MOTOROLA
intelligence everywhere™

digital dna™

Power Line Modem Reference Design

Designer Reference Manual

56800 Hybrid Controller

DRM035/D
Rev. 0, 03/2003

MOTOROLA.COM/SEMICONDUCTORS

Power Line Modem Reference Design

Designer Reference Manual — Rev 0

by:

Zdenek Kaspar
Jaromir Chocholac
portions by Milan Brejl, PhD and Frantisek Dobes
MCSL - Motorola Czech Systems Laboratories
Roznov p. Radhostem
Czech Republic
zdenek.kaspar@motorola.com
jaromir.chocholac@motorola.com

Revision history

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://www.motorola.com/semiconductors

The following revision history table summarizes changes contained in this document. For your convenience, the page number designators have been linked to the appropriate location.

Revision history

Date	Revision Level	Description	Page Number(s)
January 2003	1	Initial Release	N/A

List of Sections

List of Sections

Table of Contents

Table of Contents

Appendix A. References

Appendix B. Bill of Materials and Schematics

Appendix C. Source Code Files

Appendix D. Glossary

Table of Contents

List of Figures

List of Figures

List of Tables

List of Tables

Section 1. Introduction

1.1 Contents

1.2 Application intended functionality

Power Line Modem (PLM) is a device designed to communicate through the power line (mains). This PLM implementation is using the frequency band “B” of the CENELEC EN 50065-1 regulation (frequency band 95 to 125 kHz). Device is based on the Motorola DSP56F801 Hawk 1 family and is capable of performing using European 230 V as well as US 110 V voltage. FSK modulation technique is used for communication.

1.3 Benefits of our solution

- Both FSK modulation / demodulation routines are fully handled by the DSP s/w.
- Low cost low speed solution with baudrate 10 kbps.
- Communication according to the CELENEC EN 50065-1 “Signaling on low-voltage electrical installations in the frequency range 3 kHz to 148.5 kHz” regulation.
- Transmitted data encrypted by Tiny Encryption Algorithm.
- Data consistency is secured by FEC (Forward Error Correction), 16 bit CRC (Cyclic Redundancy Check) and interleaving technique.

Introduction

Section 2. Quick Start

2.1 Contents

2.2 Introduction

In this reference design a complete description of software and hardware of the Power Line Modem (PLM) based on the DSP56F801 is given. The PLM board is a hardware platform of the Power Line Modem reference design.

The PLM is a device designed to communicate through the power line (mains), DSP56F801 is a member of Motorola's Hawk V1 family of 16-bit Digital Signal Processors (DSP).

The result of this reference design is a protocol independent media access interface for the connection of different devices coupled through a power line. The functionality of the PLM design is demonstrated by the provided application demo stored in the internal FLASH memory of the DSP. Beyond that, the PLM board enables the implementation and testing of the user software. For this purpose the board is equipped with a JTAG/OnCE interface for flash reprogramming and debugging.

In Section 2, a brief introduction to the project is given, together with a description of the connection and startup of the Power Line Modem demo application. Section 3 details the PLM board as the hardware part

of an implementation of the PLM design. A full-scale description of the PLM software is presented in section 4. In section 5, a bill of materials and schematics of the PLM board is given, and finally, in section 6, the complete source code of the PLM can be found.

2.3 Main PLM features

This chapter describes some of the most important features and parameters of the designed solution.

The Power Line Modem presented in this reference design operates in the band B of the CENELEC EN 50065-1 regulation (see 3. CENELEC EN 50065-1: “Signaling on low-voltage electrical installations in the frequency range 3 kHz to 148.5 kHz”, 1991). It operates in half-duplex mode using a Frequency Shift Keying (FSK) modulation with a communication speed of 10 kbps.

For more information regarding this topic, see **4.4 FSK communication parameters** and **4.5.6 Communication parameters**.

2.4 PLM demonstration

In this section, the connection and startup of the Power Line Modem (PLM) board demo application is described.

PLM serves as a *transparent channel*. This means that data coming in from the SCI (Serial Communication Interface) module are received, formatted to a packet (or frame), processed and then sent to the mains (power line). For return communication, the process is analogue.

This means that the only thing needed for the PLM demonstration is the controlled dataflow of the serial data. The easiest way is to use two HyperTerminal programs since this is a standard part of the MS Windows.

The settings for HyperTerminal for this kind of demonstration can be found in **2.4.1 HyperTerminal settings**.

On the other hand there are a lot of other possibilities which can be used. One of the most exciting is a system like emWare's Embedded Micro Interworking Technology, known as EMIT® software. It is a complete communication and device / information management solution for connecting numerous embedded devices to the Internet. For more information, see **http://www.emware.com**.

2.4.1 HyperTerminal settings

There are several parameters to be set in HyperTerminal:

- proper serial port (COM1, COM2, COM3...)
- communication speed (bit rate) of the PLM demonstration is 38400 bps
- 8 data-bits per character
- none parity
- 1 stop bit
- no flow control

2.4.2 Connecting the PLM boards to the PC

The board supply-current can be delivered by the AC/DC convertor mounted on the PLM board or by an external 12V AC/DC convertor.

Perform the following steps to connect the PLM board cables:

1. Connect the serial extension cable to the selected serial port of the host computer or end device.

2. Connect the other end of the serial extension cable to J2 on the PLM board. This provides the connection which allows the host computer / end device to communicate with the PLM board.

3. Connect the power supply plug to a 230V (120V) AC power source. The red Power-On LED will illuminate when the power is correctly applied.

4. Follow steps 1 to 3 for the second PLM board.

NOTE: *It is necessary that both PLM's are connected to the same phase of the mains.*

2.4.3 Demo configuration

There are several possible Power Line Modem connections. The most typical one is shown in Figure 2-1. where the *End device* sits on one side of the communication channel, on the other side the *Client control terminal* or *host computer* (for example personal computer) is located.

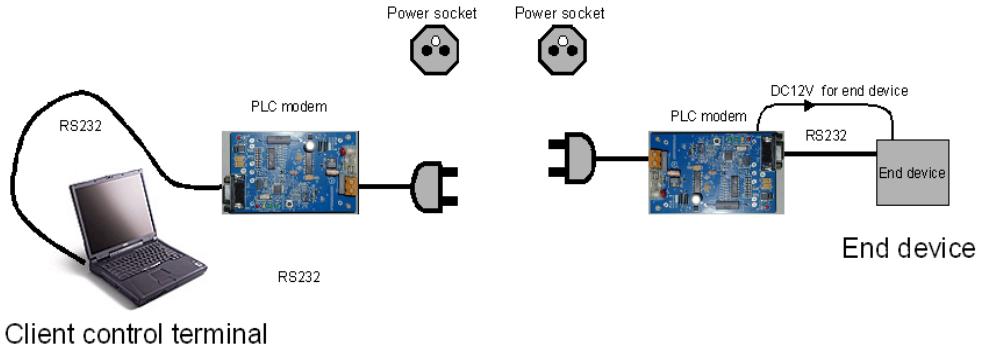


Figure 2-1. Scheme of PLM connections

For demonstration purposes personal computers with HyperTerminal programs running are used on both sides of the communication channel; the first one as an end device while the second one acts as a control terminal. For this configuration, either one PC with two serial COM ports or two PCs have to be used.

Section 3. Hardware Description

3.1 Contents

3.2 Introduction

This reference design of the Power Line Modem (PLM) provides a modem able to transmit data through a power line (mains) with a transmission speed up to 10 kbps. The PLM is based on a DSP56F801, a 16-bit Digital Signal Processor (DSP). The result of the reference design is a protocol independent media access interface for the

connection of different devices. For example, connection between appliances or connection appliances to a PC or a similar host.

The PLM board is the hardware platform for the power line modem reference design. The board supports the provided demo application, which is stored in the integrated FLASH memory of the DSP56F801.

Beyond that, the PLM board enables the implementation and testing of the user software. For that purpose, the board is equipped with a JTAG/OnCE interface for reprogramming and debugging.

3.3 Technical Data

This subsection provides technical data for both the DSP56F801 processor and the PLM board.

3.3.1 DSP56F801 Processor

The main component of the PLM board is the DSP56F801, a Motorola 16-bit DSP. Features of the DSP56F801 include:

DSP Core Features

- 16-bit DSP56800 family DSP engine with dual Harvard architecture
- As many as 40 MIPS at 80 MHz core frequency
- Single-cycle 16 × 16-bit parallel Multiplier-Accumulator (MAC)
- Two 36-bit accumulators including extension bits
- 16-bit bidirectional barrel shifter
- Parallel instruction set with unique DSP addressing modes
- Hardware DO and REP loops
- Three internal address buses
- Four internal data buses
- Instruction set supports both DSP and controller functions

- Controller style addressing modes and instructions for compact code
- Efficient C Compiler and local variable support
- JTAG/OnCE Debug Programming Interface

DSP Memory Features

- Harvard architecture permits as many as three simultaneous accesses to program and data memory
- On-chip memory including a low cost, high volume flash solution
  - 8K × 16-bit words of Program Flash
  - 1K × 16-bit words of Program RAM
  - 1K × 16-bit words of Data RAM
  - 1K × 16-bit words of Data Flash
  - 2K × 16-bit words of BootFLASH

DSP Peripheral Circuit Features

- 12-bit Analog to Digital Convertors (ADCs) which support two simultaneous conversions with two 4-pin multiplexed inputs
- General Purpose Quad Timer
- Serial Communication Interface (SCI0)
- Pulse Width Modulator module (PWMA) with 6 PWM outputs
- Serial Peripheral Interface (SPI) with configurable four-pin port
- Computer Operating Properly (COP) Watchdog timer
- Two dedicated external interrupt pins
- Eleven multiplexed General Purpose I/O (GPIO) pins
- External reset pin for hardware reset
- JTAG/OnCE for unobtrusive, processor speed-independent debugging
- Software-programmable, Phase Lock Loop-based frequency synthesizer for the DSP core clock

- Oscillation flexibility between external crystal oscillator or on-chip relaxation oscillator for lower system cost and two additional GPIO lines

3.3.2 PLM Board

Features of the PLM board include:

- DSP56F801FA80 DSP packaged in a 48-pin Plastic Quad Flat Pack (LQFP)
- Five Light-Emitting diodes (LED)
  - Power ON
  - Tx_enable
  - Data_out
  - Data_in
  - CD_out
- JTAG/OnCE interface for in-system programming and debugging
- RS232 interface for connection to PC or a similar host
- Push button for IRQA (User defined function)
- Application dedicated DSP pins accessible via a 20-pin header connector

The PLM board is shown in **Figure 3-1.**

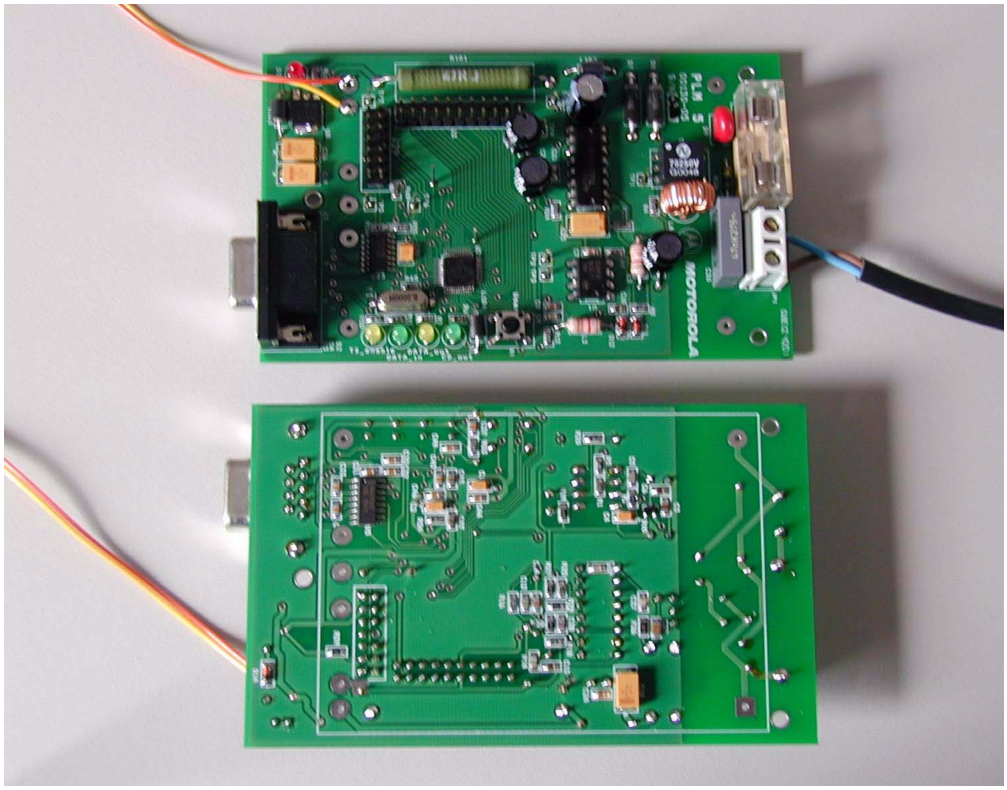


Figure 3-1. PLM Board

3.3.3 PLM Functionality

The PLM is dedicated for use in the low cost Home Interconnectivity market.

The transceiver meets the regulations for AC mains signalling of CENELEC (European Committee for Electrotechnical Standardization), FCC (Federal Communication Commission) and Industry Canada (formerly DOC).

Under FCC Section 15.107 “Limits for carrier current systems,” as well as Industry Canada guidelines, communication frequencies are allocated as shown in Figure 3-2. To protect aircraft radio navigation systems that operate between 190kHz and 525kHz, restrictions on power line communication above 185kHz have to be considered.

In conformity with CENELEC EN 50065-1 “Signalling on low-voltage electrical installations in the frequency range of 3kHz to 148.5kHz” Part 1 “General requirements, frequency bands and electromagnetic disturbances”, the communication frequencies are allocated as shown in Figure 3-2.

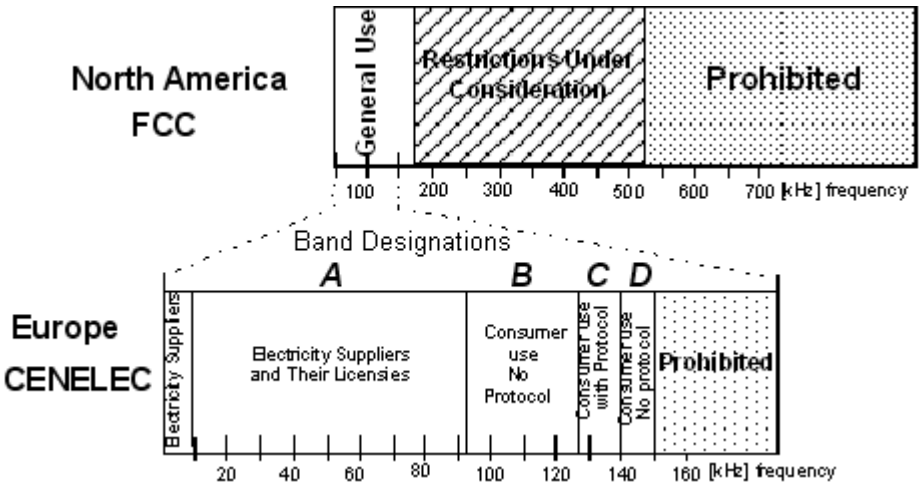


Figure 3-2. Regulatory Considerations

3.4 Power Line Modem Architecture

Schematics of the PLM board are provided in **Appendix B. Bill of Materials and Schematics**. The Power Line Modem block diagram can be seen in Figure B-1.

The PLM is a flexible system, designed to demonstrate the communication capability through the power line.

The electrical circuitry can be logically divided into following basic blocks:

- Power Stage&Coupling module
- Output Filter
- Input Stage
- Microcontroller

- RS232 interface
- Power module

3.4.1 Power Stage&Coupling module

3.4.1.1 Coupling with the Power Line

The coupling network is the interface between the power line and the low voltage transmitter output and receiver input pins of the modem. For low cost applications, when the insulation with the mains is not required, a double LC network can be used. For home applications, where insulation is mandatory, then an HF transformer should be used. Apart from the insulation with the power line, the transformer has also to perform the appropriate filtering for both the transmission and the reception. The Newport's 78250 converter transformer can be used for this application. The basic coupling network can be seen in Figure 3-3.

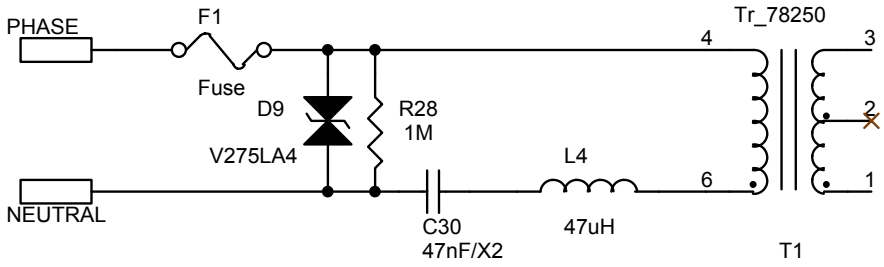


Figure 3-3. The Coupling Network

To provide an efficient transmission coupling, a 1:1 winding ratio is used. An extra LC serial filter is needed to provide rejection of unexpected harmonics in order to comply with standards. In fact, the behavior of the 1:1 winding is mainly a high pass filter, and does not provide efficient filtering of high frequency harmonics.

In reception mode, the 1:1 winding ratio, fitted with the tuning capacitor, provides a high pass filter with an efficient rejection of the 50 Hz signal. For instance, the 50 Hz amplitude is 230 V rms or 167 dBuV, and the

maximum sensitivity of the modem is 80 dBuV. To take advantage of the detection performance, the filter must reject the 50 Hz for more than 80 dB.

For the frequency band (95 kHz to 125 kHz) which is quite wide, the quality factor (Q) of the coupling filter needs to be low. Otherwise an unacceptably large attenuation at the band edges would result, that would avoid good coupling performances, sensitive to a wide range of loads. For a band-pass filter of this configuration, the quality factor is proportional to the reciprocal of the coupling capacitance. For low Q, the value of C30 needs to be large. On the other hand, the capacitance should not be too large in order to limit significantly 50 Hz mains current passing through the transformer:

The coupling capacitor C30 is used to couple the PLM with the power line and it must be a X2 type, rated for mains voltage.

The transformer possesses leakage inductance that can be tuned with the coupling capacitor to form a band-pass filter. Because the leakage inductance of the transformer 78250 is small (2 uH), some external inductance should be added to create a band-pass filter. Resistor R28 serves to discharge C30 when the device is disconnected from the power line. Varistor D9 provides protection against high voltage transients on the power line.

3.4.1.2 Modem output voltage

The maximum output voltage of a power line modem is defined by the CENELEC norm EN50065-1 and should be 116 dBuV maximum in the frequency range 95 kHz to 148.5 kHz. A measurement of the carrier amplitude on a standard CISPR16 load with a 50 Ohms spectrum analyser should be done. The CISPR16 network provides an attenuation of 6 dB, due to its structure. The maximum rms voltage measured on the analyser must then be max 122 dBuV that equals 3.56 V peak to peak. The test set-up for output voltage measurement can be seen in Figure 3-4.

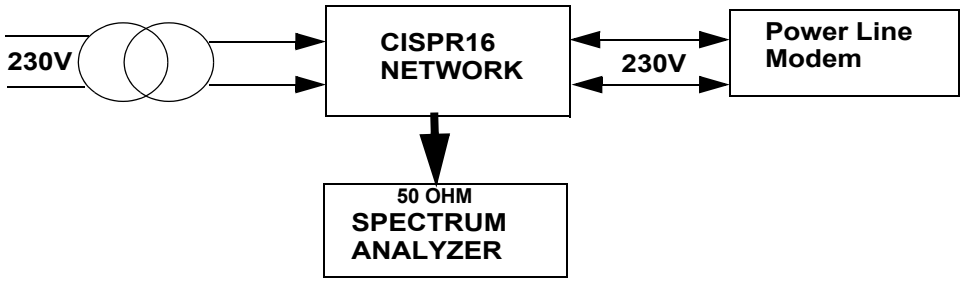


Figure 3-4. Test set-up for output voltage measurement

3.4.1.3 Power stage

The impedance of the mains network at the signalling frequencies is relatively low and varies in a wide range (1 Ω to 100 Ω). This circuit has been designed to drive a 4 Ω mains line over the 95 kHz to 125 kHz bandwidth. The signalling impedance of the mains network fluctuates as different loads are switched on or off during the day.

When transmitting, the transmitter appears as a low-impedance signal source on the mains network. If the transmitter was left in the active mode whether or not transmitting, this load would reduce the mains impedance and a signal arriving from a distant transmitter would be severely attenuated. To overcome this problem, the transmitter needs to present a high impedance to the mains network when it is not transmitting.

The TLE2301 amplifier has a 1-A output drive capability with short-circuit protection. Hence, it carries out the requirements. The TLE2301 incorporates an output 3-state facility and in addition, it has a low standby current in the 3-state mode.

The Frequency Shift Keying (FSK) modulated output signal is created by the general DSP output in form of a square wave signal. To meet CENELEC regulation, some filtering has to be done to convert the signal to a sine wave. From the harmonics point of view, only odd harmonics are contained in the square wave signal. Any frequency components above transmission band must be eliminated by a low-pass filter. The attenuation of the third harmonic must be more than -56 dB to meet

CENELEC regulation. Concerning this fact we need a low-pass filter with an attenuation slope of -120 dB/dec. The chosen solution is to use a two stage passive LC low-pass filter (-80 dB/dec) and output amplifier as an active second order low-pass filter (-40 dB/dec), in cascade.

The schematic of the output stage can be seen in Figure B-2. The output passive LC low-pass filter is described in section 3.4.2.

The output amplifier U4 and external components (capacitors C22, C28 and resistors R29, R30) create a Butterworth second order low-pass filter with cut-off frequency 110 kHz. Capacitor C28 provides a positive feedback path.

The operation can be described qualitatively:

- At low frequencies, where C22 and C28 appear as open circuits, the signal is simply buffered to the output.
- At high frequencies, where C22 and C28 appear as short circuits, the signal is shunted to ground at the amplifier's input. When f>>fc signals are attenuated by -40 dB/dec.
- Near the cut-off frequency, where the impedance of C22 and C28 is on the same order as R29 and R30, positive feedback through C28 provides Q enhancement of the signal.

The measured frequency response of the Output amplifier can be seen in Figure 3-5.

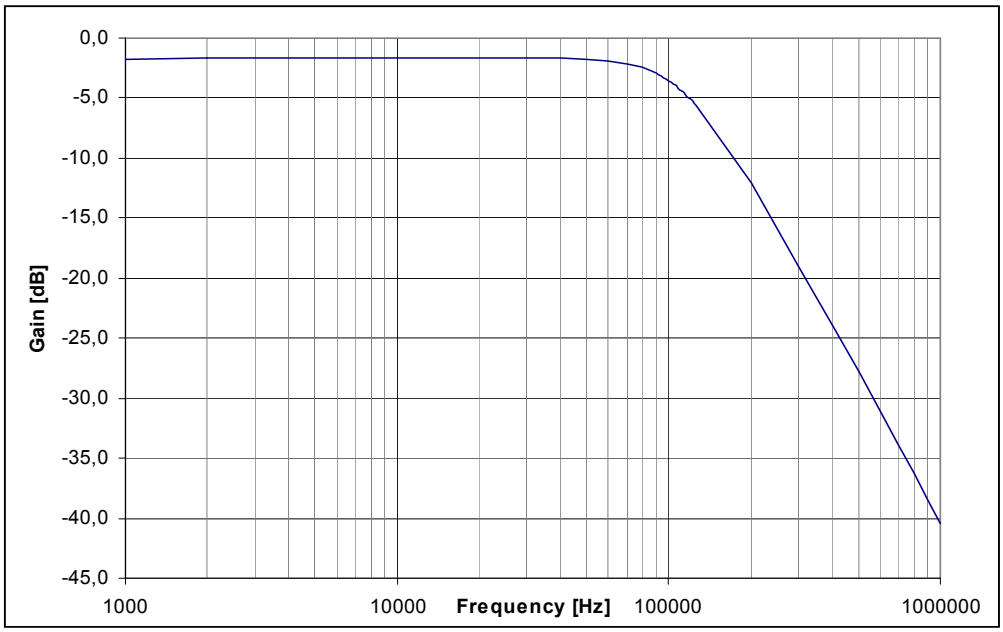


Figure 3-5. Frequency Response of the Output Amplifier

3.4.1.4 Transient and Overvoltage Protections

The Power stage of the modem has to be protected against many risks of damage, mainly due to the direct connection to the mains. Some protection against a transient overstress during power-up and an overvoltage on the power line is done. The fast recovery diodes (D7, D8) are used to clamp the surge voltage of the secondary windings and to avoid any stress and reverse voltage at the output pin of the operational amplifier. See Figure B-2.

3.4.2 Output Filter

The output filter is a simple two stage LC low-pass filter with cut off frequency of 110 kHz. The schematic of the output stage can be seen in Figure B-3. The filter is created by inductors L5, L6 and capacitors C13, C14. The measured frequency response of the Output Filter can be seen in Figure 3-6. and measured frequency response of the entire Output Stage can be seen in Figure 3-7.

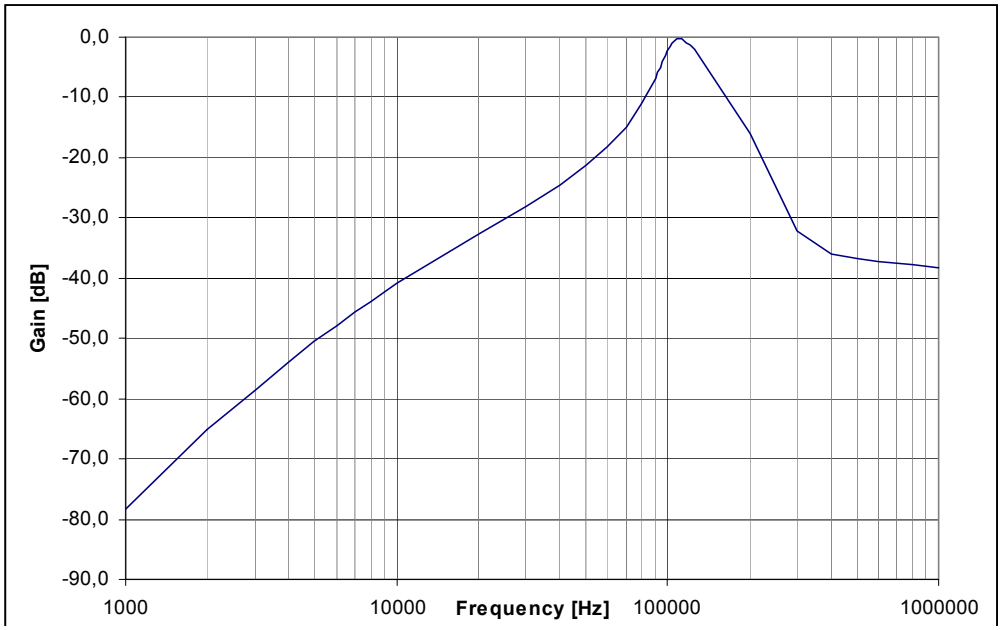


Figure 3-6. Frequency Response of the Output Filter

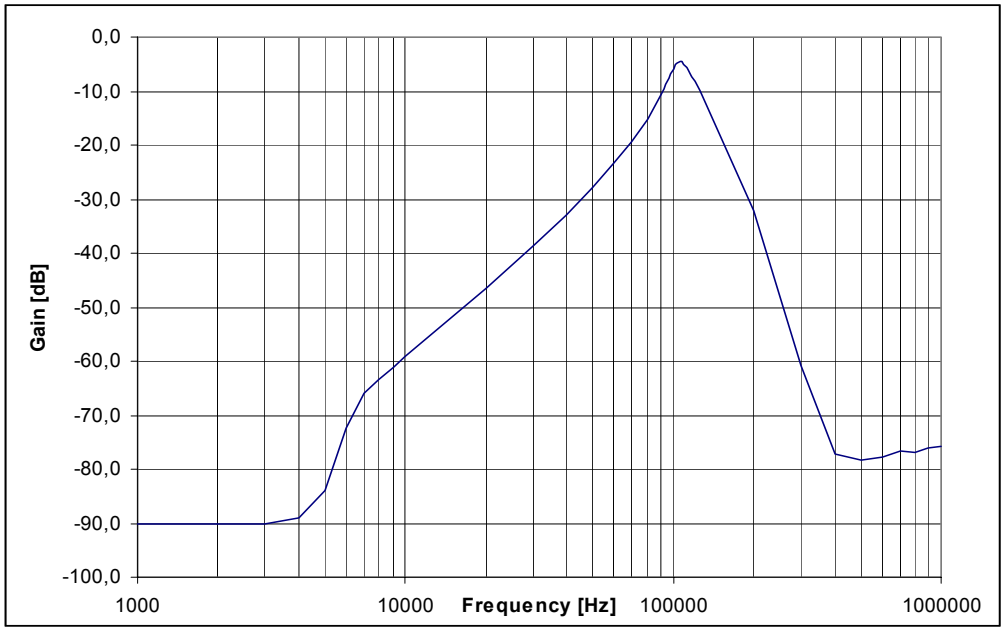


Figure 3-7. Frequency Response of the Output Stage

3.4.3 Input Stage

3.4.3.1 Input Filter

As was described in section 3.4.1.1, the transformer (T1) fitted with the tuning capacitor (C30), see Figure B-2., provides a high pass filter with an efficient rejection of the 50 Hz signal. Since the input signal is read by A/D converter, aliasing can be a problem when the input signal contains frequency components above half the A/D sampling rate. These higher frequencies can “fold over” into the lower frequency spectrum and appear as erroneous signals that cannot be distinguished from valid sampled data. By limiting the input signal bandwidth we can avoid this problem. A low-pass input filter is used to eliminate unwanted high-frequency noise and interference introduced prior sampling.

Figure B-4. shows the schematic for an Input Stage. The inductors (L1, L2) and capacitors (C3, C6) create the input high-pass filter with a cut-off frequency of 110 kHz.

3.4.3.2 Transient and Overvoltage Protections

The dual diode D10 serves to clamp the voltage level applied to the input of the input amplifier to the power supply range of the device.

3.4.3.3 Input Amplifier&Limiter

The schematic diagram of the input amplifier and limiter can be seen in Figure B-4. The LF351 high speed JFET input operational amplifier (U1) is used to amplify the input signal. To achieve a high input impedance, the non inverting configuration of the amplifier is used. The open loop voltage gain of the LF351 at a frequency 100 kHz is less than 40 dB (100). Note that the gain of the closed-loop should be small compared to the open-loop gain to get the accurate output driven by external components. The closed-loop gain is set up to 100, then the gain in the band is limited by the open-loop gain. The input amplifier is followed by a diode limiter to keep the amplitude of the signal in the range suitable for the A/D converter input. Next, a low-pass filter created by inductor L3 and capacitor C8 with cut-off frequency of 110 kHz is used to eliminate high-frequency components from the signal prior to sampling. The

measured frequency response of the Input Amplifier can be seen in Figure 3-8.

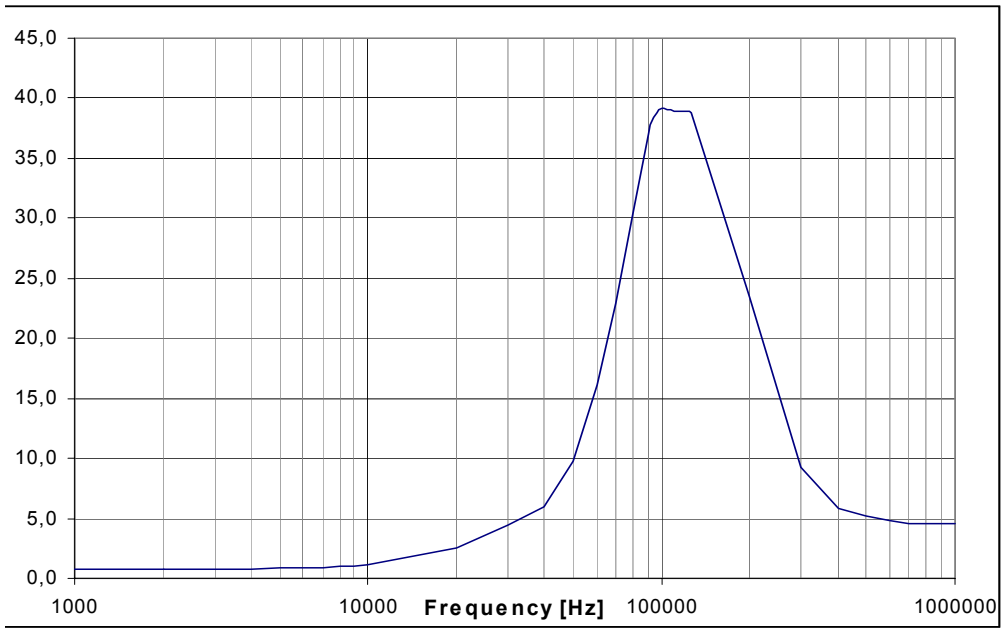


Figure 3-8. Frequency Response of the Input Amplifier

3.4.4 Microcontroller

Motorola 16-bit DSP56F801 (U20) is the main component of the PLM board. The schematic diagram can be seen in Figure B-5. The output FSK modulated signal is provided from the Timer D Channel 2 pin and the input signal is read by A/D converter channel 0. Free pins of the DSP are connected to the Extension Connector (J3) for use by an user designed application.

The External Interrupt Request A (IRQA) input is dedicated for any user specified purpose. It can be programmed to be level-sensitive or negative-edge-triggered. The push button (S1) is connected to the IRQA pin and it is bridged with capacitor (C37), to avoid noise. Pushing the button is an input event that results in the generation of an interrupt by the DSP. This interrupt can then be used by the program.

For communication status optical signalling, four LEDs (D1, D2, D3, D4) are attached to port B.

The JTAG/OnCE interface signals are connected to a JTAG Connector (J29) for reprogramming and debugging purpose.

3.4.4.1 Extended Signals

Table 3-1. Extended Signals

Signal Name	Signal Type	State During Reset	Signal Description
A0-A5	Output	Output	**PWMA0–**5—Six PWMA output pins. Setting an output control enable bit enables software to drive the PWM outputs instead of the PWM generator. 1)
TD0	Input/ Output	Input	**Timer D Channel 0**—TD0 can alternately be used as GPIOA0. After reset, the default state is the quad timer input.
TD1	Input/ Output	Input	**Timer D Channel 1**—TD1 can alternately be used as GPIOA1. After reset, the default state is the quad timer input.
FLT	Input	Input	**FAULTA**0—This Fault input pin is used for disabling selected PWMA outputs in cases where fault conditions originate off-chip.
AN1-AN3	Input	Input	**Analog inputs** to ADCA, channel 1
AN4-AN7	Input	Input	**Analog inputs** to ADCA, channel 2

1) More details in chapter 11 of *“DSP56F801/803/805/807 16-Bit Digital Signal Processor User’s Manual”*.

3.4.4.2 In-circuit JTAG/OnCE Port

A standard JTAG pin header connector is present on the PLM board to provide access from a host computer to the JTAG/OnCE port signals on the DSP device. **Table 3-2.** shows the required signals. It is

recommended to use a standard command converter to interface to the JTAG signals and the CodeWarrior tool to download the program.

Table 3-2. JTAG/OnCE Signals

Signal	Signal Description
TDI	**Test Data Input**--This input provides a serial data stream to the JTAG and the OnCE module. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.
TDO	**Test Data Output**--This tri-stateable output provides a serial data stream from the JTAG and the OnCE module. It is driven in the Shift-IR and Shift-DR controller states of the JTAG state machine and changes on the falling edge of TCK.
TCK	**Test Clock Input**--This input proves a gated clock to synchronize the test logic and shift serial data through the JTAG/OnCE port. The maximum frequency for TCK is 1/8 the maximum frequency of the DSP56F801. The TCK pin has an on-chip pull-down resistor.
TMS	**Test Mode Select Input**--This input sequences the TAP controller's state machine. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.
TRST	**Test Reset**--This input provides a reset signal to the TAP controller. This pin has an on-chip pull-up resistor.

3.4.5 RS232 Interface

The PLM board provides an RS-232 interface for connection to PC or a similar host. Refer to the RS-232 schematic diagram in Figure B-6. The RS-232 level converter (U5) transitions the SCI +3.3 V signal levels to RS-232 compatible signal levels and connects to the host's serial port via connector J2. Flow control is not provided.

3.4.6 Power Supply

A schematic of the power supply is shown in Figure B-7. Power can be supplied to the PLM board by using an external 12 Vac/dc convertor or via the AC/DC convertor mounted on the PLM board. The power supply provides 12 VDC for analog circuits and 3.3 VDC for the microcontroller and the RS232 interface. LED D5 indicates the power on state.

3.5 Board Layout

A detailed layout plans of the PLM board with the names of all components are shown in Figure 3-9. (component side) and Figure 3-10. (solder side).

