         ANDWF    PORTA, 1          ; clear A8-A10
         MOVF     ADDR_TQUAD_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 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 TQUAD *
;*****************************

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

         ; setup address bus
         BANK0
         MOVF     ADDR_TQUAD_LO, 0  ; transfer TQUAD address to latch
         MOVWF    PORTC
         MOVLW    0xF8
         ANDWF    PORTA, 1          ; clear A8-A10
         MOVF     ADDR_TQUAD_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 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 TQUAD 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
         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   7FF               ; 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 TQUAD *
;*****************************
; 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 TQUAD quadrant. A call to SETUP_TADDR 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_TQUAD
        CALL     SETUP_TADDR       ; setup address regs. for TQUAD accesses

        WR_TQUAD_L        FRMR_RST, 0x01    ; Reset framer
        WR_TQUAD_L        FRMR_RST, 0x00

        ; in the following comments, if bit is mentioned only if it is set
        WR_TQUAD_L        CDRC_CONFIG, 0x84 ; AMI, ALGSEL
        WR_TQUAD_L        XBAS_CONFIG, 0x10 ; ESF
        WR_TQUAD_L        FRMR_CONFIG, 0x30 ; ESFFA, ESF
        WR_TQUAD_L        ALMI_CONFIG, 0x10 ; ESF
        WR_TQUAD_L        IBCD_CONFIG, 0x04 ; x bit code length
        WR_TQUAD_L        IBCD_AC, 0x08       ; activate code
        WR_TQUAD_L        IBCD_DC, 0x24       ; deactivate code
        WR_TQUAD_L        SIGX_CONFIG, 0x12 ; ESF, IND
        WR_TQUAD_L        TPSC_CONFIG, 0x02 ; IND
        WR_TQUAD_L        RDS1_CONFIG, 0x04 ; RUNI
        WR_TQUAD_L        TDS1_CONFIG, 0x08 ; TUNI
        WR_TQUAD_L        RFDL_IE, 0x02       ; INTC0
        WR_TQUAD_L        RFDL_CONFIG, 0x01 ; Enable RFDL block
        WR_TQUAD_L        XFDL_CONFIG, 0x03 ; Enable XFDL block/CRC append
        WR_TQUAD_L        DL_OPT, 0x0F        ; RDLINTE, RDLEOME,
                                            ; TDLINTE, TDLUDRE

        WR_TQUAD_L        CDRC_IE, 0x40       ; Enable LOS
        WR_TQUAD_L        FRMR_IE, 0x01       ; Enable INFR
        WR_TQUAD_L        ALMI_IE, 0x07       ; Enable YEL, RED, AIS
        WR_TQUAD_L        RBOC_IE, 0x05       ; Enable IDLE, BOCE
        WR_TQUAD_L        IBCD_IE, 0x30       ; Enable LBAE, LBDE

        ; setup SIGX per channel config
        WR_TQUAD_L        SIGX_DATA, 0x07     ; Enable FIX, POL, DEB
        MOVLW             SIGX_PCH_CONFIG
        MOVWF             DATA_REG

SIGXSU_LOOP                                 ; loop to setup all DS0 channels
        WR_TQUAD_DL       SIGX_ADDR
        INCF              DATA_REG, 1
        MOVLW             SIGX_PCH_CONFIG+0x18
        XORWF             DATA_REG, 0
        BTFSS             STATUS, Z
        GOTO              SIGXSU_LOOP
```

```
         ; setup TPSC per channel config
         WR_TQUAD_L        TPSC_DATA, 0x00      ; disable everything
         MOVLW             TPSC_PCH_PCM
         MOVWF             DATA_REG

TPSCSU1_LOOP                                    ; loop to setup all DS0 channels
         WR_TQUAD_DL       TPSC_ADDR
         INCF              DATA_REG, 1
         MOVLW             TPSC_PCH_PCM+0x18
         XORWF             DATA_REG, 0
         BTFSS             STATUS, Z
         GOTO              TPSCSU1_LOOP

         WR_TQUAD_L        TPSC_DATA, 0xAA      ; set idle code = 0xAA
         MOVLW             TPSC_PCH_IDLE
         MOVWF             DATA_REG

TPSCSU2_LOOP                                    ; loop to setup all DS0 channels
         WR_TQUAD_DL       TPSC_ADDR
         INCF              DATA_REG, 1
         MOVLW             TPSC_PCH_IDLE+0x18
         XORWF             DATA_REG, 0
         BTFSS             STATUS, Z
         GOTO              TPSCSU2_LOOP