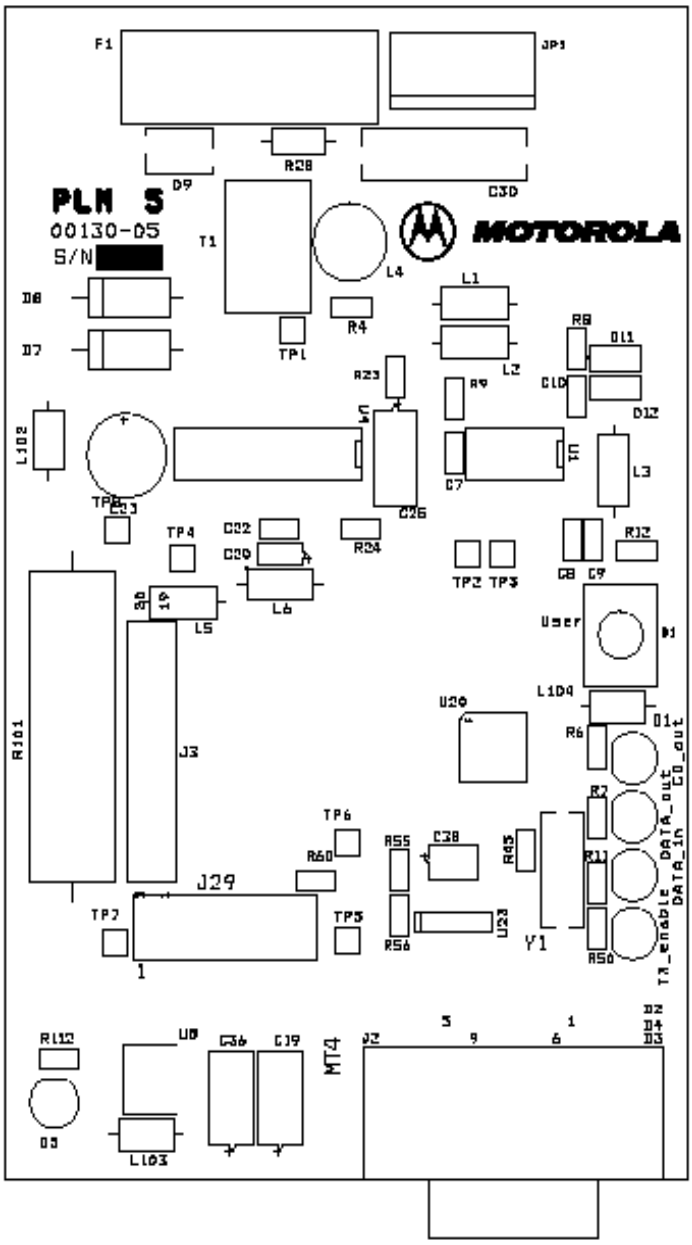


Figure 3-9. PLM Component Side Layout

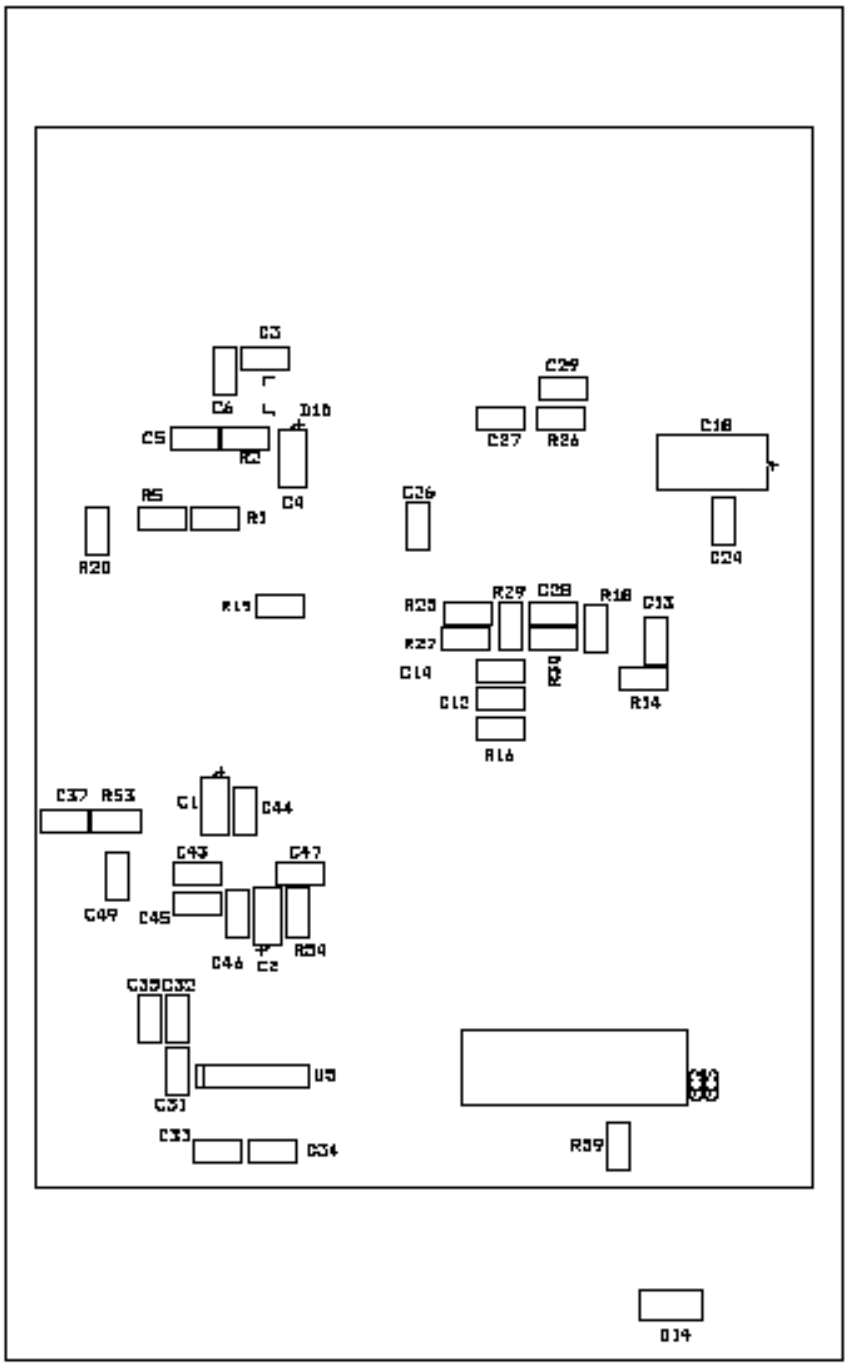


Figure 3-10. PLM Solder Side Layout

3.6 Connectors

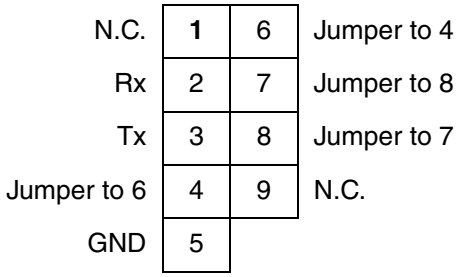
3.6.1 Expansion Connector - J3

VCC	**1**	**2**	MGND
TD1	**3**	**4**	TD0
A5	**5**	**6**	A4
A3	**7**	**8**	A2
A1	**9**	**10**	A0
VA	**11**	**12**	GND
AN7	**13**	**14**	AN6
AN5	**15**	**16**	AN4
AN3	**17**	**18**	AN2
AN1	**19**	**20**	FLT

3.6.2 JTAG/OnCE Connector - J29

TDI	**1**	**2**	GND
TDO	**3**	**4**	GND
TCK	**5**	**6**	GND
N.C.	**7**	**8**	KEY
/RESET	**9**	**10**	TMS
+3.3V	**11**	**12**	N.C.
N.C.	**13**	**14**	/J_TRST

3.6.3 RS232 Interface Connector - J2



3.7 Memory Map

The DSP56F801 has a dual Harward memory architecture, with separate program and data memory spaces.

Table 3-3. DSP56F801 Program Memory Map

From	To	Size	Content
0x0000	0x0003	4 bytes	On-Chip Boot Flash
0x0004	0x1FFF	8k - 4	On-Chip Program Flash
0x2000	0x7BFF	22k	Reserved
0x7C00	0x7FFF	1k	Program RAM
0x8000	0x87FF	2k	Boot Flash
0x8800	0xFFFF	30k	Reserved

Table 3-4. DSP56F801 Data Memory Map

From	To	Size	Content
0x0000	0x03FF	1k	On-Chip Dual Port Data RAM
0x0400	0x0BFF	2k	Reserved
0x0C00	0x0FFF	1k	On_Chip Peripheral Registers
0x1000	0x17FF	2k	On-Chip Flash
0x1800	0x1FFF	2k	Reserved

Table 3-4. DSP56F801 Data Memory Map

0x2000	0xFFF7F	56k-128	Not supported external memory access
0xFF80	0xFFFF	128bytes	On-Chip Core Configuration Registers

For a detailed description of the DSP56F801 memory map, refer to the *DSP586801/803/805/807 User's Manual, Motorola document order number DSP56F801-7UM/D - Rev. 3.0*.

Section 4. Software Module Descriptions

4.1 Contents

4.2 Introduction

This section of the reference design provides complete documentation of the Power Line Modem (PLM) software.

As described before, PLM is a device designed to communicate through the power line (mains). This implementation of PLM operates in band B of the CENELEC EN 50065-1 regulation in half-duplex mode using Frequency Shift Keying (FSK) modulation and a communication speed equal to 10 kbps. The PLM board is based on the Motorola 16-bit Digital Signal Processor DSP56F801 which is a member of Motorola Hawk V1 family.

4.2.1 Software basics

All embedded software of this project was written using CodeWarrior version 4.0.2 by Metrowerks Corporation (**http://www.metrowerks.com**).

Low level drivers for direct peripheral access were used through the development. For more information, see **4.5.2 List of the project files**.

Although this software is dedicated to the PLM board based on the DSP56F801, code is written in the way that it is fully applicable for the other member of the Motorola DSP family - the DSP56F803 device. The only necessary change is to modify the appconfig.h and linker_flash.cmd files, and add the linker_ram.cmd if an external RAM target is required. Note that the last mentioned file is not present in the PLM DSP56F801 project since this core does not allow external memory addressing.

4.2.2 Application basics

The communication itself is performed in a very straightforward way. Serial data coming into the PLM board through the SCI (Serial Communication Interface) module are received, then they are formatted

to the packet (frame), processed, and then sent out to the mains (power line). For the opposite direction of communication the process is equivalent - the packet is received through the mains, the data are checked and if there is no inconsistency error they are sent through the SCI to an appliance or a control terminal.

This approach is called the *transparent channel* or *transparent mode* of the frame (packet) oriented protocol and it is shown in Figure 4-1.

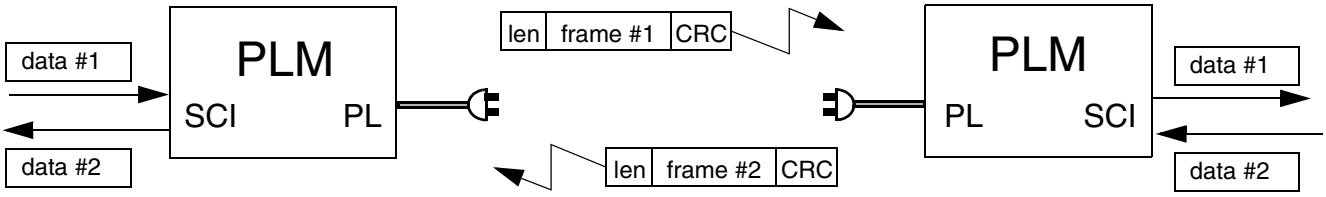


Figure 4-1. Block diagram of the communication system

4.2.3 Over the data operations basics

It is necessary to process the transmitted data packet before it can be sent out. The following operations have to take place:

- *Cyclic Redundancy Code (CRC)* computation is used to generate the *CRC field* which contains information that is added to any transmitted frame. The CRC field is used to verify the integrity of every transmitted frame since this information is checked and compared to a recalculated CRC field on the recipient's side.
- *Encryption* technique ensures the security of the transmitted data. This PLM board software utilizes the *Tiny Encryption Algorithm (TEA)* by David Wheeler and Roger Needham. TEA is a Feistel cipher with *XOR* and *and* addition as the non-linear mixing functions.
- *Forward Error Correction (FEC)* uses added redundancy information in order to correct errors which occurred during the transmission. Since the Power Line Modem operates in a very harsh and noisy environment, it is necessary to use some kind of

error detection/correction technique. This PLM implementation uses a quite straightforward method of error detection/correction called the *Linear Block Codes* with added redundancy characterized by the expression (7, 4).

- *Interleaving* is another technique which assures better consistency of the transmitted data when combined with FEC. It simply modifies the sequence of bits of the frame to be transmitted in a defined way.

NOTE: *Encryption/Decryption and Interleaving routines do not modify the length of the final frame (packet). On the other hand, the CRC and FEC routines add redundancy and therefore modify the length of the packet.*

4.2.3.1 Processing order of operations on the PL transmission side

1. CRC computation
2. TEA encryption
3. FEC coding
4. Interleaving

4.2.3.2 Processing order of operations on the PL reception side

1. De-interleaving
2. FEC decoding
3. TEA decryption
4. CRC check

4.2.4 Packet format basics

Figure 4-2. shows the format of the transferred packet (frame) before it undergoes any of the operations described above except the CRC calculation (last 2 characters of the packet).

Although the *Cntrl* value usually carries just the frame length information, it can be easily modified when necessary. For example, an extra application or protocol flags could be added to it.

n - length of the data part of the frame
N - total length of the frame, N = n + 3

Cnrtl	*Data* part (1..n) in B	*CRClow*	*CRChigh*

Figure 4-2. Format of the packet

Since the TEA encryption algorithm is implemented in the Power Line Modem, there is a restriction of the total length *N* of the frame. The restriction is the following: the total length *N* must be a multiple of 8 and therefore the length of the data part *n* is equal to *n = (N - 3).* For more information see **4.5.6 Communication parameters** and **4.6.5 Encryption / Decryption**.

NOTE: *An extra part called Header is transmitted before each packet , it is not shown in Figure 4-2. This extra part of the packet allows the bit synchronization of the FSK demodulation, see **4.3.2.3 Synchronization and windowing** for more details.*

4.3 Theory

In this section of the reference design document the theory behind the Power Line Modem implementation is given and explained. The first part provides the FSK Modulation principles, in the second part the FSK Demodulation algorithm is fully explained.

4.3.1 FSK modulation

A subset of 2-state *Frequency Shift Keying (FSK)* called *Minimal Shift Keying (MSK)* was chosen for the Power Line Modem implementation. A typical feature of this kind of modulation is the fact that a bit period T_b of the FSK modulated signal is equal to a multiple of the halves of two periods T_0 and T_1 standing for two discrete frequencies f_0 and f_1 representing logic states 0 and 1.

$$T_b = n \cdot \frac{T_0}{2} \rightarrow f_0 = n \cdot \frac{f_b}{2} \quad \text{and} \quad T_b = (n+1) \cdot \frac{T_1}{2} \rightarrow f_1 = (n+1) \cdot \frac{f_b}{2} \qquad \textbf{(EQ 4-1.)}$$

Solving these two equations the frequency separation Δf is defined as:

$$\Delta f = \frac{f_1 - f_0}{2} = \frac{f_b}{4} \quad \textbf{(EQ 4-2.)}$$

and then using a carrier frequency f_c the signaling frequencies f_0 and f_1 can be written in the following form:

$$f_0 = f_c - \frac{f_b}{4} \quad \text{and} \quad f_1 = f_c + \frac{f_b}{4} \quad \textbf{(EQ 4-3.)}$$

For the approximate bandwidth calculation of such a signal (both FSK and MSK modulation) we can write this equation:

$$B_{2FSK} \cong 2 \cdot \left(\frac{f_b}{2} + \Delta f\right) \quad \textbf{(EQ 4-4.)}$$

4.3.2 FSK demodulation

4.3.2.1 Introduction

The approach of the software FSK demodulation algorithm has the following advantages:

- number of hardware components is reduced to a minimum since they are replaced by software
- frequency of a signal element can be determined by mathematical computation (using *DTFT - Discrete Time Fourier Transformation*) that is an ultimate solution for such a noisy and harsh environment like a power line
- the output of the algorithm is the transferred message - not only a binary signal

4.3.2.2 Main idea of algorithm

The DTFT computes a continual frequency function of a given discrete-time signal. Here, the DTFT is used to compute the values F_0 and F_1 of the frequency function at 2 discrete points only - at frequencies

f_0 and f_1. Where f_0 is the frequency of a signal element corresponding to bit 0, and f_1 is the frequency of a signal element corresponding to bit 1.

$$F_0 = \sum_{n=0}^{N-1} s(n) \cdot e^{-j\omega_0 n} \qquad F_1 = \sum_{n=0}^{N-1} s(n) \cdot e^{-j\omega_1 n} \quad \textbf{(EQ 4-5.)}$$

where:

$$\omega_0 = 2\pi \cdot \frac{f_0}{f_s} \quad \text{and} \quad \omega_1 = 2\pi \cdot \frac{f_1}{f_s} \quad \textbf{(EQ 4-6.)}$$

and *s(n)* is the signal element sample, f_s is the sampling frequency.

By the comparison between F_0 and F_1 values it is decided if the signal element transfers a bit with a logical “0” or “1” value. Let’s establish a binary vector *MSG* as the received message. Then

$$MSG(j) = F_1 > F_0 \quad \textbf{(EQ 4-7.)}$$

where *j* is index of an actual bit element.

Further tasks are required to establish synchronization to the signal element within the coming FSK signal and to suppress noise influence.

4.3.2.3 Synchronization and windowing

In correspondence with the rule of digital signal minimal frequency differentiation the signal element length *T* is chosen

$$T = \frac{1}{|f_1 - f_0|} = \frac{1}{2\Delta f} \quad \textbf{(EQ 4-8.)}$$

to obtain the maximum bit rate. Then the number of samples is

$$N = f_s \cdot T \quad \textbf{(EQ 4-9.)}$$

This requirement modifies the equation for the frequency separation *Δf, as* defined in **4.3.1 FSK modulation**, into the form:

$$\Delta f = \frac{f_b}{2} \quad \textbf{(EQ 4-10.)}$$

An incoming signal is windowed by a rectangular window of length *N*. The rectangular window shape and the window length *N* are necessary to accomplish maximum frequency differentiation.

Let's establish an index *i* for indexing each signal window and corresponding variables.

The computation of $F_0(i)$ and $F_1(i)$ and the consequential comparison is done for each signal window:

$$b(i) = F_1(i) > F_0(i) \quad \textbf{(EQ 4-11.)}$$

The approximate beginning of the data burst is set from the signal window where the instantaneous value $S_B(i)$ of a short-term sliding average of the F_0 and F_1 sum crosses the doubled value $S_A(i)$ of a long-term sliding average of the F_0 and F_1 sum.

$$S_B(i) > 2 \cdot S_A(i) \quad \textbf{(EQ 4-12.)}$$

The sliding averages $S_A(i)$ and $S_B(i)$ are computed in each step as follows:

$$S_B(i) = \lambda_B S_B(i-1) + (1-\lambda_B) \cdot [F_0(i) + F_1(i)] \quad \textbf{(EQ 4-13.)}$$

if $S_B(i) < 2S_A(i)$ then

$$S_A(i) = \lambda_A S_A(i-1) + (1-\lambda_A) \cdot [F_0(i) + F_1(i)] \quad \textbf{(EQ 4-14.)}$$

otherwise the S_A long-term sliding average value is not updated:

$$S_A(i) = S_A(i-1) \quad \textbf{(EQ 4-15.)}$$

λ_A and λ_B are *Forgetting factors* which are less than but close to 1. $\lambda_A > \lambda_B$ makes the S_A value a long-term sliding average and S_B a short-term sliding average.

To achieve a synchronization of the signal windows and the signal elements a synchronization byte called a *Header* is transmitted in the pre-control (initial) part of each data burst (packet). The synchronization

byte is formed by a bit sequence *[1 0 1 0 0 1 0 1]*. The transmitter and receiver clocks are supposed to be precise enough to keep the synchronization during the whole data burst.

The best fit of the synchronization sequence is computed as the position where the divergence between the sequence of *b(i)* coming from the $F_0(i)$ and $F_1(i)$ comparison and interpolated synchronization bit sequence (interpolated Header) is minimal:

$$idx = \text{index of } min(SYN) \quad \textbf{(EQ 4-16.)}$$

where

$$SYN(i) = \sum (b \text{ XOR } [111000111000000111000111]) \quad \textbf{(EQ 4-17.)}$$

The incoming signal is windowed with 33% overlap. For this overlap the synchronization bit sequence has to be interpolated by a 3:1 ratio.

Due to this overlap, each signal element (received bit) stored in the *MSG* output buffer is calculated from 3 values of the comparison results *b(i)* (so called *subbits*). When 2 or 3 of the subbit values *b(i)* belonging to one particular bit indicate a logical “1”, that bit equal to one is added to the *MSG* output sequence buffer. Otherwise (2 or 3 subbits indicate logical “0”) bit 0 is added into the *MSG* buffer.

4.3.2.4 State model of the PLM FSK demodulation

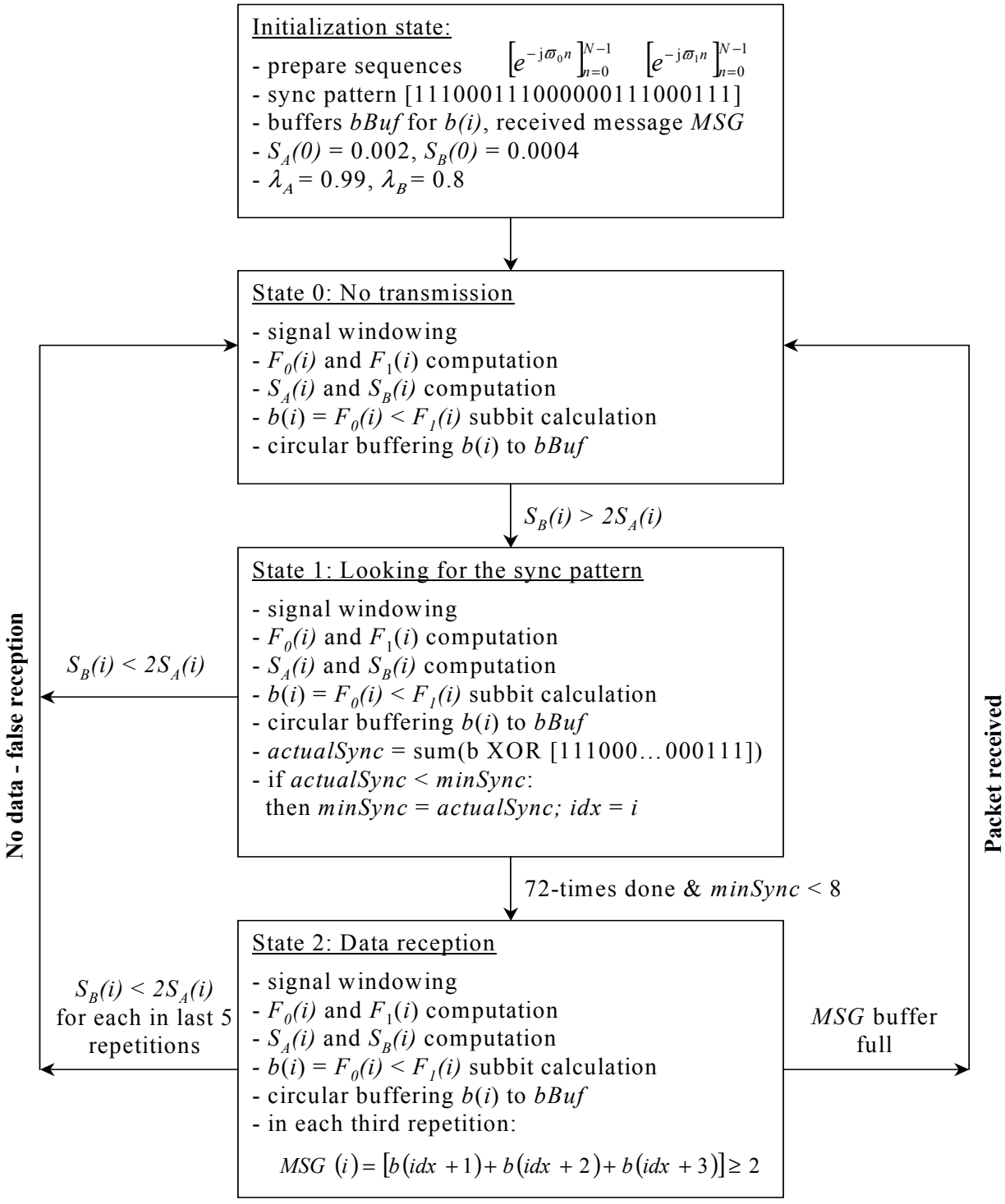


Figure 4-3. State model of the FSK demodulation

NOTE: *Values of parameters given in the previous figure were set during the testing phase of the Power Line Modem development. These tests were carried out in the noisy office environment of our lab. However some values should be changed by the user if necessary.*
Very important is a value of the multiplier in the $S_B(i) > 2 \cdot S_A(i)$ *sliding average comparison. Depending on the ratio of signal to noise its value may be changed if necessary. The source code allows you to set a value such* as *1, 2, 4... very easily using the symbolic constant* `DEMFSK_SAMULTIPLE` *which, in the source code, is used in the way* `2^(DEMFSK_SAMULTIPLE)`.

4.4 FSK communication parameters

Some of the most important parameters and the respective values of the implemented FSK algorithm are shown here, calculated and commented if necessary:

- communication is held in the CENELEC B band (95 kHz - 125 kHz)
- bit period T_b is equal to 100 µs; bit rate is therefore f_b = 10.000 bps
- f_0 and f_1 signaling frequencies can be set to these values: 100 kHz, 105 kHz, 110 kHz, 115 kHz and 120 kHz
- but possible f_0 and f_1 signaling frequency combinations are only the following (since the $\Delta f = \frac{f_b}{2}$ condition validity and because of the bandwidth):
  a) 100 kHz and 110 kHz with centre frequency f_c = 105 kHz
  b) 105 kHz and 115 kHz with centre frequency f_c = 110 kHz
  c) 110 kHz and 120 kHz with centre frequency f_c = 115 kHz
- lower frequency called f_1 signals the binary “1” value
- rough bandwidth calculation B_{2FSK} is equal to 20 kHz which is an appropriate value for all three chosen centre frequencies f_c in the CENELEC B band
- ADC (Analog to Digital Converter) sampling frequency f_s is set to 500 kHz, sampling period T_s is therefore 2 µs

- therefore, the length *N* of the rectangular window is equal to 50 samples

4.5 PLM project introduction

This section gives the introductory information and a description of the software part of the Power Line Modem project.

4.5.1 Coding convention

All source codes were written using several rules and guidelines which make the final product more readable, reusable and portable.

Here is the list of the most important ones:

- File prefix is used in each identifier that is used globally; it gives a very quick cross-reference mechanism from identifier to definition and implementation
  - variables are named in the form `fileprefix_NameOfVar`
  - for functions the form is `fileprefixNameOfFunc`
  - for a symbolic constant the form is: `FILEPREFIXCONST` (all written in upper case)
- Special prefix characters are used to further identify attributes associated with the type being specified
  - `"s"` for struct type
  - `"u"` for union type
  - `"p"` for pointer variable

All advantages mentioned above are also ensured by using the Low level drivers architecture dependent routines and on-chip peripheral drivers.

The general form of the Low level drivers command is the following:
`ioctl(peripheral_module_identifier,command,command_specific_parameter);`

This approach makes the final source code even more readable and also shortens development time.

4.5.2 List of the project files

Here is a list of all source code files of the Metrowerks CodeWarrior project:

- pl.c contains the periphery initialization, global variables declaration and the main PLM routine
- pl.h is theheader file of the main PLM routine; it contains whole set of PLM-related symbolic constants as well as the structure definition
- tmrfsk.c consists of two main timer-based routines (bit rate generation for the FSK transmission and the timeout indication of the SCI reception, both done as interrupt service routines)
- tmrfsk.h is a header file which includes all timer periphery related project macros, the GPIO related symbolic defines and function style macros are located here as well
- demfsk.c and demfsk.h contain the whole routines of the FSK demodulation (demodulation initialization, FSK demodulation itself, ADC End Of Scan interrupt service routine and a couple of other support routines); the header file includes demodulation related symbolic constants and also function style macros for ADC management using the *Timer C2* as the *ADC A TriggerTmr*
- scicomm.c and scicomm.h include all SCI periphery basis routines; Interrupt service routines for both transmission (ISR Transmitter Empty) and reception (ISR Receiver Full and ISR Receiver Error)
- coderoutines.c and coderoutines.h incorporate all data coding and decoding routines, such as FEC coding and decoding, CRC computation and the de-interleaving algorithm
- tea.c and tea.h hold the implementation of all Tiny Encryption Algorithm routines (for both encryption and decryption)
- demfskconst.c is a look-up table used for FSK demodulation

- FECtable.c is a look-up table for Forward Error Correction algorithm implementation
- CRCtable.c is a look-up table for CRC computation
- appconfig.h is the header file of the static periphery configuration made by the Low level drivers suite
- linker_flash.cmd is the linker command file of the Metrowerks CodeWarrior project

The complete source code routines of the Low level drivers is stored in `\src` subdirectory of the project.

4.5.3 Used DSP peripherals

This section briefly describes all used DSP peripheral components used in the project.

A list and short description of the *Quad Timer* modules used is given in the following table.

Table 4-1. Quad Timers

QTimer	Symbolic name	Purpose	ISR function
C2	TriggerTmr	trigger for ADC	-
D1	BitTmr	bit rate generation timer for the FSK modulation	`tmrfskBitISR`
D2	CarrierTmr	carrier generation timer for the FSK modulation	-
D3	TimeOutTmr	SCI reception timeout tmr	`tmrfskTimeOutISR`

NOTE: *The dedicated input/output pin TD2 (GPIOA2) of the QTimer D2 is used for the carrier frequency generation (set as an output).*

Usage of the *Analog to Digital Converter (ADC)* is given in a next table.

Table 4-2. ADC A

Sample Time Slot	Input analog pin	Purpose	ISR function
Sample 0	AN 0	data collection for FSK demodulation	`demfskEndOfScanISR`

NOTE: *Data sampling (PL reception) is controlled (started and stopped) via the TriggerTmr C2.*

A list and description of the *GPIO* pins used is given in the following table.

Table 4-3. GPIO

GPIO	Direction	Symbolic name	Purpose	control / signal
GPIOB4	output	`TXENABLE`	enable / disable the transmit amplifier	control
GPIOB5	output	`TXD`	transmitted data signalization	signal
GPIOB6	output	`RXD`	received data signalization	signal
GPIOB7	output	`CD`	carrier detection signalization	signal

NOTE: *The control signal influences the behavior of the Power Line Modem, the signaling ones are used just for the LED indications.*
For the DSP56F803 project the table would be exactly the same with the only exception that there is the GPIOD port used instead of the GPIOB which is not available on the DSP56F801 core.

For a description of the SCI periphery module, see Table 4-4.

4.5.4 Used interrupts

All interrupts of the Power Line Modem peripherals which are used are briefly detailed in this section:

Table 4-4. Interrupts

Symbolic name / periphery module	ISR function	Type of the interrupt	Priority of the INTR	INTR enabled after start?
BitTmr / D1	`tmrfskBitISR`	output compare	2	yes
TimeOutTmr / D3	`tmrfskTimeOutISR`	output compare	1	yes
ADC A	`demfskEndOfScanISR`	end of scan interrupt	5	yes
SCI0 - transmission	`scicommTxEmptISR`	transmitter empty	1	no
SCI0 - reception	`scicommRxFullISR`	receiver full	1	no
SCI0 - reception	`scicommRxErrISR`	receiver error	2	no

4.5.5 Main variables of the project

In this section an enumeration of the most important variables is given together with brief descriptions.

The four main communication buffers (variables are declared in pl.c, data structure in pl.h) are:

- `pl_uRxFromSCI pl_RxFromSCI` is a buffer dedicated to SCI reception operations
- `pl_uTxToSCI pl_TxToSCI` serves for the SCI transmission operations
- `pl_uRxFromPL pl_RxFromPL` stores the final data frames received from the Power Line
- `pl_uRxFromSCI pl_TxToPL` is a buffer dedicated to Power Line transmission

Data types of all buffer variables are defined as unions of two structure types - a dedicated structure and a simple array. This approach makes the operations over the buffers very flexible.

```
typedef union  // complete union of the SCI reception
```

```
{
    pl_sStructRxFromSCI Struct; // frame AS STRUCTURE
    pl_sArrayRxFromSCI Array;   // frame AS ARRAY
} pl_uRxFromSCI;
```

NOTE: *Note that the* `pl_uRxFromSCI` *data type is used for* `pl_RxFromSCI` *as well as for* `pl_TxToPL` *variables.*

- `Word16 xBuf[XBUFLENGTH]` (declared in demfsk.c) is a circular buffer of samples as they are read from the AN0 pin of the ADC A module during its `ADCEndOfScanISR` routine.
- `Word16 demfsk_NewFrmCounter` (declared in demfsk.c) is a counter for the `ADCEndOfScanISR` routine, it is decremented each time the function is performed
- `UWord32 demfsk_MSGBuf[DEMFSK_MSGBUFLEN]` (declared in demfsk.c) is another buffer aimed at FSK demodulation and therefore PL reception. Rough binary data (before any manipulation is done with them) are stored there as a result of the FSK demodulation routine.
- `pl_sFlags pl_Flags` is a following structure which contains the state and another ”error” flag of the PL modem device (taken from pl.h):

```
typedef struct
{
    UWord16 ModeOfModem : 4;    /* Mode of the modem */
/* Here are the possible states of pl_FlgModeOfModem variable */
/*   State:         Description of PL Modem Mode: */
/*  STATE0    No operation, no communication of modem */
/*  STATE1    SCI reception could be started, RxFromSCI buffer
              is ready */
/*  STATE2    SCI reception in progress */
/*  STATE3    SCI reception has been finished */
/*  STATE4    PL transmission could be started, TxToPL buffer
              is ready */
/*  STATE5    PL transmission in progress */
/*  STATE6    PL / SCI transmission has been finished */
/*  STATE7    PL reception has been started */
/*  STATE8    PL reception in progress, FSK demodulation in */
/*            Demstate 0 (waiting until F0 or F1 is present) */
/*  STATE9    PL reception in progress, FSK demodulation in */
/*            Demstate 1 (finding synchronization pattern) */
/*  STATE10   PL reception in progress, FSK demodulation in */
/*            Demstate 2 (data reception) */
/*  STATE11   PL reception in progress, FSK demodulation in */
```

```
/*            Demstate 3 (data reception finished) */
/*  STATE12   SCI transmission could be started, TxToSCI buff
              is ready */
/*  STATE13   SCI transmission in progress */

   UWord16 DataError : 1; /* Data Error occured in Rx PL frame */
                          /* bad CRC code or bad data length */
                           /* 0 - no error */
                           /* 1 - error occured */
} pl_sFlags;
```

NOTE: *If desirable, extra application flags can be added by user very easily into this bit array structure.*

- `const tea_uKey pl_TeaKey = {1, 2, 3, 4, 5, 6, 7, 8}` is an encryption key for the TEA (Tiny Encryption Algorithm) computation

4.5.6 Communication parameters

As mentioned in **4.2.4 Packet format basics**, there is a length limitation of the data part of the packet due to the TEA algorithm usage. Moreover, the FSK demodulation routine requires that the length of the frame to be received is known. In order to choose the proper length of the packet from the application point of view (**Table 4-5. Length of the communication packets**), the following symbolic constant should be set correctly in pl.h:

```
#define PL_FRAMETYPE    LONG            /* choose:  SHORT
                                                    MEDIUM
                                                    LONG */
// if SHORT is used, length of the data part of packet is 13w
// if MEDIUM is used, length of the data part of packet is 21w
// if LONG is used, length of the data part of packet is 29w
// Note when FEC is OFF, just lower 8 bits of the word are used
//                   ON, lower 14bits of the word carry the data
```

Table 4-5. Length of the communication packets

PL_FRAME TYPE	Length of cntrl part [w]	Length of data part [w]	Length of CRC part [w]	Total length [w]
SHORT	1	13	2	16
MEDIUM	1	21	2	24

Table 4-5. Length of the communication packets

PL_FRAME TYPE	Length of cntrl part [w]	Length of data part [w]	Length of CRC part [w]	Total length [w]
LONG	1	29	2	32

NOTE: *This table is valid for both* PL_FECTYPE *possible values (see next chapter). When it is equal to* PL_NOFEC*, just the lower 8 bits are stored in each word of the buffer, otherwise 14-bit long values are used (6 bits of redundant information added by FEC).*

There are other symbolic constants (defines) to be set according to the application requirements in pl.c file:

- the following setting allows the user to control the FEC technique used during the communication (switched FEC on or off):
```
#define PL_FECTYPE  PL_1STFEC
/* choose PL_NOFEC or PL_1STFEC */
```

For more information about FEC see **4.6.4 FEC calculation**.