         WR_TQUAD_L        TPSC_DATA, 0xC0      ; set signaling value = 0
         MOVLW             TPSC_PCH_SIG         ; and enable signalling
         MOVWF             DATA_REG           ; from internal source

TPSCSU3_LOOP                                    ; loop to setup all DS0 channels
         WR_TQUAD_DL       TPSC_ADDR
         INCF              DATA_REG, 1
         MOVLW             TPSC_PCH_SIG+0x18
         XORWF             DATA_REG, 0
         BTFSS             STATUS, Z
         GOTO              TPSCSU3_LOOP

         WR_TQUAD_L        SIGX_CONFIG, 0x13 ; set PCCE, enabling SIGX
         WR_TQUAD_L        TPSC_CONFIG, 0x03 ; set PCCE, enabling TPSC

         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 TQUAD Interrupt
         CALL     TQUAD_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 TQUAD interrupt
         ; this check is here to minimize interrupt latency when
         ; there are multiple TQUAD interrupts pending
         BTFSS    PORTB, 0 ; skip if interrupt is not pending
TINT_LOOP
         CALL     TQUAD_INT         ; if there is still a pending TQUAD
                                    ;  int, then service it now
         BTFSS    PORTB, 0 ; loop until no TQUAD 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


; 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 TQUAD INTERRUPTS *
;**********************************

TQUAD_INT
         BCF      INTCON, INTF      ; clear interrupt flag

         ; determine interrupting quadrant and set up TQUAD address regs.
         RD_TQUAD 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_TADDR       ; setup the upper address lines
                                    ;  to address interrupting quadrant
                                    ;  of the TQUAD

         ; read in TQUAD int source registers for interrupting quadrant
         RD_TQUAD_L        INT1_1   ; read register
         MOVWF             INT1     ; save value
         RD_TQUAD_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
          RETURN                     ; end of subroutine

; Alarm Integrator block
ALMI
          RD_TQUAD_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_TQUAD_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_TQUAD_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_TQUAD_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_TQUAD_L        MSTR_DIAG         ; read in diagnostic register
        BCF      DATA_REG, 4                ; clear line loopback bit
        WR_TQUAD_DL       MSTR_DIAG         ; update register
        RETURN                     ; return from subroutine
PLA
        RD_TQUAD_L        MSTR_DIAG
        BSF      DATA_REG, 5                ; activate payload loopback
        WR_TQUAD_DL       MSTR_DIAG         ; update register
        RETURN                     ; return from subroutine
PLD
        RD_TQUAD_L        MSTR_DIAG
        BCF      DATA_REG, 5                ; deactivate payload loopback
        WR_TQUAD_DL       MSTR_DIAG         ; update register
        RETURN                     ; return from subroutine
ULD
        RD_TQUAD_L        MSTR_DIAG         ; universal loopback deactivate
```

```
        BCF     DATA_REG, 4             ; clear all loopbacks
        BCF     DATA_REG, 5
        WR_TQUAD_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_TQUAD_L      MSTR_DIAG
        BSF     DATA_REG, 4             ; activate line loopback
        WR_TQUAD_DL     MSTR_DIAG
        RETURN                  ; return from subroutine

; Inband Loopback Code Detector block
IBCD
        RD_TQUAD_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_TQUAD_L      MSTR_DIAG
        BSF     DATA_REG, 4             ; activate line loopback
        WR_TQUAD_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_TQUAD_L      MSTR_DIAG
        BCF     DATA_REG, 4             ; deactivate line loopback
        WR_TQUAD_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_TQUAD_L        XFDL_IS            ; read TQUAD 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_TQUAD_DL       XFDL_DATA          ; write it to the TQUAD
        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_TQUAD_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_TQUAD_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_TQUAD_L        XFDL_CONFIG        ; we are finished send user packet, so
        BSF      DATA_REG, 4                ; set EOM bit in XFDL
        WR_TQUAD_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_TQUAD_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_TQUAD_L        XFDL_CONFIG
         BCF      DATA_REG, 3               ; turn off XFDL interrupts
         WR_TQUAD_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_TADDR
        RD_TQUAD_L      XFDL_CONFIG
        BSF     DATA_REG, 3     ; enable XFDL interrupts
        WR_TQUAD_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_TQUAD_L      XFDL_IS         ; read TQUAD 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_TQUAD_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_TQUAD_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_TQUAD_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_TQUAD_DL       XFDL_CONFIG
          BC_QUAD_BIT       QUADRANT, XFDL_BUSY ; clear FDL busy bit
          RETURN
SEND_PENDING_USR
          RD_TQUAD_L        XFDL_CONFIG       ; we are finished, so
          BSF      DATA_REG, 4               ; set EOM bit in XFDL
          WR_TQUAD_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_TQUAD_L        RFDL_DATA        ; read data byte
        MOVWF    TEMP_RFDL_DATA            ; save the read data
        RD_TQUAD_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