- in order to perform TEA encryption over the buffers this line should be placed in pl.h file:
```
#define PL_TEACRYPT 1
/* if defined perform TEA encryption */
```
- as mentioned in **4.5.6 Communication parameters** chapter, there are several f_0 and f_1 possible signaling frequency combinations; the following conditional macro definition placed in pl.h allows the user to choose one of them:

```
/* Choose the carrier frequencies */
#if 0
#define PL_CARRIERLOW   CARRIERLOW_110KHZ10KBPS
#define PL_CARRIERHGH   CARRIERHGH_100KHZ10KBPS
#endif

#if 1
#define PL_CARRIERLOW   CARRIERLOW_115KHZ10KBPS
#define PL_CARRIERHGH   CARRIERHGH_105KHZ10KBPS
#endif

#if 0
#define PL_CARRIERLOW   CARRIERLOW_120KHZ10KBPS
#define PL_CARRIERHGH   CARRIERHGH_110KHZ10KBPS
```

```
#endif
```

4.5.7 Linker command file modifications

Several linker command file modifications were done in the original Low level drivers stationary template during development, see linker_flash.cmd for the whole file listing:

- look-up table variables (taken from CRCtable.c, FECtable.c and demfskconst.c files) are placed in the `xflash` (data flash) memory area

```
.main_Application_constants :
{
    _consts_start= .;

    #place your constants here: const.c (.data)
    FECtable.c (.data)   # place constants into the XFlash area
    CRCtable.c (.data)
    demfskconst.c (.data)
    _consts_size= . - _consts_start;
    F_Xdata_start_in_ROM = .;
} > .xflash
```

- 3 variables of the demfsk.c (`demfsk_NewFrmCounter, pxBuf` and `prevSample`) are stored in the very beginning of the internal data memory area called `avail`. They are used in the ADCA *End Of Scan* interrupt service routine `demfskEndOfScanISR()`. The reason why they are placed there is the cycle time reduction.

```
.internal_memory_30:
{
    OBJECT (FprevSample, demfsk.c)
    OBJECT (FpxBuf, demfsk.c)
    OBJECT (Fdemfsk_NewFrmCounter, demfsk.c)
} > .avail
```

- two circular buffers of the FSK demodulation routine have to be aligned properly in the data memory area:

```
. = ALIGN(0x80);     # these definitions must be above * (.bss)
OBJECT (FxBuf, demfsk.c)
. = ALIGN(0x80);
OBJECT (FbBuf, demfsk.c)
```

- size of the .`data` and .`stack` memory areas were modified

4.5.8 Memory usage

The following table shows the PLM software memory allocation:

Table 4-6. Memory usage

Type of memory	Total size (w)	Used memory (w)	Free memory (%)
program flash (P.FLASH)	2000_h	$9F1_h$	approx. 69%
data (X.RAM) for stack	100_h	-	-
data (X.RAM)	$2C0_h$	$1E1_h$	approx. 32%
data flash (X.FLASH)	800_h	384_h	approx. 56%

4.6 PLM Implementation

In this section the complete descriptions of the key software modules of the PLM are given.

4.6.1 Modulation

The theoretical introduction of this topic is presented in **4.3.1 FSK modulation**. All FSK modulation coding can be found in tmrfsk.c and tmrfsk.h files.

As mentioned in **3.4.1.3 Power stage**, the FSK modulation output of the DSP is in a square wave form and it became a harmonic signal after the HW filtration.

The *CarrierTmr* Quad Timer D2 (see 4.5.3 Used DSP peripherals) is dedicated to carrier generation. It is configured to “Count repeatedly” (*Count Once ONCE* bit is cleared) and “Count until compare, then re-initialize” (*Count Length LENGTH* bit is set). The *Output Mode* is set to “Toggle OFLAG output on successful compare” while the “OFLAG output bit is enabled” by setting the *Output Enable (OEN)* bit.

These settings prepare the timer for autonomous FSK generation, its frequency is given by the value *CarrierTmr (*Qtimer D2) *Compare Register #1*.

The *BitTmr* Quad Timer D1 (see 4.5.3 Used DSP peripherals) is aimed for the bit rate generation of the PL transmission. It simply means that during its Output compare ISR the proper value of the *CarrierTmr's Compare Register #1* is loaded according to the current bit value. Two function style macros are used for this purpose (taken from tmrfsk.c):

```
tmrfskSetCarrierHigh(); // set logical "1" carrier
tmrfskSetCarrierLow();  // set logical "0" carrier
```

The used modulation technique is shown in Figure 4-4.

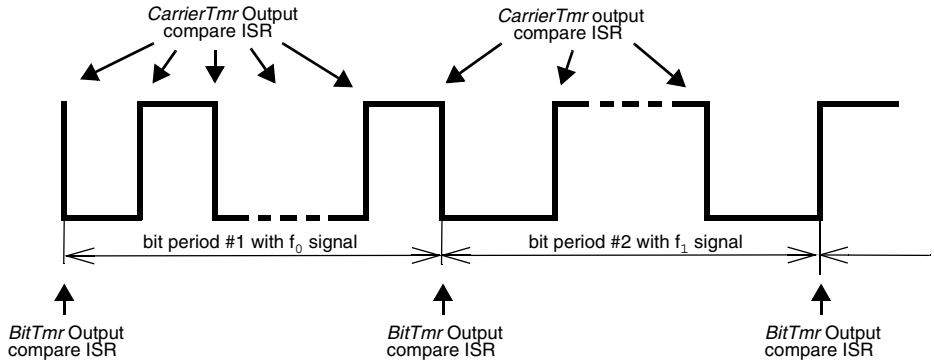


Figure 4-4. FSK generation principle

Here is the main part of the `tmrfskBitISR` routine of the *BitTmr* timer (tmrfsk.c):

```
if (mask & pl_TxToPL.Array.Byte[index]) // current bit is "1"
{
    if (ioctl(QTIMER_D2, QT_READ_COMPARE_REG1, NULL) ==
        PL_CARRIERLOW); // if previous value was logical "0"
    {
        while (ioctl(QTIMER_D2, QT_READ_COUNTER_REG, NULL) >=
                        PL_CARRIERHGH - TMRFSK_SAFETYRESERVE);
        tmrfskSetCarrierHigh(); // set logical "1" carrier
        tmrfskTxDLEDOn();       // Set transmit LED indication
    }
}
else                                    // current bit is "0"
{
    if (ioctl(QTIMER_D2, QT_READ_COMPARE_REG1, NULL) ==
        PL_CARRIERHGH); // if previous value was logical "1"
    {
        while (ioctl(QTIMER_D2, QT_READ_COUNTER_REG, NULL) >=
```

```
                                PL_CARRIERHGH - TMRFSK_SAFETYRESERVE);
            tmrfskSetCarrierLow();  // set logical "0" carrier
            tmrfskTxDLEDOff();  // Clear transmit LED indication
        }
    }
```

NOTE: *A new value of the Compare Register #1 of the CarrierTmr is not loaded immediately when the counter is too close to the compare value. For this purpose, two while() conditions are in the code. A slight delay (less than or equal to 0.75µs) is added to the bit period (100µs for the 10kbps) which does not have any negative influence on the result.*

NOTE: *A value of the Compare Register #1 is not modified when the current bit value is equal to the prior one.*

4.6.2 Demodulation

The theory behind the FSK demodulation is given in **4.3.2 FSK demodulation** section. All FSK demodulation routine can be found in demfsk.c and demfsk.h files.

4.6.2.1 Data sampling

The ADCA module is used (see **4.5.3 Used DSP peripherals** for more details) for data sampling during the PL reception phase. It is triggered by the *TriggerTmr*, therefore the ADC module samples each 2 µs with the *End of scan interrupt enabled (EOSIE)* bit set.

The ADCA `demfskEndOfScanISR` routine stores the analog values of its AN0 pin into a dedicated `xBuf` circular buffer and also decrements the `demfsk_NewFrmCounter` counter variable (see 4.5.5 Main variables of the project).

NOTE: *There are two versions of the* `demfskEndOfScanISR` *function made by conditional compilation. The first one as described in the previous paragraph, is used in the project and therefore primary recommended. The second one adds a highpass filter in the form y(n) = x(n)-x(n-1) to the functionality of the former one. Although it is probably not so suitable for such a well-filtered input signal as is present in this PLM board design*

(see 3.4.3 Input Stage), according to the tests it is possible to use it as well.

4.6.2.2 Demodulation algorithm

As shown in the code listing containing the demodulation part of the main loop from pl.c file, the demodulation routine `demfskDem()` is called when the `demfsk_NewFrmCounter` variable crosses a value of zero.

```
while(1)
{
    if ((demfsk_NewFrmCounter <= 0) &&      /* condition for the SW FSK Demodul. */
        (pl_FlgModeOfModem >= STATE7) &&    /* mode equal to PL reception */
        (pl_FlgModeOfModem <= STATE10))
            demfskDem();                    /* call SW FSK Demodulation routine */

    ...

}
```

The `demfskDem()` routine contains the complete implementation of the algorithm shown in **4.3.2.4 State model of the PLM FSK demodulation**.

As any other source codes of the project, the `demfskDem()` routine is commented very precisely so just a couple of additional notes can be mentioned regarding the implementation itself:

- As described in **4.3.2.3 Synchronization and windowing** each signal element (received bit) is calculated from the 3 results (subbits) of the `demfskDem()` routine. When 2 or 3 of the subbit values belonging to one particular bit indicate a logical “1” that bit is equal to one. Otherwise (2 or 3 subbits indicate a logical “0”) bit 0 is added to the *MSG* buffer.
- The variable `demfsk_NewFrmCounter` is set to 17, 16 and then to 17 in order to have $(17+16+17)\cdot T_s = (17+16+17)\cdot 2\mu s = 100\mu s$ time period equal to bit period T_b of the signal with a 10 kbps bitrate.

The technique used here is shown in the following figure:

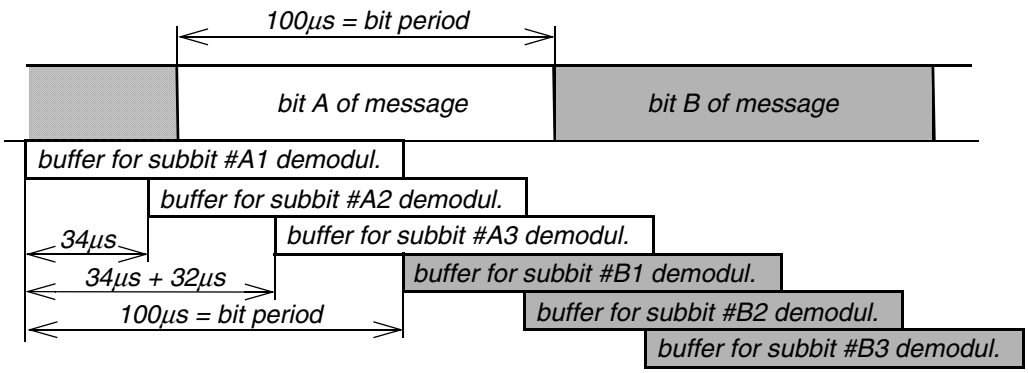


Figure 4-5. Scheme of `demfskDem()` function calling

- When a given number of bits (see 4.3.2.4 State model of the PLM FSK demodulation) of the message is received (stored in the resulting `demfsk_MSGBuf` message buffer) the PL reception phase is stopped.
- `UWord16 demState` variable is used as a State variable mentioned in 4.3.2.4.

4.6.3 CRC calculation

The Cyclic Redundancy Code (CRC) method is used to verify the integrity of every frame sent. An additional field is added to every data block at the time of transmission and then it is checked at the time of reception for correctness. One of the well-known 16-bit CRC polynoms called *CRC-16* is used in this Power Line Modem application: $x^{16}+ x^{15}+ x^{2}+ 1$.

Further mathematical details can be found in **4. IEEE Micro magazine: "A Tutorial on CRC Computations", article in IEEE Micro magazine, August 1988**.

A lookup table computation algorithm has been chosen to implement the CRC calculation. A table `const UWord16 CRCtable[256]` located in CRCtable.c file is stored in the data flash memory area. The CRC calculation routine (taken fromcoderoutines.c) is as follows:

```
Word16 codeCRCCalc(UWord16 *buffer, UWord16 n)
{
  Word16 crc = 0;

  while (n--)
   crc = ((crc >> 8) & 0xff) ^ CRCtable[(crc ^ *buffer++) & 0xff];
  return crc;
}
```

The same routine is used for both CRC field computations - before the transmission as well as after the data reception. The CRC field is generated from the *cntrl* part and the *data* part of the packet (see **4.2.4 Packet format basics**).

If the received frame is correct, the result of the CRC computation of this packet has to be equal to the CRC field that was generated (and added to the *CRC* part of the frame) during transmission.

4.6.4 FEC calculation

As mentioned in **4.2.3 Over the data operations basics**, the Forward Error Correction (FEC) technique is using added redundancy information in order to correct the errors of transmission. During transmission the redundancy data are calculated and added into a data stream, while during reception this added information is used for error detection and correction.

This reference design uses quite a straightforward method of FEC called *Linear Block Codes*. In general, block codes break up the data stream into k-bit blocks and (n-k) check (parity) bits are added to these blocks. In the literature, it is referred as a (n, k) block code.

A *FEC coder* outputs a unique n-bit *codeword v* for each of the 2^k possible input k-bit blocks called *messages u*, on the other hand the *FEC decoder* generates k-bit long *decoded received sequence u'* for each of the 2^n possible n-bit inputs so called *received sequences r*. The following ***Figure 4-6. Error control coding path*** gives a graphical explanation of the current chapter.

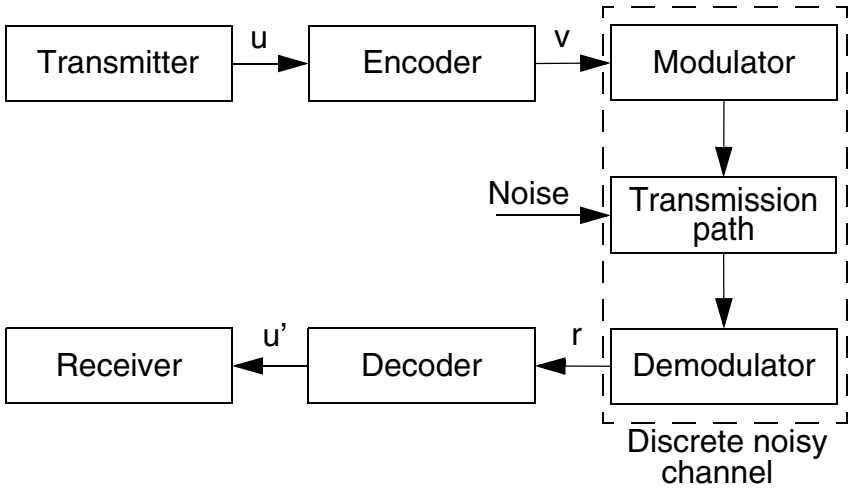


Figure 4-6. Error control coding path

Added redundancy and therefore the quality of the used FEC technique is defined by the expression (7, 4). It means that each codeword has 4 data bits and 3 redundant parity bits.

The minimal Hamming distance d_{min} (the minimal distance between two codewords) of this configuration is $d_{min} = 3$. Knowing that $d_{min} >= 2t + 1$, where t is the number of errors that can be corrected, the used FEC algorithm is able to correct one bit error in a block of 7bits.

More details about this topic can be found in: **5. Lee, Charles: "Error-control block codes for communication engineers", Artech House inc., 2000**.

Implementation of the FEC algorithm is based on two look-up tables placed in FECtable.c file. These tables were taken from the **http://www.tisl.ukans.edu/~paden/Reference/ECC/index.html**, but this page is probably no longer available. However, similar tables can be found for example in above mentioned book.

- `const UWord16 FECtableCoder[16]` is the look-up table dedicated to the FEC encoder
- `const UWord16 FECtableDecoder[128]` is the look-up table of the FEC decoder

The following piece of code (taken from the `codeMoveAndFECBuff` routine in demfsk.c) is responsible for the FEC coding:

```
for (i = FRAME_PRELEN; i < length + FRAME_PRELEN; i++)
{
    temp = ((pl_RxFromSCI.Array.Byte[i] >> 4) & 0x0F);
    temp = FECtableCoder[temp] << 7;
    temp += FECtableCoder[(pl_RxFromSCI.Array.Byte[i] & 0x0F)];
    pl_TxToPL.Array.Byte[i] = temp;
}
```

For FEC decoding the following code is used (taken from the `codePLtoSCI` routine in demfsk.c):

```
for (i = 0; i < FRAME_TOTALLEN; i++)    // FEC Decoder
{
   temp = FECtableDecoder[(pl_RxFromPL.Array.Byte[i] &
          0x3F80)>>7];
   pl_RxFromPL.Array.Byte[i] = (temp << 4) +
       FECtableDecoder[(pl_RxFromPL.Array.Byte[i] & 0x007F)];
}
```

To summarize the idea, the FEC coding routine replaces the two nibbles of data to be sent by the two 7-bit long blocks taken from the look-up table while the FEC decoding generates two nibbles back from the 14-bit long block.

4.6.5 Encryption / Decryption

The *Encryption* technique ensures the security of the transmitted data. This PLM board software utilizes *The Tiny Encryption Algorithm (TEA)* by David Wheeler and Roger Needham. For more information, see **6. David Wheeler and Roger Needham: "TEA, a Tiny Encryption Algorithm", Computer Laboratory, Cambridge University, 1994, ftp://ftp.cl.cam.ac.uk/papers/djw-rmn/djw-rmn-tea.html**.

TEA is a Feistel cipher with *XOR* and *and* addition as the non-linear mixing functions. It uses a 128-bit long encryption key (stored in `pl_TeaKey`) and a 64-bit long temporary buffer called `tea_IO` for calculation. This is the reason for the frame length restriction mentioned in **4.2.4 Packet format basics**.

The encryption/decryption implementation simply divides the whole packet into parts 8-Bytes long, performs the encryption/decryption over each of these parts and then forms them back to the frame.

As an example, here is a `teaEncryptBuff` routine taken from the tea.c, for the reverse decryption approach the principle is analogous:

```
void teaEncryptBuff(UWord16 *ptr, UWord16 roundLen)
{
    UWord16 i;
    UWord16 j = 1;
    UWord16 *backPtr;       /* pointer for back transfer */

    backPtr = ptr;          /* save a pointer */
    do
    {
        for (i = 0; i < 4; i++)     /* 8bit => 16bit */
            tea_IO.w[i] = *ptr++ + (*ptr++ << 8);
                                /* just 8bit values at *Ptr */
        teaCode();                  /* perform an encryption */
        for (i = 0; i < 4; i++)
        {                           /* 16bit => 8bit */
            *backPtr++ = (tea_IO.w[i] & 0x00FF);
            *backPtr++ = (tea_IO.w[i] & 0xFF00) >> 8;
        }
    } while ( 8*j++ < roundLen);
                            /* the length is a multiple of 8 */
}
```

4.6.6 Interleaving

The sequence of transmitted bits is modified in a way shown in **Figure 4-7. Interleaving technique of PL transmission**. The *depth of interleaving* is therefore equal to the length of the frame to be transmitted. The de-interleaving procedure during the PL reception phase is similar.

The interleaving algorithm is implemented directly in the `tmrfskBitISR` routine (placed in tmrfsk.c, see **4.3.1 FSK modulation** for more details) just by changing the order of transmitted bits.

The de-interleaving routine is called `deinterleave()` and is located in coderoutines.c.

More details about the implementation itself can be found in Figure 4-9.

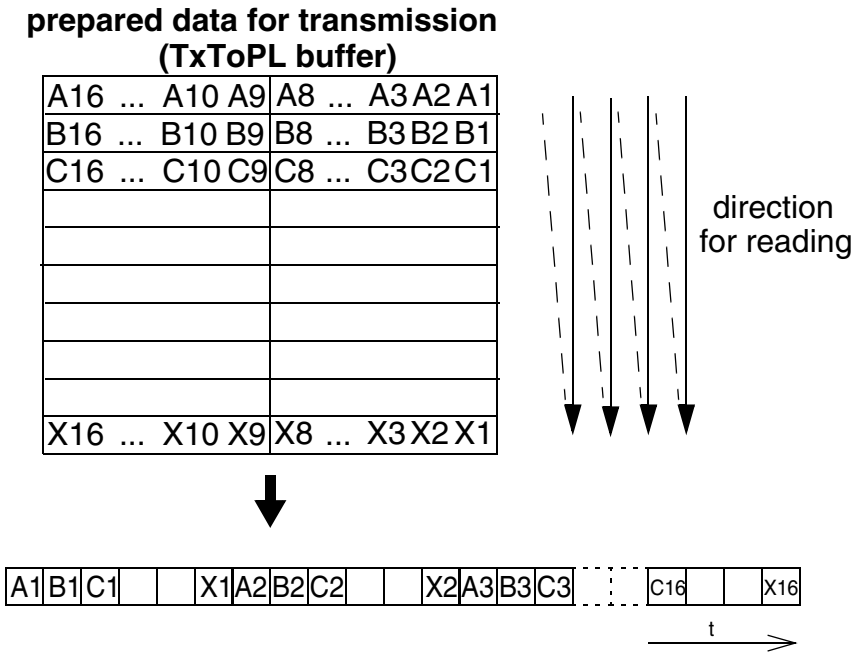


Figure 4-7. Interleaving technique of PL transmission

NOTE: *The Header part is sent in a normal linear way, there is no interleaving used.*

4.6.7 States of the PL modem

State diagram of the PLM software is shown in **Figure 4-8. State diagram of the Power Line Modem**.

Abbreviations used in the figure:

- `pl_FlgModeOfModem` variable -> MODE
- `demState` variable -> state
- powerline reception -> PL Rx
- powerline transmission -> PL Tx
- serial communication reception -> SCI Rx
- serial communication transmission -> SCI Tx

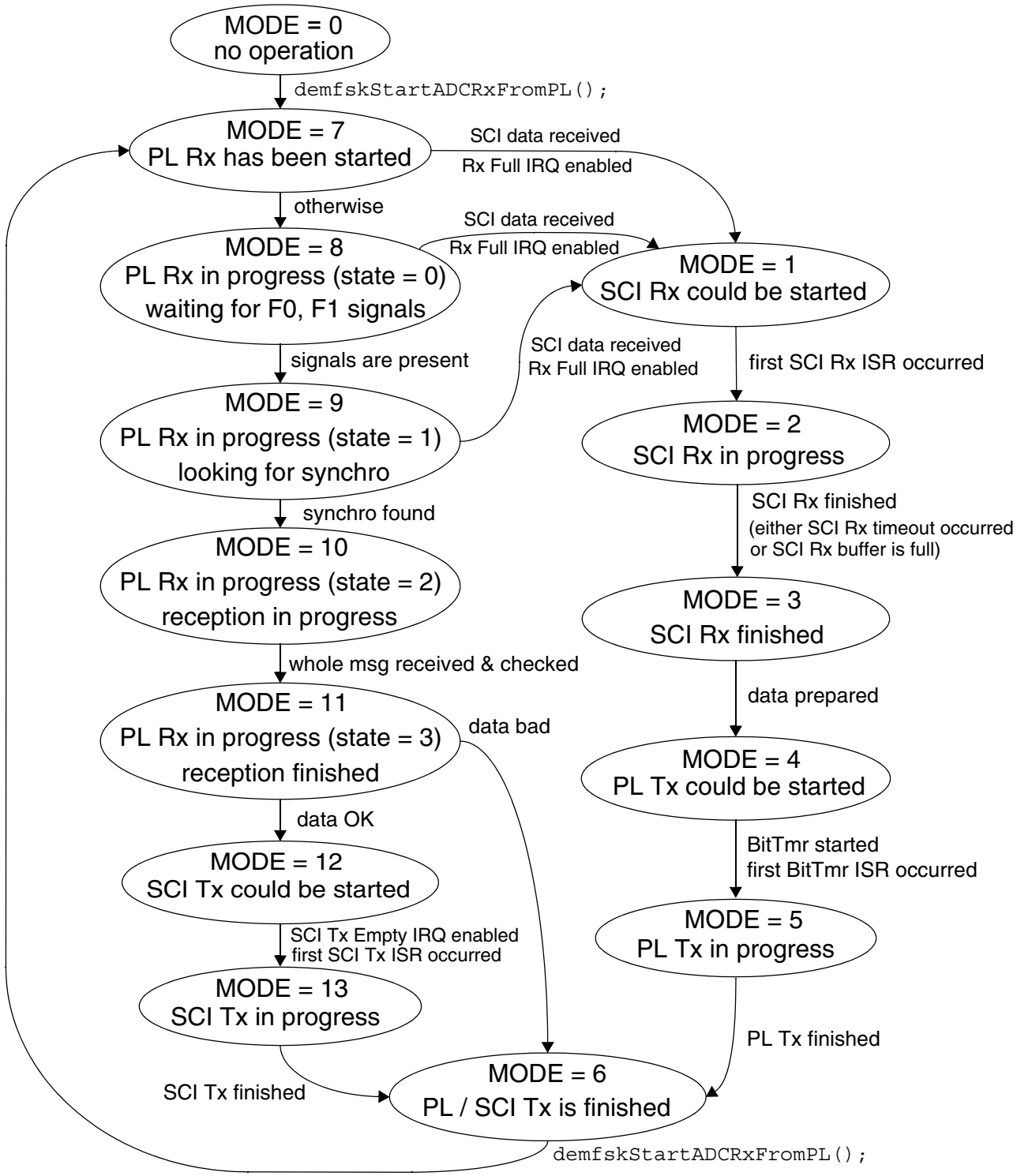


Figure 4-8. State diagram of the Power Line Modem

NOTE: *The left column of the state diagram describes the whole PL reception / SCI transmission phase (see 4.6.9), the right one the SCI reception / PL transmission part (see 4.6.8 for more details).*

NOTE: *By calling the `demfskStartADCRxFromPL()` function style macro PL data sampling is started and therefore the whole PL reception phase is activated.*

4.6.8 SCI reception / PL transmission phase

As it is shown in **Figure 4-8. State diagram of the Power Line Modem**, this phase is started by the SCI reception located in `scicommRxFullISR` routine (`pl_FlgModeOfModem` = 1 and 2). The SCI reception (data stored in `pl_RxFromSCI` buffer) is finished (`pl_FlgModeOfModem` = 3) when:

- the `pl_RxFromSCI` buffer is full
- or if an SCI reception time out occurs by the `tmrfskTimeOutISR` routine of the *TimeOutTmr* QTimer D3 (see the timer section in **4.5.3 Used DSP peripherals**)

When SCI reception is over then a `codeSCItoPL()` routine (coderoutines.c) is called. It is aimed for data preparation before the PL transmission; specifically it performs:

  – CRC calculation
  – TEA Encryption (if enabled)
  – FEC coding (if enabled)
  – prepared data are stored in the `pl_TxToPL` buffer
  – `pl_FlgModeOfModem` is set to 4

And then the PL transmission part itself can be started. It goes through the following steps:

- by calling the function style macro `tmrfskSetTxEnable()` the output amplifier is switched on

- by calling the function style macro `tmrfskStartCarrierTmr()` the *CarrierTmr* QTimer D2 (see 4.5.3 Used DSP peripherals) is started as the FSK carrier generator
- delay approx. 0.8 ms
- by the `tmrfskStartBitTmr()` function style macro the *BitTmr* QTimer D1 (see 4.5.3) is enabled for the bit rate generation during the PL transmission

NOTE: *Please notice that the interleaving is implemented in this part of the code (as mentioned in 4.6.6).*

- `tmrfskBitISR` routine of the *BitTmr* (see 4.6.1 Modulation) is used during the PL transmission of the `pl_TxToPL` data buffer (`pl_FlgModeOfModem` = 5)
- when the whole `pl_TxToPL` buffer is sent, transmission is finished:
  - `tmrfskClearTxEnable()` switches off the transmit amplifier
  - `tmrfskStopCarrierTmr()` stops the *CarrierTmr* timer
  - `tmrfskStopBitTmr()` stops the *BitTmr* timer
  - `pl_FlgModeOfModem` is set to 6
  - then by calling the `demfskStartADCRxFromPL()` function style macro the PL data sampling (whole PL reception) is started as shown in **Figure 4-8. State diagram of the Power Line Modem**. The `pl_FlgModeOfModem` variable value is set to 7.

NOTE: *A delay included before the PL transmission is necessary for the proper data demodulation on the reception side.*

NOTE: *The algorithm can be easily rewritten by the user when necessary using the current state diagram structure. Most probably a built-in application itself would generate the data for PL transmission, in such a case the SCI reception routines would be omitted.*

4.6.9 PL reception / SCI transmission phase

The PL reception algorithm (`pl_FlgModeOfModem` in the range from 7 to 11) shown in Figure 4-8. is detailed in **4.3.2.4 State model of the PLM FSK demodulation**.

When the whole received message is stored in the `demfsk_MSGBuf` buffer (`pl_FlgModeOfModem` = 12) the `codePLtoSCI()` routine is called. It then performs:

- de-interleaving (data are moved from the `demfsk_MSGBuf` buffer to `pl_RxFromPL` during this operation, see **4.5.5 Main variables of the project**)
- FEC decoding (if enabled)
- TEA decryption (if enabled)
- check the length and the CRC of the received message

If there were data consistency errors the received data are then thrown out, SCI transmission is omitted (`pl_FlgModeOfModem` set to 6) and PL reception is restarted (`pl_FlgModeOfModem` set to 7), otherwise:

- data are moved to the `pl_TxToSCI` buffer
- `pl_FlgModeOfModem` is set to 12, SCI Tx Empty IRQ is enabled
- when the first SCI Tx ISR occurred `pl_FlgModeOfModem` is set to 13 and the SCI transmission is in progress
- when the whole `pl_TxToSCI` buffer is sent, `pl_FlgModeOfModem` is set to 6 and PL reception is restarted (`pl_FlgModeOfModem` is equal to 7)

CAUTION: *During PL reception (`pl_FlgModeOfModem` in the range from 7 to 11) there should not be any ISR activated (except the ADCA End of scan) since almost all computation power is consumed by the PL demodulation algorithm. This is the reason why there is a condition testing of the SCI Receiver Data Register Full Flag (bit RDRF) in the main loop (see 4.6.11 Main loop description).*

4.6.10 Buffer details

In the following figure, detailed parameters of the used communication buffers (such as ranges, options) can be found. For more information, see also **4.5.5 Main variables of the project**, **4.5.6 Communication parameters** and also **4.6.6 Interleaving**.

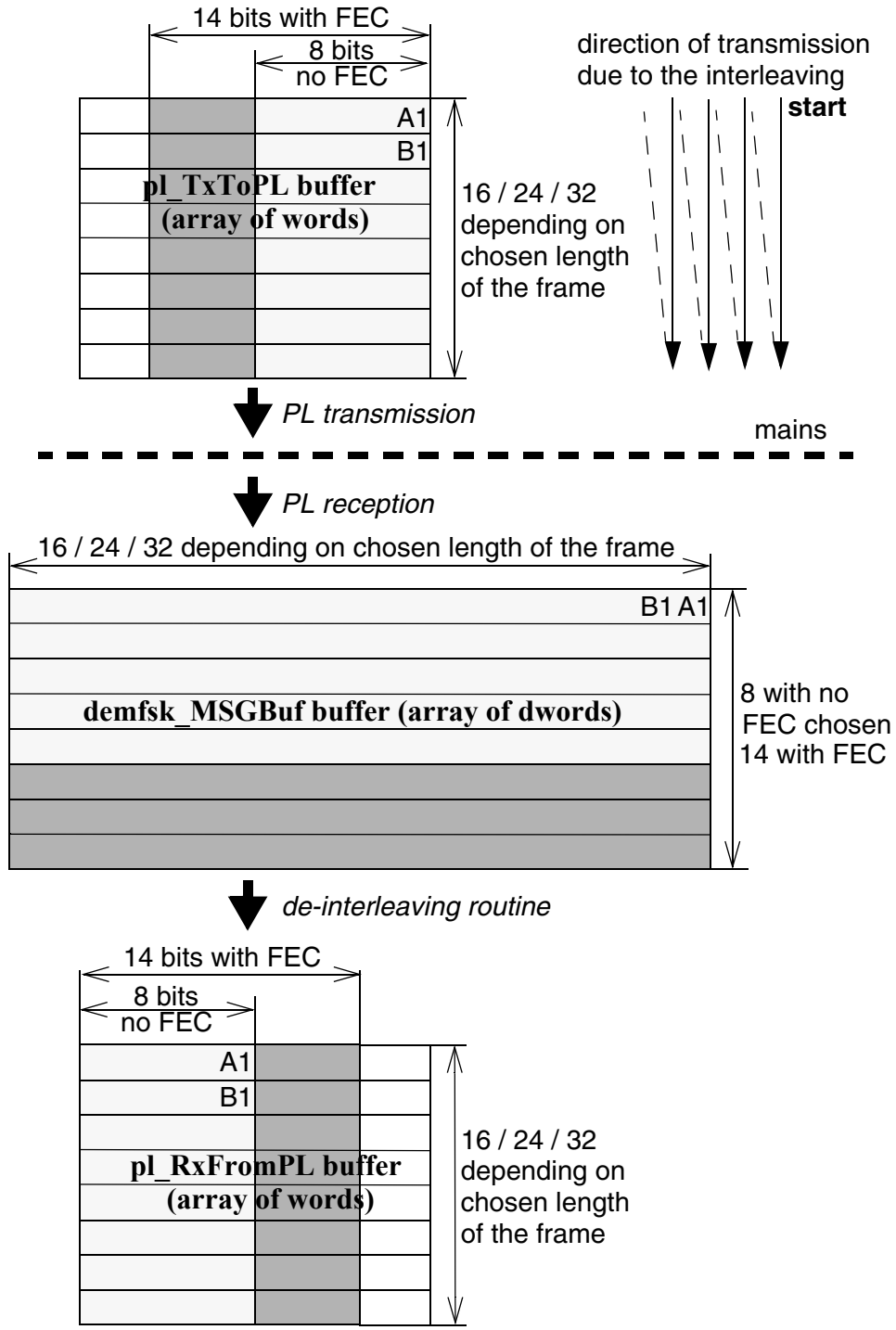


Figure 4-9. Detailed information about the buffers used

4.6.11 Main loop description

The flowchart of the main loop is shown in Figure 4-10.

Abbreviations in this figure are described here:

- `pl_FlgModeOfModem` variable -> MODE
- `demfsk_NewFrmCounter` -> Counter
- powerline reception -> PL Rx
- serial communication reception -> SCI Rx

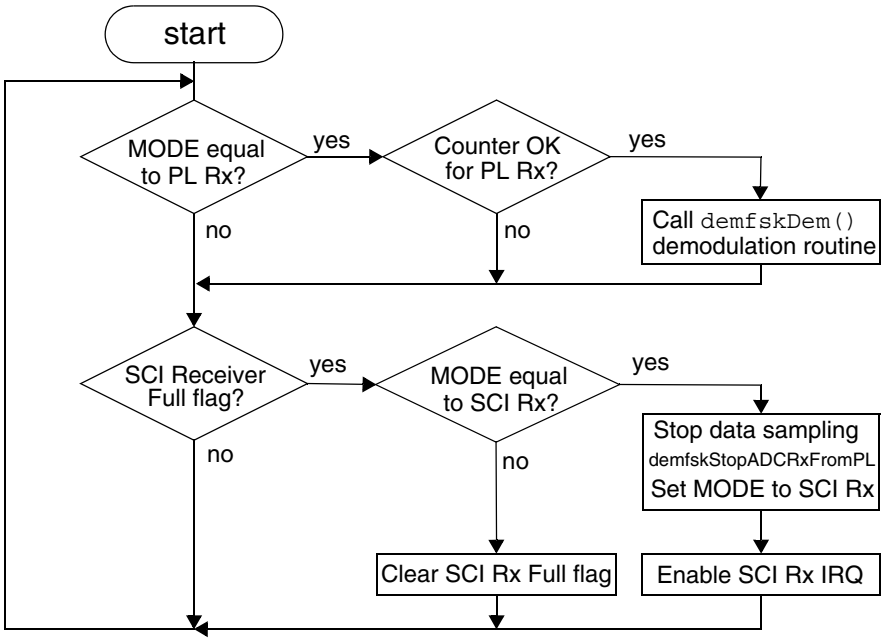


Figure 4-10. Main loop flowchart

NOTE: *The main routine is not written according to the coding rules such as no function calling in the interrupt service routines (ISR), short ISRs, testing of global flags in the main loop..., etc. It has one and only one reason - it is optimized for the speed and efficiency of the PL algorithm.*
For example, almost all global flags are tested in respective ISRs since there is no PL computation at that time. If these tests were done in the main loop, it would be too time consuming during the PL reception phase.

Software Module Descriptions

Appendix A. References

1. MOTOROLA INC.: “DSP56F80x User’s Manual”, Motorola User’s Manual, 2000, http://e-www.motorola.com
2. MOTOROLA INC.: “DSP56800 Family Manual”, Motorola Family Manual, 2000, http://e-www.motorola.com
3. CENELEC EN 50065-1: “Signaling on low-voltage electrical installations in the frequency range 3 kHz to 148.5 kHz”, 1991
4. IEEE Micro magazine: “A Tutorial on CRC Computations”, article in IEEE Micro magazine, August 1988
5. Lee, Charles: “Error-control block codes for communication engineers”, Artech House inc., 2000
6. David Wheeler and Roger Needham: “TEA, a Tiny Encryption Algorithm”, Computer Laboratory, Cambridge University, 1994, ftp://ftp.cl.cam.ac.uk/papers/djw-rmn/djw-rmn-tea.html
7. MOTOROLA INC.: “AN2262/D: Wireless HC08 Modem”, 2002

References

Appendix B. Bill of Materials and Schematics

B.1 Contents

This section includes:

- PLM_5 board bill of materials - Table B-1.
- PLM_5 board schematics
  - PLM_BLOCKS - Figure B-1.
  - Power Stage&Coupling- Figure B-2.
  - Output Filter - Figure B-3.
  - Input Stage - Figure B-4.
  - Microcontroller - Figure B-5.
  - RS232 Interface - Figure B-6.
  - Power - Figure B-7.