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_TQUAD_L        MSTR_DIAG        ; clear all loopbacks
         BCF      DATA_REG, 4               ; clear line loopback
         BCF      DATA_REG, 5               ; clear payload loopback
         WR_TQUAD_DL       MSTR_DIAG        ; update TQUAD
         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
;        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

; This routine sets the upper 2 bits of the TQUAD address register
; according to the quadrant to be accessed
; Input:        W register contains the quadrant
; Results:      ADDR_TQUAD_HI and ADDR_TQUAD_LO are set up
; **NB** WORK5 is affected by this routine !!!
SETUP_TADDR
         MOVWF    WORK5
         CLRF     ADDR_TQUAD_LO             ; clear address registers
         CLRF     ADDR_TQUAD_HI
         BSF      ADDR_TQUAD_LO, 7  ; copy bit 0 of quadrant to
         BSF      ADDR_TQUAD_HI, 0  ; bit 7 of LSB of TQUAD address reg
         BTFSS    WORK5, 0          ; and bit 1 of quadrant to
```

```
        BCF     ADDR_TQUAD_LO, 7  ; bit 0 of MSB of TQUAD address reg
        BTFSS   WORK5, 1
        BCF     ADDR_TQUAD_HI, 0
        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 ~0x664
```

```
;********************************* 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    0x08              ; check to see if this is a signalling
         ANDLW    0xF8              ; related command
         BTFSC    STATUS, Z
         GOTO     SIG_CMDS ; jump to signalling command handling
         MOVF     SCRATCH, 0        ; restore W for next compare
         XORLW    SIG_VAL           ; check for request to change signalling value
         BTFSC    STATUS, Z
         GOTO     SIG_CMDS ; jump to signalling command handling
         MOVF    SCRATCH, 0         ; restore W for next compare
         XORLW    IDLE_VAL ; check for request to change idle code
         BTFSC    STATUS, Z
         GOTO     SIG_CMDS ; jump to signalling command handling
         MOVF    SCRATCH, 0         ; restore W for next compare
         XORLW    READ_REG        ; check for TQUAD reg read
         BTFSC    STATUS, Z
         GOTO    REG_READ
         MOVF    SCRATCH, 0         ; restore W for next compare
         XORLW   WRITE_REG       ; check for TQUAD 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
         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_TQUAD_LO
         RD_RAM   REG_ADR_HI
         MOVWF    ADDR_TQUAD_HI
         PAGE0
         CALL     READ_TQUAD
         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_TQUAD_LO
         RD_RAM   REG_ADR_HI
         MOVWF    ADDR_TQUAD_HI
         RD_RAM   REG_DATA
         MOVWF    DATA_REG
         PAGE0
         CALL     WRITE_TQUAD
         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_TADDR
         PAGE1
         RD_RAM   RAM_XBOC          ; read in BOC to transmit
         WR_TQUAD_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_TADDR
```

```
        PAGE1
        WR_TQUAD_L      XBOC_CODE, BOC_TERMINATE   ; end BOC transmission
        PAGE1
        RETURN                      ; return from subroutine