Table B-1. PLM_5 board bill of materials

Part	Value
C1,C2,C4,C20	2.2uF/10V
C3	6.8nF
C13	5.6nF
C5,C9,C11,C12	10nF
C6,C8	680pF
C14	330pF
C7,C24,C31,C32,C33,C34,C35, C37,C43,C44,C45,C46,C47,C49	100nF
C10	3.3nF
C18,C19,C36	33uF/16V

Table B-1. PLM_5 board bill of materials

C22	100pF
C23	220uF/16V
C25	100uF/16V
C26	47pF
C27	39pF
C28	220pF
C29	470nF
C30	47nF/X2
C38	4.7uF/10V
D1, D2, D3, D4, D5	LED 3mm
D7, D8	Diode P6KE10A
D9	V275LA4
D10	Diode BAV99LT1
D11, D12, D14	Diode 1N4148
F1	Fuse
JP1	Connector 2 screws
J2	Connector Cannon 9
J3	Header 10x2
J29	Header 10x2
L1, L5	Inductor 0.33mH
L2, L3, L6	Inductor 3.3mH
L4	Inductor 47uH
L102, L103, L104	Ferrite Bead
R1,R2,R8,R9,R18,R24,R27,R29, R30,R53,R55,R56,R59,R60	10k
R4, R5	100R
R6,R7,R11,R20,R50,R54,R112	1k
R19, R12	4.7k
R16	18k
R23	33R

Table B-1. PLM_5 board bill of materials

R25	100k
R26	4.7R
R28	1M
R45	10M
R101	47R/4W
S1	Pushbutton
T1	Trafo 78250
U1	LF351
U4	TLE2301
U5	MAX3232ECAE
U8	MC33269DT_3.3
U20	DSP56F801FA80
U23	74AC00
Y1	Xtal 8MHz

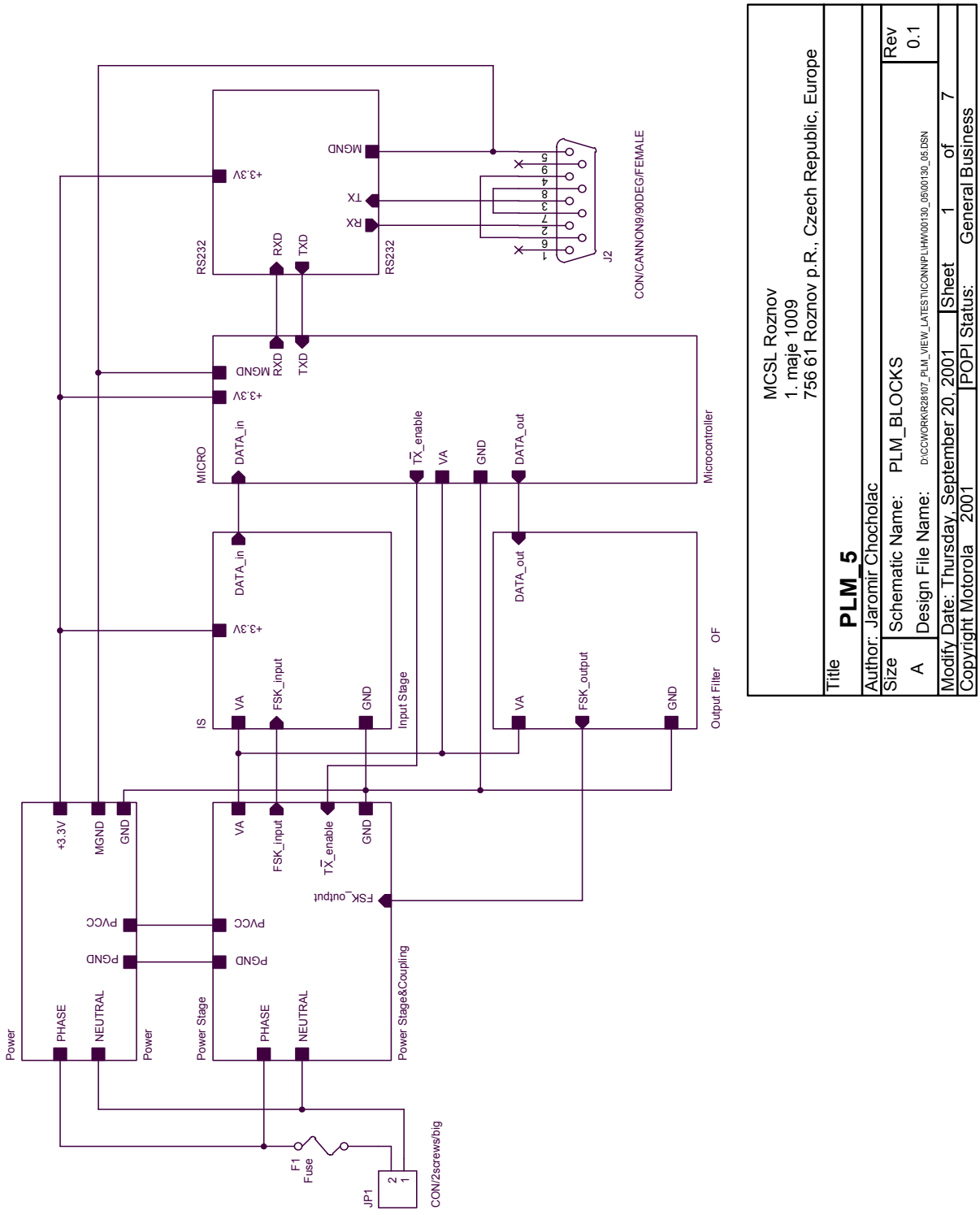


Figure B-1. PLM_BLOCKS

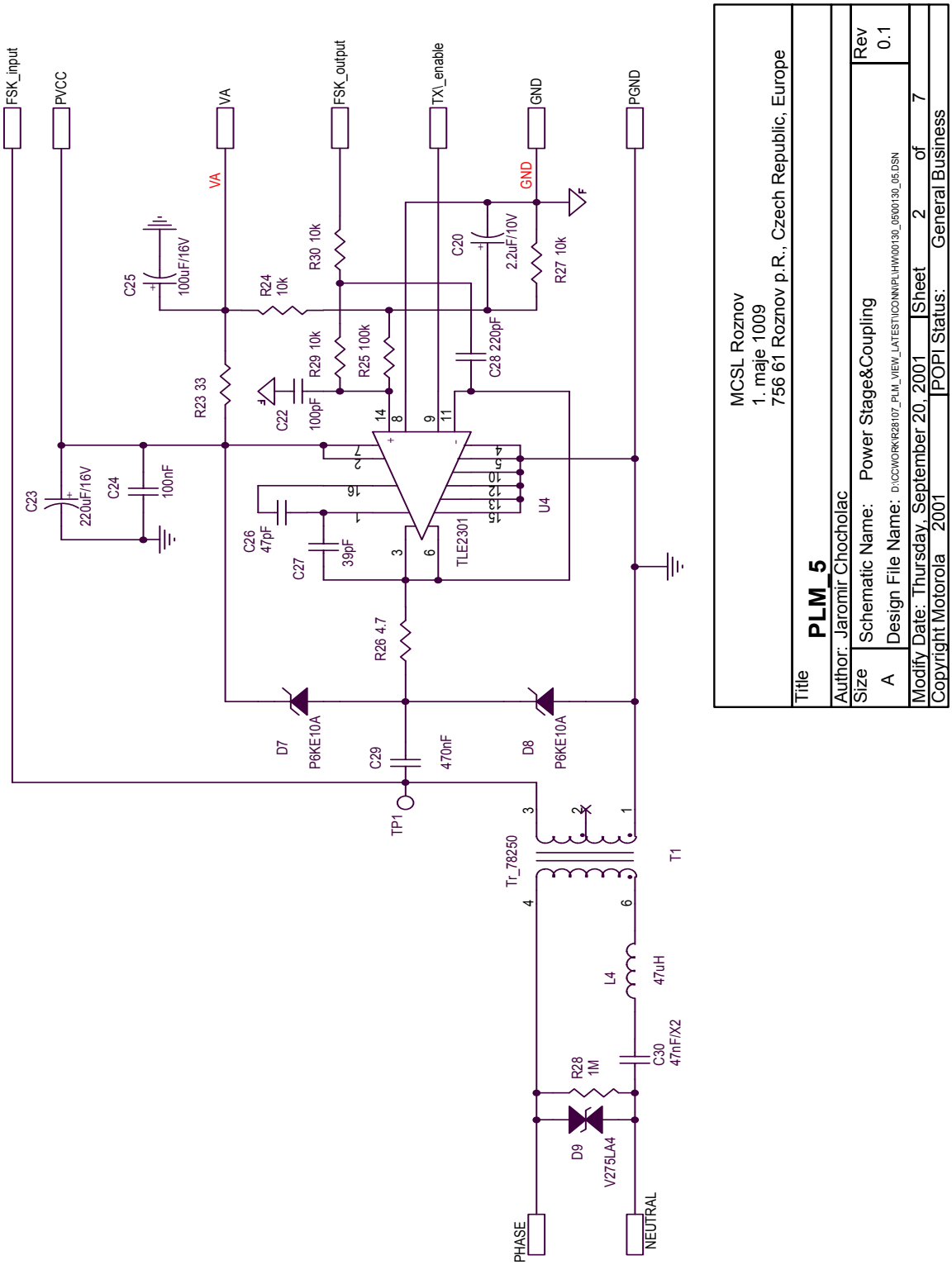


Figure B-2. Power Stage&Coupling

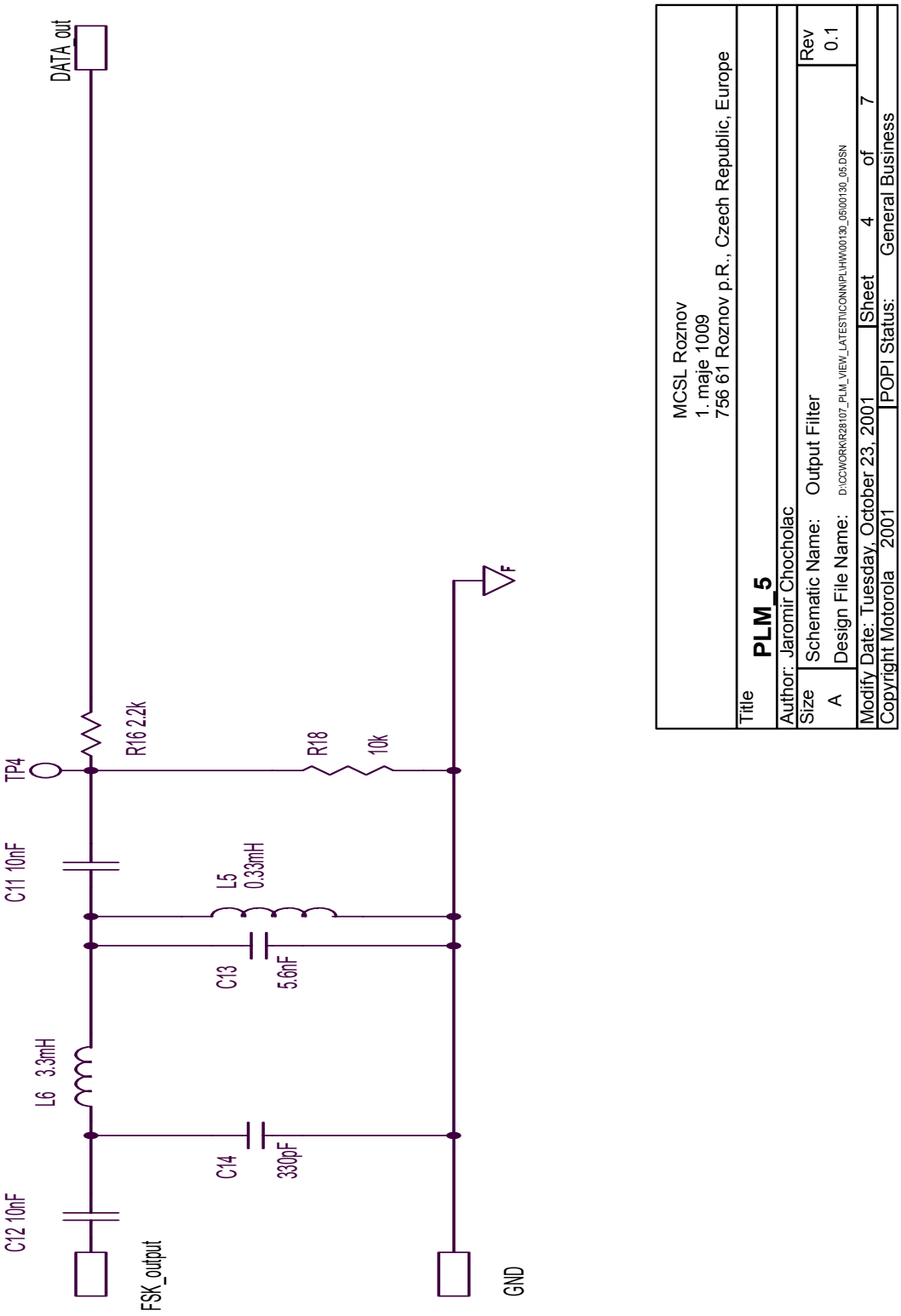
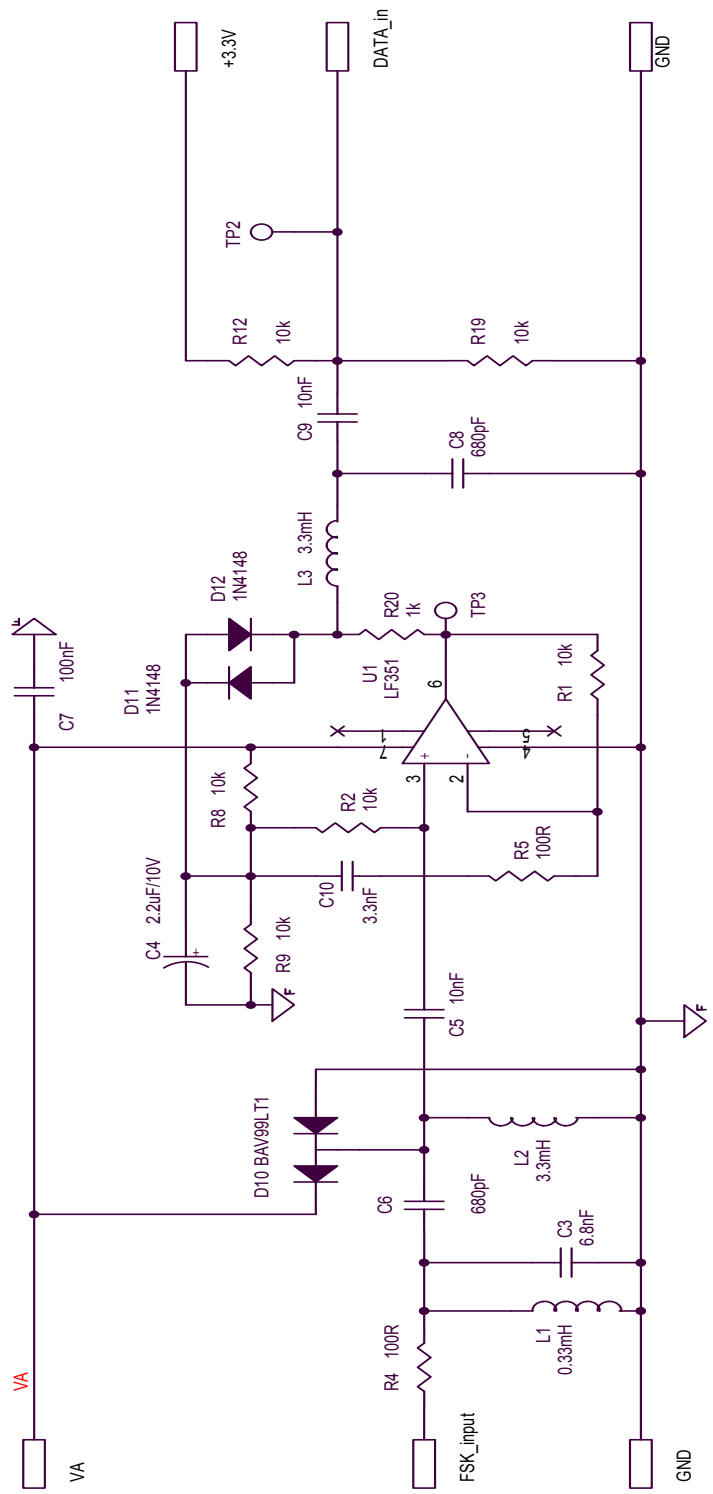


Figure B-3. Output Filter



MCSL Roznov
1. maje 1009
756 61 Roznov p.R., Czech Republic, Europe

Title **PLM_5**

Author: Jaromir Chocholac

Size A | Schematic Name: Input Stage | Rev 0.1

Design File Name: D:\CCWORK\R28107_PLM_VIEW_LATEST\CONN\PLHW00130_05\00130_05.DSN

Modify Date: Thursday, October 11, 2001 | Sheet 3 of 7

Copyright Motorola 2001 | POPI Status: General Business

Figure B-4. Input Stage

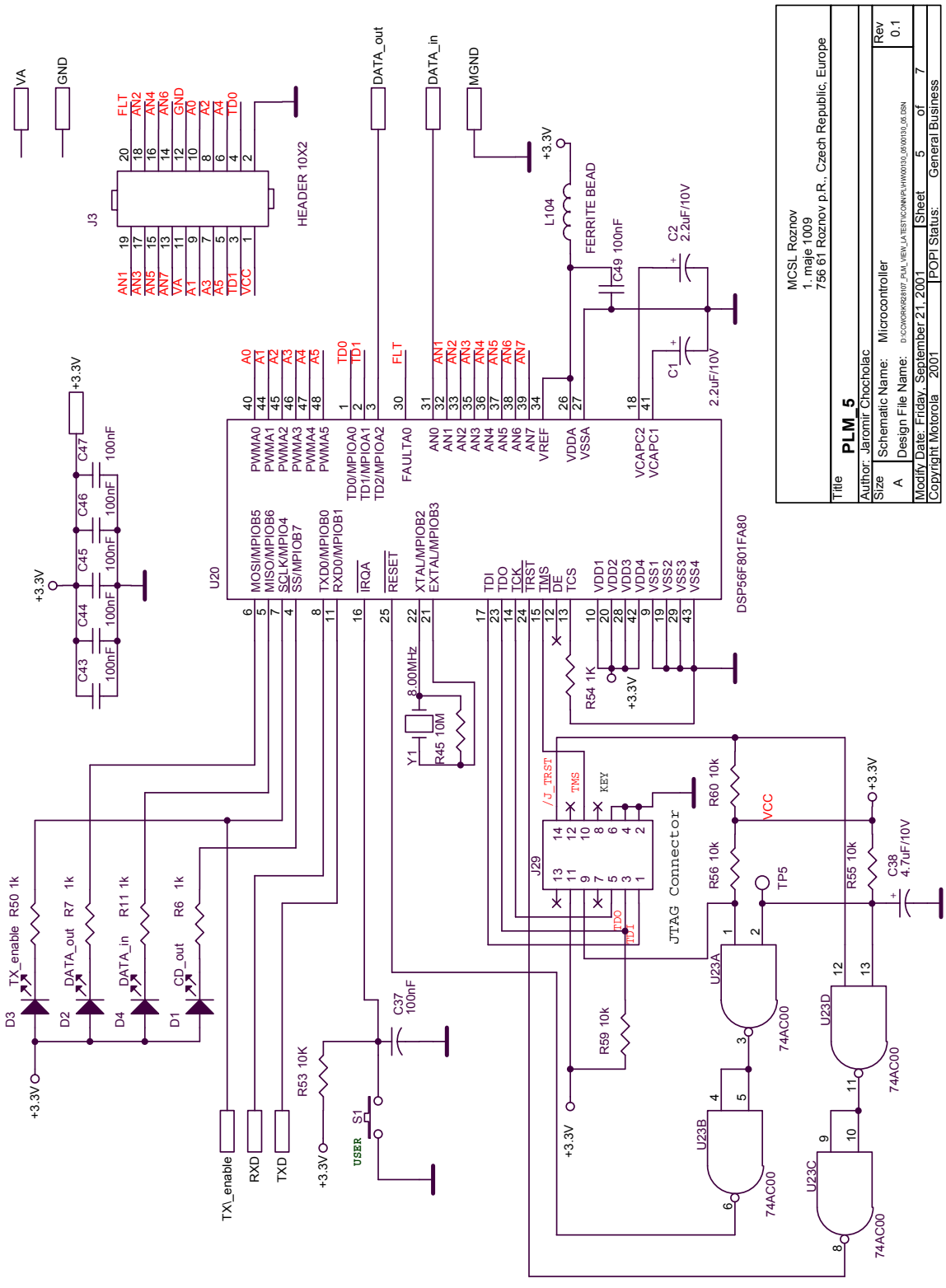


Figure B-5. Microcontroller

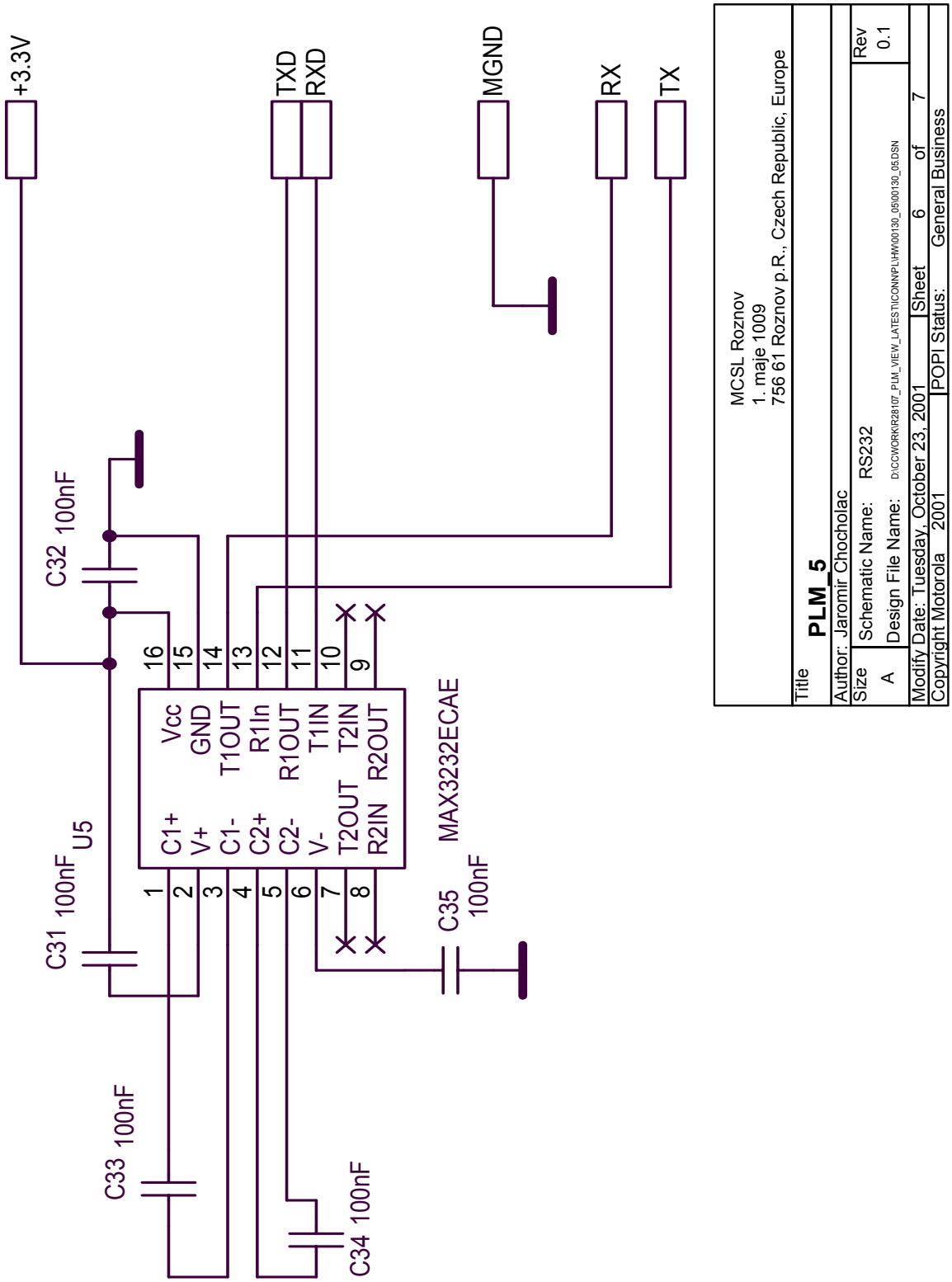


Figure B-6. RS232 Interface

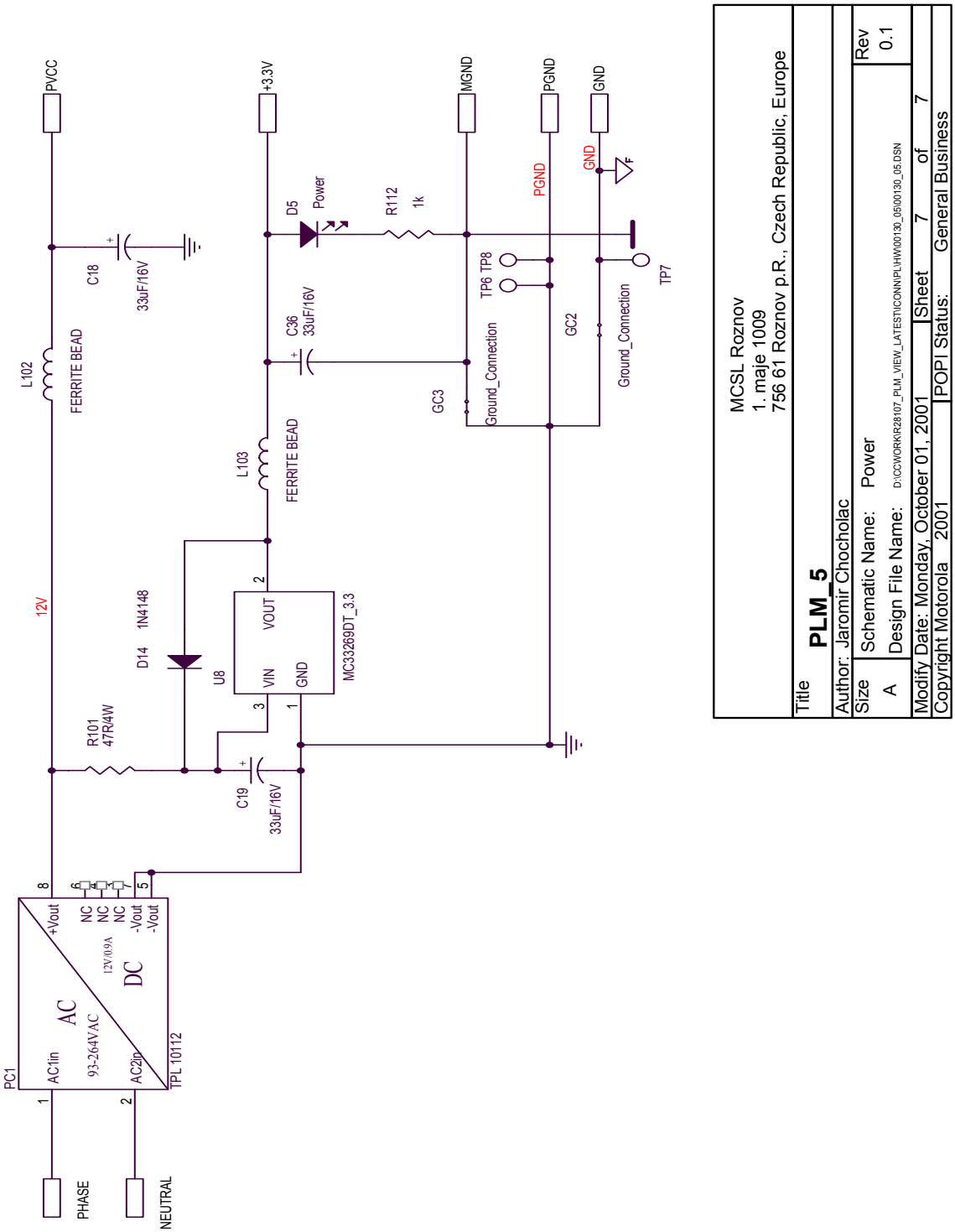


Figure B-7. Power

Appendix C. Source Code Files

C.1 Contents

C.2 Introduction

This subsection is comprised of the source code used by this design reference.

C.3 pl.c

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: pl.c
*
* DESCRIPTION: Powerline modem main routine
*
* MODULES INCLUDED:
*       main()
*       plProjectInit()
*
*****************************************************************************/

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "types.h"
#include "arch.h"
#include "periph.h"
#include "mc.h"

#include "appconfig.h"

#include "adc.h"
#include "cop.h"
#include "gpio.h"
#include "itcn.h"
#include "qtimer.h"
#include "sci.h"

#include "pl.h"
#include "tea.h"
#include "scicomm.h"
#include "tmrfsk.h"
#include "demfsk.h"
#include "coderoutines.h"

/*****************************************************************************/
/*              P R O T O T Y P E S                                          */
/*****************************************************************************/
void plProjectInit(void);

/*****************************************************************************/
```

```
/*              GLOBAL VARIABLES                                              */
/****************************************************************************/
// Note that structure type pl_uRxFromSCI is used for the RxFromSCI as well as
// for the TxToPL buffers
    pl_uRxFromSCI       pl_RxFromSCI;
        /* Buffer dedicated for SCI reception */
    pl_uRxFromSCI       pl_TxToPL;
        /* Buffer dedicated for Power Line transmission */
    pl_uRxFromPL        pl_RxFromPL;
        /* Buffer dedicated for Power Line reception */
    pl_uTxToSCI         pl_TxToSCI;
        /* Buffer dedicated for SCI transmission */

    volatile pl_sFlags  pl_Flags;   /* contains the state and another flags
                                       of PL modem device */
    const tea_uKey      pl_TeaKey = { 1, 2, 3, 4, 5, 6, 7, 8 };
        /* Key for TEA (Tiny Encryption Algorithm) computation */

/* extern of SW FSK Demodulation variable taken from the demfsk.c file */
    extern volatile Word16 demfsk_NewFrmCounter;
        /* used as a counter in ADCEndOfScanISR */
    extern UWord32 demfsk_MSGBuf[DEMFSK_MSGBUFLEN];
        /* buffer of received message of FSK demodulation routine */

/****************************************************************************
*
* Module: void plProjectInit(void)
*
* Description:
*       This function initializes the core peripherals (static configuration
*       is taken from the appconfig.h file generated by config tool)
*       Note that some core peripheral parameters depend on the global defines
*       situated in pl.h file
*       It initialises global variables as well.
*
* Returns: None
*
* Global Data:
*       pl_Flags - flag pl_FlgModeOfModem is set to STATE 0 "No operation"
*                - flag pl_FlgDataError is cleared
*       For the global data description of the demfskInit() routine see the
*       description of the routine itself
*       PL_COPINUSE is a symbolic constant, it defined the Watch Dog is used
*       DSP56F803 is a symbolic constant, if defined the 56F803 core is used
*       DSP56F801 is a symbolic constant, if defined the 56F801 core is used
*       PL_PLBAUDRATE is a symbolic constant which controls the PL
*           communication speed
*       PL_CARRIERLOW is a symbolic constant which controls the frequecy of one
*           of PL carrier
*       PL_TIMEOUTVALUE is a symbolic constant which controls the time-out value
*           of TimeOut timer TmrD3
```

Source Code Files

```
*       PL_SCIBAUDRATE is a symbolic constant which controls the SCI
*           communication speed
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
void plProjectInit(void)
{
/*******************************************************************************/
/*  PERIPHERY INITIALIZATION                                                   */
/*******************************************************************************/
    /* init COP if desired */
#ifdef PL_COPINUSE
    ioctl(COP, COP_INIT, NULL);
    ioctl(COP, COP_DEVICE, COP_ENABLE);
    ioctl(COP, COP_CLEAR_COUNTER, NULL);    // the COP service sequence
#endif
/* Initially the COP module is disabled by startup.asm code but if there is
/* the global define
/* #define PL_COPINUSE              /* if defined the Watch Dog is used */
/* placed in the pl.h file, it finally switch the COP on */

    /* init GPIO */
#ifdef DSP56F803
    ioctl( GPIO_E, GPIO_INIT, NULL);
#endif
#ifdef DSP56F801
    ioctl( GPIO_B, GPIO_INIT, NULL);
#endif

    tmrfskClearTxEnable();  // set initial pin value
    tmrfskTxDLEDOff();      // set initial pin value
    tmrfskRxDLEDOff();      // set initial pin value
    tmrfskCDLEDOff();       // set initial pin value

    /* init ADCA */
    ioctl(ADC_A, ADC_INIT, NULL);

    /* init TmrC2 - TriggerTmr for ADC */
    ioctl(QTIMER_C2, QT_INIT, NULL);

    /* init TmrD1 - BitTmr */
    ioctl( QTIMER_D1, QT_INIT, NULL );
    ioctl( QTIMER_D1, QT_WRITE_COMPARE_REG1, PL_PLBAUDRATE);
/* There is no valid definition of TmrD1 Compare register 1 value in appconfig.h
/* configuration, it depends on the global define placed in the pl.h file:
/* #define PL_PLBAUDRATE        PL_10000BPS     /* choose:  PL_10000BPS */
```

```
    /* init TmrD2 - CarrierTmr */
    ioctl(QTIMER_D2, QT_INIT, NULL);
    ioctl(QTIMER_D2, QT_WRITE_COMPARE_REG1, PL_CARRIERLOW);
/* There is no valid definition of TmrD2 Compare register 1 value in appconfig.h
/* configuration, it depends on the global define placed in the pl.h file:
/* #define PL_CARRIERLOW   CARRIERLOW_110KHZ10KBPS */

    /* init TmrD3 - TimeOutTmr */
    ioctl(QTIMER_D3, QT_INIT, NULL);
    ioctl(QTIMER_D3, QT_WRITE_COMPARE_REG1, PL_TIMEOUTVALUE);
/* There is a zero value of TmrD3 Compare register 1 written in appconfig.h
/* configuration, this register is filled according the global define
/* #define PL_TIMEOUTVALUE 1000         /* time out of SCI receive */
/* placed in the pl.h file */

    /* init SCI */
    ioctl( SCI_0, SCI_INIT, NULL);
    ioctl( SCI_0, SCI_SET_BAUDRATE_DIV, PL_SCIBAUDRATE);
/* There is no definition for SCI baudrate value in appconfig.h configuration,
/* it depends on the global define placed in the pl.h file:
/* #define PL_SCIBAUDRATE SCI_BAUD_38400 /* choose:         SCI_BAUD_38400 */
                                          /* not tested:     SCI_BAUD_4800
                                                             SCI_BAUD_9600
                                                             SCI_BAUD_19200 */

/* Enable Interrupts in three steps */
/* - first, set interrupt priorities in Group Priority Registers (GPR)
     according to defined ITCN_INT_PRIORITY_xx in appconfig.h. */
    ioctl(ITCN, ITCN_INIT_GPRS, NULL);
/* - second, in Interrupt Priority Register (IPR):
   -- enable interrupt channels according to defined ITCN_INT_PRIORITY_xx
      in appconfig.h
   -- configure external interrupts according to defined config items
      in appconfig.h */
    ioctl(ITCN, ITCN_INIT_IPR, NULL);
/* - third, enable maskable interrupts in Status Register SR, bits I1 and I0 */
    archEnableInt();        // Enable maskable (Level 0) interrupts

/*************************************************************************/
/*  VARIABLES INITIALIZATION                                             */
/*************************************************************************/
    pl_FlgModeOfModem = STATE0; // set mode - No operation
    pl_FlgDataError = 0;        // clear the flag
    demfskInit();
}

/*************************************************************************
*
* Module: void main()
*
```

Source Code Files

```
* Description:
*       Main routine of the PowerLine modem project
*
* Returns: None
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
void main (void)
{
    UWord16 temp;

    plProjectInit();        // initializes the core peripherals and variables

    demfskStartADCRxFromPL();   // start PL data sampling (PL reception)
    pl_FlgModeOfModem = STATE7; // set Mode of modem flag

/*******************************************************************************/
/*    I M P O R T A N T   N O T E                                              */
/*******************************************************************************/
// This main loop is very critical because of its speed!!!
// Call demfskDem() routine as frequently as possible
    while(1)
    {
        if ((demfsk_NewFrmCounter <= 0) &&  // condition for the SW FSK Demodul.
            (pl_FlgModeOfModem >= STATE7) &&    // mode equal to PL reception
            (pl_FlgModeOfModem <= STATE10))
                demfskDem();                // call SW FSK Demodulation routine

        if (ioctl(SCI_0, SCI_GET_RX_FULL, NULL)) // SCI Rx is full
        {
            if ((pl_FlgModeOfModem >= STATE6) && // SCITx or PLTx is finished or
                (pl_FlgModeOfModem <= STATE9))   // PLRx is started
            {                                    // (in demState = 0 or 1)
                demfskStopADCRxFromPL();     // stop PL data sampling
                                             // (stop Rx from the PL side)
                pl_FlgModeOfModem = STATE1;  // set Mode of modem flag
                ioctl(SCI_0, SCI_RX_ERROR_INT, SCI_ENABLE);  // enable interrupt
                ioctl(SCI_0, SCI_RX_FULL_INT, SCI_ENABLE);   // enable interrupt
            }
            else
            {
                if (ioctl(SCI_0, SCI_GET_RX_FULL, NULL))    // clear flag
                   temp = ioctl(SCI_0, SCI_READ_DATA, NULL);
                if (ioctl(SCI_0, SCI_GET_ERROR, NULL))      // clear flags
                   ioctl(SCI_0, SCI_CLEAR_STATUS_REG, NULL);
            }
```

```
        }

        #ifdef PL_COPINUSE
            ioctl(COP, COP_CLEAR_COUNTER, NULL);    // the COP service sequence
        #endif
    }
}