SIG_CMDS
        RD_RAM   SIG_QUAD ; get the quadrant to affect
        MOVF     DATA_REG, W
        PAGE0
        CALL     SETUP_TADDR
        RD_RAM   SIG_CHAN ; get the channel to affect
        PAGE1
        MOVLW    0x80
        IORWF    DATA_REG, 1
        MOVF     SCRATCH, 0         ; restore mailbox code for compare
        XORLW    SIG_VAL            ; check for change signalling value command
        BTFSC    STATUS, Z
        GOTO     SIG_VAL_TO
        MOVF     SCRATCH, 0         ; restore mailbox code for compare
        XORLW    IDLE_VAL ; check for change idle code command
        BTFSC    STATUS, Z
        GOTO     IDLE_VAL_TO
        MOVF     SCRATCH, 0         ; restore W for next compare
        XORLW    SIG_DMW_ON         ; check for DMW code insertion on command
        BTFSC    STATUS, Z
        GOTO     SIG_TURN_DMW_ON
        MOVF     SCRATCH, 0         ; restore W for next compare
        XORLW    SIG_DMW_OFF        ; check for DMW code insertion off command
        BTFSC    STATUS, Z
        GOTO     SIG_TURN_DMW_OFF
        MOVF     SCRATCH, 0         ; restore W for next compare
        XORLW    SIG_ON             ; check for signalling on command
        BTFSC    STATUS, Z
        GOTO     SIG_TURN_ON
        MOVF     SCRATCH, 0         ; restore W for next compare
        XORLW    SIG_OFF            ; check for signalling off command
        BTFSC    STATUS, Z
        GOTO     SIG_TURN_OFF
        MOVF     SCRATCH, 0         ; restore W for next compare
        XORLW    SIG_SRC_INT        ; check for signalling source to int. command
        BTFSC    STATUS, Z
        GOTO     SIG_SRC_TO_INT
        MOVF     SCRATCH, 0         ; restore W for next compare
        XORLW    SIG_SRC_EXT        ; check for signalling source to ext. command
        BTFSC    STATUS, Z
        GOTO     SIG_SRC_TO_EXT
        MOVF     SCRATCH, 0         ; restore W for next compare
        XORLW    SIG_IDLE_OFF       ; check for idle code insertion off command
        BTFSC    STATUS, Z
        GOTO     SIG_TURN_IDLE_OFF
        ; fall through to SIG_TURN_IDLE_ON (codes are filled)

SIG_TURN_IDLE_ON
        MOVF     DATA_REG, 0
        MOVWF    WORK
```

```
          WR_TQUAD_DL       TPSC_ADDR
          RD_TQUAD_L        TPSC_DATA
          MOVLW    0xDF             ; force DMW to 0
          ANDWF    DATA_REG, 1      ;
          MOVLW    0x40             ; force IDLC to 1 (turn idle code insertion on)
          IORWF    DATA_REG, 1      ;
          PAGE1
          GOTO     SIG_CMDS_FINISH

SIG_TURN_IDLE_OFF
          MOVF     DATA_REG, 0
          MOVWF    WORK
          WR_TQUAD_DL       TPSC_ADDR
          RD_TQUAD_L        TPSC_DATA
          MOVLW    0xBF             ; force IDLC to 0 (turn idle code insertion off)
          ANDWF    DATA_REG, 1      ;
          PAGE1
          GOTO     SIG_CMDS_FINISH

SIG_TURN_DMW_ON
          MOVF     DATA_REG, 0
          MOVWF    WORK
          WR_TQUAD_DL       TPSC_ADDR
          RD_TQUAD_L        TPSC_DATA
          MOVLW    0xBF             ; force IDLC to 0
          ANDWF    DATA_REG, 1
          MOVLW    0x20             ; force DMW to 1 (turn DMW insertion on)
          IORWF    DATA_REG, 1      ;
          PAGE1
          GOTO     SIG_CMDS_FINISH

SIG_TURN_DMW_OFF
          MOVF     DATA_REG, 0
          MOVWF    WORK
          WR_TQUAD_DL       TPSC_ADDR
          RD_TQUAD_L        TPSC_DATA
          MOVLW    0xDF             ; force DMW to 0 (turn DMW insertion off)
          ANDWF    DATA_REG, 1      ;
          PAGE1
          GOTO     SIG_CMDS_FINISH

SIG_TURN_ON
          MOVLW    0x30
          ADDWF    DATA_REG, F
          MOVF     DATA_REG, W
          MOVWF    WORK
          WR_TQUAD_DL       TPSC_ADDR
          RD_TQUAD_L        TPSC_DATA
          MOVLW    0x40             ; force SIGC1 to 1 (turn signalling on)
          IORWF    DATA_REG, F      ;
          PAGE1
          GOTO     SIG_CMDS_FINISH

SIG_TURN_OFF
          MOVLW    0x30
```

```
          ADDWF    DATA_REG, F
          MOVF     DATA_REG, W
          MOVWF    WORK
          WR_TQUAD_DL       TPSC_ADDR
          RD_TQUAD_L        TPSC_DATA
          MOVLW    0xBF             ; force SIGC1 to 0 (turn signalling off)
          ANDWF    DATA_REG, F      ;
          PAGE1
          GOTO     SIG_CMDS_FINISH