/* IMPORTANT NOTE:
/* Call this condition with demfskDem() routine as frequently as possible */
/*
        if ((demfsk_NewFrmCounter <= 0) &&  // condition for the SW FSK Demodul.
            (pl_FlgModeOfModem >= STATE7) &&    // mode equal to PL reception
            (pl_FlgModeOfModem <= STATE10))
                demfskDem();                // call SW FSK Demodulation routine
*/

/* IMPORTANT NOTE:
/* Although this condition should be placed here in main loop, it is situated
/* at the end of SCI Rx routine tmrfskTimeOutISR() because of the main loop
/* speed optimalization. */
/*
        if (pl_FlgModeOfModem == STATE3)    // pl_RxFromSCI buff is full
        {
            ioctl(SCI_0, SCI_RX_FULL_INT, SCI_DISABLE); // disable interrupt
            if (ioctl(SCI_0, SCI_GET_RX_FULL, NULL))    // clear flag
                Dummy = ioctl(SCI_0, SCI_READ_DATA, NULL);
            codeSCItoPL();                  // prepare data from SCI to PL
            pl_FlgModeOfModem = STATE4;     // set Mode of Modem
            tmrfskSetTxEnable();            // switch on the transmitter
            tmrfskStartCarrierTmr();        // start generation of FSK carrier
            archDelay(0x1FFF);              // Tx of the carrier before the
            archDelay(0x1FFF);              // header and data part transmission
            archDelay(0x1FFF);              // total 0.8ms
            archDelay(0x1FFF);
            tmrfskStartBitTmr();            // start FSK transmission
        }
*/

/* IMPORTANT NOTE:
/* Although this condition should be placed here in main loop, it is situated
/* at the end of demfskDem() routine because of the main loop speed
/* optimalization. */
/*
        if (pl_FlgModeOfModem == STATE11)   // pl_RxFromPL finished
        {
            codePLtoSCI();                  // prepare data from PL to SCI
            if (pl_FlgDataError == 0)       // check the data consistency
            {
                pl_FlgModeOfModem = STATE12;// set Mode of Modem
                ioctl(SCI_0, SCI_TX_EMPTY_INT, SCI_ENABLE); // enable SCI Tx IRQ
```

```
            }
            else                            // Bad data consistency
            {
                pl_FlgDataError = 0;        // clear flag
                pl_FlgModeOfModem = STATE6; // set Mode of Modem
                demfskStartADCRxFromPL();   // start PL data sampling
                                            // (start Rx from the PL side)
                pl_FlgModeOfModem = STATE7; // set Mode of Modem
            }
        }
*/

/* IMPORTANT NOTE:
/* Although this condition should be placed here in main loop, it is situated
/* at the end of TxToPL and TxToSCI routines because of the main loop speed
/* optimalization. */
/*
        if (pl_FlgModeOfModem == STATE6)    // pl_TxToPL or pl_TxToSCI finished
        {
            demfskStartADCRxFromPL();       // start PL data sampling
                                            // (PL reception)
            pl_FlgModeOfModem = STATE7;     // set Mode of Modem
        }
*/
```

C.4 pl.h

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: pl.h
*
* DESCRIPTION: Header file for Powerline modem main routine (pl.c)
*
* MODULES INCLUDED: None
*
*****************************************************************************/
#ifndef _PL_H
#define _PL_H

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "types.h"
```

```
/*****************************************************************************/
/* G L O B A L   P L   M O D E M   S T A T E   M A C H I N E   D E F I N E S */
/*****************************************************************************/
// NOTE: These defines are used for the pl_FlgModeOfModem variable definition
//      state:                description of PL Modem Mode:
#define STATE0     0     /* No operation, no communication of modem */
#define STATE1     1     /* SCI reception could be started, RxFromSCI buffer */
                         /* is ready */
#define STATE2     2     /* SCI reception in progress */
#define STATE3     3     /* SCI reception has been finished */
#define STATE4     4     /* PL transmission could be started, TxToPL buffer */
                         /* is ready */
#define STATE5     5     /* PL transmission in progress */
#define STATE6     6     /* PL / SCI transmission has been finished */
#define STATE7     7     /* PL reception has been started */
#define STATE8     8     /* PL reception in progress, FSK demodulation in */
                         /* Demstate 0 (waiting until F0 or F1 is present) */
#define STATE9     9     /* PL reception in progress, FSK demodulation in */
                         /* Demstate 1 (finding synchronization pattern) */
#define STATE10   10     /* PL reception in progress, FSK demodulation in */
                         /* Demstate 2 (data reception) */
#define STATE11   11     /* PL reception in progress, FSK demodulation in */
                         /* Demstate 3 (data reception finished) */
#define STATE12   12     /* SCI transmission could be started, TxToSCI buffer */
                         /* is ready */
#define STATE13   13     /* SCI transmission in progress */

/*****************************************************************************/
/* G L O B A L   D E F I N E S                                               */
/*****************************************************************************/
#define PL_SCIBAUDRATE  SCI_BAUD_38400  /* choose:      SCI_BAUD_38400 */
                                        /* not tested:  SCI_BAUD_4800
                                                        SCI_BAUD_9600
                                                        SCI_BAUD_19200 */
#define PL_PLBAUDRATE   TMRFSK_10000BPS /* choose:      TMRFSK_10000BPS */

#define PL_FRAMETYPE    LONG            /* choose:  SHORT
                                                    MEDIUM
                                                    LONG */
// if SHORT type is used, length of the data part of packet is 13 words
// if MEDIUM type is used, length of the data part of packet is 21 words
// if LONG type is used, length of the data part of packet is 29 words
// Note when FEC is OFF, just lower 8 bits of the word are used
//                  ON, lower 14bits of the word carry the data

#define PL_FECTYPE  PL_1STFEC           /* choose   PL_NOFEC
                                                    PL_1STFEC */

#define PL_TEACRYPT 1                   /* if defined perform TEA encryption */
```

Source Code Files

```
#define PL_TIMEOUTVALUE 1000            /* time out of SCI receive */
                                        /* 1000 is tested for SCI_BAUD_38400 */

//#define PL_COPINUSE                   /* if defined the Watch Dog is used */

/* Choose the carrier frequencies */
#if 0
#define PL_CARRIERLOW   CARRIERLOW_110KHZ10KBPS
#define PL_CARRIERHGH   CARRIERHGH_100KHZ10KBPS
#endif

#if 1
#define PL_CARRIERLOW   CARRIERLOW_115KHZ10KBPS
#define PL_CARRIERHGH   CARRIERHGH_105KHZ10KBPS
#endif

#if 0
#define PL_CARRIERLOW   CARRIERLOW_120KHZ10KBPS
#define PL_CARRIERHGH   CARRIERHGH_110KHZ10KBPS
#endif
/*****************************************************************************/
/* D E B U G    D E F I N E S                                                */
/*****************************************************************************/
//#define PL_NOINTERLEAVING     // if defined the PL transmission didn't perform
                                // the interleaving
/* NOTE: use this define just for testing of PL transmission because it modifies
/* only the PL Tx routine, the PL reception needs the interleaving!

/*****************************************************************************/
/* P R I V A T E   D E F I N E S                                             */
/*****************************************************************************/
/* NOTE: when carrier is called "low" (used for log. "0")
         => frequencies are higher and vice versa */
#define CARRIERLOW_110KHZ10KBPS   182 - 1 /* half period of 110kHz, 10kBps */
#define CARRIERHGH_100KHZ10KBPS   200 - 1 /* half period of 100kHz, 10kBps */

#define CARRIERLOW_115KHZ10KBPS   174 - 1 /* half period of 115kHz, 10kBps */
#define CARRIERHGH_105KHZ10KBPS   190 - 1 /* half period of 105kHz, 10kBps */

#define CARRIERLOW_120KHZ10KBPS   167 - 1 /* half period of 120kHz, 10kBps */
#define CARRIERHGH_110KHZ10KBPS   182 - 1 /* half period of 110kHz, 10kBps */

#define SHORT       0
#define MEDIUM      1
#define LONG        2

#define PL_NOFEC    8
#define PL_1STFEC   14

#if (PL_FECTYPE == PL_NOFEC)
    #define PL_TXMASK       0x0080
```

```
#else
    #define PL_TXMASK       0x2000
#endif

#define PL_HEADERTXMASK     0x0080

#define FRAME_PRELEN   1    // length of pre-control part of packet [in bytes]
                            // 1B of header
#define FRAME_HEADER 0xA5   // this is the header of the frame

#define FRAME_CNTRLLEN 3    // length of the non-data part of packet [in bytes]
                            // 1B of length, 2B of CRC
#if (PL_FRAMETYPE == SHORT)
    #define FRAME_DATALEN 13    // length of the data part of packet [in bytes]
#elif (PL_FRAMETYPE == MEDIUM)
    #define FRAME_DATALEN 21    // length of the data part of packet [in bytes]
#else
    #define FRAME_DATALEN 29    // length of the data part of packet [in bytes]

#endif
    #define FRAME_TOTALLEN  (FRAME_DATALEN + FRAME_CNTRLLEN)
                                // length of the data and CNTRL part of packet

#if (PL_FECTYPE == PL_NOFEC)
    #define FRAME_TOTALBITS (FRAME_TOTALLEN * 8)  // total number of bits for Rx
#else
    #define FRAME_TOTALBITS (FRAME_TOTALLEN * 14) // total number of bits for Rx
#endif

/*****************************************************************************/
/* S T R U C T U R E S                                                       */
/*****************************************************************************/
typedef struct                  // frame AS STRUCTURE of the SCI reception
// Note that this structure is used also for PL transmission
{
    UWord16 Header[FRAME_PRELEN];// header
    UWord16 Cntrl;              // len
    UWord16 Data[FRAME_DATALEN];// data part
    UWord16 CRC[2];             // 2B of CRC, low byte first
} pl_sStructRxFromSCI;

typedef struct                  // frame AS ARRAY of the SCI reception
{
    UWord16 Byte[FRAME_PRELEN + FRAME_DATALEN + FRAME_CNTRLLEN];
} pl_sArrayRxFromSCI;

typedef union                   // complete union of the SCI reception
{
    pl_sStructRxFromSCI Struct; // frame AS STRUCTURE of the SCI reception
    pl_sArrayRxFromSCI Array;   // frame AS ARRAY of the SCI reception
} pl_uRxFromSCI;
```

Source Code Files

```
/*****************************************************************************/

typedef struct                  // frame AS STRUCTURE of the PL reception
{
    UWord16 Cntrl;              // len
    UWord16 Data[FRAME_DATALEN];// data part
    UWord16 CRC[2];             // 2B of CRC, low byte first
} pl_sStructRxFromPL;

typedef struct                  // frame AS ARRAY of the PL reception
{
    UWord16 Byte[FRAME_CNTRLLEN + FRAME_DATALEN];
} pl_sArrayRxFromPL;

typedef union                   // complete union of the PL reception
{
    pl_sStructRxFromPL Struct;  // frame AS STRUCTURE of the PL reception
    pl_sArrayRxFromPL Array;    // frame AS ARRAY of the PL reception
} pl_uRxFromPL;
/*****************************************************************************/

typedef struct                  // frame AS STRUCTURE of the SCI transmission
{
    UWord16 Cntrl;              // len
    UWord16 Data[FRAME_DATALEN];// data part
} pl_sStructTxToSCI;

typedef struct                  // frame AS ARRAY of the SCI transmission
{
    UWord16 Byte[FRAME_CNTRLLEN + FRAME_DATALEN - 2];   // minus 2B of CRC
} pl_sArrayTxToSCI;

typedef union                   // complete union of the SCI transmission
{
    pl_sStructTxToSCI Struct;   // frame AS STRUCTURE of the SCI transmission
    pl_sArrayTxToSCI Array;     // frame AS ARRAY of the SCI transmission
} pl_uTxToSCI;
/*****************************************************************************/

typedef struct
{
    UWord16 ModeOfModem : 4;    /* Mode of the modem */
/* Here are the possible states of pl_FlgModeOfModem variable */
/*   State:         Description of PL Modem Mode: */
/*  STATE0    No operation, no communication of modem */
/*  STATE1    SCI reception could be started, RxFromSCI buffer is ready */
/*  STATE2    SCI reception in progress */
/*  STATE3    SCI reception has been finished */
/*  STATE4    PL transmission could be started, TxToPL buffer is ready */
/*  STATE5    PL transmission in progress */
/*  STATE6    PL / SCI transmission has been finished */
```

```
/*  STATE7    PL reception has been started */
/*  STATE8    PL reception in progress, FSK demodulation in */
/*            Demstate 0 (waiting until F0 or F1 is present) */
/*  STATE9    PL reception in progress, FSK demodulation in */
/*            Demstate 1 (finding synchronization pattern) */
/*  STATE10   PL reception in progress, FSK demodulation in */
/*            Demstate 2 (data reception) */
/*  STATE11   PL reception in progress, FSK demodulation in */
/*            Demstate 3 (data reception finished) */
/*  STATE12   SCI transmission could be started, TxToSCI buff is ready */
/*  STATE13   SCI transmission in progress */

    UWord16 DataError : 1; /* Data Error occured in Rx PL frame */
                           /* bad CRC code or bad data length */
                           /* 0 - no error */
                           /* 1 - error occured */
} pl_sFlags;

/*****************************************************************************/
/* SHORT-CUT DEFINES                                                         */
/*****************************************************************************/
#define pl_FlgModeOfModem           pl_Flags.ModeOfModem
#define pl_FlgDataError             pl_Flags.DataError

#endif
```

C.5 tmrfsk.c

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: tmrfsk.c
*
* DESCRIPTION: This file consists of all timer-based routines needed for the PL
*       modem (such as FSK bit rate ISR and Timeout ISR).
*
* MODULES INCLUDED:
*       tmrfskBitISR()
*       tmrfskTimeOutISR()
*
*****************************************************************************/

/*****************************************************************************/
/*              I N C L U D E S                                              */
```

Source Code Files

```
/*****************************************************************************/
#include "types.h"
#include "arch.h"
#include "periph.h"
#include "appconfig.h"

#include "qtimer.h"
#include "gpio.h"
#include "sci.h"

#include "tmrfsk.h"
#include "demfsk.h"
#include "coderoutines.h"

#include "pl.h"

/*****************************************************************************/
/*              GLOBAL VARIABLES                                             */
/*****************************************************************************/
    extern pl_uRxFromSCI       pl_TxToPL;
        /* Buffer dedicated for Power Line transmission */
    extern volatile pl_sFlags  pl_Flags;    /* contains the state and another
                                               flags of PL modem device */

/*****************************************************************************
*
* Module: void tmrfskBitISR(void)
*
* Description:
*       This function is the ISR of the Bit Timer (TmrD1). It is used during the
*       PL transmission, it generates the interrupt each bit period. After each
*       bit period it is necessary to set the proper timing values for FSK
*       carrier generation. This FSK carrier generation is done by CarrierTmr
*       (TmrD2).
*       Note that transmission consists of two steps:
*       1) Tx of HEADER part which is sent in normal (non-interleaved) way
*       2) Tx of the rest of the packet is sent using the interleaving technique
*          (when PL_NOINTERLEAVING is not defined) or without the interleaving
*          (PL_NOINTERLEAVING is defined)
*
* Returns: None
*
* Global Data:
*       pl_Flags - flag pl_FlgModeOfModem - if the Mode was set to STATE4,
*                       "PL transmission could be started" then it is switched
*                       to STATE5 "PL transmission in progress". When PL Tx is
*                       finished, it is set to STATE6 "PL / SCI transmission
*                       has been finished" and then to STATE7 "PL reception has
*                       been started"
*       pl_TxToPL is a buffer to be sent
*       PL_NOINTERLEAVING is a symbolic constant, if defined the interleaving
```

```
*           over the transmission is NOT performed
*       PL_TXMASK is a symbolic constant which controls the mask of the PL
*           transmission
*       PL_HEADERTXMASK is a symbolic constant which controls the mask of the PL
*           transmission of the Header
*       PL_CARRIERLOW and PL_CARRIERHGH are symbolic constants describing both
*           carrier frequencies
*       PL_FECTYPE is a symbolic constant describing the type of used FEC
*           correction
*       FRAME_TOTALLEN is a symbolic constant describing the total length of
*           the whole packet [in B] to be sent
*       FRAME_PRELEN is a symbolic constant describing the length of header
*            part of the packet
*
* Arguments: None
*
* Range Issues: Only when pl_FlgModeOfModem is equal to STATE4 or STATE5, the
*           PL transmission is performed
*
* Special Issues: None
*
*******************************************************************************/
#pragma interrupt
void tmrfskBitISR(void)
{
    static UWord16 mask;            // mask in array of transmit frame
    static UWord16 index;           // index in array of transmit frame
    static bool txHeader;           // Tx of header part in prograss [yes / no]

    ioctl( QTIMER_D1, QT_CLEAR_FLAG, QT_COMPARE_FLAG);

#ifndef PL_NOINTERLEAVING           // PL Tx with INTERLEAVING
    if ( pl_FlgModeOfModem == STATE4)   // test Mode of Modem condition
    {
        pl_FlgModeOfModem = STATE5;     // set Mode of Modem
        index = 0;
        mask = 1;
        txHeader = 1;                   // send Header part now
    }
    if ( pl_FlgModeOfModem == STATE5)   // test Mode of Modem condition
    {
        if (mask <= PL_TXMASK)
        {
            if (mask & pl_TxToPL.Array.Byte[index])     // currect bit is "1"
            {
                if (ioctl(QTIMER_D2, QT_READ_COMPARE_REG1, NULL) ==
                    PL_CARRIERLOW); // if previous value was logical "0"
                {
                    while (ioctl(QTIMER_D2, QT_READ_COUNTER_REG, NULL) >=
                                    PL_CARRIERHGH - TMRFSK_SAFETYRESERVE);
                    tmrfskSetCarrierHigh(); // set logical "1" carrier
```

Source Code Files

```
                tmrfskTxDLEDOn();       // Set the transmit LED indication
            }
        }
        else                                        // currect bit is "0"
        {
            if (ioctl(QTIMER_D2, QT_READ_COMPARE_REG1, NULL) ==
                PL_CARRIERHGH); // if previous value was logical "1"
            {
                while (ioctl(QTIMER_D2, QT_READ_COUNTER_REG, NULL) >=
                                PL_CARRIERHGH - TMRFSK_SAFETYRESERVE);
                tmrfskSetCarrierLow();  // set logical "0" carrier
                tmrfskTxDLEDOff();      // Clear the transmit LED indication
            }
        }

        if (txHeader == 1)  // Header part (just 8bits) is beeing transmited
        {                   // in linear way (no interleaving)
            mask <<= 1;     // movement in packet array in a row direct
            if (mask > PL_HEADERTXMASK)   // check the 8bit length
            {
                txHeader = 0;   // Header part has been transmitted
                index = FRAME_PRELEN;
                mask = 1;
            }
        }
        else            // Interleaving of the data packet part
        {               // movement in packet array in a column direction
            if (index == FRAME_TOTALLEN + FRAME_PRELEN - 1)
            {
                mask <<= 1;
                index = FRAME_PRELEN;
            }
            else
                index++;
        }
    }
#else
// PL Tx with NO INTERLEAVING
    if ( pl_FlgModeOfModem == STATE4)         // test Mode of Modem condition
    {
        pl_FlgModeOfModem = STATE5;           // set Mode of Modem
        index = 0;
#if (PL_FECTYPE == PL_NOFEC)
        mask = 1;                           // Mask for 8bit Header with no FEC
#else
        pl_TxToPL.Array.Byte[0] <<= 6;      // right shift of Header
        mask = 0x40;                        // Mask for 8bit Header with FEC
#endif
    }
    if ( pl_FlgModeOfModem == STATE5)       // test Mode of Modem condition
    {
```

```
        if (index < FRAME_TOTALLEN + FRAME_PRELEN)
        {
            if (mask & pl_TxToPL.Array.Byte[index])
            {
                if (ioctl(QTIMER_D2, QT_READ_COMPARE_REG1, NULL) ==
                    PL_CARRIERLOW); // if previous value was logical "0"
                {
                    while (ioctl(QTIMER_D2, QT_READ_COUNTER_REG, NULL) >=
                                    PL_CARRIERHGH - TMRFSK_SAFETYRESERVE);
                    tmrfskSetCarrierHigh(); // set logical "1" carrier
                    tmrfskTxDLEDOn();       // Set the transmit LED indication
                }
            }
            else
            {
                if (ioctl(QTIMER_D2, QT_READ_COMPARE_REG1, NULL) ==
                    PL_CARRIERHGH); // if previous value was logical "1"
                {
                    while (ioctl(QTIMER_D2, QT_READ_COUNTER_REG, NULL) >=
                                   PL_CARRIERHGH - TMRFSK_SAFETYRESERVE);
                    tmrfskSetCarrierLow();  // set logical "0" carrier
                    tmrfskTxDLEDOff();  // Clear the transmit LED indication
                }
            }

            if (mask == PL_TXMASK)
            {
                mask = 1;
                index++;
            }
            else
                mask <<= 1;
        }
#endif
        else    // end of frame
        {
            tmrfskClearTxEnable();  // Disable the transmit amplifier
            tmrfskStopCarrierTmr(); // Stop the carrier generation Tmr
            tmrfskStopBitTmr();     // Stop the Bit period Tmr
            tmrfskTxDLEDOff();      // set initial pin value
            archDelay(0x1FFF);      // wait a while after the transmission
            archDelay(0x1FFF);      // in order to settle the line
            pl_FlgModeOfModem = STATE6;     // set Mode of Modem
            demfskStartADCRxFromPL();       // start PL data sampling
                                            // (PL reception)
            pl_FlgModeOfModem = STATE7;     // set Mode of Modem
        }
    }
}

/*****************************************************************************
```

Source Code Files

```
*
* Module: void tmrfskTimeOutISR(void)
*
* Description:
*       This function is the ISR of the TimeOut Timer (TmrD3). It generates the
*       timeout to indicate that the SCI reception was stopped before fulfilling
*       the whole SCI Rx buffer. It simply gives the order to stop the SCI
*       reception and start the PL transmission part.
*
* Returns: None
*
* Global Data:
*       pl_Flags - flag pl_FlgModeOfModem - the Mode is inicially set to STATE3,
*                       "SCI reception has been finished" but after the
*                       codeSCItoPL() routine the data for PL transmission is
*                       ready so it switches to STATE4 "PL transmission could
*                       be started".
*       For the global data description of the codeSCItoPL() routine see the
*       description of the routine itself
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
#pragma interrupt
void tmrfskTimeOutISR(void)
{
    UWord16 temp;

    archPushAllRegisters();
    ioctl( QTIMER_D3, QT_CLEAR_FLAG, QT_COMPARE_FLAG);      // clear the flag
    pl_FlgModeOfModem = STATE3;     // set Mode of Modem
    ioctl(SCI_0, SCI_RX_FULL_INT, SCI_DISABLE); // disable interrupt
    if (ioctl(SCI_0, SCI_GET_RX_FULL, NULL))    // clear flag
        temp = ioctl(SCI_0, SCI_READ_DATA, NULL);
    codeSCItoPL();                  // prepare data from SCI to PL
    pl_FlgModeOfModem = STATE4;     // set Mode of Modem
    tmrfskSetTxEnable();            // switch on the transmitter
    tmrfskStartCarrierTmr();        // start generation of FSK carrier
    archDelay(0x1FFF);              // Tx of the carrier before
    archDelay(0x1FFF);              // the header and data part transmission
    archDelay(0x1FFF);              // total 0.8ms
    archDelay(0x1FFF);
    tmrfskStartBitTmr();            // start FSK transmission
    archPopAllRegisters();
}
```

C.6 tmrfsk.h

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: tmrfsk.h
*
* DESCRIPTION: This file consists of all timer-based macro defines needed for
*       the PL modem. It also incorporates the GPIO-based macro defines.
*
* MODULES INCLUDED: None
*
*****************************************************************************/
#ifndef _TMRFSK_H
#define _TMRFSK_H

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "types.h"
#include "arch.h"
#include "periph.h"

#include "qtimer.h"
#include "gpio.h"

/*****************************************************************************/
/*              P R O T O T Y P E S                                          */
/*****************************************************************************/
void tmrfskBitISR(void);
void tmrfskTimeOutISR(void);

/*****************************************************************************/
/* PL Transmission And Baudrate Speed Defines                                */
/*****************************************************************************/
#define TMRFSK_10000BPS             4000 - 1    /* 10000pbs */
#define TMRFSK_19200BPS             2084 - 1    /* 19200pbs */

#define TMRFSK_SAFETYRESERVE    30

/*****************************************************************************/
/* GPIO Mapping Defines                                                      */
/*****************************************************************************/
#define TXENABLE  0x0010  /* bit No. 4  transmission disabled / enabled pin */
#define TXD       0x0020  /* bit No. 5  transmission data pin */
```

Source Code Files

```
#define RXD       0x0040  /* bit No. 6  reception data pin */
#define CD        0x0080  /* bit No. 7  carrier detection pin */

/*************************************************************************/
/*              M A C R O S                                              */
/*************************************************************************/
/* Timer D1 section */
#define tmrfskStartBitTmr()     \
        ioctl(QTIMER_D1, QT_WRITE_COUNTER_REG, 0);                        \
        ioctl(QTIMER_D1, QT_SET_COUNT_MODE, QT_COUNT_RISING_EDGES_MODE)
    /* clear and start the Bit Timing Tmr [D1] */
#define tmrfskStopBitTmr()  \
        ioctl(QTIMER_D1, QT_SET_COUNT_MODE, QT_NO_OPERATION)
    /* stop the Bit Timing Tmr [D1] */

/* Timer D2 section */
#define tmrfskSetCarrierHigh()      \
        ioctl(QTIMER_D2, QT_WRITE_COMPARE_REG1, PL_CARRIERHGH)
    // switch tmr D2 oscillation to High transmit frequency */
#define tmrfskSetCarrierLow()       \
        ioctl(QTIMER_D2, QT_WRITE_COMPARE_REG1, PL_CARRIERLOW)
    /* switch tmr D2 oscillation to Low transmit frequency */
#define tmrfskStartCarrierTmr()     \
        ioctl(QTIMER_D2, QT_WRITE_COUNTER_REG, 0);                        \
        ioctl(QTIMER_D2, QT_SET_COUNT_MODE, QT_COUNT_RISING_EDGES_MODE)
    /* clear and start the Carrier Timing Tmr [D2] */
#define tmrfskStopCarrierTmr()  \
        ioctl(QTIMER_D2, QT_SET_COUNT_MODE, QT_NO_OPERATION);             \
        ioctl(QTIMER_D2, QT_FORCE_OFLAG, 0);                              \
        ioctl(QTIMER_D2, QT_EXT_OFLAG_FORCE, QT_ENABLE)
    /* stop the Carrier Timing Tmr [D2], force the OFLAG bit to be logic. "0" */

/* Timer D3 section */
#define tmrfskStartTimeOutTmr()     \
        ioctl(QTIMER_D3, QT_SET_COUNT_MODE, QT_COUNT_RISING_EDGES_MODE)
    /* start the Time Out Timer [D3] */
#define tmrfskStopTimeOutTmr()      \
        ioctl(QTIMER_D3, QT_SET_COUNT_MODE, QT_NO_OPERATION);             \
        ioctl(QTIMER_D3, QT_WRITE_COUNTER_REG, 0x0000)
    /* stop the Time Out Timer [D3] and write a start value into */
#define tmrfskClearTimeOutTmr() ioctl(QTIMER_D3, QT_WRITE_COUNTER_REG, 0x0000)
    /* set TimeOut Tmr to a start value [D3] */

/* GPIO section */
#ifdef DSP56F803
/* Note that TxEnable output pin control the function of the modem
/* If the PowerLine modem should:
/*      perform a transmission then pin TxENABLE is cleared and the indication
/*      LED is switched on
/*      not perform a transmission then pin TxENABLE is set and the indication
/*      LED is switched off
```

```
/* See that signal TxEnable is inverted; to enable it has to be cleared
/* and vice versa */
    #define tmrfskSetTxEnable()     ioctl(GPIO_E, GPIO_CLEAR_PIN, TXENABLE)
    #define tmrfskClearTxEnable()   ioctl(GPIO_E, GPIO_SET_PIN, TXENABLE)
/* Note that TxD output pin is only for signalization
/* If the PowerLine modem transmission data value (Tx) is:
/*      logical "1" then pin TxD is cleared to 0 and LED is switched on
/*      logical "0" then pin TxD is set to 1 and LED is switched off */
    #define tmrfskTxDLEDOn()        ioctl(GPIO_E, GPIO_CLEAR_PIN, TXD)
    #define tmrfskTxDLEDOff()       ioctl(GPIO_E, GPIO_SET_PIN, TXD)
/* Note that RxD output pin is only for signalization
/* If the PowerLine modem reception data value (Rx) is:
/*      logical "1" then pin RxD is cleared to 0 and LED is switched on
/*      logical "0" then pin RxD is set to 1 and LED is switched off */
    #define tmrfskRxDLEDOn()        ioctl(GPIO_E, GPIO_CLEAR_PIN, RXD)
    #define tmrfskRxDLEDOff()       ioctl(GPIO_E, GPIO_SET_PIN, RXD)
/* Note that CD output pin is only for signalization
/* If the PowerLine modem reception is:
/*      processing then pin CD is cleared to 0 and LED is switched on
/*      not processing then pin CD is set to 1 and LED is switched off */
    #define tmrfskCDLEDOn()         ioctl(GPIO_E, GPIO_CLEAR_PIN, CD)
    #define tmrfskCDLEDOff()        ioctl(GPIO_E, GPIO_SET_PIN, CD)
#endif

#ifdef DSP56F801
/* Note that TxEnable output pin control the function of the modem
/* If the PowerLine modem should:
/*      perform a transmission then pin TxENABLE is cleared and the indication
/*      LED is switched on
/*      not perform a transmission then pin TxENABLE is set and the indication
/*      LED is switched off
/* See that signal TxEnable is inverted; to enable it has to be cleared
/* and vice versa */
    #define tmrfskSetTxEnable()     ioctl(GPIO_B, GPIO_CLEAR_PIN, TXENABLE)
    #define tmrfskClearTxEnable()   ioctl(GPIO_B, GPIO_SET_PIN, TXENABLE)
/* Note that TxD output pin is only for signalization
/* If the PowerLine modem transmission data value (Tx) is:
/*      logical "1" then pin TxD is cleared to 0 and LED is switched on
/*      logical "0" then pin TxD is set to 1 and LED is switched off */
    #define tmrfskTxDLEDOn()        ioctl(GPIO_B, GPIO_CLEAR_PIN, TXD)
    #define tmrfskTxDLEDOff()       ioctl(GPIO_B, GPIO_SET_PIN, TXD)
/* Note that RxD output pin is only for signalization
/* If the PowerLine modem reception data value (Rx) is:
/*      logical "1" then pin RxD is cleared to 0 and LED is switched on
/*      logical "0" then pin RxD is set to 1 and LED is switched off */
    #define tmrfskRxDLEDOn()        ioctl(GPIO_B, GPIO_CLEAR_PIN, RXD)
    #define tmrfskRxDLEDOff()       ioctl(GPIO_B, GPIO_SET_PIN, RXD)
/* Note that CD output pin is only for signalization
/* If the PowerLine modem reception is:
/*      processing then pin CD is cleared to 0 and LED is switched on
/*      not processing then pin CD is set to 1 and LED is switched off */
```

```
    #define tmrfskCDLEDOn()         ioctl(GPIO_B, GPIO_CLEAR_PIN, CD)
    #define tmrfskCDLEDOff()        ioctl(GPIO_B, GPIO_SET_PIN, CD)
#endif

#endif
```

C.7 demfsk.c

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: demfsk.c
*
* DESCRIPTION: Source file for the SW FSK demodulator
*
* MODULES INCLUDED:
*       demfskInit()
*       demfskDem()
*       demfskEndOfScanISR()
*       calcDTFT()
*       slidAverage()
*       numOnes()
*
*****************************************************************************/

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "types.h"
#include "arch.h"
#include "periph.h"
#include "appconfig.h"

#include "qtimer.h"
#include "gpio.h"
#include "adc.h"
#include "sci.h"

#include "demfsk.h"
#include "tmrfsk.h"
#include "coderoutines.h"
#include "pl.h"

#undef add
```

```
#undef sub

/*************************************************************************/
/*              P R O T O T Y P E S                                      */
/*************************************************************************/
asm Word16 calcDTFT(Word16 *pCoeff);
asm void slidAverage(Word16 *Si, Word16 lambi, Word16 f);
UWord16 numOnes(UWord32 tempVar);

/*************************************************************************/
/*              GLOBAL VARIABLES                                         */
/*************************************************************************/
    volatile Word16 demfsk_NewFrmCounter;
        /* used as a counter in ADCEndOfScanISR */
    UWord32 demfsk_MSGBuf[DEMFSK_MSGBUFLEN];
        /* buffer of received message of FSK demodulation routine */
    extern volatile pl_sFlags  pl_Flags;    /* contains the state and another
                                               flags of PL modem device */

/*************************************************************************/
/*              LOOK-UP TABLE GLOBAL VARIABLES                           */
/*************************************************************************/
extern const Word16 K100[2*DEMFSK_FRAMELEN]; // FSK demodulator const of 100kHz
extern const Word16 K105[2*DEMFSK_FRAMELEN]; // FSK demodulator const of 105kHz
extern const Word16 K110[2*DEMFSK_FRAMELEN]; // FSK demodulator const of 110kHz
extern const Word16 K115[2*DEMFSK_FRAMELEN]; // FSK demodulator const of 115kHz
extern const Word16 K120[2*DEMFSK_FRAMELEN]; // FSK demodulator const of 120kHz

/*************************************************************************/
/*              GLOBAL VARIABLES OF THE FILE                             */
/*************************************************************************/
Word16  SA;                 /* long-term sliding average of (F0+F1) */
Word16  SB;                 /* short-term sliding average of (F0+F1) */
Word16  lambA;              /* forgetting factor (for long-term) */
Word16  lambB;              /* forgetting factor (for short-term) */
UWord16 demState;           /* state of demodulation process */
volatile UWord16 jj;        /* step [17 16 17] variable for proper
                               demfsk_NewFrmCounter computation */
UWord16 *pidx;              /* pointer to the subbit where the synchronization
                               pattern was detected */
UWord16 eachThird;          /* help counter variable since 1bit of message is
                               derived from 3 subsequent frames (subbits) */
UWord32 last24SubBits;      /* last 24 received subbits, LSB is the newest one
                               it is used for header pattern synchronization */
Word16 xBuf[XBUFLENGTH];    /* circular buffer of input samples from ADC, xBuf
                               base address must be multiple of 2^k, defined in
                               linker command file */
UWord16 bBuf[BBUFLENGTH];   /* circular buffer of decoded subbits from frames
                               calculations, bBuf base address must be multiple
                               of 2^k, defined in linker command file */
Word16 *pxBuf;              /* pointer to the xBuf buffer used when sample (read
```

Source Code Files

```
                                from ADC) is stored to xBuf */
UWord16 *pbBuf;                 /* pointer to the bBuf buffer */
Word16 *pK0Base;                /* base address of the e ^ (-j * Omega0 * n)
                                   coefficients */
Word16 *pK1Base;                /* base address of the e ^ (-j * Omega1 * n)
                                   coefficients */
Word16 *pInFrame;               /* pointer (reading data pointer for DTFT
                                   calculations) to the sample buffer xBuf
                                   (modulo in xBuf!!) */
Word16 *prevSample;             /* previous input sample */
UWord32 *pMSGBuf;               /* pointer to the MSGBuf buffer */

/*************************************************************************
* Module:      void demfskInit(void)
*
* Description: Initialization of FSK demodulation
*
* Returns: None
*
* Global Data:
*       SA - set initial value to long-term sliding average of (F0+F1)
*       SB - set initial value to short-term sliding average of (F0+F1)
*       lambA - set const. value to forgetting factor (for long-term average)
*       lambB - set const. value to forgetting factor (for short-term average)
*       *pbBuf - pointer is set to the bBuf (circular buffer of decoded subbits
*           from frames calculations)
*       *pxBuf - pointer (pointer for saving the ADC samples) is set to the xBuf
*           (circular buffer of samples read from ADC)
*       *pInFrame - pointer (reading data pointer for DTFT calculations) is
*           initially set near the xBuf (circular buffer of samples read from
*           ADC) buffer end
*       *pMSGBuf - pointer is set to the demfsk_MSGBuf (buffer of received
*           message of FSK demodulation routine)
*       demState - state of demodulation process is set to 0
*       last24SubBits - last 24 received subbits is cleared
*       jj - step [17 16 17] variable is cleared
*       demfsk_NewFrmCounter - a counter in ADCEndOfScanISR is cleared
*       *pK0Base - set base address of the e ^ (-j * Omega0 * n) coefficients
*       *pK1Base - set base address of the e ^ (-j * Omega1 * n) coefficients
*       K100[2 * DEMFSK_FRAMELEN] - array of FSK dem. coefficients for 100kHz
*       K105[2 * DEMFSK_FRAMELEN] - array of FSK dem. coefficients for 105kHz
*       K110[2 * DEMFSK_FRAMELEN] - array of FSK dem. coefficients for 110kHz
*       K115[2 * DEMFSK_FRAMELEN] - array of FSK dem. coefficients for 115kHz
*       K120[2 * DEMFSK_FRAMELEN] - array of FSK dem. coefficients for 120kHz
*       PL_CARRIERLOW and PL_CARRIERHGH are symbolic constants describing both
*           carrier frequencies
*       CARRIERLOW_110KHZ10KBPS, CARRIERLOW_115KHZ10KBPS,
*       CARRIERLOW_120KHZ10KBPS, CARRIERHGH_100KHZ10KBPS,
*       CARRIERHGH_105KHZ10KBPS and CARRIERHGH_110KHZ10KBPS are symbolic
*           constants describing each carrier frequency in term of number
*
```

```
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
void demfskInit(void)
{
    SA          = FRAC16(0.002);
    SB          = FRAC16(0.0004);
    lambA       = FRAC16(1.0 - 0.01);
    lambB       = FRAC16(1.0 - 0.2);
    pbBuf       = bBuf;
    pxBuf       = xBuf;
    pInFrame    = xBuf + XBUFLENGTH - 33;
    pMSGBuf     = demfsk_MSGBuf;
    demState    = 0;
    last24SubBits = 0;
    jj          = 1;
    demfsk_NewFrmCounter = 17;
    #if (PL_CARRIERLOW == CARRIERLOW_110KHZ10KBPS)  // set FSK Dem. frequencies
        pK0Base     = (Word16 *) K110;              // for F0 calculation
    #elif (PL_CARRIERLOW == CARRIERLOW_115KHZ10KBPS)
        pK0Base     = (Word16 *) K115;
    #elif (PL_CARRIERLOW == CARRIERLOW_120KHZ10KBPS)
        pK0Base     = (Word16 *) K120;
    #endif
    #if (PL_CARRIERHGH == CARRIERHGH_100KHZ10KBPS)  // set FSK Dem. frequencies
        pK1Base     = (Word16 *) K100;              // for F1 calculation
    #elif (PL_CARRIERHGH == CARRIERHGH_105KHZ10KBPS)
        pK1Base     = (Word16 *) K105;
    #elif (PL_CARRIERHGH == CARRIERHGH_110KHZ10KBPS)
        pK1Base     = (Word16 *) K110;
    #endif
}

/*******************************************************************************
* Module:      void demfskDem(void)
*
* Description: This function performs the FSK demodulation routine. It
*       processes one frame of data samples saved in xBuf and perform the DTFT
*       (Discrete time Fourier transformation) calculation over them.
*       It determines if samples contain the valid PL data or not. There are
*       four possible states of the PL reception stored in demState variable:
*           = 0 : waiting state until F0 or F1 frequency components is present
*           = 1 : F0 or F1 present, finding the synchronization pattern state
*           = 2 : synchronization pattern was found, data reception state
*           = 3 : whole message received and saved; demState value is cleared
*                 immediately to 0 so this state is not treated by switch
*                 condition.
```

```
*       The resulting message is stored to demfsk_MSGBuf variable.
*       Note that 1 bit value is calculated from 3 subsequent frames, so lets
*       call the demodulation result coming from one frame "subbit".
*
* Returns: None
*
* Global Data:
*       pl_Flags - flag pl_FlgModeOfModem - there are 5 possible modes of PL
*                       reception: (Note that these modes are controled by the
*                       demState variable described above)
*                       STATE7 - PL reception has been started
*                       STATE8 - PL reception in progress, FSK demodulation in
*                                Demstate 0 (waiting until F0 or F1 is present)
*                       STATE9 - PL reception in progress, FSK demodulation in
*                                Demstate 1 (finding synchronization pattern)
*                       STATE10- PL reception in progress, FSK demodulation in
*                                Demstate 2 (data reception)
*                       STATE11- PL reception in progress, FSK demodulation in
*                                Demstate 3 (data reception finished)
*                - flag pl_FlgDataError (modified by codePLtoSCI() routine) is
*                       checked and if set it is cleared
*       demfsk_NewFrmCounter - a counter in ADCEndOfScanISR set either to 16 or
*           17. This demfskDem() routine is called when this variable is equal
*           to 0 (the frame is ready for computation) since it is decremented
*           each time the ADCEndOfScanISR routine is performed.
*       jj - step [17 16 17] variable for proper demfsk_NewFrmCounter settings
*       *pK0Base - base address of the e ^ (-j * Omega0 * n) coefficients
*       *pK1Base - base address of the e ^ (-j * Omega1 * n) coefficients
*       last24SubBits - each calculated subbit is stored into this variable,
*           which is used for the header pattern synchronization
*       *pbBuf - each calculated subbit is stored into bBuf (circular buffer
*           of decoded subbits from frames calculations)
*       SA - updated long-term sliding average of (F0+F1) is stored here
*       SB - updated short-term sliding average of (F0+F1) is stored here
*       lambA - constant value of forgetting factor (for long-term average)
*       lambB - constant value of forgetting factor (for short-term average)
*       demState - state of demodulation process as described above
*       *pidx - pointer is set to the subbit where the synchronization pattern
*           was detected
*       demfsk_MSGBuf[DEMFSK_MSGBUFLEN] - is a buffer of received message of
*           the FSK demodulation routine
*       *pMSGBuf - pointer to the demfsk_MSGBuf (buffer of received message)
*       eachThird - help counter variable since 1bit of message is derived from
*           3 subsequent frames (subbits)
*       *pInFrame - data reading pointer in xBuf for DTFT calculations
*       For the global data description of the codePLtoSCI() routine see the
*       description of the routine itself
*       DEMFSK_SAMULTIPLE - a symbolic constant describing the multiple of SA
*           sliding average for comparison if (SB < 2^(DEMFSK_SAMULTIPLE) * SA)
*       BBUFLENGTH - a symbolic constant describing the length of the bBuf
*       DEMFSK_SYNCPATTERN is a symbolic constant describing the synchronization
```

```
*           pattern
*       FRAME_TOTALLEN - a symbolic constant describing the total length [in B]
*           of transmitted interleaved packet. The numBitsMSGBufDWord variable
*          (counter of bits in one 32 bit long word) uses this constant
*       FRAME_TOTALBITS - a symbolic constant describing the data length [in
*           bits] of transmitted packet. The numBitsReceived variable (counter
*           of received bits of message) is compared with this value and if it
*           is equal, the reception is finished
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
void demfskDem(void)
{

    volatile UWord16 tempBit111; // bit value result calculated from 3
                                 // subsequent frames (subbits)
    static UWord16 minSync;
        // minimal number of errors in comparison with Header Sync pattern
    static UWord16 demCount;
        // used as a counter of frames as soon as F0 or F1 appeared
    static UWord16 numBitsReceived;     // counter of received bits of message
    static UWord16 numBitsMSGBufDWord;
        // counter of bits in one 32 bit long word of the MSGBuff since the
        // number of bits saved in 1 Dword depends on the data length [in B] of
        // transmitted interleaved packet

    Word16 f0, f1;
        // f0 determines the level of frequency 0 component in sampled signal
        // f1 determines the level of frequency 1 component in sampled signal
    UWord16 actualSync;
        // actual number of errors in comparison with the Header Sync pattern

    demfsk_NewFrmCounter = 16;  // set the condition for next FSK Dem calling
    if (jj & 1)                 // it is called in the way [17, 16, 17]
    {
        demfsk_NewFrmCounter++;
    }

    /* f0 and f1 calculation, saturation mode must be off */
    f0 = calcDTFT(pK0Base);
    f1 = calcDTFT(pK1Base);
    /* f0, f1 comparison and store decoded bit to bBuf buffer */
    last24SubBits <<= 1;
    last24SubBits &= 0x00FFFFFF;
    if (f1 > f0)
    {
```

Source Code Files

```
        *pbBuf = 1;             // save subbit value to bBuf
        last24SubBits |= 1;     // save subbit value to last24SubBits
    }
    else
    {
        *pbBuf = 0;
    }
    pbBuf++;                    // modulo addressed buffer
    if (pbBuf >= bBuf + BBUFLENGTH)     // modulo addressing
        pbBuf = bBuf;

    asm {               /* calculation f0 += f1; */
        move    f1,a    /* saturation mode must be off */
        add     f0,a
        asr     a
        rnd     a
        move    a,f0
    }

    slidAverage(&SB, lambB, f0);        // short-term sliding average calcul
    if (((SB >> DEMFSK_SAMULTIPLE) < SA) && (demState != 2))
                                        // if (SB < 2^(DEMFSK_SAMULTIPLE) * SA)
    {
        slidAverage(&SA, lambA, f0);    // long-term sliding average calcul
    }

    switch (demState)
    {
        case 0: /* wait state until f0 or f1 is present */
        {
            if ((SB >> DEMFSK_SAMULTIPLE) > SA)
                                        // if (SB > 2^(DEMFSK_SAMULTIPLE) * SA)
            {
                demState = 1;
                minSync = 24;
                demCount = 0;
            }
            break;
        }
        case 1: /* f0 or f1 is present, find synchronization pattern state */
        {
            demCount++;
            actualSync = numOnes(last24SubBits ^ DEMFSK_SYNCPATTERN);
                        // correlation with the synchronization header pattern
            if (actualSync < minSync)
            {
                minSync = actualSync;
                pidx = pbBuf;
            }
                                                        // low f0 or f1 level
            if ((SB >> DEMFSK_SAMULTIPLE) < SA)
```

```
                                    // if (SB < 2^(DEMFSK_SAMULTIPLE) * SA)
        {
            demState = 0;
        }
        if (demCount == 72)     // stop looking for the Sync header pattern
        {
            if (minSync < 8)    // the Sync header pattern was found
            {
                pMSGBuf = demfsk_MSGBuf;
                demState = 2;
                eachThird = 1;
                demCount = 0;
                numBitsReceived = 0;
                numBitsMSGBufDWord = FRAME_TOTALLEN;    // set counter
                (*pMSGBuf) = 0; // clears the MSGBuf
            }
            else                // the Sync header pattern was not found
            {
                demState = 0;   // start from the demodulation state 0 again
            }
        }
        break;
    }   // end case 1

    case 2: /* synchronization pattern was found, data reception state */
    {
        if (eachThird == 3)     // 3 frames (subbits) carry just 1 bit info
        {
            numBitsReceived++;
            eachThird = 1;

            asm { /* calculate tempBit111 = *(pidx) + *(pidx+1) + *(pidx+2);
                             pidx += 3; */
                move    m01,r1          /* store value of m01 */
                move    pidx,r0         /* r0 - address of actual bit */
                move    #BBUFLENGTH-1,m01   /* r0-modulo addressing in bBuf,
                                    bBuf address must be multiple of 2^k */
                nop
                move    x:(r0)+,a       /* a - *pidx */
                move    x:(r0)+,x0      /* x0 - *(pidx + 1) */
                add     x0,a    x:(r0)+,x0
                add     x0,a            /* a - *pidx+*(pidx+1)+*(pidx+2) */
                move    r0,pidx         /* store updated pidx pointer */
                move    a,tempBit111    /* store result */
                move    r1,m01          /* restore value of m01 */
                nop                     /* due to pipelining */
            }

            (*pMSGBuf) <<= 1;       // MSG result buffer
            if (tempBit111 > 1)
            {
```

Source Code Files

```
                    (*pMSGBuf) |=  1;   // write a bit value into Dword MSG
                    tmrfskRxDLEDOn();   // RxD LED control
                }
                else
                    tmrfskRxDLEDOff();  // RxD LED control

                numBitsMSGBufDWord--;
                if (!numBitsMSGBufDWord) // Dword of MSGBuf is full (received)
                {
                    numBitsMSGBufDWord = FRAME_TOTALLEN;    // set counter
                    pMSGBuf++;          // next Dword of MSGBuf is empty
                    (*pMSGBuf) = 0;     // clears the MSGBuf
                }
            }
            else
            {
                eachThird += 1;
            }

            if (numBitsReceived == FRAME_TOTALBITS) // whole message received
            {
                demfskStopADCRxFromPL();    // stop PL data sampling
                                            // (stop Rx from the PL side)
                demState = 3;               // whole message received
            }
            if (((SB >> DEMFSK_SAMULTIPLE) < SA) &&
                (numBitsReceived < FRAME_TOTALBITS / 4))
            {           // if noise detected (F0 or F1 is not present now)
                demCount++;
                if (demCount > 5)
                {       // if noise detected (F0 or F1 was not 5 times present)
                    demState = 0;
                }
            }
            break;
        }  // end case 2
    }   // end switch

    asm {   /* Update pointer pInFrame [17 16 17] using modulo arithmetic */
        move    m01,r1              /* store value of m01 */
        move    pInFrame,r0         /* r0 - address of input frame */
        move    #16,n
        move    #XBUFLENGTH-1,m01   /* r0 - modulo addressing in xBuf, xBuf */
                                    /* address must be multiple of 2^k */
        bftstl  #0x1,jj             /* if (jj & 1)      */
        bcs     __NO17              /* {                */
        lea     (r0)+               /*     pInFrame++;  */
                                    /* }                */
__NO17:
        lea     (r0)+n              /*  pInFrame += 16; */
        move    r0,pInFrame         /* store updated pInFrame pointer */
```

```
        move    r1,m01              /* restore value of m01 */
        nop                         /* due to pipelining */
    }

    jj++;
    if (jj & 4)         // jj is from the range < 1,3 >
    {
        jj = 1;
    }

    pl_FlgModeOfModem = demState + 8;   // set Mode of modem flag
    if (demState == 3)
    {                                   // whole message received
        demState = 0;
        pl_FlgModeOfModem = STATE11;    // set Mode of modem flag

        codePLtoSCI();                  // prepare data from PL to SCI
        if (pl_FlgDataError == 0)       // check the data consistency
        {
            pl_FlgModeOfModem = STATE12;// set Mode of Modem
            ioctl(SCI_0, SCI_TX_EMPTY_INT, SCI_ENABLE); // enable SCI Tx IRQ
        }
        else                            // Bad data consistency
        {
            pl_FlgDataError = 0;        // clear flag
            pl_FlgModeOfModem = STATE6; // set Mode of Modem
            demfskInit();               // reinitialise FSK demodulator
            demfskStartADCRxFromPL();   // start PL data sampling
                                        // (start Rx from the PL side)
            pl_FlgModeOfModem = STATE7; // set Mode of Modem
        }
    }

    if (demState == 0)                  // PL reception LED signalization
    {
        tmrfskCDLEDOff();               // CD LED control
        tmrfskRxDLEDOff();              // RxD LED control
    }
    else
        tmrfskCDLEDOn();                // CD LED control
}

/*****************************************************************************
* Module:      demfskEndOfScanISR()
*
* Description: ADC A End of Scan ISR (Interrupt Service Routine)
*              First version:
*              - reads sample from ADC A and stores it in circular xBuf buffer
*              Second version:
*              - adds highpass filter saved value y(n) equal to actual sample
*                x(n) minus the previous sample x(n-1)
```

Source Code Files

```
*
* Returns: None
*
* Global Data:
*       addrPXBUF - symbolic constant, address of pxBuf pointer
*       addrNEWFRMCOUNTER - symbolic constant, address of the
*           demfsk_NewFrmCounter variable
*       addrPREVSAMPLE - symbolic constant, address of prevSample variable
*       XBUFLENGTH  - symbolic constant, length of xBuf (circular buffer of
*           input samples read from ADC)
*
* Arguments:   None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
asm void demfskEndOfScanISR(void)
{
#if 1 /* version without highpass filter */
    lea     (sp)+               /* saving registers */
    move    y0,x:(sp)+          /* saving registers */
    move    r0,x:(sp)+          /* saving registers */
    move    m01,x:(sp)          /* saving registers */

    move    x:<addrPXBUF,r0     /* r0 - pointer to xBuf buffer, "moves pxBuf,r0" */
    move    #XBUFLENGTH-1,m01   /* modulo addressing */
    move    x:0x0e89,y0         /* y0 - input sample, &ArchIO.AdcA.ResultReg[0] */
    move    y0,x:(r0)+          /* input sample to xBuf buffer */
    move    r0,x:<addrPXBUF     /* store pointer to xBuf, "move r0,pxBuf" */

    decw    x:<addrNEWFRMCOUNTER    /* decw demfsk_NewFrmCounter */
                                    /* a counter in ADCEndOfScanISR */

    bfset   #0x800,x:0x0e86     /* clear EOSI flag, &ArchIO.AdcA.Control1Reg */

    move    x:(sp)-,m01         /* restoring registers */
    move    x:(sp)-,r0          /* restoring registers */
    move    x:(sp)-,y0          /* restoring registers */
    rti
#else
    /* version with added highpass filter y(n) = x(n)-x(n-1) */
    lea     (sp)+               /* saving registers */
    move    y0,x:(sp)+          /* saving registers */
    move    r0,x:(sp)+          /* saving registers */
    move    m01,x:(sp)          /* saving registers */

    move    x:<addrPXBUF,r0     /* r0 - pointer to xBuf buffer,
                                   "moves pxBuf,r0" */
    move    #XBUFLENGTH-1,m01   /* modulo addressing */
```

```
    move    x:0x0e89,y0         /* y0 - input sample,
                                   &ArchIO.AdcA.ResultReg[0] */
    sub     x:<addrPREVSAMPLE,y0    /* y0 - actual sample - previous sample,
                                       "sub     prevSample,y0" */
    move    y0,x:(r0)+          /* input sample to xBuf buffer */
    move    r0,x:<addrPXBUF     /* store pointer to xBuf, "move r0,pxBuf" */
    move    x:0x0e89,y0         /* y0 - input sample,
                                   &ArchIO.AdcA.ResultReg[0] */
    move    y0,x:<addrPREVSAMPLE    /* store actual sample,
                                       "move y0,prevSample" */

    decw    x:<addrNEWFRMCOUNTER    /* decw demfsk_NewFrmCounter */
                                    /* a counter in ADCEndOfScanISR */

    bfset   #0x800,x:0x0e86     /* clear EOSI flag, &ArchIO.AdcA.Control1Reg */

    move    x:(sp)-,m01         /* restoring registers */
    move    x:(sp)-,r0          /* restoring registers */
    move    x:(sp)-,y0          /* restoring registers */
    rti
#endif
}

/*************************************************************************
* Module: asm Word16 calcDTFT(Word16 *pCoeff)
*
* Description: Calculation of DTFT coefficient Fi (i = 0,1):
*   abs(Fi)^2 =
*   = ( sum{Input(n)*CoeffReal(n)/(2^DEMFSK_FSCALE)}^2
*     + sum{Input(n)*CoeffImag(n)/(2^DEMFSK_FSCALE)}^2 ) / 2
*   for n = 0 to 49.
*
* Returns:
*       Function returns abs(Fi)^2
*
* Global Data:
*       *pInFrame - pointer (reading data pointer for DTFT calculations) to
*           the sample buffer xBuf (modulo in xBuf!!)
*       XBUFLENGTH  - symbolic constant, length of xBuf (circular buffer of
*           input samples read from ADC)
*       DEMFSK_FRAMELEN - symbolic constant, length of frame [in samples]
*       DEMFSK_FSCALE   - symbolic constant, scaling factor
*
* Arguments:
*       pCoeff - pointer to table of coefficients, coefficients must be in the
*           order: real0, imag0, real1, imag1...
*
* Range Issues: None
*
* Special Issues: saturation mode must be off
*
```

Source Code Files

```
* Others: - buffer of input samples should be in internal data RAM memory
*         - tables of coefficients should be in XFlash data memory area (56F801
*           source or FLASH target in 56F803 source) or in internal RAM data
*           memory area (RAM target in 56F803 source)
*************************************************************************/
asm Word16 calcDTFT(Word16 *pCoeff)
{
    move    m01,r1                  /* store value of m01 */
    move    pInFrame,r0             /* r0 - address of input samples */
    move    #XBUFLENGTH-1,m01       /* r0 - modulo addressing in xBuf,
                                       xBuf address must be multiple of 2^k */
    move    r2,r3                   /* r3 - address of coefficients */

    /* calculation of real and imag part of F0, no pipeline dependency */
    clr     a           x:(r0)+,y1  /* a - real part of F0, y1 - input sample */
    clr     b           x:(r3)+,x0  /* b - imag part of F0,
                                       x0 - real part of coeff. */
    do      #DEMFSK_FRAMELEN,__END
    mac     x0,y1,a                 x:(r3)+,x0  /* x0 - imag. part of coeff. */
    mac     x0,y1,b     x:(r0)+,y1  x:(r3)+,x0  /* y1 - input sample,
                                                   x0 - real part of coeff. */
__END:

    do      #DEMFSK_FSCALE,__END1   /* scaling */
    asr     a                       /* a - a / 2 ^ DEMFSK_FSCALE */
    asr     b                       /* b - b / 2 ^ DEMFSK_FSCALE */
__END1:

    rnd     a
    move    a,y0
    mpy     y0,y0,a         /* a - (real part of F0)^2 */
    rnd     b
    move    b,y0
    mac     y0,y0,a         /* a - (real part + imag part of F0)^2 */
    asr     a               /* a - a/2, scaling */
    rnd     a
    move    a,y0            /* the return value to y0 */
    move    r1,m01          /* restore value of m01 */
    rts
}

/*************************************************************************
* Module: asm void slidAverage(Word16 *Si, Word16 lambi, Word16 f)
*
* Description: Calculation of Si (i = A,B): Si = lambi*Si + (1-lambi)*f
*
* Returns: Result is returned in Si
*
* Global Data: None
*
* Arguments:
```

```
*       *Si - pointer to Si (SA or SB) sliding average
*       lambi - forgetting factor lambA or lambB
*       f - new value for sliding average calculation
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
asm void slidAverage(Word16 *Si, Word16 lambi, Word16 f)
{
                                    /* r2 - Si, y0 - lambi, y1 - f */
    move    x:(r2),x0               /* x0 - Si */
    mpy     y0,x0,a                 /* a  - lambi*Si */
    add     #-32768,y0              /* y0 - (-1 + lambi) */
    macr    -y1,y0,a
    move    a,x:(r2)                /* store result */
    rts
}

/*******************************************************************************
* Module: UWord16 numOnes(UWord32 tempVar)
*
* Description: Function returns the number of ones in lower 24 bits of parameter
*              tempVar.
*
* Returns: Number of ones in lower 24 bits of parameter tempVar
*
* Global Data: None
*
* Arguments:
*       tempVar - variable where to calculate
*
* Range Issues: none
*
* Special Issues: none
*******************************************************************************/
UWord16 numOnes(UWord32 tempVar)
{
    asm         /* A  -  tempVar, Y0 - result */
    {
        clr     b
        clr     y0
        clr     y1
        do      #24,__END
        asr     a
        adc     y,b
     __END:
        move    b0,y0
    }
}
```

C.8 demfsk.h

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: demfsk.h
*
* DESCRIPTION: Header file for the SW FSK demodulator
*
* MODULES INCLUDED: None
*
*****************************************************************************/
#ifndef _DEMFSK_H
#define _DEMFSK_H

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "pl.h"

/*****************************************************************************/
/* Defines for SW FSK Demodulation                                           */
/*****************************************************************************/
#define DEMFSK_MSGBUFLEN    14      // length of MSGBuf buffer
/* for the FRAME_TOTALLEN equal to 16,24 or 32 and PL_FECTYPE equal to PL_NOFEC
/*      the length should be >= 8 */
/* for the FRAME_TOTALLEN equal to 16,24 or 32 and PL_FECTYPE equal to PL_1STFEC
/*      the length should be >= 14 */

#define DEMFSK_FRAMELEN     50      // length of frame [in samples]
#define DEMFSK_SYNCPATTERN  0x00E381C7
    /* synchronization pattern is equal to 1110 0011 1000 0001 1100 0111 */

#define XBUFLENGTH      100         // length of xBuf buffer
#define BBUFLENGTH      100         // length of bBuf buffer
#define DEMFSK_FSCALE   4
    // both real and imag parts of F0 and F1 are scalled down by 2^DEMFSK_FSCALE

#define DEMFSK_SAMULTIPLE    1  /* SA multiple for comparison
                                   (SB < 2 ^ (DEMFSK_SAMULTIPLE) * SA)*/


/*****************************************************************************/
/* Defines for memory mapping used in ADC End Of Scan ISR routine            */
/*****************************************************************************/
```

```
// Note that the sequence of these defines has to be identical with the sequence
// as defined in linker_ram and linker_flash command files
#define addrPREVSAMPLE              0x0
#define addrPXBUF                   0x1
#define addrNEWFRMCOUNTER           0x2

/*****************************************************************************/
/*              P R O T O T Y P E S                                          */
/*****************************************************************************/
void  demfskInit(void);
void  demfskDem(void);
asm void demfskEndOfScanISR(void);

/*****************************************************************************/
/*              M A C R O S                                                  */
/*****************************************************************************/
/* Timer C2 section */
#define demfskStartADCRxFromPL()        \
        ioctl(QTIMER_C2, QT_SET_COUNT_MODE, QT_COUNT_RISING_EDGES_MODE)
    // start the Tmr C2 which triggers the ADC conversion =>
    // it starts the data sampling for the SW FSK Demodulation routine
#define demfskStopADCRxFromPL()         \
        ioctl(QTIMER_C2, QT_SET_COUNT_MODE, QT_NO_OPERATION);    \
        ioctl(ADC_A, ADC_CLEAR_STATUS_EOSI, NULL)
    // stop the Tmr C2 which triggers the ADC conversion =>
    // it stops the data sampling for the SW FSK Demodulation routine
    // and clears the ADC End of Scan Interrupt flag as well

#endif
```

C.9 coderoutines.c

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: coderoutines.c
*
* DESCRIPTION: This file contains the routines of data coding / decoding,
*       specifically: FEC code / decode, CRC computation
*
* MODULES INCLUDED:
*       codeCRCCalc()
*       codeSCItoPL()
*       codeMoveAndFECBuff()
```

Source Code Files

```
*       codePLtoSCI()
*       deintrleave()
*
*****************************************************************************/

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "types.h"
#include "arch.h"
#include "periph.h"
#include "sci.h"

#include "pl.h"
#include "tea.h"
#include "coderoutines.h"
#include "demfsk.h"

/*****************************************************************************/
/*              GLOBAL VARIABLES                                             */
/*****************************************************************************/
    extern pl_uRxFromSCI       pl_RxFromSCI;
        /* Buffer dedicated for SCI reception */
    extern pl_uRxFromSCI       pl_TxToPL;
        /* Buffer dedicated for Power Line transmission */
    extern pl_uRxFromPL        pl_RxFromPL;
        /* Buffer dedicated for Power Line reception */
    extern pl_uTxToSCI         pl_TxToSCI;
        /* Buffer dedicated for SCI transmission */

    extern volatile pl_sFlags  pl_Flags;    /* contains the state and another
                                               flags of PL modem device */

/* extern of SW FSK Demodulation variable taken from the demfsk.c file */
    extern UWord32 demfsk_MSGBuf[DEMFSK_MSGBUFLEN];
        /* buffer of received message of FSK demodulation routine */

/*****************************************************************************/
/*                  LOOK-UP TABLE GLOBAL VARIABLES                           */
/*****************************************************************************/
    extern const UWord16 CRCtable[256];         // table of the 16bit CRC codes
    extern const UWord16 FECtableCoder[16];
    /* table of the linear block Forward error correction - coding part */
    extern const UWord16 FECtableDecoder[128];
    /* table of the linear block Forward error correction - decoding part */

/*****************************************************************************/
/*              P R O T O T Y P E S                                          */
/*****************************************************************************/
    Word16 codeCRCCalc(UWord16 *Buffer, UWord16 n);
    void codeMoveAndFECBuff(UWord16 fec_Mode, UWord16 Length);
```

```
    asm void deintrleave(UWord32 *pInput, UWord16 *pOutput,
                         UWord16 numRows, UWord16 numColumns);
/*****************************************************************************
*
* Module: Word16 codeCRCCalc(UWord16 *buffer, UWord16 n)
*
* Description:
*       This function generates the 16 bit CRC      16    15   2
*       using the following polynom:               X   + X  + X  + 1
*
* Returns: calculated CRC value
*
* Global Data:
*       CRCtable[256] - look-up table for 16 bit CRC computation
*
* Arguments:
*       *buffer - pointer to buffer to be calculated
*       n - length of the buffer
*
* Range Issues: None
*
* Special Issues: None
*
* Others: - look-up table called CRCTable should be located in XFlash data
*           memory area (56F801 source or FLASH target in 56F803 source) or in
*           internal or external RAM data memory area (RAM target in 56F803
*           source)
*
*****************************************************************************/
Word16 codeCRCCalc(UWord16 *buffer, UWord16 n)
{
    Word16 crc = 0;

    while (n--)
        crc = ((crc >> 8) & 0xff) ^ CRCtable[(crc ^ *buffer++) & 0xff];
    return crc;
}

/*****************************************************************************
*
* Module: void codeSCItoPL(void)
*
* Description:
*       This routine completely prepare the data from the SCI Rx buffer side to
*       PL Tx side: it clears the rest of data bytes of pl_RxFromSCI buffer,
*       calculate the CRC of the frame, call the TEA encyption algorith and
*       finally call the routine that moves the pl_RxFromSCI buffer to
*       pl_TxToPL buffer and perform FEC during this move.
*
* Returns: None
*
```

Source Code Files

```
* Global Data:
*       pl_RxFromSCI - a buffer of SCI received data
*       pl_TxToPL - a buffer of data prepared for PL transmission
*       FRAME_DATALEN - a symbolic constant describing the data length [in B]
*           of packet.
*       For the global data description of the codeCRCCalc() routine see the
*       description of the routine itself
*       PL_TEACRYPT - a symbolic constant, if defined the Tiny Encryption is
*           processed over the packets
*       For the global data description of the teaEncryptBuff() routine see the
*       description of the routine itself
*       FRAME_TOTALLEN - a symbolic constant describing the total length of
*           the whole packet [in B] to be sent
*       For the global data description of the codeMoveAndFECBuff() routine see
*       the description of the routine itself
*       PL_FECTYPE - a symbolic constant describing the type of used FEC
*           correction
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
void codeSCItoPL(void)
{
    UWord16 temp;

    for (temp = pl_RxFromSCI.Struct.Cntrl; temp < FRAME_DATALEN; temp++)
            pl_RxFromSCI.Struct.Data[temp] = 0; // clear the rest of the data

    // CRC calculation  (1B of CNTRL + Data)
    temp = codeCRCCalc(&pl_RxFromSCI.Struct.Cntrl,
                        pl_RxFromSCI.Struct.Cntrl + 1);
        // CRC is calculated only over the really used data
    pl_RxFromSCI.Struct.CRC[0] = (UWord16) 0x00FF & temp;       // low CRC byte
    pl_RxFromSCI.Struct.CRC[1] = (UWord16) (0xFF00 & temp) >> 8;// high CRC byte

    #ifdef PL_TEACRYPT      // ENCRYPT
        teaEncryptBuff(&pl_RxFromSCI.Struct.Cntrl, FRAME_TOTALLEN);
    #endif

    codeMoveAndFECBuff(PL_FECTYPE, FRAME_TOTALLEN);
    // Move & perform FEC (if required) from pl_RxFromSCI to pl_TxToPL buffer
}

/*******************************************************************************
*
* Module:  void codeMoveAndFECBuff(UWord16 fec_Mode, UWord16 length)
*
```

```
* Description:
*       This routine moves the pl_RxFromSCI to pl_TxToPL buffer and also
*       perform the FEC coding (if required).
*
* Returns: None
*
* Global Data:
*       pl_RxFromSCI - a buffer of SCI received data
*       pl_TxToPL - a buffer of data prepared for PL transmission
*       FECtableCoder - table of the coding part of the linear block Forward
*           error correction
*       FRAME_PRELEN - a symbolic constant describing the length [in B]
*           pre-control [header] part of packet
*       PL_NOFEC and PL_1STFEC - symbolic constants defining the type of used
*           FEC coding
*
* Arguments: FEC_Mode:
*               - either PL_NOFEC [no FEC correction]
*               - or PL_1STFEC - [FIRST version of FEC correction]
*            length - an actual length of the pl_RxFromSCI / pl_TxToPL buffer
*
* Range Issues: None
*
* Special Issues: None
*
* Others: - look-up table called FECtableCoder could be located in XFlash data
*           memory area (56F801 source or FLASH target in 56F803 source) or in
*           internal or external RAM data memory area (RAM target in 56F803
*           source)
*******************************************************************************/
void codeMoveAndFECBuff(UWord16 fec_Mode, UWord16 length)
{
    UWord16 i, temp;

    for (i = 0; i < FRAME_PRELEN; i++)      // transfer the header part of frame
        pl_TxToPL.Array.Byte[i] = pl_RxFromSCI.Array.Byte[i];

    if (fec_Mode == PL_NOFEC)   // 8 data bits represents ONE byte
    {                           // transfer packet
        for (i = FRAME_PRELEN; i < length + FRAME_PRELEN; i++)
            pl_TxToPL.Array.Byte[i] = pl_RxFromSCI.Array.Byte[i];
    }
    else
    if (fec_Mode == PL_1STFEC)  // 14 coded bits represents ONE byte
    {                           // transfer & FEC packet
        for (i = FRAME_PRELEN; i < length + FRAME_PRELEN; i++)
        {
            temp = ((pl_RxFromSCI.Array.Byte[i] >> 4) & 0x0F);
            temp = FECtableCoder[temp] << 7;
            temp += FECtableCoder[(pl_RxFromSCI.Array.Byte[i] & 0x0F)];
            pl_TxToPL.Array.Byte[i] = temp;
```

```
        }
    }
}

/*****************************************************************************
*
* Module: void codePLtoSCI(void)
*
* Description:
*       This routine completely transfer the data from the PL Rx buffer side to
*       SCI Tx side: it call the de-interleaving routine, align the PL Rx
*       buffer, if FEC is used then perform the de-FEC algorithm, check the
*       consistency of the received packet (using the length and CRC values)
*       and finally move the pl_RxFromPL to the pl_TxToSCI buffer.
*
*   Returns: None
*
*   Global Data:
*       For the global data description of the deintrleave() routine see the
*       description of the routine itself
*       demfsk_MSGBuf - buffer of received message of FSK demodulation routine
*       pl_RxFromPL is a buffer of received data from the PL side
*       FECtableDecoder - table of the decoding part of the linear block Forward
*           error correction
*       For the global data description of the teaDecryptBuff() routine see the
*       description of the routine itself
*       pl_Flags - flag pl_FlgDataError is set if either CRC or length error
*           occured
*       FRAME_TOTALLEN - a symbolic constant describing the total length of
*           the whole packet [in B] to be sent
*       PL_FECTYPE - a symbolic constant describing the type of used FEC
*           correction
*       PL_NOFEC and PL_1STFEC - symbolic constants defining the type of used
*           FEC coding
*       PL_TEACRYPT - a symbolic constant, if defined the Tiny Encryption is
*           processed over the packets
*       FRAME_DATALEN - a symbolic constant describing the data length [in B]
*           of packet.
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
* Others: - look-up table called FECtableDecoder could be located in XFlash data
*           memory area (56F801 source or FLASH target in 56F803 source) or in
*           internal or external RAM data memory area (RAM target in 56F803
*           source)
*****************************************************************************/
void codePLtoSCI(void)
```

```
{
    UWord16 i, temp;

    deintrleave(demfsk_MSGBuf, pl_RxFromPL.Array.Byte, FRAME_TOTALLEN,
                PL_FECTYPE);
        // do the complete deinterleaving of the demfsk_MSGBuf
    for (i = 0; i < FRAME_TOTALLEN; i++)    // align the RxFromPl buffer
    {
        #if (PL_FECTYPE == PL_1STFEC)       // if 1st method of FEC coding
            pl_RxFromPL.Array.Byte[i] >>= 2;
        #elif (PL_FECTYPE == PL_NOFEC)      // if NO FEC coding
            pl_RxFromPL.Array.Byte[i] >>= 8;
        #endif
    }

    // if "NO FEC" correction type is set, do nothing because Data in a buffer
    // are in a non-coded form
    #if (PL_FECTYPE == PL_1STFEC)       // if 1st method of FEC coding chosen
    for (i = 0; i < FRAME_TOTALLEN; i++)    // FEC Decoder
    {
        temp = FECtableDecoder[(pl_RxFromPL.Array.Byte[i] & 0x3F80) >> 7];
        pl_RxFromPL.Array.Byte[i] = (temp << 4) +
                    FECtableDecoder[(pl_RxFromPL.Array.Byte[i] & 0x007F)];
    }
    #endif
    #ifdef PL_TEACRYPT      // decrypt the Rx buffer
        teaDecryptBuff(pl_RxFromPL.Array.Byte, FRAME_TOTALLEN);
    #endif

    if ((pl_RxFromPL.Struct.Cntrl >= 1) &&          // if length is OK
        (pl_RxFromPL.Struct.Cntrl <= FRAME_DATALEN))
    {
        temp = (pl_RxFromPL.Struct.CRC[1] << 8) + pl_RxFromPL.Struct.CRC[0];
        // original CRC value
        if (codeCRCCalc(pl_RxFromPL.Array.Byte,pl_RxFromPL.Struct.Cntrl+1) ==
                                                                        temp)
        // if calculated CRC value is equal to the original, the packet is OK
        {
            // store only CNTRL and DATA
            for (i = 0; i < pl_RxFromPL.Struct.Cntrl + 1; i++)
                pl_TxToSCI.Array.Byte[i] = pl_RxFromPL.Array.Byte[i];
        }
        else
        {
            pl_FlgDataError = 1; // set bad data consistency (Data Error) flag
        }                        // bad CRC
    }
    else
    {
        pl_FlgDataError = 1;     // set bad data consistency (Data Error) flag
    }                            // bad length
```

Source Code Files