SIG_SRC_TO_INT
          MOVLW    0x30
          ADDWF    DATA_REG, F
          MOVF     DATA_REG, W
          MOVWF    WORK
          WR_TQUAD_DL       TPSC_ADDR
          RD_TQUAD_L        TPSC_DATA
          MOVLW    0x80             ; force SIGC0 to 1 (source=internal)
          IORWF    DATA_REG, F      ;
          PAGE1
          GOTO     SIG_CMDS_FINISH

SIG_SRC_TO_EXT
          MOVLW    0x30
          ADDWF    DATA_REG, F
          MOVF     DATA_REG, W
          MOVWF    WORK
          WR_TQUAD_DL       TPSC_ADDR
          RD_TQUAD_L        TPSC_DATA
          MOVLW    0x7F             ; force SIGC0 to 0 (source=BTSIG)
          ANDWF    DATA_REG, F      ;
          PAGE1
          GOTO     SIG_CMDS_FINISH

SIG_VAL_TO
          MOVLW    0x30
          ADDWF    DATA_REG, F
          MOVF     DATA_REG, W
          MOVWF    WORK
          WR_TQUAD_DL       TPSC_ADDR
          RD_TQUAD_L        TPSC_DATA
          MOVLW    0xF0             ; preserve SIGC0 and SIGC1
          ANDWF    DATA_REG, W      ; erase old signal code
          MOVWF    SCRATCH
          RD_RAM   SIGDATA_TX_RAM
          MOVLW    0x0F
          ANDWF    DATA_REG, F
          MOVF     SCRATCH, W
          IORWF    DATA_REG, F      ; insert new signal code
          PAGE1
          GOTO     SIG_CMDS_FINISH

IDLE_VAL_TO
          MOVLW    0x18
          ADDWF    DATA_REG, W
```

```
         MOVWF    WORK
         RD_RAM   IDLE_CODE_RAM

SIG_CMDS_FINISH
         WR_TQUAD_DL       TPSC_DATA
         MOVLW    0x7F
         ANDWF    WORK, W
         MOVWF    DATA_REG
         WR_TQUAD_DL       TPSC_ADDR
         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

;****************
;* SIGX ROUTINE *
;****************

SIGX
         CLRF     WORK3                     ; start with quadrant 0
         CLRF     WORK4

SIGXCOPY_LOOP
         MOVLW    SIGX_PCH_DATA             ; point to first SIGX data reg.
         MOVWF    WORK2

SIGXCOPY
         INTSON                             ; give interrupts a chance
         INTSOFF
```

```
        MOVF    WORK2, W          ; copy current channel no.
        MOVWF   DATA_REG
        MOVF    WORK3, W
        PAGE0
        CALL    SETUP_TADDR               ; setup for current quadrant
        PAGE1
        WR_TQUAD_DL       SIGX_ADDR       ; read SIGX data reg.
        RD_TQUAD_L        SIGX_DATA
        MOVLW   0x0F                      ; mask out dont care bits
        ANDWF   DATA_REG, F
        MOVLW   HIGH SIGDATA1_RX_RAM      ; set RAM address to
        MOVWF   ADDR_RAM_HI               ; current signalling register
        MOVLW   LOW SIGDATA1_RX_RAM
        ADDWF   WORK4, W
        MOVWF   ADDR_RAM_LO
        PAGE0
        CALL    WRITE_RAM                 ; copy signalling data to RAM
        PAGE1
        INCF    WORK2, 1
        INCF    WORK4, 1
        MOVLW   SIGX_PCH_DATA+0x18        ; loop until all 24 channels copied
        XORWF   WORK2, 0
        BTFSS   STATUS, Z
        GOTO    SIGXCOPY

        INCF    WORK3, 1
        MOVLW   0x04                      ; loop until all 4 quadrants done
        XORWF   WORK3, 0
        BTFSS   STATUS, Z
        GOTO    SIGXCOPY_LOOP