```
}

/*************************************************************************
* Module:      asm void deintrleave(UWord32 *pInput, UWord16 *pOutput,
*                    UWord16 numRows, UWord16 numColumns)
*
* Description: Function performes de-intearleaving of input data pInput
*              and returns de-interleaved data in buffer pOutput.
*              The 14 / 8 data bits (with FEC or without FEC) are located in
*              upper 14 / 8 bits of words in pOutput buffer.
*
* Returns: None
*
* Global Data: None
*
* Arguments:
*   Inputs: pInput - pointer to input buffer
*           pOutput - pointer to output buffer
*           numRows - 16, 24 or 32 (number of transfered bytes in one packet)
*           numColumns - 14 or 8 (with FEC or without FEC)
*
* Range Issues: None
*
* Special Issues: None
*
* Others: routine doesn't require linear addressing (i.e. m01 to be $FFFF)
*************************************************************************/
asm void deintrleave(UWord32 *pInput, UWord16 *pOutput,
                     UWord16 numRows, UWord16 numColumns)
{   /* r2 - pInput, r3 - pOutput, y0 - numRows, y1 - numColumns */
    move    y0,n            /* calculate end address of Output buffer */
    nop                     /* pipeline dependency */
    lea     (r3)+n
    lea     (r3)-
    move    r3,r0           /* r0 - pOutput+numRows, r0 and r3 changed because
                               r3 uses always linear addressing and enables
                               "index by short displacement" addressing mode */
__OUTER:
    move    r0,r3           /* r3 - pointer to output buffer */
    move    x:(r2+1),a
    move    x:(r2)+,a0
    lea     (r2)+
    do      y0,__INNER
    move    x:(r3),x0
    asr     a               /* c - LSB */
    ror     x0              /* x0 - shift right */
    move    x0,x:(r3)-
__INNER:
    decw    y1              /* decrement outer loop count */
    bgt     __OUTER         /* branch to top of OUTER loop if not done */
    rts
```

```
}
```

C.10 coderoutines.h

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: coderoutines.h
*
* DESCRIPTION: A header file for coderoutines.c
*
* MODULES INCLUDED: None
*
*****************************************************************************/
#ifndef _CODEROUTINES_H
#define _CODEROUTINES_H

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "types.h"

/*****************************************************************************/
/*              P R O T O T Y P E S                                          */
/*****************************************************************************/
    void codeSCItoPL();         /* prepare data from SCI to PL */
    void codePLtoSCI();         /* prepare data from PL to SCI */

#endif
```

C.11 scicomm.c

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: scicomm.c
*
```

Source Code Files

```
* DESCRIPTION: This file consists of all SCI-based routines needed for the
*       PL modem (such as SCI Rx Full ISR and SCI Tx Empty ISR)
*
* MODULES INCLUDED:
*       scicommRxErrISR()
*       scicommRxFullISR()
*       scicommTxEmpISR()
*
*****************************************************************************/

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "types.h"
#include "arch.h"
#include "periph.h"
#include "appconfig.h"

#include "sci.h"
#include "gpio.h"

#include "scicomm.h"
#include "tmrfsk.h"
#include "demfsk.h"
#include "coderoutines.h"

#include "pl.h"

/*****************************************************************************/
/*              GLOBAL VARIABLES                                             */
/*****************************************************************************/
    extern pl_uRxFromSCI       pl_RxFromSCI;
        /* Buffer dedicated for SCI reception */
    extern pl_uTxToSCI         pl_TxToSCI;
        /* Buffer dedicated for SCI transmission */
    extern volatile pl_sFlags  pl_Flags;    /* contains the state and another
                                               flags of PL modem device */

/*****************************************************************************
*
* Module: void scicommRxErrISR(void)
*
* Description:
*       This function is the ISR of the SCI Receiver error. It clears the Noise
*       flag (NF), Parity error flag (PF), Framing error flag (FE) and Overrun
*       flag (OR) whenever it is needed.
*
* Returns: None
*
* Global Data: None
*
```

```
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
#pragma interrupt
void scicommRxErrISR(void)
{
    UWord16 temp;

    temp = ioctl(SCI_0, SCI_GET_STATUS_REG, NULL);
    ioctl(SCI_0, SCI_CLEAR_STATUS_REG, NULL);
}

/*******************************************************************************
*
* Module: void scicommRxFullISR(void)
*
* Description:
*       This function is the ISR of the SCI Rx Full. It reads the new data, save
*       them to the pl_RxFromSCI buffer. There are two way how this SCI Receive
*       could be finished, either it fills up the whole pl_RxFromSCI buffer or
*       communication Timeout (generated by Timeout Tmr TmrD3) occurs.
*
* Returns: None
*
* Global Data:
*       pl_Flags - flag pl_FlgModeOfModem - if the Mode was set to STATE1 -
*                       "SCI reception could be started", the SCI Rx is started
*                       and mode is switched to STATE2 "SCI reception in
*                       progress".
*                       When SCI Rx is finished the mode is set to STATE3 "SCI
*                       reception has been finished". The codeSCItoPL() prepares
*                       the data for PL Tx packet and mode is set to STATE4 "PL
*                       transmission could be started"
*       pl_RxFromSCI - a buffer of SCI reception
*       For the global data description of the codeSCItoPL() routine see the
*       description of the routine itself
*       FRAME_HEADER - a symbolic constant descriing the header value of packet
*           over the transmission is NOT performed
*
* Arguments: None
*
* Range Issues: Only when pl_FlgModeOfModem is equal to STATE1 or STATE2, the
*           SCI reception is performed
*
* Special Issues: None
*
*******************************************************************************/
```

Source Code Files

```
#pragma interrupt
void scicommRxFullISR(void)
{
    static UWord16 index;
    UWord16 temp;

    archPushAllRegisters();
    tmrfskClearTimeOutTmr();            // reset the time-out timer
    if ( pl_FlgModeOfModem == STATE1)   // test Mode of Modem condition
    {
        pl_FlgModeOfModem = STATE2;     // set Mode of Modem
        index = 0;
        pl_RxFromSCI.Struct.Cntrl = 0;  // clear the "Control" of the frame
        pl_RxFromSCI.Struct.Header[0] = FRAME_HEADER;   // set header
        tmrfskStartTimeOutTmr();        // start the time-out timer
    }

    if ( pl_FlgModeOfModem == STATE2)   // test Mode of Modem condition
    {
        temp = ioctl(SCI_0, SCI_GET_STATUS_REG, NULL);
        pl_RxFromSCI.Struct.Data[index] = ioctl(SCI_0, SCI_READ_DATA, NULL);
        index++;
        pl_RxFromSCI.Struct.Cntrl = index;  // set Length
        if (index == FRAME_DATALEN)     // the RxFromSCI buffer is full
        {
            tmrfskStopTimeOutTmr();     // stop the time-out timer
            pl_FlgModeOfModem = STATE3; // set Mode of Modem

            ioctl(SCI_0, SCI_RX_FULL_INT, SCI_DISABLE); // disable interrupt
            ioctl(SCI_0, SCI_RX_ERROR_INT, SCI_DISABLE); // disable interrupt

            codeSCItoPL();                  // prepare data from SCI to PL
            pl_FlgModeOfModem = STATE4;     // set Mode of Modem
            tmrfskSetTxEnable();            // switch on the transmitter
            tmrfskStartCarrierTmr();        // start generation of FSK carrier
            archDelay(0x1FFF);              // Tx of the carrier before the
            archDelay(0x1FFF);              // header and data part transmission
            archDelay(0x1FFF);              // total 0.8ms
            archDelay(0x1FFF);
            tmrfskStartBitTmr();            // start FSK transmission
        }
    }
    archPopAllRegisters();
}

/*****************************************************************************
*
* Module: void scicommTxEmpISR(void)
*
* Description:
*       This function is the ISR of the SCI Transmitter empty. It sends the
```

```
*       data part of packet. The length of data part is taken from the pl_TxToSCI
*       variable itself.
*
* Returns: None
*
* Global Data:
*       pl_Flags - flag pl_FlgModeOfModem - if the Mode was set to STATE12 -
*                       "SCI transmission could be started", the SCI Tx is
*                       started and mode is switched to STATE13 "SCI
*                       transmission in progress".
*                       When whole packet was transmitted the mode is set to
*                       STATE6 "PL / SCI transmission has been finished". Than
*                       the ADC data sampling is started and mode is set to
*                       STATE7 "PL reception has been started"
*       pl_TxToSCI - a buffer to be sent
*
* Arguments: None
*
* Range Issues: Only when pl_FlgModeOfModem is equal to STATE12 or STATE13, the
*           SCI transmission is performed

* Special Issues: None
*
*******************************************************************************/
#pragma interrupt
void scicommTxEmpISR(void)
{
    UWord16 temp;
    static UWord16 index;

    if (pl_FlgModeOfModem == STATE12)   // pl_TxToSCI is ready
    {
        pl_FlgModeOfModem = STATE13;    // set Mode of Modem
        index = 0;
    }
    if (pl_FlgModeOfModem == STATE13)   // pl_TxToSCI is being processed
    {
        temp = ioctl(SCI_0, SCI_GET_STATUS_REG, NULL);
                                        // clear Transmit Register Empty Flag
        ioctl(SCI_0, SCI_WRITE_DATA, pl_TxToSCI.Struct.Data[index]);  // SCI Tx
        index++;

        if (index >= pl_TxToSCI.Struct.Cntrl)   // whole packet was transmitted
        {
            ioctl(SCI_0, SCI_TX_EMPTY_INT, SCI_DISABLE); // disable SCI Tx IRQ
            pl_FlgModeOfModem = STATE6;     // set Mode of Modem
            demfskStartADCRxFromPL();       // start PL data sampling
                                            // (PL reception)
            pl_FlgModeOfModem = STATE7;     // set Mode of Modem
        }
    }
```

```
}
```

C.12 scicomm.h

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: scicomm.h
*
* DESCRIPTION: A header file for scicomm.c
*
* MODULES INCLUDED: None
*
*****************************************************************************/
#ifndef _SCICOMM_H
#define _SCICOMM_H

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "types.h"

/*****************************************************************************/
/*              P R O T O T Y P E S                                          */
/*****************************************************************************/
void scicommTxEmpISR(void);
void scicommRxErrISR(void);
void scicommRxFullISR(void);

#endif
```

C.13 tea.c

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: tea.c
```

```
*
* DESCRIPTION: This file contains the complete Tiny Encryption Algorithm (TEA)
*              implementation
*
* MODULES INCLUDED:
*       teaCode()
*       teaDecode()
*       teaEncryptBuff()
*       teaDecryptBuff()
*
*****************************************************************************/

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "pl.h"
#include "tea.h"
#include "types.h"

/*****************************************************************************/
/*              P R O T O T Y P E S                                          */
/*****************************************************************************/
    void teaCode(void);
    void teaDecode(void);

/*****************************************************************************/
/*              GLOBAL VARIABLES                                             */
/*****************************************************************************/
    extern const tea_uKey pl_TeaKey;
        /* Key for TEA (Tiny Encryption Algorithm) computation */

/*****************************************************************************/
/*              LOCAL VARIABLES OF THE FILE                                  */
/*****************************************************************************/
    tea_uIO tea_IO;     // own buffer for TEA computation, 64bits long

/*****************************************************************************/
/* TEA (The Tiny Encryption Algorithm) authors description                   */
/*****************************************************************************/
/*
The Tiny Encryption Algorithm (TEA) by David Wheeler and Roger Needham
of the Cambridge Computer Laboratory.

Placed in the Public Domain by David Wheeler and Roger Needham.
ftp://ftp.cl.cam.ac.uk/papers/djw-rmn/djw-rmn-tea.html

**** ANSI C VERSION (New Variant) ****

Authors notes:

TEA is a Feistel cipher with XOR and and addition as the non-linear
```

```
mixing functions.

Takes 64 bits of data in tea_IO (dw[0] and dw[1]) and returns the 64 bits long
result again into tea_IO. Takes 128 bits of key in k[0] - k[3].

TEA can be operated in any of the modes of DES. Cipher Block
Chaining is, for example, simple to implement.

n is the number of iterations. 32 is ample, 16 is sufficient, as few
as eight may be OK. The algorithm achieves good dispersion after six
iterations. The iteration count can be made variable if required.

Note this is optimised for 32-bit CPUs with fast shift capabilities.
It can very easily be ported to assembly language on most CPUs.

teaDelta is chosen to be the real part of the golden ratio
Sqrt(5/4) - 1/2 ~ 0.618034 multiplied by 2^32.

This version has been amended to foil two weaknesses identified by
David A. Wagner (daw@cs.berkeley.edu): 1) effective key length of
old-variant TEA was 126 not 128 bits 2) a related key attack was possible
although impractical.
*/

/*****************************************************************************/
/*              N O T E S                                                    */
/*****************************************************************************/
/*
The Tiny Encryption Algorithm operates over its own 64bit long buffer, so the
final buffer to be encrypted / decrypted has to be multiple of the 8B value!
*/

/*****************************************************************************
*
* Module: void teaCode(void)
*
* Description:
*       This function perform the TEA encryption over the its own 64-bit long
*        buffer
*
* Returns: None
*
* Global Data:
*       tea_IO - 64bits long buffer for TEA computation
*       y - a short-cut define for tea_IO.dw[0]
*       z - a short-cut define for tea_IO.dw[1]
*       pl_TeaKey - Key for TEA (Tiny Encryption Algorithm) computation
*       k[4] - a short-cut define for pl_TeaKey.dw[4]
*       TeaDelta - TEA constant chosen to be the real part of the golden ratio
*       Sqrt(5/4) - 1/2 ~ 0.618034 multiplied by 2^32
*
```

```
* Arguments: None
*
* Range Issues: Note that it performs the encryption over the 64 bits, so the
*               final buffer to be encrypted should be a multiple of this value!
*
* Special Issues: None
*
*******************************************************************************/
void teaCode(void)
{
    Word16 i = 32;
    UWord32 sum = 0;

    while(i-- > 0)
    {
        y += (z << 4 ^ z >> 5) + z ^ sum + k[sum & 3];
        sum += TeaDelta;
        z += (y << 4 ^ y >> 5) + y ^ sum + k[sum >> 11 & 3];
    }
}

/*******************************************************************************
*
* Module: void teaDecode(void)
*
* Description:
*       This function perform the TEA decryption over the own 64-bit long buffer
*
* Returns: None
*
* Global Data:
*       tea_IO - 64bits long buffer for TEA computation
*       y - a short-cut define for tea_IO.dw[0]
*       z - a short-cut define for tea_IO.dw[1]
*       pl_TeaKey - Key for TEA (Tiny Encryption Algorithm) computation
*       k[4] - a short-cut define for pl_TeaKey.dw[4]
*       TeaDelta - TEA constant chosen to be the real part of the golden ratio
*       Sqrt(5/4) - 1/2 ~ 0.618034 multiplied by 2^32
*
* Arguments: None
*
* Range Issues: Note that it performs the decryption over the 64 bits, so the
*               final buffer to be encrypted should be a multiple of this value!
*
* Special Issues: None
*
*******************************************************************************/
void teaDecode(void)
{
    Word16 i = 32;
    UWord32 sum = 0xC6EF3720;   // sum = teaDelta << 5
```

Source Code Files

```
                                    // in general sum = teaDelta * n;
    while(i-- > 0)
    {
        z -= (y<<4 ^ y>>5) + y ^ sum + k[sum>>11 & 3];
        sum -= TeaDelta;
        y -= (z<<4 ^ z>>5) + z ^ sum + k[sum&3];
    }
}

/*****************************************************************************
*
* Module: void teaEncryptBuff(UWord16 *ptr, UWord16 roundLen)
*
* Description:
*        This function calls the TEA encryption algorithm and move the data to
*        and back to the temp buffer
*
* Returns: None
*
* Global Data:
*       tea_IO - 64bits long buffer for TEA computation
*       For the global data description of the teaCode() routine see the
*       description of the routine itself

* Arguments:
*       *ptr - pointer to the data-buffer
*       roundLen - the length of the buffer to be encrypted (must be a multiple
*           of 8, this project uses following length values: 16, 24 and 32)
*
* Range Issues: roundLen value has to be a multiple of 8
*
* Special Issues: None
*
*****************************************************************************/
void teaEncryptBuff(UWord16 *ptr, UWord16 roundLen)
{
    UWord16 i;
    UWord16 j = 1;
    UWord16 *backPtr;       // pointer for back transfer

    backPtr = ptr;          // save a pointer
    do
    {
        for (i = 0; i < 4; i++)                     // 8bit => 16bit
            tea_IO.w[i] = *ptr++ + (*ptr++ << 8);   // just 8bit values at *Ptr
        teaCode();                                  // perform an encryption
        for (i = 0; i < 4; i++)
        {
            *backPtr++ = (tea_IO.w[i] & 0x00FF);        // 16bit => 8bit
            *backPtr++ = (tea_IO.w[i] & 0xFF00) >> 8;
        }
```

```
    } while ( 8*j++ < roundLen);    // the length is a multiple of 8
}

/*****************************************************************************
*
* Module: void teaDecryptBuff(UWord16 *ptr, UWord16 roundLen)
*
* Description:
*        This function calls the TEA decryption algorithm and move the data to
*        and back to the temp buffer
*
* Returns: None
*
* Global Data:
*       tea_IO - 64bits long buffer for TEA computation
*       For the global data description of the teaCode() routine see the
*       description of the routine itself

* Arguments:
*       *ptr - pointer to the data-buffer
*       roundLen - the length of the buffer to be encrypted (must be a multiple
*           of 8, this project uses following length values: 16, 24 and 32)
*
* Range Issues: roundLen value has to be a multiple of 8
*
* Special Issues: None
*
*****************************************************************************/
void teaDecryptBuff(UWord16 *ptr, UWord16 roundLen)
{
    UWord16 i;
    UWord16 j = 1;
    UWord16 *backPtr;       // pointer for back transfer

    backPtr = ptr;          // save a pointer
    do
    {
        for (i = 0; i < 4; i++)                     // 8bit => 16bit
            tea_IO.w[i] = *ptr++ + (*ptr++ << 8);   // just 8bit values at *Ptr
        teaDecode();                                // perform an encryption
        for (i = 0; i < 4; i++)
        {
            *backPtr++ = (tea_IO.w[i] & 0x00FF);        // 16bit => 8bit
            *backPtr++ = (tea_IO.w[i] & 0xFF00) >> 8;
        }
    } while ( 8*j++ < roundLen);    // the length is a multiple of 8
}
```

C.14 tea.h

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: tea.h
*
* DESCRIPTION: This file is a header file for tea.c
*
* MODULES INCLUDED: None
*
*****************************************************************************/
#ifndef _TEA_H
#define _TEA_H

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "pl.h"
#include "types.h"

/*****************************************************************************/
/*              S T R U C T U R E S                                          */
/*****************************************************************************/
typedef union           /* TEA buffer */
{
    UWord16 w[4];
    UWord32 dw[2];
} tea_uIO;

typedef union           /* TEA Encryption key */
{
    UWord16 w[8];
    UWord32 dw[4];
} tea_uKey;

/*****************************************************************************/
/* L O C A L   D E F I N E S                                                 */
/*****************************************************************************/
#define TeaDelta 0x9E3779B9 /* TEA constant chosen to be the real part of the
                  golden ratio Sqrt(5/4) - 1/2 ~ 0.618034 multiplied by 2^32 */

#define y tea_IO.dw[0]      /* a short-cut define for tea_IO */
#define z tea_IO.dw[1]      /* a short-cut define for tea_IO */
#define k pl_TeaKey.dw      /* a short-cut define for pl_TeaKey */
```

```
/*************************************************************************/
/*              P R O T O T Y P E S                                      */
/*************************************************************************/
    void teaEncryptBuff(UWord16 *Ptr, UWord16 RoundLen);
    void teaDecryptBuff(UWord16 *Ptr, UWord16 RoundLen);

#endif
```

C.15 CRCtable.c

```
/*************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*************************************************************************
*
* FILE NAME: CRCtable.c
*
* DESCRIPTION: This file contains table of the 16bit CRC codes. Linker command
*       file locates this file either into the XFlash data memory area (56F801
*       source or FLASH target in 56F803 source) or to internal RAM data memory
*       area (RAM target in 56F803 source).
*
* MODULES INCLUDED: None
*
*************************************************************************/

/*************************************************************************/
/*              I N C L U D E S                                          */
/*************************************************************************/
#include "types.h"

/*************************************************************************/
/* CRC Constants (Look-up table located at XFlash or XRAM data memory area)   */
/*************************************************************************/
/*
CRC-Berechnung und Implementierungstips

Dies ist nicht der richtige Ort, um die Theorie der zyklischen Redundanz-
überprüfung (cyclic redundancy check, CRC) zu erläutern. Hierzu sei auf
die Arbeit von Michael Röhner, DC4OX [2] verwiesen. Dieser Abschnitt
schildert nur die für eine Implementierung notwendigen Details.

Als Prüfpolynom wird das CRC16-Polynom verwendet. Dieses hat die Gestalt

         16    15   2
```

```
        X   + X  + X  + 1

Der CRC-Generator wird mit 0 vorbesetzt. Berechnet wird der CRC über alle
Datenbytes einschließlich des Kommandobytes 0x80.

Bekanntlich wird im KISS-Protokoll die Abgrenzung der Rahmen mit dem
FEND-Zeichen (0xc0) durchgeführt. Der Fall, daß dieses Zeichen im
Datenstrom vorkommt, wird gesondert behandelt. Dieser Vorgang wird
SLIP-Encoding genannt.

Der CRC muß berechnet werden, bevor das SLIP-Encoding stattfindet, und
ueberprüft werden, nachdem das SLIP-Decoding stattgefunden hat. Dafür
gib es mehrere Gründe:

- Die CRC Bytes könnten FESC, TFEND, FEND usw enthalten.
- Der SLIP En/Decoder wird in manchen Host-Implementierungen (z.B. WAMPES)
  auch  unabhängig von KISS benutzt, um beispielsweise die Verbindung zum
  Unix-Kernel herzustellen. In diesem Fall wären CRC-Überprüfungen zwar
  auch wünschenswert, werden aber von der anderen Seite nicht verstanden.

Die CRCs gehören also logisch zum KISS Layer.

Die Berechnung findet wie folgt statt:
- CRC-Generator mit 0 vorbesetzen.
- Alle Datenbytes nacheinander in den Algorithmus hineintun, einschließlich
  der beiden CRC-Bytes.
- Am Ende muß wieder 0 im CRC-Generator stehen. Ist der Wert ungleich 0, so
  ist ein Übertragungsfehler aufgetreten und der Rahmen muß verworfen werden.

Verschiedene Algorithmen für den CRC-Generator werden in [2] beschrieben.
Hier sei ein einfacher tabellengesteuerter Algorithmus in der Programmier-
sprache C angegeben, der den CRC eines Puffers (buf) der Länge n berechnet.

<snip>

Literatur

[1] Karn, Phil, KA9Q; Proposed "Raw" TNC Functional Spec, 6.8.1986;
    veröffentlicht in den USENET-News;
[2] Röhner, Michael, DC4OX; Was ist CRC?; veröffentlicht im Packet-Radio
    Mailbox-Netz, Mai 1988
[3] FTP Software, Inc.; PC/TCP Version 1.09 Packet Driver Specification;
    Wakefield, MA 1989
[4] Schiefer, Jan, DL5UE; WAMPES - Weiterentwicklung; Vortrags-Skriptum
    des 5. überregionalen Packet-Radio-Treffens; Frankfurt 1989;
*/
const UWord16 CRCtable[256] =
{
    0x0000, 0xc0c1, 0xc181, 0x0140, 0xc301, 0x03c0, 0x0280, 0xc241,
    0xc601, 0x06c0, 0x0780, 0xc741, 0x0500, 0xc5c1, 0xc481, 0x0440,
    0xcc01, 0x0cc0, 0x0d80, 0xcd41, 0x0f00, 0xcfc1, 0xce81, 0x0e40,
```

```
    0x0a00, 0xcac1, 0xcb81, 0x0b40, 0xc901, 0x09c0, 0x0880, 0xc841,
    0xd801, 0x18c0, 0x1980, 0xd941, 0x1b00, 0xdbc1, 0xda81, 0x1a40,
    0x1e00, 0xdec1, 0xdf81, 0x1f40, 0xdd01, 0x1dc0, 0x1c80, 0xdc41,
    0x1400, 0xd4c1, 0xd581, 0x1540, 0xd701, 0x17c0, 0x1680, 0xd641,
    0xd201, 0x12c0, 0x1380, 0xd341, 0x1100, 0xd1c1, 0xd081, 0x1040,
    0xf001, 0x30c0, 0x3180, 0xf141, 0x3300, 0xf3c1, 0xf281, 0x3240,
    0x3600, 0xf6c1, 0xf781, 0x3740, 0xf501, 0x35c0, 0x3480, 0xf441,
    0x3c00, 0xfcc1, 0xfd81, 0x3d40, 0xff01, 0x3fc0, 0x3e80, 0xfe41,
    0xfa01, 0x3ac0, 0x3b80, 0xfb41, 0x3900, 0xf9c1, 0xf881, 0x3840,
    0x2800, 0xe8c1, 0xe981, 0x2940, 0xeb01, 0x2bc0, 0x2a80, 0xea41,
    0xee01, 0x2ec0, 0x2f80, 0xef41, 0x2d00, 0xedc1, 0xec81, 0x2c40,
    0xe401, 0x24c0, 0x2580, 0xe541, 0x2700, 0xe7c1, 0xe681, 0x2640,
    0x2200, 0xe2c1, 0xe381, 0x2340, 0xe101, 0x21c0, 0x2080, 0xe041,
    0xa001, 0x60c0, 0x6180, 0xa141, 0x6300, 0xa3c1, 0xa281, 0x6240,
    0x6600, 0xa6c1, 0xa781, 0x6740, 0xa501, 0x65c0, 0x6480, 0xa441,
    0x6c00, 0xacc1, 0xad81, 0x6d40, 0xaf01, 0x6fc0, 0x6e80, 0xae41,
    0xaa01, 0x6ac0, 0x6b80, 0xab41, 0x6900, 0xa9c1, 0xa881, 0x6840,
    0x7800, 0xb8c1, 0xb981, 0x7940, 0xbb01, 0x7bc0, 0x7a80, 0xba41,
    0xbe01, 0x7ec0, 0x7f80, 0xbf41, 0x7d00, 0xbdc1, 0xbc81, 0x7c40,
    0xb401, 0x74c0, 0x7580, 0xb541, 0x7700, 0xb7c1, 0xb681, 0x7640,
    0x7200, 0xb2c1, 0xb381, 0x7340, 0xb101, 0x71c0, 0x7080, 0xb041,
    0x5000, 0x90c1, 0x9181, 0x5140, 0x9301, 0x53c0, 0x5280, 0x9241,
    0x9601, 0x56c0, 0x5780, 0x9741, 0x5500, 0x95c1, 0x9481, 0x5440,
    0x9c01, 0x5cc0, 0x5d80, 0x9d41, 0x5f00, 0x9fc1, 0x9e81, 0x5e40,
    0x5a00, 0x9ac1, 0x9b81, 0x5b40, 0x9901, 0x59c0, 0x5880, 0x9841,
    0x8801, 0x48c0, 0x4980, 0x8941, 0x4b00, 0x8bc1, 0x8a81, 0x4a40,
    0x4e00, 0x8ec1, 0x8f81, 0x4f40, 0x8d01, 0x4dc0, 0x4c80, 0x8c41,
    0x4400, 0x84c1, 0x8581, 0x4540, 0x8701, 0x47c0, 0x4680, 0x8641,
    0x8201, 0x42c0, 0x4380, 0x8341, 0x4100, 0x81c1, 0x8081, 0x4040
};
```

C.16 FECtable.c

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* FILE NAME: FECtable.c
*
* DESCRIPTION: This file contains table of the linear block Forward error
*       correction codes. Linker command file locates this file either into the
*       XFlash data memory area (56F801 source or FLASH target in 56F803 source)
*       or to internal RAM area (RAM target in 56F803 source)
*
* MODULES INCLUDED: None
```

Source Code Files

```
*
*****************************************************************************/

/*****************************************************************************/
/*              I N C L U D E S                                              */
/*****************************************************************************/
#include "types.h"

/*****************************************************************************/
/* FEC Constants (Look-up table located at XFlash or XRAM data memory area)  */
/*****************************************************************************/
/* Linear Block Code taken from the following address:
/* http://www.tisl.ukans.edu/~paden/Reference/ECC/linear/index.html */
    const UWord16 FECtableCoder[16] =
    {
        0x00,   //  0   0 0 0 0 0 0 0   0 0 0 0
        0x51,   //  1   1 0 1 0 0 0 1   0 0 0 1
        0x72,   //  2   1 1 1 0 0 1 0   0 0 1 0
        0x23,   //  3   0 1 0 0 0 1 1   0 0 1 1
        0x34,   //  4   0 1 1 0 1 0 0   0 1 0 0
        0x65,   //  5   1 1 0 0 1 0 1   0 1 0 1
        0x46,   //  6   1 0 0 0 1 1 0   0 1 1 0
        0x17,   //  7   0 0 1 0 1 1 1   0 1 1 1
        0x68,   //  8   1 1 0 1 0 0 0   1 0 0 0
        0x39,   //  9   0 1 1 1 0 0 1   1 0 0 1
        0x1A,   //  A   0 0 1 1 0 1 0   1 0 1 0
        0x4B,   //  B   1 0 0 1 0 1 1   1 0 1 1
        0x5C,   //  C   1 0 1 1 1 0 0   1 1 0 0
        0x0D,   //  D   0 0 0 1 1 0 1   1 1 0 1
        0x2E,   //  E   0 1 0 1 1 1 0   1 1 1 0
        0x7F,   //  F   1 1 1 1 1 1 1   1 1 1 1
    };

    const UWord16 FECtableDecoder[128] =
    {
        0x0, 0x0, 0x0, 0x3, 0x0, 0xD, 0x6, 0x7,
        0x0, 0xD, 0xA, 0xB, 0xD, 0xD, 0xE, 0xD,
        0x0, 0x1, 0xA, 0x7, 0x4, 0x7, 0x7, 0x7,
        0xA, 0x9, 0xA, 0xA, 0xC, 0xD, 0xA, 0x7,
        0x0, 0x3, 0x3, 0x3, 0x4, 0x5, 0xE, 0x3,
        0x8, 0x9, 0xE, 0x3, 0xE, 0xD, 0xE, 0xE,
        0x4, 0x9, 0x2, 0x3, 0x4, 0x4, 0x4, 0x7,
        0x9, 0x9, 0xA, 0x9, 0x4, 0x9, 0xE, 0xF,
        0x0, 0x1, 0x6, 0xB, 0x6, 0x5, 0x6, 0x6,
        0x8, 0xB, 0xB, 0xB, 0xC, 0xD, 0x6, 0xB,
        0x1, 0x1, 0x2, 0x1, 0xC, 0x1, 0x6, 0x7,
        0xC, 0x1, 0xA, 0xB, 0xC, 0xC, 0xC, 0xF,
        0x8, 0x5, 0x2, 0x3, 0x5, 0x5, 0x6, 0x5,
        0x8, 0x8, 0x8, 0xB, 0x8, 0x5, 0xE, 0xF,
        0x2, 0x1, 0x2, 0x2, 0x4, 0x5, 0x2, 0xF,
        0x8, 0x9, 0x2, 0xF, 0xC, 0xF, 0xF, 0xF,
```

```
    };
```

C.17 demfskconst.c

```
/*************************************************************************
*
* Motorola Inc.
* (c) Copyright 2001 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*************************************************************************
*
* FILE NAME: demfskconst.c
*
* DESCRIPTION: This file contains the FSK demodulator constants. Linker command
*       file locates this file either into the XFlash data memory area (56F801
*       source or FLASH target in 56F803 source) or to internal RAM data memory
*       area (RAM target in 56F803 source)

* MODULES INCLUDED: None
*
*************************************************************************/

/*************************************************************************/
/*              I N C L U D E S                                          */
/*************************************************************************/
#include "types.h"
#include "demfsk.h"

/*************************************************************************/
/* Coefficient Look-up tables (located at XFlash or XRAM data memory area)    */
/*************************************************************************/
/* { Omega = 2 * pi * f / fsamp }
/* { e ^ ( -j * Omega * n) }
/* where:   f is the nominal frequency
/*          fsamp is the sample frequency
/*          n is the integer number from range <0, 49> */
/*************************************************************************/
/* f = 100kHz                                                            */
/*************************************************************************/
const Word16 K100[2 * DEMFSK_FRAMELEN] = {
   /* real part          imag part */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 0 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 1 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 2 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 3 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 4 */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 5 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 6 */
```

Source Code Files

```
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 7 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 8 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 9 */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 10 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 11 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 12 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 13 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 14 */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 15 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 16 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 17 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 18 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 19 */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 20 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 21 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 22 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 23 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 24 */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 25 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 26 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 27 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 28 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 29 */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 30 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 31 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 32 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 33 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 34 */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 35 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 36 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 37 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 38 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 39 */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 40 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 41 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 42 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 43 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 44 */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 45 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 46 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 47 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 48 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 49 */
};

/*****************************************************************************/
/* f = 105kHz                                                                */
/*****************************************************************************/
const Word16 K105[2 * DEMFSK_FRAMELEN] = {
   /* real part          imag part */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 0 */
```

```
    FRAC16(+0.248690), FRAC16(-0.968583),  /* 1 */
    FRAC16(-0.876307), FRAC16(-0.481754),  /* 2 */
    FRAC16(-0.684547), FRAC16(+0.728969),  /* 3 */
    FRAC16(+0.535827), FRAC16(+0.844328),  /* 4 */
    FRAC16(+0.951057), FRAC16(-0.309017),  /* 5 */
    FRAC16(-0.062791), FRAC16(-0.998027),  /* 6 */
    FRAC16(-0.982287), FRAC16(-0.187381),  /* 7 */
    FRAC16(-0.425779), FRAC16(+0.904827),  /* 8 */
    FRAC16(+0.770513), FRAC16(+0.637424),  /* 9 */
    FRAC16(+0.809017), FRAC16(-0.587785),  /* 10 */
    FRAC16(-0.368125), FRAC16(-0.929776),  /* 11 */
    FRAC16(-0.992115), FRAC16(+0.125333),  /* 12 */
    FRAC16(-0.125333), FRAC16(+0.992115),  /* 13 */
    FRAC16(+0.929776), FRAC16(+0.368125),  /* 14 */
    FRAC16(+0.587785), FRAC16(-0.809017),  /* 15 */
    FRAC16(-0.637424), FRAC16(-0.770513),  /* 16 */
    FRAC16(-0.904827), FRAC16(+0.425779),  /* 17 */
    FRAC16(+0.187381), FRAC16(+0.982287),  /* 18 */
    FRAC16(+0.998027), FRAC16(+0.062791),  /* 19 */
    FRAC16(+0.309017), FRAC16(-0.951057),  /* 20 */
    FRAC16(-0.844328), FRAC16(-0.535827),  /* 21 */
    FRAC16(-0.728969), FRAC16(+0.684547),  /* 22 */
    FRAC16(+0.481754), FRAC16(+0.876307),  /* 23 */
    FRAC16(+0.968583), FRAC16(-0.248690),  /* 24 */
    FRAC16(-0.000000), FRAC16(-1.000000),  /* 25 */
    FRAC16(-0.968583), FRAC16(-0.248690),  /* 26 */
    FRAC16(-0.481754), FRAC16(+0.876307),  /* 27 */
    FRAC16(+0.728969), FRAC16(+0.684547),  /* 28 */
    FRAC16(+0.844328), FRAC16(-0.535827),  /* 29 */
    FRAC16(-0.309017), FRAC16(-0.951057),  /* 30 */
    FRAC16(-0.998027), FRAC16(+0.062791),  /* 31 */
    FRAC16(-0.187381), FRAC16(+0.982287),  /* 32 */
    FRAC16(+0.904827), FRAC16(+0.425779),  /* 33 */
    FRAC16(+0.637424), FRAC16(-0.770513),  /* 34 */
    FRAC16(-0.587785), FRAC16(-0.809017),  /* 35 */
    FRAC16(-0.929776), FRAC16(+0.368125),  /* 36 */
    FRAC16(+0.125333), FRAC16(+0.992115),  /* 37 */
    FRAC16(+0.992115), FRAC16(+0.125333),  /* 38 */
    FRAC16(+0.368125), FRAC16(-0.929776),  /* 39 */
    FRAC16(-0.809017), FRAC16(-0.587785),  /* 40 */
    FRAC16(-0.770513), FRAC16(+0.637424),  /* 41 */
    FRAC16(+0.425779), FRAC16(+0.904827),  /* 42 */
    FRAC16(+0.982287), FRAC16(-0.187381),  /* 43 */
    FRAC16(+0.062791), FRAC16(-0.998027),  /* 44 */
    FRAC16(-0.951057), FRAC16(-0.309017),  /* 45 */
    FRAC16(-0.535827), FRAC16(+0.844328),  /* 46 */
    FRAC16(+0.684547), FRAC16(+0.728969),  /* 47 */
    FRAC16(+0.876307), FRAC16(-0.481754),  /* 48 */
    FRAC16(-0.248690), FRAC16(-0.968583),  /* 49 */
};
```

Source Code Files

```
/*****************************************************************************/
/* f = 110kHz                                                                */
/*****************************************************************************/
const Word16 K110[2 * DEMFSK_FRAMELEN] = {
   /* real part          imag part */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 0 */
   FRAC16(+0.187381), FRAC16(-0.982287),  /* 1 */
   FRAC16(-0.929776), FRAC16(-0.368125),  /* 2 */
   FRAC16(-0.535827), FRAC16(+0.844328),  /* 3 */
   FRAC16(+0.728969), FRAC16(+0.684547),  /* 4 */
   FRAC16(+0.809017), FRAC16(-0.587785),  /* 5 */
   FRAC16(-0.425779), FRAC16(-0.904827),  /* 6 */
   FRAC16(-0.968583), FRAC16(+0.248690),  /* 7 */
   FRAC16(+0.062791), FRAC16(+0.998027),  /* 8 */
   FRAC16(+0.992115), FRAC16(+0.125333),  /* 9 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 10 */
   FRAC16(-0.876307), FRAC16(-0.481754),  /* 11 */
   FRAC16(-0.637424), FRAC16(+0.770513),  /* 12 */
   FRAC16(+0.637424), FRAC16(+0.770513),  /* 13 */
   FRAC16(+0.876307), FRAC16(-0.481754),  /* 14 */
   FRAC16(-0.309017), FRAC16(-0.951057),  /* 15 */
   FRAC16(-0.992115), FRAC16(+0.125333),  /* 16 */
   FRAC16(-0.062791), FRAC16(+0.998027),  /* 17 */
   FRAC16(+0.968583), FRAC16(+0.248690),  /* 18 */
   FRAC16(+0.425779), FRAC16(-0.904827),  /* 19 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 20 */
   FRAC16(-0.728969), FRAC16(+0.684547),  /* 21 */
   FRAC16(+0.535827), FRAC16(+0.844328),  /* 22 */
   FRAC16(+0.929776), FRAC16(-0.368125),  /* 23 */
   FRAC16(-0.187381), FRAC16(-0.982287),  /* 24 */
   FRAC16(-1.000000), FRAC16(-0.000000),  /* 25 */
   FRAC16(-0.187381), FRAC16(+0.982287),  /* 26 */
   FRAC16(+0.929776), FRAC16(+0.368125),  /* 27 */
   FRAC16(+0.535827), FRAC16(-0.844328),  /* 28 */
   FRAC16(-0.728969), FRAC16(-0.684547),  /* 29 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 30 */
   FRAC16(+0.425779), FRAC16(+0.904827),  /* 31 */
   FRAC16(+0.968583), FRAC16(-0.248690),  /* 32 */
   FRAC16(-0.062791), FRAC16(-0.998027),  /* 33 */
   FRAC16(-0.992115), FRAC16(-0.125333),  /* 34 */
   FRAC16(-0.309017), FRAC16(+0.951057),  /* 35 */
   FRAC16(+0.876307), FRAC16(+0.481754),  /* 36 */
   FRAC16(+0.637424), FRAC16(-0.770513),  /* 37 */
   FRAC16(-0.637424), FRAC16(-0.770513),  /* 38 */
   FRAC16(-0.876307), FRAC16(+0.481754),  /* 39 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 40 */
   FRAC16(+0.992115), FRAC16(-0.125333),  /* 41 */
   FRAC16(+0.062791), FRAC16(-0.998027),  /* 42 */
   FRAC16(-0.968583), FRAC16(-0.248690),  /* 43 */
   FRAC16(-0.425779), FRAC16(+0.904827),  /* 44 */
   FRAC16(+0.809017), FRAC16(+0.587785),  /* 45 */
```

```
   FRAC16(+0.728969), FRAC16(-0.684547),  /* 46 */
   FRAC16(-0.535827), FRAC16(-0.844328),  /* 47 */
   FRAC16(-0.929776), FRAC16(+0.368125),  /* 48 */
   FRAC16(+0.187381), FRAC16(+0.982287),  /* 49 */
};

/*****************************************************************************/
/* f = 115kHz                                                                */
/*****************************************************************************/
const Word16 K115[2 * DEMFSK_FRAMELEN] = {
   /* real part          imag part */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 0 */
   FRAC16(+0.125333), FRAC16(-0.992115),  /* 1 */
   FRAC16(-0.968583), FRAC16(-0.248690),  /* 2 */
   FRAC16(-0.368125), FRAC16(+0.929776),  /* 3 */
   FRAC16(+0.876307), FRAC16(+0.481754),  /* 4 */
   FRAC16(+0.587785), FRAC16(-0.809017),  /* 5 */
   FRAC16(-0.728969), FRAC16(-0.684547),  /* 6 */
   FRAC16(-0.770513), FRAC16(+0.637424),  /* 7 */
   FRAC16(+0.535827), FRAC16(+0.844328),  /* 8 */
   FRAC16(+0.904827), FRAC16(-0.425779),  /* 9 */
   FRAC16(-0.309017), FRAC16(-0.951057),  /* 10 */
   FRAC16(-0.982287), FRAC16(+0.187381),  /* 11 */
   FRAC16(+0.062791), FRAC16(+0.998027),  /* 12 */
   FRAC16(+0.998027), FRAC16(+0.062791),  /* 13 */
   FRAC16(+0.187381), FRAC16(-0.982287),  /* 14 */
   FRAC16(-0.951057), FRAC16(-0.309017),  /* 15 */
   FRAC16(-0.425779), FRAC16(+0.904827),  /* 16 */
   FRAC16(+0.844328), FRAC16(+0.535827),  /* 17 */
   FRAC16(+0.637424), FRAC16(-0.770513),  /* 18 */
   FRAC16(-0.684547), FRAC16(-0.728969),  /* 19 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 20 */
   FRAC16(+0.481754), FRAC16(+0.876307),  /* 21 */
   FRAC16(+0.929776), FRAC16(-0.368125),  /* 22 */
   FRAC16(-0.248690), FRAC16(-0.968583),  /* 23 */
   FRAC16(-0.992115), FRAC16(+0.125333),  /* 24 */
   FRAC16(-0.000000), FRAC16(+1.000000),  /* 25 */
   FRAC16(+0.992115), FRAC16(+0.125333),  /* 26 */
   FRAC16(+0.248690), FRAC16(-0.968583),  /* 27 */
   FRAC16(-0.929776), FRAC16(-0.368125),  /* 28 */
   FRAC16(-0.481754), FRAC16(+0.876307),  /* 29 */
   FRAC16(+0.809017), FRAC16(+0.587785),  /* 30 */
   FRAC16(+0.684547), FRAC16(-0.728969),  /* 31 */
   FRAC16(-0.637424), FRAC16(-0.770513),  /* 32 */
   FRAC16(-0.844328), FRAC16(+0.535827),  /* 33 */
   FRAC16(+0.425779), FRAC16(+0.904827),  /* 34 */
   FRAC16(+0.951057), FRAC16(-0.309017),  /* 35 */
   FRAC16(-0.187381), FRAC16(-0.982287),  /* 36 */
   FRAC16(-0.998027), FRAC16(+0.062791),  /* 37 */
   FRAC16(-0.062791), FRAC16(+0.998027),  /* 38 */
   FRAC16(+0.982287), FRAC16(+0.187381),  /* 39 */
```

Source Code Files

```
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 40 */
   FRAC16(-0.904827), FRAC16(-0.425779),  /* 41 */
   FRAC16(-0.535827), FRAC16(+0.844328),  /* 42 */
   FRAC16(+0.770513), FRAC16(+0.637424),  /* 43 */
   FRAC16(+0.728969), FRAC16(-0.684547),  /* 44 */
   FRAC16(-0.587785), FRAC16(-0.809017),  /* 45 */
   FRAC16(-0.876307), FRAC16(+0.481754),  /* 46 */
   FRAC16(+0.368125), FRAC16(+0.929776),  /* 47 */
   FRAC16(+0.968583), FRAC16(-0.248690),  /* 48 */
   FRAC16(-0.125333), FRAC16(-0.992115),  /* 49 */
};

/*****************************************************************************/
/* f = 120kHz                                                                */
/*****************************************************************************/
const Word16 K120[2 * DEMFSK_FRAMELEN] = {
   /* real part          imag part */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 0 */
   FRAC16(+0.062791), FRAC16(-0.998027),  /* 1 */
   FRAC16(-0.992115), FRAC16(-0.125333),  /* 2 */
   FRAC16(-0.187381), FRAC16(+0.982287),  /* 3 */
   FRAC16(+0.968583), FRAC16(+0.248690),  /* 4 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 5 */
   FRAC16(-0.929776), FRAC16(-0.368125),  /* 6 */
   FRAC16(-0.425779), FRAC16(+0.904827),  /* 7 */
   FRAC16(+0.876307), FRAC16(+0.481754),  /* 8 */
   FRAC16(+0.535827), FRAC16(-0.844328),  /* 9 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 10 */
   FRAC16(-0.637424), FRAC16(+0.770513),  /* 11 */
   FRAC16(+0.728969), FRAC16(+0.684547),  /* 12 */
   FRAC16(+0.728969), FRAC16(-0.684547),  /* 13 */
   FRAC16(-0.637424), FRAC16(-0.770513),  /* 14 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 15 */
   FRAC16(+0.535827), FRAC16(+0.844328),  /* 16 */
   FRAC16(+0.876307), FRAC16(-0.481754),  /* 17 */
   FRAC16(-0.425779), FRAC16(-0.904827),  /* 18 */
   FRAC16(-0.929776), FRAC16(+0.368125),  /* 19 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 20 */
   FRAC16(+0.968583), FRAC16(-0.248690),  /* 21 */
   FRAC16(-0.187381), FRAC16(-0.982287),  /* 22 */
   FRAC16(-0.992115), FRAC16(+0.125333),  /* 23 */
   FRAC16(+0.062791), FRAC16(+0.998027),  /* 24 */
   FRAC16(+1.000000), FRAC16(+0.000000),  /* 25 */
   FRAC16(+0.062791), FRAC16(-0.998027),  /* 26 */
   FRAC16(-0.992115), FRAC16(-0.125333),  /* 27 */
   FRAC16(-0.187381), FRAC16(+0.982287),  /* 28 */
   FRAC16(+0.968583), FRAC16(+0.248690),  /* 29 */
   FRAC16(+0.309017), FRAC16(-0.951057),  /* 30 */
   FRAC16(-0.929776), FRAC16(-0.368125),  /* 31 */
   FRAC16(-0.425779), FRAC16(+0.904827),  /* 32 */
   FRAC16(+0.876307), FRAC16(+0.481754),  /* 33 */
```

```
   FRAC16(+0.535827), FRAC16(-0.844328),  /* 34 */
   FRAC16(-0.809017), FRAC16(-0.587785),  /* 35 */
   FRAC16(-0.637424), FRAC16(+0.770513),  /* 36 */
   FRAC16(+0.728969), FRAC16(+0.684547),  /* 37 */
   FRAC16(+0.728969), FRAC16(-0.684547),  /* 38 */
   FRAC16(-0.637424), FRAC16(-0.770513),  /* 39 */
   FRAC16(-0.809017), FRAC16(+0.587785),  /* 40 */
   FRAC16(+0.535827), FRAC16(+0.844328),  /* 41 */
   FRAC16(+0.876307), FRAC16(-0.481754),  /* 42 */
   FRAC16(-0.425779), FRAC16(-0.904827),  /* 43 */
   FRAC16(-0.929776), FRAC16(+0.368125),  /* 44 */
   FRAC16(+0.309017), FRAC16(+0.951057),  /* 45 */
   FRAC16(+0.968583), FRAC16(-0.248690),  /* 46 */
   FRAC16(-0.187381), FRAC16(-0.982287),  /* 47 */
   FRAC16(-0.992115), FRAC16(+0.125333),  /* 48 */
   FRAC16(+0.062791), FRAC16(+0.998027),  /* 49 */
};
```

C.18 appconfig.h

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2000 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* File Name: appconfig.h
*
* Description: file for static configuration of the application
*              (initial values, interrupt vectors)
*
* Modules Included:
*
*****************************************************************************/

#ifndef __APPCONFIG_H
#define __APPCONFIG_H

/*.*************************************************************
*
*    RADEGAST configuration file generated by Hawk Configuration Tool
*
*************************************************************.*/

#define   DSP56F801
#define   EXTCLK   8000000L
```

Source Code Files

```
/*.
    OCCS, COP & External interrupts  configuration
    ----------------------------------------
    Core freq.=80.000 MHz, IPBus freq.=40.000 MHz
    COP disabled, COP period =838.86 ms
    External interrupts:  None
.*/
#define COP_TIMEOUT_REG            0x0fff
extern void Start(void);
#define INT_VECTOR_ADDR_1          Start


/*.
    Quad Timer C2 configuration
    ---------------------------
    Count mode: No operation
    Primary count source: Prescaler (IP BUS clock divide by 1)
    Secondary count source: Counter #0 input pin
    Input polarity: True polarity
    Output polarity: True polarity
    Input capture mode: Capture disabled, input edge flag INTdisabled
    Output capture mode: Toggle OFLAG output on succesful compare
    Count once: Count repeatedly
    Count direction: Count up
    Coinit disabled, Master mode disabled, Output disabled
    Interrupts:  None
.*/
#define QT_C2_CONTROL_REG          0x1023
#define QT_C2_COMPARE_REG1         0x0027


/*.
    Quad Timer D1 configuration
    ---------------------------
    Count mode: No operation
    Primary count source: Prescaler (IP BUS clock divide by 1)
    Secondary count source: Counter #0 input pin
    Input polarity: True polarity
    Output polarity: True polarity
    Input capture mode: Capture disabled, input edge flag INTdisabled
    Output capture mode: Asserted while counter is active
    Count once: Count repeatedly
    Count direction: Count up
    Coinit disabled, Master mode disabled, Output disabled
    Interrupts:  Compare interrupt
.*/
#define QT_D1_CONTROL_REG          0x1020
#define QT_D1_STATUS_CONTROL_REG   0x4000
extern void tmrfskBitISR(void);
#define INT_VECTOR_ADDR_31         tmrfskBitISR
```

```
#define ITCN_INT_PRIORITY_31       0x0002


/*.
    Quad Timer D2 configuration
    ---------------------------
    Count mode: No operation
    Primary count source: Prescaler (IP BUS clock divide by 1)
    Secondary count source: Counter #0 input pin
    Input polarity: True polarity
    Output polarity: True polarity
    Input capture mode: Capture disabled, input edge flag INTdisabled
    Output capture mode: Toggle OFLAG output on succesful compare
    Count once: Count repeatedly
    Count direction: Count up
    Coinit disabled, Master mode disabled, Output enabled
    Interrupts:  None
.*/
#define QT_D2_CONTROL_REG          0x1023
#define QT_D2_STATUS_CONTROL_REG   0x0001


/*.
    Quad Timer D3 configuration
    ---------------------------
    Count mode: No operation
    Primary count source: Prescaler (IP BUS clock divide by 128)
    Secondary count source: Counter #0 input pin
    Input polarity: True polarity
    Output polarity: True polarity
    Input capture mode: Capture disabled, input edge flag INTdisabled
    Output capture mode: Asserted while counter is active
    Count once: Count until compare and stop
    Count direction: Count up
    Coinit disabled, Master mode disabled, Output disabled
    Interrupts:  Compare interrupt
.*/
#define QT_D3_CONTROL_REG          0x1e60
#define QT_D3_STATUS_CONTROL_REG   0x4000
extern void tmrfskTimeOutISR(void);
#define INT_VECTOR_ADDR_33         tmrfskTimeOutISR
#define ITCN_INT_PRIORITY_33       0x0001


/*.
    Analog to digital converter A configuration
    -------------------------------------------
    Clock frequency = 5.000 MHz
    Trigger source: SYNC input
    Scan mode: Triggered Sequential
    Sample 0 mapped to AN0, zero crossing disabled
```

Source Code Files

```
    Interrupts:  End of scan
.*/
#define ADC_A_CONTROL_REG1          0x1804          /*Stop off */
#define ADC_A_CONTROL_REG2          0x0003
#define ADC_A_CHANNEL_LIST_REG1     0x3210
#define ADC_A_CHANNEL_LIST_REG2     0x7654
#define ADC_A_SAMPLE_DISABLE_REG    0x00fe          /*only Sample 0 is enabled */
#define ADC_A_OFFSET_REG0           0x3ffc          /*Offset for signed results */
extern void demfskEndOfScanISR(void);
#define INT_VECTOR_ADDR_55          demfskEndOfScanISR
#define ITCN_INT_PRIORITY_55        0x0005


/*.
    Serial communication interface 0 configuration
    -----------------------------------------------
    Baud rate: Not defined Bd
    Receiver: enabled
    Transmitter: enabled
    Data word length: 8 bits
    Parity: None
    Polarity: True polarity
    Wake-up condition: By idle
    Wait mode function: SCI disabled in Wait Mode
    Loop mode: Disabled
    Interrupts:  None
.*/
#define SCI_0_CONTROL_REG           0x000c          /*SCI Rx Full ISR disabled */
extern void scicommTxEmpISR(void);
#define INT_VECTOR_ADDR_51          scicommTxEmpISR
#define ITCN_INT_PRIORITY_51        0x0001
extern void scicommRxFullISR(void);
#define INT_VECTOR_ADDR_53          scicommRxFullISR
#define ITCN_INT_PRIORITY_53        0x0001
extern void scicommRxErrISR(void);
#define INT_VECTOR_ADDR_52          scicommRxErrISR
#define ITCN_INT_PRIORITY_52        0x0002


/*.
    GPIO B configuration
    ---------------------
    Pin 0 - Direction: Input, Mode: Peripheral, Pull-up: Disable
    Pin 1 - Direction: Input, Mode: Peripheral, Pull-up: Disable
    Pin 2 - Direction: Input, Mode: Peripheral, Pull-up: Disable
    Pin 3 - Direction: Input, Mode: Peripheral, Pull-up: Disable
    Pin 4 - Direction: Output, Mode: GPIO, Pull-up: Disable
    Pin 5 - Direction: Output, Mode: GPIO, Pull-up: Disable
    Pin 6 - Direction: Output, Mode: GPIO, Pull-up: Disable
    Pin 7 - Direction: Output, Mode: GPIO, Pull-up: Disable
    GPIO interrupt disabled
```

```
.*/
#define GPIO_B_DATA_DIRECTION_REG  0x00f0
#define GPIO_B_PERIPHERAL_ENABLE_REG 0x000f

/*.         End of autogenerated code
********************************************** ..*/

/*****************************************************************************
* N O T E S
*****************************************************************************/
/*      C O P
/* Initially the COP module is disabled by startup.asm code but when there is
/* the global define
/* #define PL_COPINUSE       /* if defined the Watch Dog is used */
/* placed in the pl.h file, it finally switch the COP on */

/*      T M R   D 1
/* There is no definition for TmrD1 Compare register 1 value in appconfig.h
/* configuration, it depends on the global define placed in the pl.h file:
/* #define PL_PLBAUDRATEPL_10000BPS/* choose: PL_10000BPS */

/*      T M R   D 2
/* There is no valid definition of TmrD2 Compare register 1 value in appconfig.h
/* configuration, it depends on the global define placed in the pl.h file:
/* #define PL_CARRIERLOW   CARRIERLOW_110KHZ10KBPS */

/*      T M R   D 3
/* There is a zero value of TmrD3 Compare register 1 written in appconfig.h
/* configuration, this register is filled according the global define
/* #define PL_TIMEOUTVALUE 1000/* time out of SCI receive */
/* placed in the pl.h file */

/*      S C I
/* There is no definition for SCI baudrate value in appconfig.h configuration,
/* it depends on the global define placed in the pl.h file:
/* #define PL_SCIBAUDRATE   SCI_BAUD_38400/* choose: SCI_BAUD_38400 */
                                          /* not tested: SCI_BAUD_4800
                                          SCI_BAUD_9600
                                          SCI_BAUD_19200 */

/*****************************************************************************
*
* Interrupt vectors definition
*
*****************************************************************************/

/*
Example of interrupt vector definition:

         extern void userISRFunction(void);  prototype of the ISR must be
                                             placed in your code
```

```
        #define INT_VECTOR_ADDR_yy          userISRFunction
        #define ITCN_INT_PRIORITY_yy     value 0-7 (0 = disabled,
                                              (1 = lowest  interrupt priority,
                                               7 = highest interrupt priority)

  where:
        yy  is interrupt vector number
*/

/*****************************************************************************
*
* Default components initialization values
*
*****************************************************************************/

/*
Example of initialization values definition for GPIO:

        #define  GPIO_x_PERIPHERAL_ENABLE_REG    0x0000
        #define  GPIO_x_DATA_DIRECTION_REG       0x0000

  where:
        x  is GPIO port
*/

#endif
```

C.19 linker_flash.cmd

```
MEMORY {

    .pflash (RX)  : ORIGIN = 0x0000, LENGTH = 0x2000  # program flash memory
    .pram   (RWX) : ORIGIN = 0x7C00, LENGTH = 0x0400  # program ram memory
    .bflash (RX)  : ORIGIN = 0x8000, LENGTH = 0x0800  # boot flash memory
    .avail  (RW)  : ORIGIN = 0x0000, LENGTH = 0x0030  # available
    .cwregs (RW)  : ORIGIN = 0x0030, LENGTH = 0x0010  # C temp registrs in
                                                      # CodeWarrior
    .data   (RW)  : ORIGIN = 0x0040, LENGTH = 0x02C0  # data
    .stack  (RW)  : ORIGIN = 0x0300, LENGTH = 0x0100  # stack
    .regs   (RW)  : ORIGIN = 0x0C00, LENGTH = 0x0400  # periperal registers
    .xflash (R)   : ORIGIN = 0x1000, LENGTH = 0x0800  # flash memory to place
                                                      # constant and initialized
                                                      # values for data
    .onchip (RW)  : ORIGIN = 0xFF80, LENGTH = 0x0080  # on-chip core
                                                      # configuration registers
}

FORCE_ACTIVE {FconfigInterruptVector, boot_start}
```

```
FORCE_ACTIVE {FK100, FK105, FK110, FK115, FK120}
FORCE_ACTIVE {FprevSample, FpxBuf, Fdemfsk_NewFrmCounter}

SECTIONS {
    .main_Application_code :
    {
        config.c (.text)
        *(Startup.text)
        *(Main.text)
        *(rtlib.text)
        *(fp_engine.text)

        *(.text)
    } > .pflash

    .flash_booting :
    {
        *(Boot.text)
    } > .bflash

    .internal_memory_30:
    {
        OBJECT (FprevSample, demfsk.c)
        OBJECT (FpxBuf, demfsk.c)
        OBJECT (Fdemfsk_NewFrmCounter, demfsk.c)
    } > .avail

    .main_Application_constants :
    {
        _consts_start= .;

        #place your constants here: const.c (.data)
        FECtable.c (.data)          # place constants into the XFlash area
        CRCtable.c (.data)
        demfskconst.c (.data)
        _consts_size= . - _consts_start;
        F_Xdata_start_in_ROM = .;
    } > .xflash

    .main_application_data : AT (ADDR(.xflash)+_consts_size)
    #init values of global variables are placed in xflash after constants
    {
        F_StackAddr    = ADDR(.stack);
        F_StackEndAddr = ADDR(.stack) + SIZEOF(.stack) / 2  - 1;
        F_Xdata_start_in_RAM = .;
        _data_start= .;

        * (.data)
        * (fp_state.data)
        * (rtlib.data)
```

Source Code Files

```
        . = ALIGN(0x80);            # these definitions must be above * (.bss)
        OBJECT (FxBuf, demfsk.c)
        . = ALIGN(0x80);
        OBJECT (FbBuf, demfsk.c)

        * (.bss)
        * (rtlib.bss.lo)

        F_Xdata_size = . - _data_start;
    } > .data

    FArchIO   = ADDR(.regs);
    FArchCore = ADDR(.onchip);
}
```

Appendix D. Glossary

A — See “accumulators (A and B or D).”

accumulators (A and B or D) — Two 8-bit (A and B) or one 16-bit (D) general-purpose registers in the CPU. The CPU uses the accumulators to hold operands and results of arithmetic and logic operations.

acquisition mode — A mode of PLL operation with large loop bandwidth. Also see ’tracking mode’.

address bus — The set of wires that the CPU or DMA uses to read and write memory locations.

addressing mode — The way that the CPU determines the operand address for an instruction. The M68HC12 CPU has 15 addressing modes.

ALU — See “arithmetic logic unit (ALU).”

analogue-to-digital converter (ATD) — The ATD module is an 8-channel, multiplexed-input successive-approximation analog-to-digital converter.

arithmetic logic unit (ALU) — The portion of the CPU that contains the logic circuitry to perform arithmetic, logic, and manipulation operations on operands.

asynchronous — Refers to logic circuits and operations that are not synchronized by a common reference signal.

ATD — See “analogue-to-digital converter”.

B — See “accumulators (A and B or D).”

baud rate — The total number of bits transmitted per unit of time.

BCD — See “binary-coded decimal (BCD).”

binary — Relating to the base 2 number system.

Glossary

binary number system — The base 2 number system, having two digits, 0 and 1. Binary arithmetic is convenient in digital circuit design because digital circuits have two permissible voltage levels, low and high. The binary digits 0 and 1 can be interpreted to correspond to the two digital voltage levels.

binary-coded decimal (BCD) — A notation that uses 4-bit binary numbers to represent the 10 decimal digits and that retains the same positional structure of a decimal number. For example,

234 (decimal) = 0010 0011 0100 (BCD)

bit — A binary digit. A bit has a value of either logic 0 or logic 1.

branch instruction — An instruction that causes the CPU to continue processing at a memory location other than the next sequential address.

break module — The break module allows software to halt program execution at a programmable point in order to enter a background routine.

breakpoint — A number written into the break address registers of the break module. When a number appears on the internal address bus that is the same as the number in the break address registers, the CPU executes the software interrupt instruction (SWI).

break interrupt — A software interrupt caused by the appearance on the internal address bus of the same value that is written in the break address registers.

bus — A set of wires that transfers logic signals.

bus clock — See "CPU clock".

byte — A set of eight bits.

CAN — See "Motorola scalable CAN."

CCR — See “condition code register.”

central processor unit (CPU) — The primary functioning unit of any computer system. The CPU controls the execution of instructions.

CGM — See “clock generator module (CGM).”

clear — To change a bit from logic 1 to logic 0; the opposite of set.

clock — A square wave signal used to synchronize events in a computer.

clock generator module (CGM) — The CGM module generates a base clock signal from which the system clocks are derived. The CGM may include a crystal oscillator circuit and/or phase-locked loop (PLL) circuit.

comparator — A device that compares the magnitude of two inputs. A digital comparator defines the equality or relative differences between two binary numbers.

computer operating properly module (COP) — A counter module that resets the MCU if allowed to overflow.

condition code register (CCR) — An 8-bit register in the CPU that contains the interrupt mask bit and five bits that indicate the results of the instruction just executed.

control bit — One bit of a register manipulated by software to control the operation of the module.

control unit — One of two major units of the CPU. The control unit contains logic functions that synchronize the machine and direct various operations. The control unit decodes instructions and generates the internal control signals that perform the requested operations. The outputs of the control unit drive the execution unit, which contains the arithmetic logic unit (ALU), CPU registers, and bus interface.

COP — See "computer operating properly module (COP)."

CPU — See “central processor unit (CPU).”

CPU12 — The CPU of the MC68HC12 Family.

CPU clock — Bus clock select bits BCSP and BCSS in the clock select register (CLKSEL) determine which clock drives SYSCLK for the main system, including the CPU and buses. When EXTALi drives the SYSCLK, the CPU or bus clock frequency (f_o) is equal to the EXTALi frequency divided by 2.

CPU cycles — A CPU cycle is one period of the internal bus clock, normally derived by dividing a crystal oscillator source by two or more so the high and low times will be equal. The length of time required to execute an instruction is measured in CPU clock cycles.

CPU registers — Memory locations that are wired directly into the CPU logic instead of being part of the addressable memory map. The CPU always has direct access to the information in these registers. The CPU registers in an M68HC12 are:

- A (8-bit accumulator)
- B (8-bit accumulator)
  - D (16-bit accumulator formed by concatenation of accumulators A and B)
- IX (16-bit index register)
- IY (16-bit index register)

Glossary

- SP (16-bit stack pointer)
- PC (16-bit program counter)
- CCR (8-bit condition code register)

cycle time — The period of the operating frequency: $t_{CYC} = 1/f_{OP}$.

D — See “accumulators (A and B or D).”

decimal number system — Base 10 numbering system that uses the digits zero through nine.

duty cycle — A ratio of the amount of time the signal is on versus the time it is off. Duty cycle is usually represented by a percentage.

ECT — See “enhanced capture timer.”

EEPROM — Electrically erasable, programmable, read-only memory. A nonvolatile type of memory that can be electrically erased and reprogrammed.

EPROM — Erasable, programmable, read-only memory. A nonvolatile type of memory that can be erased by exposure to an ultraviolet light source and then reprogrammed.

enhanced capture timer (ECT) — The HC12 Enhanced Capture Timer module has the features of the HC12 Standard Timer module enhanced by additional features in order to enlarge the field of applications.

exception — An event such as an interrupt or a reset that stops the sequential execution of the instructions in the main program.

fetch — To copy data from a memory location into the accumulator.

firmware — Instructions and data programmed into nonvolatile memory.

free-running counter — A device that counts from zero to a predetermined number, then rolls over to zero and begins counting again.

full-duplex transmission — Communication on a channel in which data can be sent and received simultaneously.

hexadecimal — Base 16 numbering system that uses the digits 0 through 9 and the letters A through F.

high byte — The most significant eight bits of a word.

illegal address — An address not within the memory map

illegal opcode — A nonexistent opcode.

index registers (IX and IY) — Two 16-bit registers in the CPU. In the indexed addressing modes, the CPU uses the contents of IX or IY to determine the effective address of the operand. IX and IY can also serve as a temporary data storage locations.

input/output (I/O) — Input/output interfaces between a computer system and the external world. A CPU reads an input to sense the level of an external signal and writes to an output to change the level on an external signal.

instructions — Operations that a CPU can perform. Instructions are expressed by programmers as assembly language mnemonics. A CPU interprets an opcode and its associated operand(s) and instruction.

inter-IC bus (I^2C) — A two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange between devices.

interrupt — A temporary break in the sequential execution of a program to respond to signals from peripheral devices by executing a subroutine.

interrupt request — A signal from a peripheral to the CPU intended to cause the CPU to execute a subroutine.

I/O — See “input/output (I/0).”

jitter — Short-term signal instability.

latch — A circuit that retains the voltage level (logic 1 or logic 0) written to it for as long as power is applied to the circuit.

latency — The time lag between instruction completion and data movement.

least significant bit (LSB) — The rightmost digit of a binary number.

logic 1 — A voltage level approximately equal to the input power voltage (V_{DD}).

logic 0 — A voltage level approximately equal to the ground voltage (V_{SS}).

low byte — The least significant eight bits of a word.

M68HC12 — A Motorola family of 16-bit MCUs.

mark/space — The logic 1/logic 0 convention used in formatting data in serial communication.

mask — 1. A logic circuit that forces a bit or group of bits to a desired state. 2. A photomask used in integrated circuit fabrication to transfer an image onto silicon.

MCU — Microcontroller unit. See “microcontroller.”

Glossary

memory location — Each M68HC12 memory location holds one byte of data and has a unique address. To store information in a memory location, the CPU places the address of the location on the address bus, the data information on the data bus, and asserts the write signal. To read information from a memory location, the CPU places the address of the location on the address bus and asserts the read signal. In response to the read signal, the selected memory location places its data onto the data bus.

memory map — A pictorial representation of all memory locations in a computer system.

MI-Bus — See "Motorola interconnect bus".

microcontroller — Microcontroller unit (MCU). A complete computer system, including a CPU, memory, a clock oscillator, and input/output (I/O) on a single integrated circuit.

modulo counter — A counter that can be programmed to count to any number from zero to its maximum possible modulus.

most significant bit (MSB) — The leftmost digit of a binary number.

Motorola interconnect bus (MI-Bus) — The Motorola Interconnect Bus (MI Bus) is a serial communications protocol which supports distributed real-time control efficiently and with a high degree of noise immunity.

Motorola scalable CAN (msCAN) — The Motorola scalable controller area network is a serial communications protocol that efficiently supports distributed real-time control with a very high level of data integrity.

msCAN — See "Motorola scalable CAN".

MSI — See "multiple serial interface".

multiple serial interface — A module consisting of multiple independent serial I/O sub-systems, e.g. two SCI and one SPI.

multiplexer — A device that can select one of a number of inputs and pass the logic level of that input on to the output.

nibble — A set of four bits (half of a byte).

object code — The output from an assembler or compiler that is itself executable machine code, or is suitable for processing to produce executable machine code.

opcode — A binary code that instructs the CPU to perform an operation.

open-drain — An output that has no pullup transistor. An external pullup device can be connected to the power supply to provide the logic 1 output voltage.

operand — Data on which an operation is performed. Usually a statement consists of an operator and an operand. For example, the operator may be an add instruction, and the operand may be the quantity to be added.

oscillator — A circuit that produces a constant frequency square wave that is used by the computer as a timing and sequencing reference.

OTPROM — One-time programmable read-only memory. A nonvolatile type of memory that cannot be reprogrammed.

overflow — A quantity that is too large to be contained in one byte or one word.

page zero — The first 256 bytes of memory (addresses $0000–$00FF).

parity — An error-checking scheme that counts the number of logic 1s in each byte transmitted. In a system that uses odd parity, every byte is expected to have an odd number of logic 1s. In an even parity system, every byte should have an even number of logic 1s. In the transmitter, a parity generator appends an extra bit to each byte to make the number of logic 1s odd for odd parity or even for even parity. A parity checker in the receiver counts the number of logic 1s in each byte. The parity checker generates an error signal if it finds a byte with an incorrect number of logic 1s.

PC — See “program counter (PC).”

peripheral — A circuit not under direct CPU control.

phase-locked loop (PLL) — A clock generator circuit in which a voltage controlled oscillator produces an oscillation which is synchronized to a reference signal.

PLL — See "phase-locked loop (PLL)."

pointer — Pointer register. An index register is sometimes called a pointer register because its contents are used in the calculation of the address of an operand, and therefore points to the operand.

polarity — The two opposite logic levels, logic 1 and logic 0, which correspond to two different voltage levels, V_{DD} and V_{SS}.

polling — Periodically reading a status bit to monitor the condition of a peripheral device.

port — A set of wires for communicating with off-chip devices.

prescaler — A circuit that generates an output signal related to the input signal by a fractional scale factor such as 1/2, 1/8, 1/10 etc.

program — A set of computer instructions that cause a computer to perform a desired operation or operations.

Glossary

program counter (PC) — A 16-bit register in the CPU. The PC register holds the address of the next instruction or operand that the CPU will use.

pull — An instruction that copies into the accumulator the contents of a stack RAM location. The stack RAM address is in the stack pointer.

pullup — A transistor in the output of a logic gate that connects the output to the logic 1 voltage of the power supply.

pulse-width — The amount of time a signal is on as opposed to being in its off state.

pulse-width modulation (PWM) — Controlled variation (modulation) of the pulse width of a signal with a constant frequency.

push — An instruction that copies the contents of the accumulator to the stack RAM. The stack RAM address is in the stack pointer.

PWM period — The time required for one complete cycle of a PWM waveform.

RAM — Random access memory. All RAM locations can be read or written by the CPU. The contents of a RAM memory location remain valid until the CPU writes a different value or until power is turned off.

RC circuit — A circuit consisting of capacitors and resistors having a defined time constant.

read — To copy the contents of a memory location to the accumulator.

register — A circuit that stores a group of bits.

reserved memory location — A memory location that is used only in special factory test modes. Writing to a reserved location has no effect. Reading a reserved location returns an unpredictable value.

reset — To force a device to a known condition.

SCI — See "serial communication interface module (SCI)."

serial — Pertaining to sequential transmission over a single line.

serial communications interface module (SCI) — A module that supports asynchronous communication.

serial peripheral interface module (SPI) — A module that supports synchronous communication.

set — To change a bit from logic 0 to logic 1; opposite of clear.

shift register — A chain of circuits that can retain the logic levels (logic 1 or logic 0) written to them and that can shift the logic levels to the right or left through adjacent circuits in the chain.

signed — A binary number notation that accommodates both positive and negative numbers. The most significant bit is used to indicate whether the number is positive or negative, normally logic 0 for positive and logic 1 for negative. The other seven bits indicate the magnitude of the number.

software — Instructions and data that control the operation of a microcontroller.

software interrupt (SWI) — An instruction that causes an interrupt and its associated vector fetch.

SPI — See "serial peripheral interface module (SPI)."

stack — A portion of RAM reserved for storage of CPU register contents and subroutine return addresses.

stack pointer (SP) — A 16-bit register in the CPU containing the address of the next available storage location on the stack.

start bit — A bit that signals the beginning of an asynchronous serial transmission.

status bit — A register bit that indicates the condition of a device.

stop bit — A bit that signals the end of an asynchronous serial transmission.

subroutine — A sequence of instructions to be used more than once in the course of a program. The last instruction in a subroutine is a return from subroutine (RTS) instruction. At each place in the main program where the subroutine instructions are needed, a jump or branch to subroutine (JSR or BSR) instruction is used to call the subroutine. The CPU leaves the flow of the main program to execute the instructions in the subroutine. When the RTS instruction is executed, the CPU returns to the main program where it left off.

synchronous — Refers to logic circuits and operations that are synchronized by a common reference signal.

timer — A module used to relate events in a system to a point in time.

toggle — To change the state of an output from a logic 0 to a logic 1 or from a logic 1 to a logic 0.

tracking mode — A mode of PLL operation with narrow loop bandwidth. Also see ‘acquisition mode.’

Glossary

two's complement — A means of performing binary subtraction using addition techniques. The most significant bit of a two's complement number indicates the sign of the number (1 indicates negative). The two's complement negative of a number is obtained by inverting each bit in the number and then adding 1 to the result.

unbuffered — Utilizes only one register for data; new data overwrites current data.

unimplemented memory location — A memory location that is not used. Writing to an unimplemented location has no effect. Reading an unimplemented location returns an unpredictable value.

variable — A value that changes during the course of program execution.

VCO — See "voltage-controlled oscillator."

vector — A memory location that contains the address of the beginning of a subroutine written to service an interrupt or reset.

voltage-controlled oscillator (VCO) — A circuit that produces an oscillating output signal of a frequency that is controlled by a dc voltage applied to a control input.

waveform — A graphical representation in which the amplitude of a wave is plotted against time.

wired-OR — Connection of circuit outputs so that if any output is high, the connection point is high.

word — A set of two bytes (16 bits).

write — The transfer of a byte of data from the CPU to a memory location.

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

DRM035/D

Freescale Semiconductor, Inc.