        INTSON                            ; turn interrupts back on
        BCF     TIMEFLAG, 1               ; clear the 3ms flag
        RETURN                            ; end of 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
G_BITS
          INTSOFF
          PAGE0
          MOVF      WORK2, W
          CALL      SETUP_TADDR
          RD_TQUAD_L          BEEM1             ; read MSB of bit error count
          MOVLW     0x01                        ; mask off unused bits
          ANDWF     DATA_REG, F
          WR_RAM_DP           BEEM1_RAM         ; copy it to RAM
          MOVF      DATA_REG, W
          INTSON
          PAGE1
          BTFSS     STATUS, Z                   ; is MSB of bit error count > 0 ?
          GOTO      BE_256PLUS
          INTSOFF
          PAGE0
          MOVF      WORK2, W
          CALL      SETUP_TADDR
          RD_TQUAD_L          BEEL1             ; read LSB of bit error count
          WR_RAM_DP           BEEL1_RAM         ; copy it to RAM
          MOVF      DATA_REG, W
          INTSON
          PAGE1
          BTFSC     STATUS, Z                   ; is LSB of bit error count = 0 ?
          GOTO      SEF_BIT                     ; yes then no need to set any G bits
          XORLW     0x01
          BTFSS     STATUS, Z                   ; is LSB of bit error 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 bit error 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 bit error 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 bit error count < 101 ?
         GOTO     BE_101PLUS
         BSF      INDF, PRM_G4             ; yes then set G4
         GOTO     SEF_BIT
BE_101PLUS
         BSF      INDF, PRM_G5             ; bit error count must be 100<x<256
                                           ; so set G5
         GOTO     SEF_BIT
BE_256PLUS
         INTSOFF
         PAGE0
         MOVF     WORK2, W
         CALL     SETUP_TADDR
         RD_TQUAD_L        BEEL1
         WR_RAM_DP         BEEL1_RAM
         MOVF     DATA_REG, W
         INTSON
         PAGE1
         SUBLW    0x3F
         BTFSS    STATUS, C                ; is LSB of bit error count < 64 ?
         GOTO     BE_320PLUS
         BSF      INDF, PRM_G5             ; yes then set G5
         GOTO     SEF_BIT
BE_320PLUS
         BSF      INDF, PRM_G6             ; bit error count must be x>319
                                           ; so set G6
SEF_BIT
         INTSOFF
         PAGE0
         MOVF     WORK2, W
         CALL     SETUP_TADDR
         RD_TQUAD_L        OOF1             ; read severly errored frame count
         MOVLW    0x07                     ; mask off unused bits
         ANDWF    DATA_REG, F
         WR_RAM_DP         OOF1_RAM ; copy it to RAM
         MOVF     DATA_REG, W
         INTSON
         PAGE1
         BTFSC    STATUS, Z                ; is severely errored frame count >= 1
                                          ?
         GOTO     FE_BIT
         INCF     FSR, F                   ; yes then
         BSF      INDF, PRM_SE             ; set SE
         DECF     FSR, F
         INTSOFF                           ; need to copy framing error count
         PAGE0
         MOVF     WORK2, W          ; anyway
         CALL     SETUP_TADDR
         RD_TQUAD_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
         PAGE1
```

```
        GOTO    LV_BIT
FE_BIT
        INTSOFF
        PAGE0
        MOVF    WORK2, W
        CALL    SETUP_TADDR
        RD_TQUAD_L      FER1
        MOVLW   0x1F                    ; mask off unused bits
        ANDWF   DATA_REG, F
        WR_RAM_DP       FER1_RAM
        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_TADDR
        RD_TQUAD_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_TADDR
        RD_TQUAD_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
        INTSOFF
        PAGE0
        MOVF    WORK2, W
        CALL    SETUP_TADDR
        RD_TQUAD_L      ELST_IS         ; read elastic store interrupt status
                                        ; to see if a slip has occured
```

```
          INTSON
          PAGE1
          BSF       INDF, PRM_SL              ; set SL if SLIPI in ELST_IS is set
          BTFSS     DATA_REG, 0
          BCF       INDF, PRM_SL
LB_BIT
          INCF      FSR, F                    ; set LB if this quadrant is in
                                              ; payload loopback
          INTSOFF
          PAGE0
          MOVF      WORK2, W
          CALL      SETUP_TADDR
          RD_TQUAD_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_TADDR
          RD_TQUAD_L        XFDL_CONFIG
          BSF       DATA_REG, 3               ; enable XFDL interrupts
          WR_TQUAD_DL       XFDL_CONFIG
          PAGE1
          INTSON
          RETURN                              ; end of GENERATE_PRM

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

PMON
          BCF       TIMEFLAG, 0               ; clear the 1 sec. flag
          ; Write to TQUAD to update all PMON counters
          INTSOFF
          WR_TQUAD   GLOBAL_PMON, 00
          INTSON
```

```
        PAGE1
        ; delay 6us to allow for update latency
        MOVLW   15              ; delay constant
        MOVWF   WORK2           ; use work2 as counter

PMON_LOOP
        DECFSZ  WORK2, 1 ; decrement and skip if 0
        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

PMC-951003 Issue date: August, 1996.

Printed in Canada